MATH36032 Project Two

Student ID no:10098348

April 18, 2021

1 Introduction

The goal of this project is to model the dynamics of a Fox-Rabbit chase using ordinary differential equations (referred to as ODEs henceforth) in two different scenarios. The initial positions of the fox, rabbit, warehouse and burrow are:

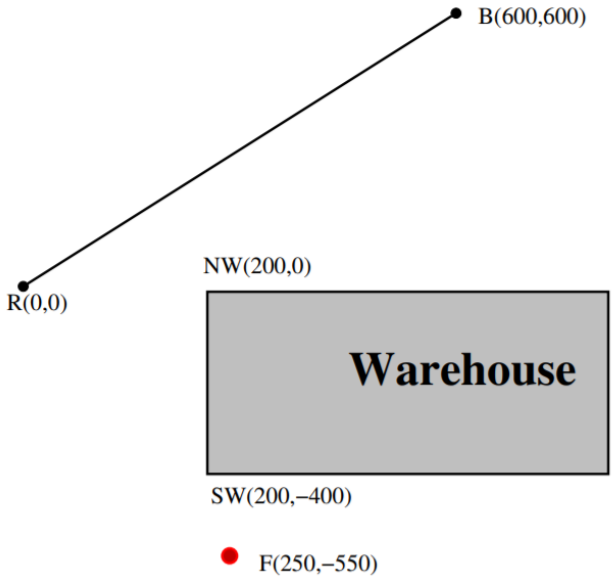

Figure 1: positions of the fox (F), rabbit (R) burrow (B) and warehouse[1]

The rabbit and the fox begin at the positions shown. The rabbit moves towards the burrow in a straight line, instantaneously. The fox pursues the rabbit such that its velocity vector is always pointed towards the rabbit. If the vector is blocked (if the fox cannot see the rabbit) by the south-west (SW) corner of the warehouse then the fox runs towards the corner and if upon reaching the corner, the vector is still blocked then the fox runs along the SW-NW side of the warehouse until it can see the rabbit again. I wasn't able to figure out how to code whether the fox could see the rabbit so that doesn't figure into my solutions.

2 Scenario One

The velocities of the rabbit and the fox are constant, at 13ms^{-1} and 16ms^{-1}, respectively. Since the rabbit is moving along the diagonal of a triangle with base length and height 600, the angle at which it is moving relative to the x-axis is $\pi/4$ radians. So the horizontal velocity of the rabbit is $13\cos(\pi/4)$ and the vertical velocity is $13\sin(\pi/4)$.

One of the main functions of the program is foxode1, which solves 2 differential equations to calculate the position, z, of the fox at time t. First we calculate a vector r, the position of the rabbit at time t. We do this by multiplying the horizontal and vertical velocities by t. Now, the equations for the horizontal and vertical velocity of the fox are given by

$$\dot{z}(1) = \frac{16(r(1) - z(1))}{\sqrt{(r(1) - z(1))^2 + (r(2) - z(2))^2)}}$$

$$\dot{z}(2) = \frac{16(r(2) - z(2))}{\sqrt{(r(1) - z(1))^2 + (r(2) - z(2))^2)}}$$

where $r(1), r(2)$ represent the first and second elements of the vector r, corresponding to the horizontal and vertical positions of the rabbit and similarly $z(1), z(2)$ represent the horizontal and vertical positions of the fox.

As can be seen in figure 2, instead of taking the values 13 and 16, the function takes generic values s_r and s_f for the speeds of the rabbit and fox. We also initialise the vector dzdt as a 2 by 1 column

```
function dzdt = foxode1(t, z, s_r, s_f)
%definition of ODE for the Fox-Rabbit chase

r = [s_r*cos(pi/4)*t s_r*sin(pi/4)*t]; %the position of the rabbit

dist = sqrt((r(1)-z(1))^2 + (r(2)-z(2))^2);%distance between fox and rabbit
dzdt = zeros(2,1); %2x1 column vector
dzdt(1) = s_f*(r(1)-z(1))/dist; %horizantal velocity
dzdt(2) = s_f*(r(2)-z(2))/dist; %vertical velocity
end
```

Figure 2: My code listing for function foxode1.

vector before setting it equal to the ODEs we defined above. We will later use the ode45 solver to solve the ODEs defined.

