MATH36031 Project 2

10680317

Nov 2023

1 Introduction

This project is about a simulation for the fox-rabbit chasing problem under the given two situations. Figure 2 is the background for this project. The starting point of red path (0,0) is the initial point for fox. Rabbit starts from the initial point of green path which is (0,800). The burrow is located at point (800(-sin(π/3),800cos(π/3))), and the rabbit moves along a circular path from its initial point to the burrow. Points E (-500,-350) and A (-350,620) are two points of the fence. While the fox is chasing behind the rabbit, if the fence blocks the sight of the fox, the fox will run directly to the point A. If the fox gets the vision of the rabbit, it will chase after it.

The first situation follows the assumption that rabbit runs at a speed of 12m/s and fox also runs at a constant speed of 17m/s. The second situation assumes that they are both running at two diminishing speeds, which are two functions related to t. The aim for this report is: (i) Find the time and the position at the instant when the rabbit is either caught or reaches the burrow. (ii) Total distance covered by the fox over the time, and provide graphical representation for fox and rabbit.

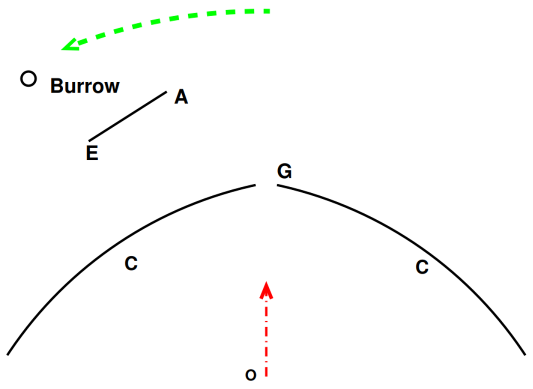

Figure 1: Problem description

2 Main functions

2.1 Ode45

`ode45`[1] is one of the main functions used in this project, which is the key step to solving the Ordinary Differential Equations (Odes). This function implements a Runge-Kutta method [1] with a variable time

[1]In this report, all the code variables will be demonstrated in text-mode.

step for efficient computation. The basic structure of the function is:

$$\texttt{[t, y, te, ze, zi] = ode45(odefun, tspan, y0, options).} \tag{1}$$

In equation 2, the variables on the left-hand side include `te`, which indicates the time points at which the events specified in the options occur, and `ze`, which yields the relevant solutions at these event times. The variable `zi` represents the indices of these events. The outcomes `t` and `y` are solutions to the Odes outlined in the function. The application of `ode45` in this project is instrumental in tracing the fox's trajectory and determining if the rabbit manages to escape.

2.2 Event Function

The general form of event function is given as:

$$\texttt{[value, isterminal, direction] = myEventsF cn(t, y).} \tag{2}$$

The event function comprises multiple variables. `value(i)` represents a mathematical expression for the i th event, which is triggered when `value(i)` equals zero. `isterminal(i)` indicates whether `ode45` should cease operation, with `1` signifying a stop and `0` indicating continuation. Additionally, `direction(i)` specifies the trend of `value` approaching 0. A `+1` identifies zeros only where the event function is on an upward trend, while `-1` identifies zeros where the function is decreasing.

2.3 Sight

This function is used to determine whether or not the fence block the sight of the fox. The full demonstration for this function can be found in figure 2.

From (**Line3**) to (**Line6**), we define the end points of two line segment from the fox to rabbit and the fence from point E to point A. From (**Line8**) to (**Line10**), the `doLinesIntersect` function is called to determine whether or not the two lines intersect. The function (`isClear =~ isBlocked`) returns true if there is an intersection, and false if not.

From (**12**) to (**25**), we will use the following mathematical formulas to determine the intersection:

$$t = \frac{(a_2 - c_2)(d_1 - c_1) - (a_1 - c_1)(d_2 - c_2)}{\text{denominator}} \tag{3}$$

$$u = \frac{(b_1 - a_1)(a_2 - c_2) - (b_2 - a_2)(a_1 - c_1)}{\text{denominator}} \tag{4}$$

$$\text{denominator} = (b_1 - a_1)(d_2 - c_2) - (b_2 - a_2)(d_1 - c_1) \tag{5}$$

If the denominator is 0, the lines are parallel or coincident, and they only intersect if `t` and `u` are within the range [0,1].

