Project 2

10626728

1 Introduction

1.1 Outlining the Problem

This project considers the problem of a fox chasing a rabbit as it attempts to escape and run towards its burrow. Using differential equations to model the positions of both the fox and the rabbit at particular times, we will set up and solve a system of ODEs (Ordinary Differential Equations) that we can then solve in order to find out the locations of both animals at given times. Using some MATLAB tools, such as *ODE Event Locations* as an option within *ode45*, we can determine when to stop the integration involved in solving this system and thus, determine at what particular time the rabbit escaped or was caught.

Initially, the rabbit will be located at $(0, 0)$ and the fox will be at $(-250, -550)$. The rabbit will move in a straight path towards its burrow, located at $(-600, 600)$, with initial speed $s_{r_0} = 13m/s$. The fox will move with initial speed $s_{f_0} = 16m/s$, and will move along a path in one of the three following ways:

1. If the fox can see the rabbit, then the fox's direction of travel will be directed towards the rabbit's position. In other words, the direction of the velocity vector of the fox will be towards the exact location of the rabbit.
2. If the view of the rabbit is blocked by the south wall of an impassable warehouse, then the fox will run directly to the southeast corner of the warehouse.
3. If the view of the rabbit is subsequently blocked by the warehouse, then the fox will run directly north until it can see the rabbit again, at which point it will follow the same path as outlined in the first point.

The rabbit is considered caught by the fox if the distance between the fox and the rabbit is less than or equal to 0.1 meters. The warehouse is considered to extend infinitely to the west, and so the northeast and southeast corners can be referred to as north and south respectively without the risk of confusion. Both the fox and the rabbit can be modelled as particles so their sizes can be neglected - as with most mathematical models. The initial configurations of the fox and rabbit, together with the environment they will be moving within can be seen in figure 1.

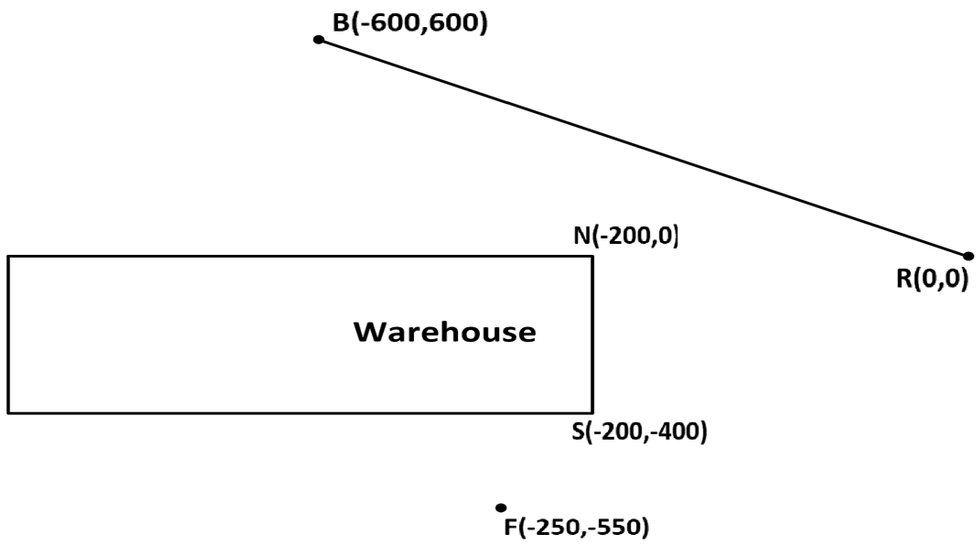


Figure 1: The initial configuration of the problem, with the locations of the fox (F), rabbit (R), burrow (B), and north (N) and south (S) corners. The Coordinates are given in meters.

In both sections of the problem we will need to find: the time T when the rabbit escaped or was caught, the position of the fox at this time T, and the distance travelled by the fox in time T along its path. A plot of both paths is also required.

