Project 2 Report - Fox and Rabbit Dynamics

UID: 10450192

November 25, 2021

1 Introduction to the problem

The assignment considers the dynamics of a fox's hunt for a rabbit (Figure 1). The latter (red diamond) starts at position $(0, 800)$ and tries to escape to its burrow (rec circle) located at $800(\sin(\pi/3), \cos(\pi/3))$ by following a circular path of radius 800 (NB one can wonder if this path is the safest one for the rabbit - it is longer then a straight path to the burrow but is not as close to the fox). The fox (blue triangle) starts at position $(0, 0)$ but has to first leave a circular enclosure (dotted circle) through a gate located at $(0, 300)$. After that, it will move directly towards the rabbit (blue path) unless it cannot see the rabbit because its sight is blocked by a barrier with endpoints A and B. In that case, it will first move towards point A and then return to following the rabbit directly (red path). Importantly, the two paths on the diagram are simplified as the actual path of the fox may first lead him towards the rabbit until it loses sight of it, at which point it will start moving towards A and then return to following the rabbit. The fox catches the rabbit if the distance between them is less or equal to 0.1m. If the rabbit reaches the burrow before that happens, he successfully escapes. Finally, there are two cases to be analysed: a) the rabbit and the fox both move at constant speeds $s_{r0} = 13\text{ms}^{-1}$ and $s_{f0} = 16\text{ms}^{-1}$ respectively and b) their speeds change with distance as $s_0 \exp(-\mu d(t))$ where d is the distance traveled at time t with $\mu_r = 0.0008\text{m}^{-1}$ and $\mu_f = 0.0002\text{m}^{-1}$ for rabbit and fox respectively.

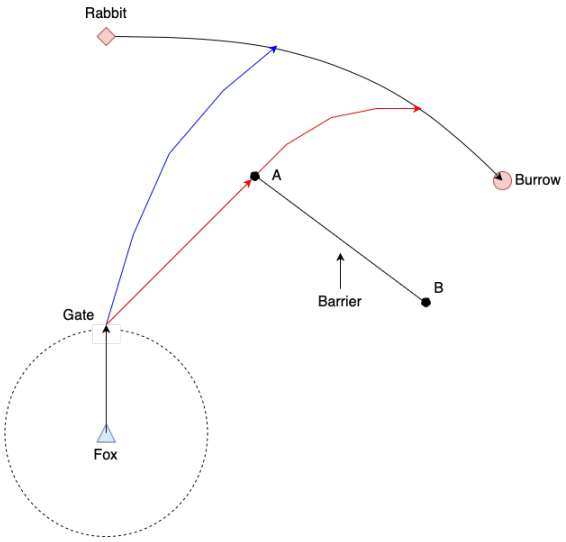

Figure 1: A schematic diagram of the situation at the beginning and some of possible paths.

2 Deriving the Equations of Motion

2.1 Speed Function

We begin this section by noticing that the first scenario with constant speeds is simply a special case of the second scenario for $\mu_r = \mu_f = 0$, which implies $s(t) = s_0 \exp(0 \cdot d(t)) = s_0$. This means we can effectively derive one set of equations of motion for variable speeds, which will work equally well for constant speeds (though its numerical solution may take longer to compute). Next, we turn to the expression for speed at distance d. We can rewrite $d(t)$ as $\int_0^t s(\tau)d\tau$ which can be readily substituted into the expression for s:

$$s(t) = s_0 \exp(-\mu \int_0^t s(\tau)d\tau).$$

We can now divide both sides by s_0 and take the natural logarithm:

$$\ln(\frac{s(t)}{s_0}) = -\mu \int_0^t s(\tau)d\tau.$$

If we now take the time derivative of both sides we obtain a separable, first-order ODE:

$$\dot{s} = -\mu s^2,$$

where $\dot{s}$ denotes the time derivative of s as per standard physics convention (which will be used throughout this report). Solving the above equation with initial condition $s(0) = s_0$ yields:

$$s = \frac{1}{\mu t + 1/s_0}. \tag{1}$$

Equation (1) is very useful for our purpose because it means we have an analytic expression for s at any time t and do not have to concern ourselves with calculating the distance travelled by rabbit or fox at any point (which would be either done by another ODE or integration within our primary ODE, significantly slowing down the calculation).

