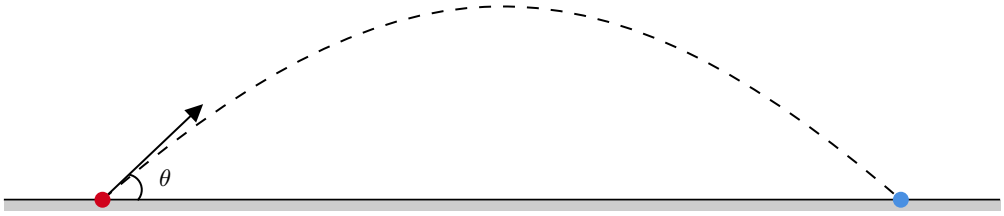
Problem Solving By Computer - Project 2

1 Introduction

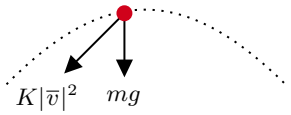
In this project, we will look at a classic of scientific computing, namely ballistics calculations. Our setup is as follows. A cannonball of weight 6kg is fired from the origin with an initial speed of 450ms^{-1} and at an angle θ, as shown in the figure below. For simplicity, we will ignore the size of the cannonball.



Although we have chosen particular parameters for our model, we will generalise our code so that it works for any other reasonable parameters. That is, we will decouple our chosen model parameters from the implementation itself, so that we can easily adapt our model when the requirements change. The complete source code is listed in the appendix.

2 Modelling with air resistance

In practice, projectiles are affected not only by gravity, but also by air resistance. We will assume that only two forces are exerted on the cannonball: gravity and the friction force $F = K|\vec{v}|^2$, opposite to the direction of velocity $\vec{v}$, and proportional to the speed squared $|\vec{v}|^2$ with constant $K = 0.00002\text{kgm}^{-1}$. Further, we will assume the gravitational constant to be 9.8ms^{-1}. We will make use of Newton's Second Law of Motion to formulate a system of differential equations to model the motion of the cannonball.



Newton's Second Law of Motion states that the sum of forces $\vec{F}$ acting on an object is equal to the mass of the object, m, multiplied by the acceleration of the object, $\vec{a}$.

$$m\vec{a} = m\vec{g} - K|\vec{v}|\vec{v}$$

Let $\vec{s}(t) = [s_x(t), s_y(t)]^\top$ be the displacement vector of the cannonball from the origin at a given time t. By writing $\vec{a} = \vec{s}\,''$ and rearranging, we get the following system of second order ODEs.

$$\begin{bmatrix} s_x''(t) \\ s_y''(t) \end{bmatrix} = \begin{bmatrix} 0 \\ -9.8 \end{bmatrix} - \frac{K}{m}\sqrt{s_x'(t)^2 + s_y'(t)^2} \begin{bmatrix} s_x'(t) \\ s_y'(t) \end{bmatrix}$$

The initial conditions for our system of ODEs is given by

$$s(0) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \text{and} \quad s'(0) = \begin{bmatrix} v_0 \cos\theta \\ v_0 \sin\theta \end{bmatrix}$$

As MATLAB's ODE solver, **ode45**, only solves first order ODEs of the form $z' = f(t, z)$, we must reduce our second order system to a first order system. In order to do this, we define $z = [s_x(t), s_y(t), v_x(t), v_y(t)]^\top$ where s_x, s_y are the component-wise displacement functions and v_x, v_y are the component-wise velocity functions, and rewrite the system of equations as follows.

$$z' = \begin{bmatrix} s_x \\ s_y \\ v_x \\ v_y \end{bmatrix}' = \begin{bmatrix} v_x \\ v_y \\ -\frac{K}{m} v_x \sqrt{v_x^2 + v_y^2} \\ -g - \frac{K}{m} v_y \sqrt{v_x^2 + v_y^2} \end{bmatrix}$$

The initial conditions for our first order system is given by

$$z(0) = \begin{bmatrix} s_x(0) \\ s_y(0) \\ v_x(0) \\ v_y(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 450\cos\theta \\ 450\sin\theta \end{bmatrix}$$