1.2 The *ode45* Function

The *ode45* function is built into MATLAB. The function solves systems of differential equations of the form $y' = f(t, y(t))$, $y(t_0) = y_0$[1, page 193]. The general form of *ode45* is $[t, y] = ode45(odefun, tspan, y0)$, where *odefun* is a function containing the differential equations to solve, $tspan = [t0\ tf]$, and $y0$ is a vector containing the initial conditions. *ode45* integrates the system of equations given by *odefun* from $t0$ to tf, using initial conditions. Each row of y will correspond to the solution of the ODE system at the specific time found in the corresponding row in t. If the system of ODEs contains more than one ODE, then each column of y will correspond to the solution of each ODE. Understanding this function and its output arrays is very important, as it will allow us to easily extract the information necessary to plot the paths of both animals once the system of differential equations has been properly defined.

1.3 ODE Event Locations

We can define a function of the general form $[value, isterminal, direction] = myEventsFcn(t, y)$. The output arguments are vectors whose ith element corresponds with the ith event. $value(i)$ is a mathematical expression that defines the ith event. When this expression is 0, the event has occurred. $isterminal(i) = 1$ if the integration is to stop when the event occurs, otherwise it is 0. $direction(i) = 0$ if all instances of the event are to be recorded, $+1$ if only those when the event expression is increasing are recorded, or -1 if only those when it is decreasing are recorded. We then use the *odeset* function to pass these events into the ODE solver with the syntax $options = odeset('Events', myEventsFcn(t, y))$, together with the new general form $[t, y, te, ye, ie] = ode45(odefun, tspan, y0, options)$. The added output arguments are:

- te, a column vector containing the times at which each events occurred,
- ye, containing the solution value at each of the times in te,
- and ie, which contains the indices of the events that occurred, and so tells us which events were triggered.

Therefore, defining one event as the rabbit reaching the burrow and another as the fox catching the rabbit, will enable us to stop the integration at those specific times and access the solutions to the differential equations at the occurrence. We can use these events to find the position of the fox and the distance the fox has travelled in each instance.

1.4 Distance Along a Curve

In the initial configuration shown in figure 1 it is clear that the fox can see the rabbit, and so at least some part of the fox's path will be curved as it chases the moving rabbit. So, we need to know how to find the length of a curve, which will allow us to find the distance travelled by the fox. We will discus how to derive the equations necessary to find the distances travelled [see 2, Chapter 12, page 449]. Suppose the position of a particle at time t is given by:

$$\gamma(t) = (x(t), y(t)).$$

Then the velocity of the particle is $\gamma'(t) = (x'(t), y'(t))$. Since speed is the magnitude of velocity, the speed of the particle is given by $|\gamma'(t)|$. Finally, since distance is the integral of speed

$$\text{total distance travelled between time a and time b} = \int_{t=a}^{t=b} \sqrt{|\gamma'(t)|}\, dt = \int_{t=a}^{t=b} \sqrt{(x'(t))^2 + (y'(t))^2}\, dt.$$

Therefore, after differentiating, we have

$$\frac{ds(t)}{dt} = \sqrt{\left(\frac{dx(t)}{dt}\right)^2 + \left(\frac{dy(t)}{dt}\right)^2}, \tag{1}$$

where $s(t)$ = distance travelled by the particle, and $x(t)$ and $y(t)$ are the x and y coordinates of the particle at time t. Solving two separate ODEs of this form using *ode45* would give us the total distance travelled by each animal.

1.5 Intersection of Two Line Segments

We need a method to mathematically determine if the warehouse obstructs the fox's sight of the rabbit, since the path of the fox changes dependent on this condition. With a little thought, we can see that the

sight is blocked if the straight line joining the location of the fox to the location of the rabbit intersects the N-S line segment of the warehouse. We can check if this happens, by first finding the equation of the straight line connecting the two animals. Then, since the N-S line segment is given by the line $x = -200$, in the range $-400 \leq y \leq 0$, we can substitute $x = -200$ into the equation of our line and check if the y value lies in this range. This idea is geometrically illustrated in figure 2. In order to help with the implementation of this, a function, called InSight, was created. *InSight*, shown in figure 5, determines if the line segment connecting two particles intersects the N-S line segment of the warehouse. The function returns *true* if the two line segments do not intersect, and returns *false* if they do intersect. Using this function, we are easily able to set up the conditions necessary to determine which path the fox takes at a given time.