3 Question 1

3.1 ODEs function

For problem 1, we have to find the corresponding ODEs. Set the fox's position as a 2-by-1 column vector `z(z1,z2)`. Similarly set the rabbit's position as `r(r1,r2)`. Since we have two constant speeds for fox and rabbit which is 17m/s and 12m/s respectively. The ODEs for this problem are:

$$\frac{dz}{dt}(1) = Sf_0 \frac{r_1(t) - z_1(t)}{\sqrt{(r_2(t) - z_2(t))^2 + (r_1(t) - z_1(t))^2}} \tag{6}$$

```
function isClear = Sight(z, r, A, E)
    % Check if the line from the fox (z) to the rabbit (r) intersects the line from E to A
    foxToRabbitStart = z;        % Line segment from fox to rabbit
    foxToRabbitEnd = r;
    FenceStart = E;               % Line segment representing the fence
    FenceEnd= A;
    % Perform intersection test
    isBlocked = doLinesIntersect(foxToRabbitStart, foxToRabbitEnd, FenceStart, FenceEnd);
    % If blocked, return false. Otherwise, return true
    isClear = ~isBlocked;
end
function intersect = doLinesIntersect(a, b, c, d)
    % Function to check if line segments ab and cd intersect
    % Calculate the denominator
    denominator = (b(1) - a(1)) * (d(2) - c(2)) - (b(2) - a(2)) * (d(1) - c(1));
    % Lines are parallel or coincident
    if denominator == 0
        intersect = false;
        return;
    end
    % Calculate parameters t and u
    t = ((a(2) - c(2)) * (d(1) - c(1)) - (a(1) - c(1)) * (d(2) - c(2))) / denominator;
    u = ((b(1) - a(1)) * (a(2) - c(2)) - (b(2) - a(2)) * (a(1) - c(1))) / denominator;
    % Check if t and u fall within the range [0, 1]
    intersect = (t >= 0 && t <= 1) && (u >= 0 && u <= 1);
end
```

Figure 2: Function code: `Sight`.

$$\frac{dz}{dt}(2) = Sf_0 \frac{r_2(t) - z_2(t)}{\sqrt{(r_2(t) - z_2(t))^2 + (r_1(t) - z_1(t))^2}} \tag{7}$$

The full demonstration of the code can be found in figure 3.
In (**Line3**), the rabbit's position `r` is calculated based on the time `t`. Where the rabbit moves in a circular path centered around the origin (0,0), with a radius of 800 units. The position changes over time, creating a dynamic chase scenario. Then we consider two cases.
For case 1 (**Line5**) to (**Line9**), if the fox sees the rabbit (`Sight=1`), and calculate the distance between the fox and the rabbit by using the Pythagorean theorem. Then calculate the horizontal velocity and vertical velocity by using formula 6 and 7. For case 2 (**Line10**) to (**Line14**), if the fox does not sees the rabbit(`Sight=0`), the fox changes to move towards point A and using the same logic that previously mentioned. At the final part of the code, we want the fox to pass through point G, so we set the horizontal velocity to 0 and vertical velocity to `s_f`, until it pass (0,300).

3.2 Event Function

This function helps in stopping the ODE solver under specific conditions related to a fox-rabbit chase scenario. The full demonstration can be found in figure 4. we define the position of the rabbit r as $\left(-800\sin\left(\frac{s_r t}{800}\right),\ 800\cos\left(\frac{s_r t}{800}\right)\right)$, and `dis` is the Euclidean distance between the fox and rabbit. For event 1, `value(1)` is set to the difference between the current distance and `mindist` is the minimum distance for the fox to catch the rabbit (0.1m). For event 2, `value(2)` calculates the horizontal distance between the rabbit and its burrow. First condition checks if the fox is close enough to catch the rabbit, and the second checks if the rabbit is close to its burrow. When either of these conditions is met, the ODE solver will stop the simulation.

```
function dzdt=foxode1(t,z,s_r,s_f,A,E)
% Definition of ODE for fox-rabbit chase.
r=[800*-sin(s_r*t/800) 800*cos(s_r*t/800)];  % Position of rabbit
% The equations related to fox have to be discussed under two different situations.
   if Sight(z,r,A,E)==1                      % If fox get the vision of rabbit
   dist=sqrt((r(1)-z(1))^2+(r(2)-z(2))^2);   % Distance between fox and rabbit.
   dzdt=zeros(2,1);                          % Set z as a column vector.
   dzdt(1)=s_f*(r(1)-z(1))/dist;             % Horizontal velocity of fox
   dzdt(2)=s_f*(r(2)-z(2))/dist;             % Vertical velocity of fox
   elseif Sight(z,r,A,E)==0                  % If fox does not get the vision of rabbit
   dist=sqrt((A(1)-z(1))^2+(A(2)-z(2))^2);   % Distance between fox and point A.
   dzdt=zeros(2,1);                          % Set z as a column vector.
   dzdt(1)=s_f*(A(1)-z(1))/dist;             % Horizontal velocity of fox
   dzdt(2)=s_f*(A(2)-z(2))/dist;             % Vertical velocity of fox
   end
if z(2)<300                                  % Set a line for fox to come out the gate
   dzdt(1)=0;
   dzdt(2)=s_f*1;
end
end
```