Finally, the implementation of the first order system in MATLAB is as follows.

```
1  function s = projection_ode(t, z, g, m, K)
2    s = zeros(4, 1);
3    s(1) = z(3);
4    s(2) = z(4);
5    s(3) = -K * sqrt(z(3)^2 + z(4)^2) * z(3) / m;
6    s(4) = -g - K * sqrt(z(3)^2 + z(4)^2) * z(4) / m;
7  end
```

2.1 Determining the point of impact with the ground

First, we observe that the method signature of **ode45** is

```
[TOUT, YOUT, TE, YE, IE] = ode45(ODEFUN, TSPAN, Y0, OPTIONS)
```

where `TSPAN = [T0 FINAL]` is the interval of integration for our system of ODEs. However, the difficulty is that we wish to terminate the integration once the projectile touches the ground, but we do not know the time such event would occur beforehand. Instead, we can use an event function which will terminate the integration once the projectile touches the ground, which is exactly when $s_y = 0$. We can write the events function as follows.

```
function [value, isTerminal, direction] = events_function(t, z)
  value = z(2);             % When the height is 0,
  isTerminal = 1;           % terminate integration,
  direction = -1;           % but only if the ball is falling
end
```

However, `TSPAN = [T0 FINAL]` is a required parameter of **ode45**, regardless of whether we provide an event function. We can derive an upper bound on the duration of flight by calculating the flight time for the model with no air resistance, which is given by

$$\frac{2v_0 \sin\theta}{g}$$

where v_0 is the initial velocity and θ is the firing angle. As $2v_0 \sin\theta/g$ is bounded from above by $2v_0/g$, we will use `TSPAN = [0, 2 * v / g]` for the interval of integration.

2.2 Solving the system of equations

We will create a function `projection_solution(g, v, m, K, theta)` which returns the solution to our initial value problem, as follows.

```
function [t, z, te, ze] = projection_solution(g, v, m, K, theta)
  ode = @(t, z) projection_ode(t, z, g, m, K);
  tspan = [0, 2 * v / g];
  ic = [0, 0, v * cos(theta), v * sin(theta)];
  options = odeset('events', events_function, 'reltol', 1e-8);
  [t, z, te, ze] = ode45(ode, tspan, ic, options);
end
```

Question: In the absence of friction force, the maximum horizontal displacement can be calculated explicitly, given by the launch angle $\theta = \pi/4$. What is the maximum horizontal distance and the associated angle, if we consider the above friction force?

3 Maximum horizontal displacement

Suppose we have a function `distance_fun(theta)` which returns the horizontal displacement when fired with angle `theta`, then we can find the maximum horizontal displacement by minimising the negative of the distance function, using `fminbnd`. Indeed, minimising the negative of a function is the same as maximising the function, and can be done as follows.

```
[theta, distance] = fminbnd(distance_fun, 0, pi / 2);
fprintf('Firing at %.6f radians yields a maximum distance of %.2fm\n', ...
        theta, -distance);
```

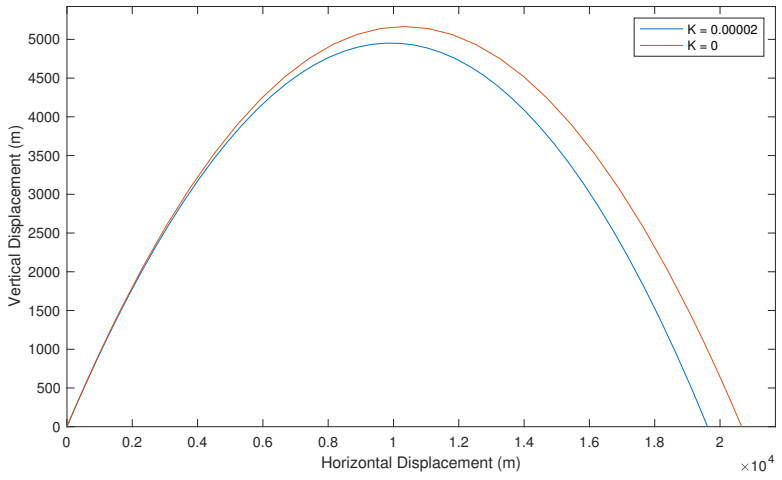
Notice that, in order to use `fminbnd`, we need to provide a function of one variable. However, we wish to generalise our code so that we can provide the model parameters g, v, m and K. To do this, we simply create a function `distance_function` which acts as a generator for our distance function of one variable `distance_fun`, as follows.

```
distance_fun = @(theta) -distance_function(g, v, m, K, theta);
```