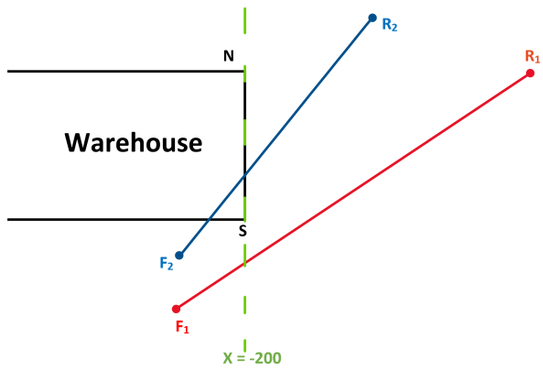


Figure 2: In red, a configuration where there is no intersection of the line segments, and in blue, a configuration where there is an intersection. The coordinates of the point of intersection between the line joining the position of the fox and rabbit and the line $x = -200$ can be found. The y coordinate can be compared to the range $-400 \leq y \leq 0$, which is the range of y values that the N-S line segment spans, to determine if the two line segments intersect.

1.6 Governing Equations

Now we will now derive the governing equations for the motion of both animals. This type of problem has already been studied multiple times, and it is specifically mentioned in the text 'MATLAB guide'[1, page 201], and so we can use this source as a reference in defining the equations needed. In order to determine the ODEs, we must first understand that we are solving for the x and y positions of both the fox and rabbit, and the distances travelled by each of them. Our solution will be a six row vector, with each row representing one of the quantities just mentioned. Therefore, we need six ODEs to solve. If we first focus only on those equations for position, it is helpful to recognise that velocity is the derivative of position, so we need to derive four equations for the velocities in the x and y components for the fox and rabbit respectively. Solving this system will be done through integration of each equation, and thus the result will be the positions of both animals in each of the two components. The velocity in each component is always defined as the product of the speed and a unit vector in the direction of travel.

In all cases, the rabbit runs from its starting point at $(0, 0)$ directly towards its burrow located at $(-600, 600)$. Therefore, a unit vector in the direction of travel given by the vector $\frac{1}{\sqrt{2}}\begin{bmatrix}-1\\1\end{bmatrix}$, and so the rabbits path is defined by:

$$\frac{dr_x(t)}{dt} = s_r\left(\frac{-1}{\sqrt{2}}\right), \quad \frac{dr_y(t)}{dt} = s_r\left(\frac{1}{\sqrt{2}}\right), \tag{2}$$

where r_x, r_y are the positions in the x and y components, and s_r is the speed of the rabbit.

Using equation 1, we find that

$$\frac{dr_d(t)}{dt} = \sqrt{\left(\frac{dr_x}{dt}\right)^2 + \left(\frac{dr_y}{dt}\right)^2}, \tag{3}$$

where r_d is the distance travelled by the rabbit.

Now we will derive the governing equations for the fox's motion. We need to consider three cases:

1. the fox running directly towards the rabbit,
2. the fox running directly towards S,
3. the fox running directly north.

Case 1: The direction of motion is always exact from the fox to the rabbit and so the path of the fox is determined by the ODEs:

$$\frac{df_x(t)}{dt} = s_f\left(\frac{r_x - f_x}{dist_{fr}}\right), \quad \frac{df_y(t)}{dt} = s_f\left(\frac{r_y - f_y}{dist_{fr}}\right), \tag{4}$$

where $dist_{fr} = \sqrt{(r_x - f_x)^2 + (r_y - f_y)^2}$, and s_f is the speed of the fox.

Case 2: We can calculate the component of the velocity in the same way, except now the fox is running from its position directly to S, located at $(-200, -400)$. Therefore we can define $dist_{fS} = \sqrt{(-200 - f_x)^2 + (-400 - f_y)^2}$, and

$$\frac{df_x(t)}{dt} = s_f\left(\frac{-200 - f_x}{dist_{fS}}\right), \quad \frac{df_y(t)}{dt} = s_f\left(\frac{-400 - f_y}{dist_{fS}}\right). \tag{5}$$

Case 3: Here the fox moves directly north, and so the components of velocity are easily found to be

$$\frac{df_x(t)}{dt} = 0, \quad \frac{df_y(t)}{dt} = s_f. \tag{6}$$

Finally, we have the distance the fox has travelled, which in all cases is given by the solution to the ODE:

$$\frac{df_d(t)}{dt} = \sqrt{\left(\frac{df_x}{dt}\right)^2 + \left(\frac{df_y}{dt}\right)^2}, \tag{7}$$

where f_d is the distance the fox has travelled.