Figure 3: ODEs function code: `foxode1`.

```
function [value, isterminal,direction]=foxrab1(t,z,s_r,mindist,B)
% ODE45 will be stopped if following two condition happened.
r=[800*-sin(s_r*t/800) 800*cos(s_r*t/800)]; % position of rabbit
   dis=sqrt((r(1)-z(1))^2+(r(2)-z(2))^2); % distance between fox and rabbit
   % If distance between fox and rabbit tends to 0, this event will be triggered.
   value(1)=dis-mindist;
   isterminal(1)=1;
   direction(1)=-1;
   % If distance between rabbit and burrow tends to 0, this event will be triggered.
   value(2)=r(1)-B(1);
   isterminal(2)=1;
   direction(2)=-1;
end
```

Figure 4: Function code: `foxrab1`.

3.3 Output of the code

In the final part of question 1, we will input the value from section 1 for Ode45 to compute the solution and the path. The background of the graph is written in the function `backgroundplot`, which includes all of the given positions in the questions. The output code and the graph can be found in the figure 5 and 7 respectively. In the ODE function, `RelTol` [2] is been used to minimise mistakes and obtain a more accurate result. From (**Line18**) to (**Line24**), the output shows that at time 69.8132s, the event 2 is been triggered and this means that the rabbit escapes to the burrow. At that moment, the fox moves to the position [-677.8306 422.3709], and the distance travelled by the fox is 1186.8m. The figure 7 illustrates that the fox's pursuit is momentarily obstructed by point A, as evidenced by the distinct change in its path. Notably, when the fox loses sight of the rabbit, its trajectory shifts to a straight line. This alteration in movement pattern persists until the fox passes point A and regains visual contact with the rabbit.

```
s_r = 12;                                          % Speed of rabbit
s_f = 17;                                          % Speed of fox
A = [-350,620];                                    % Set Point A
E = [-500,350];                                    % Set Point E
z0 = [0,0];                                        % Fox start chasing with point z0
ts = [0,100];                                      % Time span from 0 to 100 seconds
mindist = 0.1;                                     % Minimum distance
B = [800*(-sin(pi/3)), 800*cos(pi/3)];             % B represents the point of Burrow
% Solve the ODE using event function and ode45
options = odeset('Events', @(t,z)foxrab1(t,z,s_r,mindist,B),'RelTol',1e-9);
[t, z, te, ze, zi] = ode45(@(t,z)foxode1(t,z,s_r,s_f,A,E), ts, z0, options);
te,ze,fd=te*s_f,zi
backgroundplot(A, E, z0);                          % Plot background
% Plot the paths of fox and rabbit
plot(z(:,1), z(:,2), 800*-sin(s_r*t/800), 800*cos(s_r*t/800));
te =
   69.8132
ze =
 -677.8306 422.3709
fd =
   1.1868e+03
zi =
     2
```

Figure 5: Output of question 1.

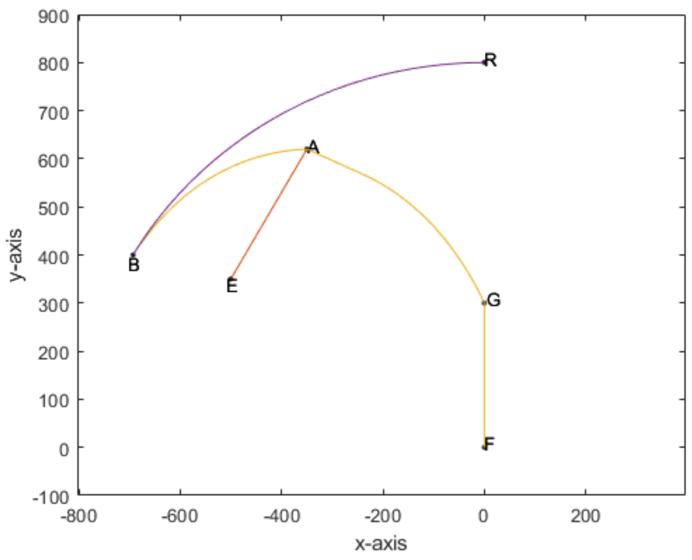

Figure 6: Question 1 graph.