All that remains is to write the generalised distance function. In fact, it turns out that we can implement `distance_function` as a wrapper function over `projection_solution`. That is, `distance_function` is a function that calls `projection_solution` and returns the horizontal displacement, as follows.

```
function d = distance_function(g, v, m, K, theta)
  [~, z] = projection_solution(g, v, m, K, theta);
  d = z(end, 1);
end
```

We conclude that the maximum distance of 19617.77m is achieved with a firing angle of 0.778422 radians, or 44.60°. Observe that when K is positive, air drag slows down the projectile, and as expected, it does not go as far as it would in the case $K = 0$.



4 Modelling moving interceptors

Suppose we wish to fire the cannonballs to destroy a target at 15000 meters away. However, there are interceptors located at 12000 meters away, which are being fired vertically at regular intervals, which could block the cannonballs. Each interceptor is 1000 meters long, is launched every 20 seconds, and moves upward with uniform velocity. The initial configuration at time $t = 0$ is shown below.

Question: What are the possible firing angles and associated firing times, such that the cannonball will destroy the target without being blocked by an interceptor?

4.1 Finding the required firing angles to reach the target

First, we observe that the firing angle required to yield a given horizontal displacement is independent of the firing time. As a result, we can start by finding the firing angles in order to reach the target horizontal displacement of 15000m. Using our `distance_function`, we can plot the horizontal displacement against the associated firing angle as follows.

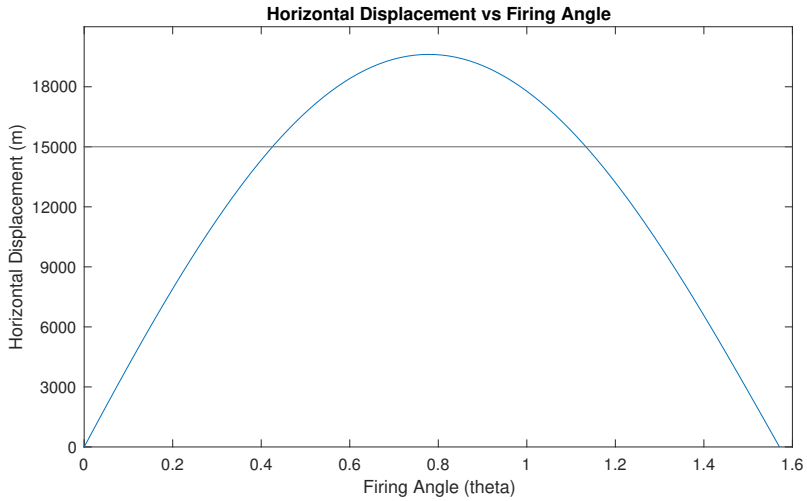
```
distance = @(theta) distance_function(g, v, m, K, theta);

angles = linspace(0, pi / 2, 100);
displacement = arrayfun(distance, angles);
displacement(1) = 0;

plot(angles, displacement);
title('Horizontal Displacement vs Firing Angle');
xlabel('Firing Angle (degrees)');
ylabel('Horizontal Displacement (m)');
yline(15000);

% Remove exponential notation from axis tick labels
ax = gca;
ax.YAxis.Exponent = 0;
```

For our chosen model parameters ($g = 9.8,\ v = 450,\ m = 6,\ K = 0.00002$), the plot for the horizontal displacement against the associated firing angle is shown below.