2 Task Pseudo-code

In task A, the fox and rabbit are assumed to be running at constant speeds, equal to their initial speeds. Therefore, $s_f = s_{f_0} = 16m/s$ and $s_r = s_{r_0} = 13m/s$. Whereas in task B, the speeds of both animals are modelled as diminishing over time and are functions of the distances already travelled. The speeds are

$$s_f(t) = s_{f_0}e^{-0.0002(f_d(t))}, \quad s_r(t) = s_{r_0}e^{-0.0008(r_d(t))},$$

where $f_d(t)$ and $r_d(t)$ are the distances travelled by the fox and rabbit respectively.

This being the only difference allows the same approach to be used for both parts of the problem, with only the definition of the speeds of both animals needing to be changed.

The following pseudo-code was used to outline the formation of the ODEs.

> For z given by $z = [f_x, f_y, f_d, r_x, r_y, r_d]^T$, and initial conditions $z0 = [-250, -550, 0, 0, 0, 0]^T$.
> We can define system of ODEs by:
> Initialise a 6x1 array of zeros called dzdt.
> If fox cannot see rabbit and $f_x < -200 \rightarrow$ dzdt(1) & dzdt(2) are ODEs in equation 5.
> If fox cannot see rabbit and $f_x \geq -200 \rightarrow$ dzdt(1) & dzdt(2) are ODEs in equation 6.
> If fox can see rabbit $\rightarrow$ dzdt(1) & dzdt(2) are ODEs in equation 4.
> Then, dzdt(3) is ODE in equation 7,
> dzdt(4) & dzdt(5) are ODEs in equation 2,
> and dzdt(6) is ODE in equation 3.
> Have six ODEs corresponding in order to the six components of z and use
> *ode45* to solve this system over a time span.

The speeds needed to define the equations above can be determined by:

> If Task A
> $\rightarrow s_f = s_{f_0} = 16m/s$ and $s_r = s_{r_0} = 13m/s$.
> If Task B
> $\rightarrow s_f = s_{f_0}e^{-0.0002(f_d)}$, where f_d is the solution to equation 7, and
> $s_r = s_{r_0}e^{-0.0008(r_d)}$, where r_d is the solution to equation 3.

Pseudo-code was also used to outline the process for determining the events function.

> Define event 1 to be distance between rabbit and burrow. When this crosses zero:
> $\rightarrow$ stop integration
> Define event 2 to be distance between fox and rabbit. When this crosses threshold (0.1m):
> $\rightarrow$ stop integration.

We can then pass these events into *ode45* as an option, and use the extra output arguments discussed in section 1.3, to determine the time the rabbit was caught or escaped, position of fox at this time, and the distance travelled by the fox. The *plot* function of MATLAB will allow us to plot the paths of both animals by producing phase plane plots[1, page 196] of the numerical solutions found by *ode45*.

3 Code Implementation

Writing the code consisted of piecing together all the parts discussed previously, including the *InSight* function (figure 5), system of ODEs (figure 4), events function (figure 7), and the solving and plotting (figure 8).

The first step was to define all of the initial conditions, shown in figure 3, that were necessary for the problem. The second value in tspan shown in line 6 is found by calculating the initial distance between the animals, and then dividing it by the difference between their initial speeds. This will be an upper bound for the time, as within this time-span the rabbit will definitely be caught or will reach the burrow.

```
1 z0 = [-250, -550, 0, 0, 0, 0]; % ICs of fox and rabbit positions and distances travelled
2 s_r0 = 13; s_f0 = 16; % Initial speeds of rabbit and fox respectively
3 burrow = [-600; 600];
4 mindist = 0.1;
5 S = [-200; -400];
6 tspan = [0 norm([z0(1) z0(2)] - [z0(4) z0(5)])/(s_f0 - s_r0)]; % Timespan to run ODE over
```