The events function - foxrab1 is the other main function in the program. It uses the inbuilt properties and functions of Matlab and its ODE solvers to stop the solution when certain conditions are met. In our case these conditions are (1) the fox catching the rabbit (when the distance between the two is smaller than a given minimum distance), or (2) the rabbit reaching the burrow. We code the first by calculating the distance between the fox and the rabbit and subtracting off the minimum distance. If its zero or negative the solution is stopped. This is done by setting isterminal to 1 and direction to -1 (as we're interested in when the value turns 0 or negative). To code the second, we subtract the horizontal and vertical co-ordinates of the rabbit from the corresponding co-ordinates of the burrow and if its zero or negative the solution is stopped.

```
function [value, isterminal, direction] = foxrab1(t, z, s_r,mindist,burrow)
r = [s_r*cos(pi/4)*t s_r*sin(pi/4)*t];
burrow = [600, 600];
value(1) = sqrt((r(1)-z(1))^2 + (r(2)-z(2))^2) - mindist; %fox catches
                                                          %rabbit
isterminal(1) = 1;
direction(1) = -1;
value(2) = (burrow(1) - r(1) > 0 && burrow(2) - r(2) > 0); %if rabbit
                                                           %reaches burrow
isterminal(2) = 1;
direction(2) = -1;
end
```

Figure 3: My code listing for function foxrab1.

The remaining work is to define the variables with the initial conditions given and set a time span for the calculation to run over and use the ode45 solver to solve the ODEs defined in foxode1. We configure options so that it uses the events function foxrab1. We call ode45 as follows:

```
[t, z, te, ze, zi] = ode45(@(t,z)foxode1(t, z, sr, sf), ts, z0, options);
```

Figure 4: The ode45 line

This solves foxode1, with the given values and taking in to account the events function. It returns vectors t and z (time and positions of the fox at time t. It also returns te - the time elapsed in the calculation, ze - the position of the fox at the end, and zi - which is either 1 or 2 depending on which event happened.

For the question we get the output:

```
te =

   65.2714


ze =

  456.4679  416.5239


zi =

     2
```

Figure 5: Outputs for question

So the rabbit reached the burrow safely after 65.2714 seconds and the horizontal and vertical positions of the fox at that time were 456.4679 and $4.16.5239$ metres, respectively.

We also plot the positions of the rabbit and fox and add a Legendre box on a graph using the code:

This generates the graph:

```
plot(z(:,1), z(:,2), sr*cos(pi/4)*t, sr*sin(pi/4)*t)
legend('Fox', 'Rabbit', 'Location', 'Best')
```

Figure 6: Code to create graph

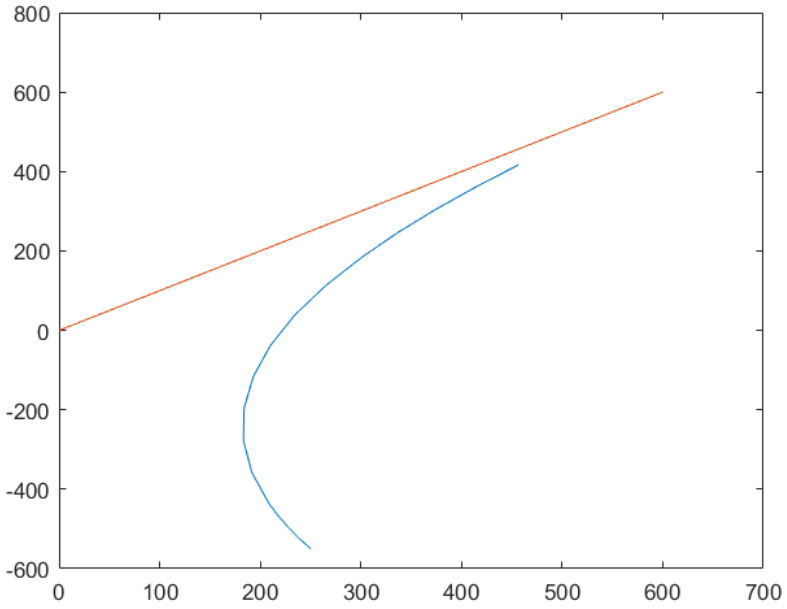

Figure 7: Graph for question 1

3 Question 2

In the new scenario the speeds of the rabbit and for are non-constant and decreasing with distance. The speed of the rabbit is given by $\mathrm{s_r} = \mathrm{s_{r_0}} \exp\left(-0.0008\mathrm{d_r(t)}\right)$ and the speed of the fox is given by $\mathrm{s_f} = \mathrm{s_{f_0}} \exp\left(-0.0002\mathrm{d_f(t)}\right)$, where $\mathrm{s_{r_0}} = 13$ and $\mathrm{s_{f_0}} = 16$ are the initial velocities of the rabbit and fox, respectively. $\mathrm{d_r(t))}$ and $\mathrm{d_f(t))}$ are the distance travelled by the rabbit and the fox at time t, respectively.