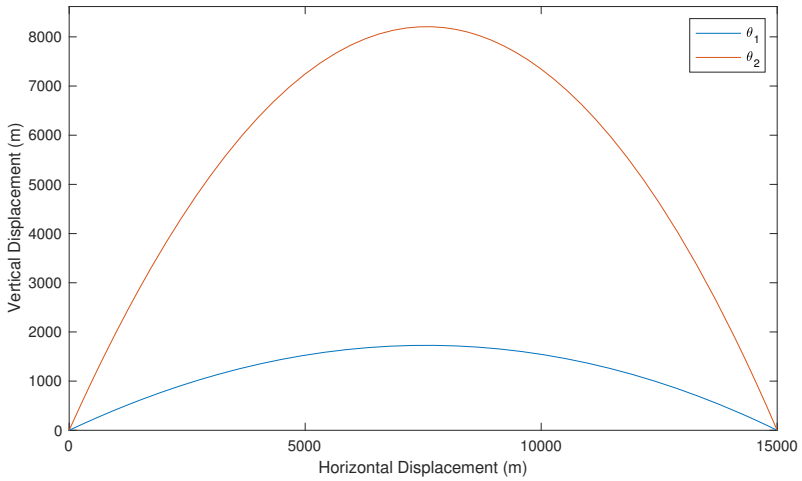
By looking at the plot, we see that the distance function $f_x(\theta)$ is a unimodal function. That is, $f_x(\theta)$ is increasing on $[0, \theta_{max}]$, and decreasing on $[\theta_{max}, \pi/2]$, where θ_{max} is the firing angle that yields the maximum horizontal displacement. In particular, suppose we wish to hit a target at distance d, then, there are three possible outcomes:

1. If $d > f_x(\theta_{max})$, then it is impossible to hit the target.
2. If $d = f_x(\theta_{max})$, then θ_{max} is precisely the only firing angle to reach this distance.
3. If $d < f_x(\theta_{max})$, then there are two solutions to the equation $f_x(\theta) = d$. One solution lies in the interval $(0, \theta_{max})$, and the other lies in the interval $(\theta_{max}, \pi/2)$.

Earlier, we saw that the firing angle which yields the maximum horizontal displacement is $\theta_{max} = 0.778422$ with $f_x(\theta_{max}) = 19617.77$m. As the distance of our target, 15000m, is less than the maximum distance, we have two possible firing angles, which can be found using the **fzero** function provided by MATLAB. Indeed, we can use **fzero** to find the values of θ where the distance function $f_x(\theta)$ and the line $y = 15000$ intersect, by finding the roots of $f_{15000}(\theta) = f_x(\theta) - 15000$. The implementation of finding the firing angles is as follows.

```
1 target_distance = 15000;
2 distance_fun = @(theta) distance_function(g, v, m, K, theta);
3 [maxTheta, ~] = fminbnd(@(theta) –distance_fun(theta), 0, pi / 2);
4
5 distance_from_target = @(theta) distance_fun(theta) – target_distance;
6 thetaOne = fzero(distance_from_target, [eps, maxTheta]);
7 thetaTwo = fzero(distance_from_target, [maxTheta, pi / 2]);
```

We conclude that the two firing angles which yield a horizontal distance of 15000m are $\theta_1 = 0.425366$ and $\theta_2 = 1.133637$. The trajectory plots for θ_1 and θ_2 are shown below.



4.2 Finding the time and height for a given horizontal displacement

In order to determine whether the trajectory of the cannonball is blocked by the interceptor, we need to know the height of the cannonball when the horizontal distance travelled by the ball is 12000m. As the interceptors are moving, we also need to know the time taken for the cannonball to reach the horizontal distance of 12000m. In order to do this, we can add a non-terminating event to our event function, which we can use to obtain the time and height at which the cannonball arrives at the interceptors. We can add a second event to our event function as follows.

```
function [value, isTerminal, direction] = events_function(t, z, distance)
  value(1) = z(1) - distance;    % When the distance is 12000,
  isTerminal(1) = 0;             % take note of the event.
  direction(1) = 1;

  value(2) = z(2);               % When the height is 0,
  isTerminal(2) = 1;             % terminate integration,
  direction(2) = -1;             % but only if the ball is falling
end
```

Notice that, for this event, we set `isTerminal = 0` to indicate that we only want to detect when this event occurs, rather than halt integration when this event occurs.