Figure 3: Initial conditions used to solve the system of ODEs

The function that defines the system of ODEs was written using the pseudo-code to help with organisation. It was important here to refer back to the initial conditions in line 1 of 3, to ensure that the order in which the ODEs have been defined was consistent with the order of the initial conditions. Thus, the governing equations had to be in the following order: fox's x component, fox's y component, fox's distance, rabbit's x component, rabbit's y component, rabbit's distance. The function defining these equations is shown in figure 4. The function *fox_rab_ODE* first initializes a $6x1$ array that will contain the differential equations to be solved. The functions of the speeds are defined, using the input parameter *task*. The entire code was written in a function *fox_rab*, which uses a single input parameter, *task*. If this input is 1 then the model in task A is used, and 2 for the model in task B, as can be seen in figure 6, by typing *help fox_rab* into the command window in MATLAB. The distances between the position of the fox and the rabbit and south corner of the warehouse respectively are also defined to improve readability of the code, in line with the ideas outlined in section 1.6. The function *InSight*, shown in figure 5 determines if the sight is blocked or not, along with the a check for the x coordinate of the fox to determine if the fox has passed S or not already, in order to decide what path the fox should follow. The ODEs are then defined in agreement with the pseudo-code and the equations derived in section 1.6.

```
1 >> help fox_rab
2   fox_rab solves a system of ODEs to model a fox chasing a rabbit
3   as it runs towards it's burrow, within a specific area. Certain conditions
4   about their running speeds, starting positions, and running paths are
5   predetermined. fox_rab can take a single input parameter:
6   input is 1, then the constant speed model is used,
7   input is 2, then the diminishing speed model is used.
```

Figure 6: Information to the user on how to use the function *fox_rab*.

The events function was defined using the pseudo-code and sources detailing similar problems[1, Listing 12.4, page 204] as a guide. One key observation in this step was to recognise that since the rabbit's path is a straight line from the origin to the point $(-600, 600)$, the gradient is -1. Therefore, the rabbit's coordinates will be given by $(r_x, r_y)|r_x = -r_y$. Thus, it suffices to only check that the x coordinate of the rabbit has reached the x coordinate of the burrow (or the same approach for y). It is not necessary to check both since, using the relation previously mentioned, once the rabbit has reached the burrow on the x axis, it must have also reached it on the y axis too. This improves the efficiency, since there is no need to calculate the modulus of the vector from the rabbits location to the burrow each time, however the limitation of this is that new events functions would have to be defined if the rabbit approached from a different path. The second event was necessary to check if the rabbit had been caught. Using the simplification outlined in section 1.1, that the rabbit is caught if the distance between the fox and the rabbit is less than or 0.1m, the second event function was defined to calculate the distance between the two animals minus the minimum distance travelled. The event function is shown in figure 7.

```
function dzdt = fox_rab_ODE(z,s_f0,s_r0, task, S)
    % Funtion sets the system of ODES for
    % position vector of and distance travelled
    % by fox and rabbit.
    dzdt = zeros(6,1);
    if task == 2
        % Diminishing speeds
        s_f = s_f0 * exp(-0.0002*z(3));
        s_r = s_r0 * exp(-0.0008*z(6));
    else
        % Constant speeds
        s_f = s_f0; s_r = s_r0;
    end
    dist_rf = sqrt((z(4)-z(1))^2+(z(5)-z(2))^2); % Calc distance from rabbit to fox
    dist_Sf = sqrt((S(1)-z(1))^2 + (S(2)-z(2))^2); % Calc distance from fox to S
    if ~InSight(z) && z(1) < -200 % S blocks view of rabbit
        dzdt(1) = s_f/dist_Sf .* (S(1)-z(1));
        dzdt(2) = s_f/dist_Sf .* (S(2)-z(2));
    elseif ~InSight(z) % N block view of rabbit
        dzdt(1) = s_f .* 0;
        dzdt(2) = s_f .* 1;
    else % View of rabbit is not blocked
        dzdt(1) = s_f/dist_rf .* (z(4)-z(1));
        dzdt(2) = s_f/dist_rf .* (z(5)-z(2));
    end
    dzdt(3) = sqrt(dzdt(1)^2+dzdt(2)^2); % Distance travelled by fox at time t
    % ODE for rabbit's path
    dzdt(4) = -s_r/sqrt(2);
    dzdt(5) = s_r/sqrt(2);
    dzdt(6) = sqrt(dzdt(4)^2+dzdt(5)^2); % Distance travelled by rabbit at time t
end
```