The main difference between question 1 and 2 is that now we must use ODEs to find the position of the rabbit. We do this with a function rabode. The ODE's we need to solve to find the position at time t are related to horizontal velocity, vertical velocity and arc

length

$$v_x = \mathrm{s_{f_0}} \cos(\pi/4) \exp\left(-0.0002\mathrm{d_f(t)}\right)$$

$$v_y = \mathrm{s_{f_0}} \sin(\pi/4) \exp\left(-0.0002\mathrm{d_f(t)}\right)$$

$$\frac{dr}{dt} = \sqrt{((v_x)^2 + (v_y)^2)}$$

Using the fact that $\int \frac{dr}{dt}\, dt = \mathrm{d_f(t)}$, we code the differential equation as

```
function drdt = rabode(t,r, s_r)
%definition of ODE for rabbit - calculate distance travlled by rabbit
    drdt = zeros(3,1);
    drdt(1) = s_r*cos(pi/4)*exp(-0.0008*r(3));
    drdt(2) = s_r*sin(pi/4)*exp(-0.0008*r(3));
    drdt(3) = sqrt((drdt(1))^2 + (drdt(2))^2);
end
```

Figure 8: Code for rabode

Similar to this and foxode1 from question 1, we code the ODEs for the position of the fox as

```
function dzdt = foxode2(t, z, s_f)
%definition of ODE for the Fox
        dist = sqrt((r(1)-z(1))^2 + (r(2)-z(2))^2);%distance between fox and
     rabbit
        dzdt = zeros(3,1); %2x1 column vector
        dzdt(1) = s_f*exp(-0.0002*z(3))*(r(1)-z(1))/dist; %horizantal
    velocity
        dzdt(2) = s_f*exp(-0.0002*z(3))*(r(2)-z(2))/dist; %vertical velocity
        dzdt(3) = sqrt(dzdt(1)^2 + dzdt(2)^2);
```

Figure 9: Code for foxode2

The events function is exactly the same as it is in question 1.

We initialise the variables and time span like we did in question 1. Though here the initial conditions are three dimensional because of the ODEs ("x-component", "y-component" and "distance component" ($\sqrt{(x^2 + y^2)}$). We then invoke the ODE solvers to solve rabode and foxode2 using the code:

```
options = odeset('Events', @(t,z)foxrab2(t,z,sr0, mindist, burrow));
[t, r] = ode45(@(t,r)rabode(t, r, sr0), ts, rz0);
[t, z, te, ze, zi] = ode45(@(t,z)foxode2(t, z, sf0), ts, fz0, options);
```

Figure 10: calls to ode45

Because rabode is solved before foxode, the variable r is usable in the latter. As with question 1, we get the outputs te, ze, zi,

```
te =

   40.1284


ze =

    0.0414   -0.0910  604.0523


zi =

     1
```

Figure 11: Outputs for question 2

According to the outputs, the fox catches the rabbit after 40.1284 seconds have elapsed.

We can make a plot the positions of the rabbit and the fox with the code:

```
 plot(r(:,1), r(:,2), z(:,1), z(:,2))
legend('Rabbit', 'Fox', 'Location', 'Best')
```

Figure 12: Plot for question 2

The plot is shown in figure 13. Clearly something has gone wrong, the fox only moves towards the rabbits initial position and does not "chase" after it. I was unable to find the problem and fix it but I think I may have defined and solved the wrong ODEs in foxode2.

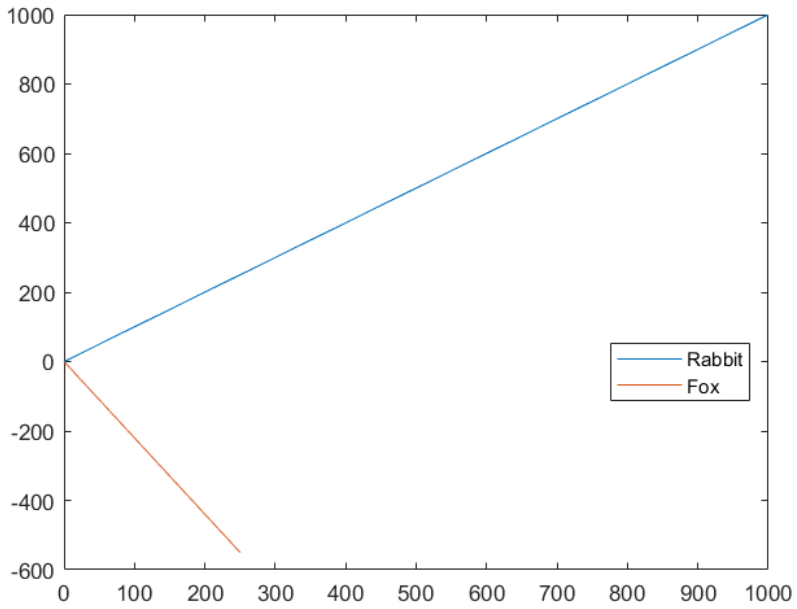

Figure 13: Plot for question 2

4 References

1. Gajjar, J., 2021. MATH36032 Project 2 - deadline 12th April 2021, time 1100hrs.. [online] https://online.manchester.ac.uk/. Available at: <https://online.manchester.ac.uk/bbcswebdav/pid-12372610-dt-content-rid-67974337_1/courses/I3133-MATH-36032-1201-2SE-009260/Proj2.pdf>[Accessed 16 April 2021].