4 Question 2

4.1 ODEs function

In question 2, instead of constant speed, the speed of fox and rabbit diminish in time and their speed at time t are given by:

$$s_f(t) = s_{f0}e^{-\mu_f d_f(t)}, \quad s_r(t) = s_{r0}e^{-\mu_r d_r(t)} \tag{8}$$

where the initial speed of fox and rabbit are 17 and 12 respectively. The speeds decay rates are given as $\mu_f = 0.0002m^{-1}$ for the fox and $\mu_r = 0.0008m^{-1}$ for the rabbit. Here $d_f(t)$ and $d_r(t)$ denote the respective distances covered by each up until any given time t, which is grater than 0.
We can design a system comprising six variables. In this context, `z` is defined as a 6-element column vector. The first two elements `z1, z2`, represent the fox's position, while `z3, z4` denote the location of the rabbit. The fifth element, `z5`, tracks the distance traveled by the fox, and the sixth element, `z6`, does the same for the rabbit. Hence, the ODEs for this question are:

$$\frac{dz}{dt}(5) = s_{f0}e^{-\mu_f d_f(t)} \tag{9}$$

$$\frac{dz}{dt}(6) = s_r(t) = s_{r0}e^{-\mu_r d_r(t)} \tag{10}$$

$$d_f(t) = z(5),\ d_r(t) = z(6) \tag{11}$$

The full demonstration of the function can be found in figure 7. We follow the basic logic of question 1 with some extra conditions. (**Line7**) and (**Line8**) states the diminishing speed of fox and rabbit using formula 8. (**Line9**) intialise a vector `dzdt` of six elements to 0. From (**Line11**) to (**Line15**), the horizontal velocity and vertical velocity of rabbit can be found by using the formula of angular velocity[3]:

$$\frac{dz}{dt}(3) = -\frac{s_r}{r}r_p(2); \frac{dz}{dt}(4) = \frac{s_r}{r}r_p(1); \tag{12}$$

`r_p` is the position of rabbit `z3` and `z4`.

4.2 Event function

The full demonstration for the event function of question 2 can be found in figure 8. The basic logic for this section is generally the same as section 3.2. Since we are using a 6-element column vector, the definition for position of fox and rabbit have to be replaced, this is showed in (**Line4**) and (**Line5**).

4.3 Output of the code

The last part of question 2 follows the same procedure as section 3.3, in addition add the rate of diminishing speed for two animals. The output code and the graph can be found in the figure 9 and 10 respectively. From (**Line16**) to (**Line23**), event 1 is been triggered and this means that the fox catches the rabbit at the time 67.2355s. At the moment, the fox moves to the position [-561.5742 569.7670] and has been travelled for 1029.4m. In figure 10, it is evident that upon crossing point G, the fox persistently follows the rabbit without deviating towards point A. This observation strongly suggests that the fox maintains a visual lock on the rabbit for the entirety of the chase.

References

[1] Wikipedia Contributors (2019). Runge–Kutta methods. [online] Wikipedia. Available at: `https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods.`

[2] uk.mathworks.com. (n.d.). Create or modify options structure for ODE and PDE solvers - MATLAB odeset - MathWorks United Kingdom. [online] Available at: `https://uk.mathworks.com/help/matlab/ref/odeset.html`