Figure 4: Code showing the governing equations.

```
function p = InSight(z)
% InSight takes a vector input that contains
% the positions of two particles. Checks if
% the line between them intersects the line
% segment x = -200, -400 <= y <= 0.
    grad = (z(5)-z(2))/(z(4)-z(1));
    y = grad.*(-200 - z(1)) + z(2);
    if -400 <= y && y <= 0
        p = false;
    else
        p = true;
    end
end
```

Figure 5: *InSight* takes one input argument, z, and compares the coordinates of two particles contained within z to the line segment $x = -200, -400 \leq y \leq 0$.

```
function [value,isterminal,direction] = events(z, burrow, mindist)
    % Events function to stop ODE solver if rabbit reaches the burrow
    % or if fox catches the rabbit
    value(1) = z(4) - burrow(1); % Rabbit reaches the burrow x-coord
    isterminal(1) = 1;
    direction(1) = -1;
    value(2) = sqrt((z(4)-z(1))^2+(z(5)-z(2))^2) - mindist; % Fox catches the rabbit
    isterminal(2) = 1;
    direction(2) = -1;
end
```

Figure 7: Events function containing both events equations to determine if the rabbit has reached the burrow or if the rabbit has been caught. The value of *isterminal* is 1 for both, since in both cases the integration should stop. The *direction* is -1 for both, since in both cases the given equation should decrease through 0 to trigger the event.

The *odeset* function was used to pass the events function into the ODE solver, and the plotting function within MATLAB was used to construct a plot of the paths of both animals. Additionally, by using the extra output arguments $[te, ye, ie]$ we can easily see which events were triggered and the time at which

they were, as well as the positions and distances travelled by both animals at this point. The code used to implement all of this is shown in figure 8. Additionally, a few statements have been written to display the information required in an easy to read manner. The statements make use of the additional outputs from the events function, and also the *num2str* function in order to display the numerical values necessary.

```
%ODE solver
option = odeset('MaxStep',0.01,'Events',@(t,z)events(z,burrow,mindist));
[~, pfoxrab, te, ze, ie] = ode45(@(t,z)fox_rab_ODE(z,s_f0,s_r0,task,S), tspan, z0, option);

xfox = pfoxrab(:,1);
yfox = pfoxrab(:,2);
xrab = pfoxrab(:,4);
yrab = pfoxrab(:,5);

% Plots of fox and rabbit paths, and warehouse for context
plot(xfox, yfox, xrab, yrab); hold on
rectangle('Position',[-1000 -400 800 400]), text(-400,-200,'Warehouse')
legend('Fox', 'Rabbit')
xlim([-650 0]), ylim([-600 650])
xlabel('x coordinate (meters)'), ylabel('y coordinate (meters)')
hold off

% Display outputs with context

if ie == 1
    disp(['Rabbit reached the burrow before it was caught, in ', num2str(te(1)), ' seconds.' ...
    ' At this point the fox was at (', num2str(ze(1)), ', ', num2str(ze(2)), ').' ])
elseif ie == 2
    disp(['Rabbit was caught in ', num2str(te(1)), ' seconds at ' ...
    '(', num2str(ze(1)), ', ', num2str(ze(2)), ').' ])
end
disp(['The fox travelled ', num2str(ze(3)), ' meters in the ', num2str(te(1)), ' seconds.'])
```

Figure 8: The events function is passed into ode45 by the *odeset* function. The system of ODEs is solved, with the solution to the ODEs stored as the columns of *pfoxrab*. The relevant columns can be plotted against each other in order to show the paths of both animals[1, page 196]. The warehouse is also plotted and labelled for clarity and to give context the the paths of the animals.

4 Outputs

4.1 Task A

The output from calling the function for task A is shown in figure 9. This clearly shows that with all of the initial conditions outlined in section 1.1, the rabbit will escape the fox in 65.2514s, and the fox will be at the position $(-448.3923, 410.7324)$. By using the Pythagorean Theorem, we can check the validity of this solution. Moving at 13m/s over a distance of $\sqrt{(-600)^2 + (600)^2} \approx 848.5281$m should take $\frac{848.5281}{13} \approx 65.2714$s. This estimate agrees with the output shown in figure 9 to 1 decimal place. The curved distance travelled by the fox can approximated by 2 separate straight lines between the approximate start and end point of each curved section. Recognising that this is an underestimate of the true distance travelled, we find an approximation of this distance to be 1034.92m. This is indeed a slight underestimate of the true distance travelled, which is found to be 1044.3423m. One final check shows that the distance travelled by the fox in the time given is also valid, as $\frac{1044.3423}{65.2714} = 15.999999$m/s, which is the running speed of the fox, accounted for rounding of the outputs in the MATLAB display. As a result of these checks we can be confident that the code gives a valid solution. The plot of the paths of both animals is shown in figure 11a.

```
>> fox_rab(1)
Rabbit reached the burrow before it was caught, in 65.2714 seconds. At this point the
fox was at (-448.3923, 410.7324).
The fox travelled 1044.3423 meters in the 65.2714 seconds.
```