Consequently, we need to update our `projection_solution` function to take into account the `distance` parameter, as follows. Notice that on Line 5, we use a similar design pattern as earlier, by using `events_function(t, z, distance)` to generate `events_fun(t, z)`.

```
1 function [t, z, te, ze] = projection_solution(g, v, m, K, theta, distance)
2   ode = @(t, z) projection_ode(t, z, g, m, K);
3   tspan = [0, 2 * v / g];
4   ic = [0, 0, v * cos(theta), v * sin(theta)];
5   events_fun = @(t, z) events_function(t, z, distance);
6   options = odeset('events', events_fun, 'reltol', 1e-8);
7   [t, z, te, ze] = ode45(ode, tspan, ic, options);
8 end
```

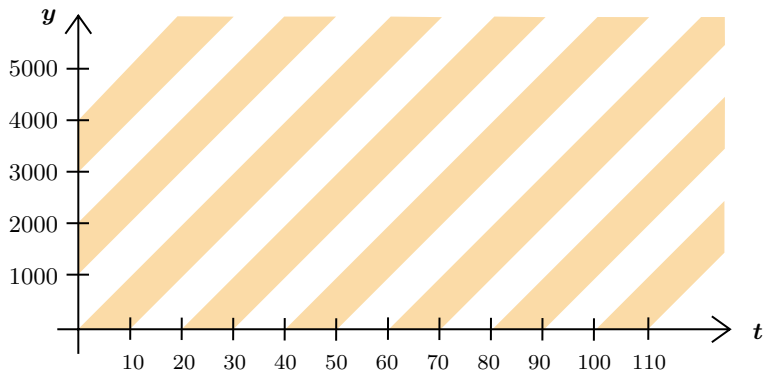
Finally, we can obtain the information (time and height of cannonball) about the event at which the cannonball arrives at the interceptors, as follows.

```
1 ic_distance = 12000;
2 [t1, z1, te1, ze1] = projection_solution(g, v, m, K, thetaOne, ic_distance);
3 [t2, z2, te2, ze2] = projection_solution(g, v, m, K, thetaTwo, ic_distance);
4
5 fprintf('[theta = %.4f]: %.4fs to reach %dm, with height %.4fm\n', ...
6         thetaOne, te1(1), ic_distance, ze1(1, 2));
7 fprintf('[theta = %.4f]: %.4fs to reach %dm, with height %.4fm\n', ...
8         thetaTwo, te2(1), ic_distance, ze2(1, 2));
```

We conclude that by firing at angle $\theta_1 = 0.425366$, it takes $t_1 = 29.8907$s to reach a distance of 12000m, with height $y_1 = 1117.1270$m. Likewise, if we fire at angle $\theta_2 = 1.133637$, it takes $t_2 = 65.0186$s to reach a distance of 12000m, with height $y_2 = 5318.9654$m.

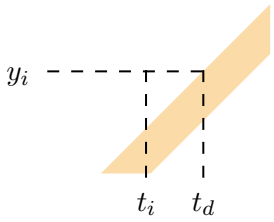
4.3 Finding the firing times to avoid the interceptors

All that remains in order to find the firing times such that the trajectory of the cannonball avoids the interceptors, is to model the interceptors. That is, we need to determine whether an interceptor is present, for a given time and height. It turns out that we can model the positions of the interceptors by a system of inequalities, as shown in the figure below.



The shaded regions indicate the presence of an interceptor, whereas the the non-shaded regions indicate the absence of an interceptor. Each shaded region is defined by two lines, the line to the left of the shaded region which indicates the top of the interceptor, and the line to the right of the shaded region which indicates the bottom of the interceptor.