[3] Gsu.edu. (2019). Rotational Quantities. [online] Available at: `http://hyperphysics.phy-astr.gsu.edu/hbase/rotq.html.`

```
function dzdt=foxode2(t,z,s_r0,s_f0,A,E,u_r,u_f)
% Definition of the second ODE.
f_p=z(1:2);                             % position of fox
r_p=z(3:4);                             % position of rabbit
d_f=z(5);                               % Distance travelled by the fox in time t
d_r=z(6);                               % Distance travelled by the rabbit in time t
s_f=s_f0*exp(-u_f*d_f);                 % diminishing speed of fox
s_r=s_r0*exp(-u_r*d_r);                 % diminishing speed of rabbit
dzdt=zeros(6,1);                        % set a vector of dzdt
% Circular path parameters
radius = 800;                           % Radius of the circle
angular_velocity = s_r / radius;        % Angular velocity (radians per unit time)
% Set rabbit's velocity for circular path
dzdt(3) = -angular_velocity * (r_p(2)); % Horizontal velocity component for circular path
dzdt(4) = angular_velocity * (r_p(1));  % Vertical velocity component for circular path
dzdt(5)=s_f;                            % Speed of fox.
dzdt(6)=s_r;                            % Speed of rabbit
   if Sight(f_p,r_p,A,E)==1        % If the rabbit in fox's sight, then direct to rabbit
   d=sqrt((r_p(1)-f_p(1))^2+(r_p(2)-f_p(2))^2);
   dzdt(1)=s_f*(r_p(1)-f_p(1))/d;   % the velocity of fox if towards rabbit
   dzdt(2)=s_f*(r_p(2)-f_p(2))/d;
   elseif Sight(f_p,r_p,A,E)==0    % If the fox's don't have sight, fox toward A
   d=sqrt((A(1)-f_p(1))^2+(A(2)-f_p(2))^2);
   dzdt(1)=s_f*(A(1)-f_p(1))/d;     % The velocity of fox if towards A
   dzdt(2)=s_f*(A(2)-f_p(2))/d;
   end
if z(2)<300
    dzdt(1)=0;
    dzdt(2)=s_f*1;
end
end
```

Figure 7: ODEs function code: `foxode2`.

```
function [value, isterminal, direction] = foxrab2(t, z, s_r, mindist, B)
    % ODE45 will be stopped if the following conditions happen.
    % Extract the positions of the fox and rabbit from the state vector z
    f_p = [z(1), z(2)]; % Position of fox
    r_p = [z(3), z(4)]; % Position of rabbit
    % Distance between fox and rabbit
    dis= norm(f_p - r_p);
    value(1)=dis-mindist;
    isterminal(1)=1;
    direction(1)= -1;
    % If distance between rabbit and burrow tends to 0, this event will be triggered.
    value(2)=r_p(1)-B(1);
    isterminal(2)=1;
    direction(2)= -1;
end
```

Figure 8: Function code: `foxrab2`.

```
s_r0=12                     % Speed of rabbit at t = 0.
s_f0=17;                    % Speed of fox at t = 0.
A=[-350,620];               % Set Point A
E=[-500,350];               % Set Point E
mindist=0.1;                % Minimum distance
u_f=0.0002;                 % Rate of the diminishing speed of fox
u_r=0.0008;                 % Rate of the diminishing speed of rabbit
z0=[0;0;0;800;0;0];         % z1 & z2 imply the position of Fox while z3 & z4 represent the
    rabbit's position. z5 is distance travelled by fox, z6 is the distance travelled by rabbit
ts = [0,100];
options2=odeset('Events', @(t,z)foxrab2(t,z,s_r,mindist,B),'RelTol',1e-9);
[t,z,te,ze,zi]=ode45(@(t,z)foxode2(t,z,s_r0,s_f0,A,E,u_r,u_f),ts,z0,options2);
te,fp=ze(3:4),fd=ze(5),zi
backgroundplot(A,E,z0);  % Plot background
plot(z(:,1),z(:,2),z(:,3),z(:,4));
te =
   67.2355
fp =
 -561.5742 569.7670
fd =
   1.0294e+03
zi =
     1
```

Figure 9: Output of question 2.

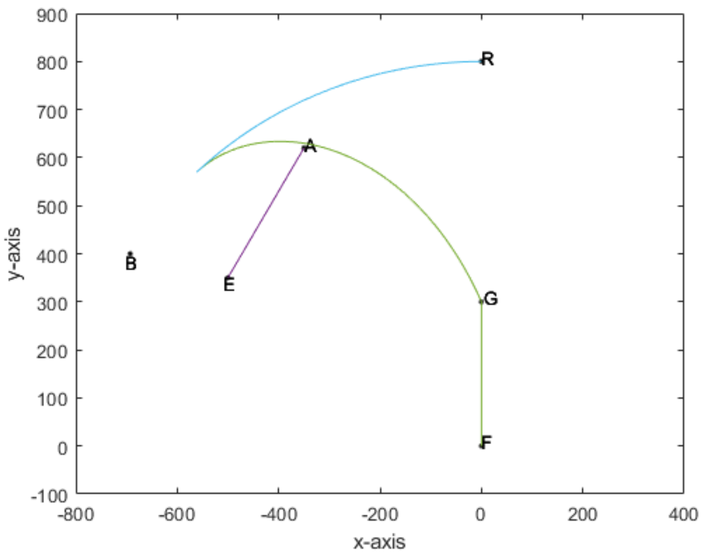

Figure 10: Question 2 graph.