Figure 9: The displayed output for task A, with the input parameter 1 using the constant speed model.

4.2 Task B

Task B requires the diminishing speed model, and so the outputs from this task should be expected to be different to that of the first task. A few things can be noted in order to check for validity of results: if the rabbit is caught it should be caught at a point before the burrow and on the rabbits path, and if the rabbit escapes it should take longer to escape than in task A. These may seem obvious however they are important to recognise outputs that would definitely be invalid. In this case, the fox catches the rabbit at the position $(-586.0435, 586.0435)$, after 90.4648s, shown in figure 10. The plot of the paths is shown in figure 11b. By doing similar checks to those in task A, we find an approximation to the distance travelled by the fox to be 1251.77m which is again a pretty good underestimate of the true total distance travelled, which is 1271.2117m. The other checks also hold for these results, and together with the assumptions about the results made earlier, we can be confident that this solution is also correct.

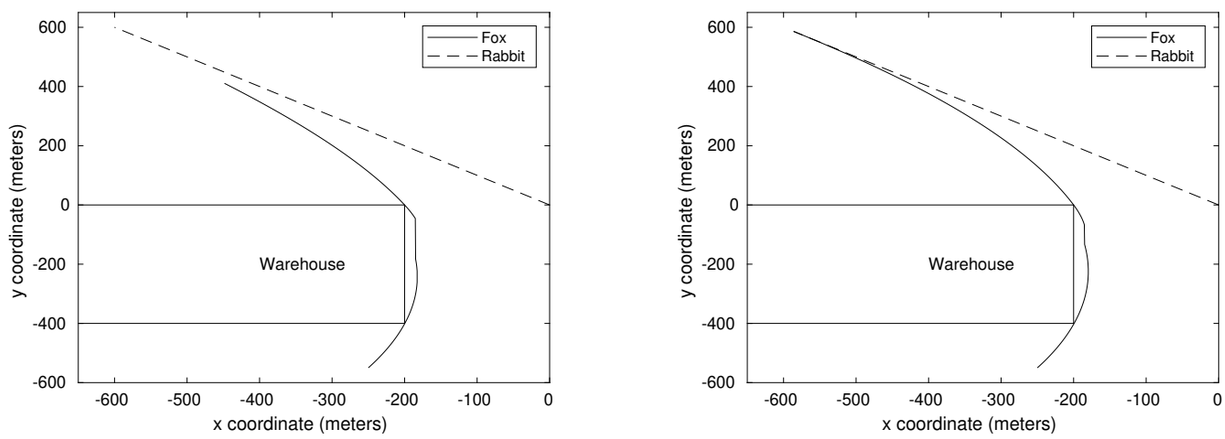
```
>> fox_rab(2)
Rabbit was caught in 90.4638 seconds at (-586.0435, 586.0435).
The fox travelled 1271.2117 meters in the 90.4638 seconds.
```

Figure 10: The displayed output for task B, with the input parameter 2 using the diminishing speed model.

A table displaying the results of the simulation is shown in 1.

Results				
Task	**Speed Model**	**Outcome**	**Time (s)**	**Position of Fox (meters)**
A	Constant	Escaped	65.2714	(-448.3923, 410.7324)
B	Diminishing	Caught	90.4638	(-586.0435, 586.0435)

Table 1: Summary of simulation results.



(a) Fox and Rabbit paths with constant speeds.

(b) Fox and Rabbit paths with diminishing speeds.

Figure 11: Plots of the paths of both animals. Left, the plot for task A and right, the plot for task B.

References

[1] Desmond J. Higham and Nicholas J. Higham. *MATLAB guide*. David Marshall, 2016.

[2] Gilbert Strang. *Calculus*. Wellesley-Cambridge Press, 1991.