5 Appendix

```
function foxrabchase1
%A fox rabbit pursuit simulation based on the values and criteria of
%question 1

sr = 13; %speed of rabbit
sf = 16; % "     " fox
z0 = [250 -550]; %ic for fox
ts = [0 norm(z0)/(sf-sr)]; % timespan
mindist = 0.1;
burrow = [600 600];
options = odeset('Events', @(t,z)foxrab1(t,z,sr, mindist, burrow));
[t, z, te, ze, zi] = ode45(@(t,z)foxode1(t, z, sr, sf), ts, z0, options);
te, ze, zi
plot(z(:,1), z(:,2), sr*cos(pi/4)*t, sr*sin(pi/4)*t)

function dzdt = foxode1(t, z, s_r, s_f)
%definition of ODE for the Fox-Rabbit chase

r = [s_r*cos(pi/4)*t s_r*sin(pi/4)*t]; %the position of the rabbit

dist = sqrt((r(1)-z(1))^2 + (r(2)-z(2))^2);%distance between fox and rabbit
dzdt = zeros(2,1); %2x1 column vector
dzdt(1) = s_f*(r(1)-z(1))/dist; %horizantal velocity
dzdt(2) = s_f*(r(2)-z(2))/dist; %vertical velocity
end

function [value, isterminal, direction] = foxrab1(t, z, s_r,mindist,burrow)
r = [s_r*cos(pi/4)*t s_r*sin(pi/4)*t];
burrow = [600, 600];
value(1) = sqrt((r(1)-z(1))^2 + (r(2)-z(2))^2) - mindist; %fox catches
                                                          %rabbit
isterminal(1) = 1;
direction(1) = -1;
value(2) = (burrow(1) - r(1) > 0 && burrow(2) - r(2) > 0); %if rabbit
                                                           %reaches burrow
isterminal(2) = 1;
direction(2) = -1;
end
end
```

Figure 14: Code for question 1

```
function foxrabchase3
sr0 = 13;
sf0 = 16;
rz0 = [0 0 0];
fz0 = [250 -550 0];
ts = [0 norm(fz0)/(sf0-sr0)];
mindist = 0.1;
burrow = [600 600];
options = odeset('Events', @(t,z)foxrab2(t,z,sr0, mindist, burrow));
[t, r] = ode45(@(t,r)rabode(t, r, sr0), ts, rz0);
[t, z, te, ze, zi] = ode45(@(t,z)foxode2(t, z, sf0), ts, fz0, options);
te, ze, zi
plot(r(:,1), r(:,2), z(:,1), z(:,2))
legend('Rabbit', 'Fox', 'Location', 'Best')

function drdt = rabode(t,r, s_r)
%definition of ODE for rabbit - calculate distance travlled by rabbit
    drdt = zeros(3,1);
    drdt(1) = s_r*cos(pi/4)*exp(-0.0008*r(3));
    drdt(2) = s_r*sin(pi/4)*exp(-0.0008*r(3));
    drdt(3) = sqrt((drdt(1))^2 + (drdt(2))^2);
end

function dzdt = foxode2(t, z, s_f)
%definition of ODE for the Fox
        dist = sqrt((r(1)-z(1))^2 + (r(2)-z(2))^2);%distance between fox and
     rabbit
        dzdt = zeros(3,1); %2x1 column vector
        dzdt(1) = s_f*exp(-0.0002*z(3))*(r(1)-z(1))/dist; %horizantal
    velocity
        dzdt(2) = s_f*exp(-0.0002*z(3))*(r(2)-z(2))/dist; %vertical velocity
        dzdt(3) = sqrt(dzdt(1)^2 + dzdt(2)^2);
end

function [value, isterminal, direction] = foxrab2(t, z, s_r, mindist, burrow
    )
        burrow = [600, 600];
        value(1) = sqrt((r(1)-z(1))^2 + (r(2)-z(2))^2) - mindist;
        isterminal(1) = 1;
        direction(1) = -1;
        value(2) = (burrow(1) - r(1) > 0 && burrow(2) - r(2) > 0);
        isterminal(2) = 1;
        direction(2) = -1;
end
end
```

Figure 15: Code for question 2