Observe that the gradient of the lines are 100 and every lower bound line passes through the point $(20n, 0)$, so the lower bound lines are of the form $y = 100(t - 20n)$, $n \in \mathbb{N} \cup \{0\}$. In order to find the maximum delay before firing the cannonball such that the cannonball misses the interceptor, we need to find the point (t_d, h_i), where t_d lies on a lower bound line and $t_i \leq t_d$.



Putting the equation of the lower bound line and the inequality together, we have

$$\frac{1}{20}\left(t_i - \frac{y_i}{100}\right) \leq n.$$

However, as n is an integer, we can find the least such n that satisfies the inequality by

$$n = \left\lceil \frac{1}{20}\left(t_i - \frac{y_i}{100}\right) \right\rceil.$$

It follows that $t_d = 0.01y_i + 20n$ and so the maximum delay we can wait before firing the cannonball is given by $t_d - t_i$. If (t_i, y_i) lies in a shaded region, we need to calculate a minimum delay that we must wait before firing the cannonball. Observe that if the maximum delay is less than 10, then (t_i, y_i) must lie in a non-shaded region, otherwise, the minimum delay can be computed by $t_d - t_i - 10$. However, for our parameters, it turns out that both (t_1, y_1) and (t_2, y_2) lie outside the shaded region, and so the minimum delay is 0. The implementation of finding the maximum delay is as follows.

```
function d = delay_function(t, h)
  n = ceil(0.05 * (t - 0.01 * h));
  td = 0.01 * h + 20 * n - t;
  d = [max(td - 10, 0), td];
end
```

We find the intervals of delay for θ_1 and θ_2 as follows.

```
ic_distance = 12000;
[t1, z1, te1, ze1] = projection_solution(g, v, m, K, thetaOne, ic_distance);
[t2, z2, te2, ze2] = projection_solution(g, v, m, K, thetaTwo, ic_distance);

d1 = delay_function(te1(1), ze1(1, 2));
d2 = delay_function(te2(1), ze2(1, 2));

fprintf('[theta = %.4f]: Min Delay = %.4f, Max Delay = %.4f\n', ...
        thetaOne, d1(1), d1(2));
fprintf('[theta = %.4f]: Min Delay = %.4f, Max Delay = %.4f\n', ...
        thetaTwo, d2(1), d2(2));
```

The above code snippet produces the following output.

```
[theta = 0.4254]: Min Delay = 0.0000, Max Delay = 1.2805
[theta = 1.1336]: Min Delay = 0.0000, Max Delay = 8.1711
```

We conclude that in order to fire with launch angle $\theta_1 = 0.425366$ without hitting any interceptors, we can fire at times $t \in [0, 1.2805)$. However, as the interceptors are fired periodically every 20 seconds, with the interceptor and the gap being of equal width, we can also fire every other 10 seconds after 1.2805. That is, for $\theta_1 = 0.425366$, we can fire at times $t \in (11.2805, 21.28025) \cup (31.28025, 41.28025) \cup \cdots \cup (11.2805 + 20n, 21.2805 + 20n)$.

Similarly, in order to avoid the interceptors when firing with launch angle $\theta_2 = 1.133637$, we can fire at times $t \in [0, 8.1711) \cup (18.1711, 28.1711) \cup \cdots \cup (18.1711 + 20n, 28.1711 + 20n)$.

4.4 Results

We have found the two possible launch angles and the associated firing times, such that the cannonball will hit the target placed 15000m away, without being blocked by an interceptor.

$$\theta_1 = 0.425366,\ t_{\theta_1} = [0, 1.2805) \cup \{(11.2805 + 20n, 21.2805 + 20n) \mid n \in \mathbb{N} \cup \{0\}\}$$

$$\theta_2 = 1.133637,\ t_{\theta_2} = [0, 8.1711) \cup \{(18.1711 + 20n, 28.1711 + 20n) \mid n \in \mathbb{N} \cup \{0\}\}$$