2.2 Rabbit's Motion

To begin, we will denote the position vector of the rabbit as $\mathbf{r}_r$, hence its velocity is $\dot{\mathbf{r}}_r$. Since we know the path taken by the rabbit will be circular and directed towards the burrow we can readily write its velocity as:

$$\dot{\mathbf{r}}_r = s_r(t) \begin{bmatrix} \cos(\theta(t)) \\ -\sin(\theta(t)) \end{bmatrix},$$

where θ is the angle between starting and current position of rabbit at time t[1]. To make the analysis more tractable we start with the case of constant speed and use a substitution $\theta(t) = \omega \cdot t$, where ω is the angular velocity. We integrate the above equation for $\dot{\mathbf{r}}_r$ with respect to time:

$$\mathbf{r}_r = \frac{s_{r0}}{\omega} \begin{bmatrix} \sin(\theta(t)) \\ \cos(\theta(t)) \end{bmatrix} + \mathbf{c}.$$

It is not difficult to see that in order for the path to be circular with radius 800m and centred at $(0, 0)$ we need $\omega = \frac{r_{r0}}{800\text{m}}$ and $\mathbf{c} = 0$. Thus we have for $\dot{\mathbf{r}}_r$ (skipping units for clarity):

$$\dot{\mathbf{r}}_r = s_{r0} \begin{bmatrix} \cos(\frac{s_{r0}t}{800}) \\ -\sin(\frac{s_{r0}t}{800}) \end{bmatrix}.$$

Next, we make an ansatz for the varying speeds case inspired by the form of the constant speed form of the ODE in which we substitute $s_r(t)$ for s_{r0}. It is not immediately obvious that this will be the correct form of the ODE but numerically solving it using MATLAB shows that it does work. The caveat is that we need to introduce θ as another variable we are solving for in our ODE and hence we finally obtain the following set of equations for motion of the rabbit:

$$\dot{\mathbf{r}}_r = s_r(t)\begin{bmatrix}\cos(\theta(t))\\ -\sin(\theta(t))\end{bmatrix} \tag{2}$$

$$\dot{\theta} = \frac{s_r(t)}{800\text{m}}. \tag{3}$$

2.3 Fox's Motion

Similarly to the notation we used for the rabbit, we will denote the fox's position and velocity as $\mathbf{r}_f$ and $\dot{\mathbf{r}}_f$ respectively. We will also need to define the position of the gate as $\mathbf{r}_{gate}$ and point A as $\mathbf{r}_A$. The fox's motion starts easily enough, for the initial stage its velocity is:

$$\dot{\mathbf{r}}_f = s_f(t)\begin{bmatrix}0\\ 1\end{bmatrix}, \tag{4}$$

as long as $\mathbf{r}_f(2) < \mathbf{r}_{gate}(2)$ (where (2) refers to the second component of a vector as per MATLAB syntax). After reaching the gate, the motion becomes slightly more involved. If the fox sees the rabbit, the equation becomes:

$$\dot{\mathbf{r}}_f = s_f(t)\frac{\mathbf{r}_r - \mathbf{r}_f}{|\mathbf{r}_r - \mathbf{r}_f|}, \tag{5}$$

where $(\mathbf{r}_r - \mathbf{r}_f)/|\mathbf{r}_r - \mathbf{r}_f| = \hat{\mathbf{r}}_{f\to r}$ is the unit vector pointing from the fox towards the rabbit[1]. Similarly, if the fox cannot see the rabbit because the barrier is in the way, the equation becomes (using the unit vector notation from above):

$$\dot{\mathbf{r}}_f = s_f(t)\hat{\mathbf{r}}_{f\to A}. \tag{6}$$

3 Implementation and Results

Overall, one main script (`main.m`) and three function scripts (`VelocitySystem.m`, `CanSeeRabbit.m` and `CatchEvents.m`) have been written in order to solve the problems in this project.

3.1 `CatchEvents.m`

This is a relatively simple function which catches 4 types of events in the run of the ode45 solver:

1. the fox reaches the gate
2. the fox stops seeing/can again see the rabbit
3. the fox catches the rabbit
4. the rabbit manages to escape by reaching the burrow.

In all of these ode45 will return all positions and the time for the event. Furthermore, if any of (3) and (4) occurs, the run of ode45 is halted. The function can be seen below.

```
function [events, terminal, dir] = CatchEvents(t, r, GATE, BURROW, BARRIER)
%CatchEvents Defines the noteworthy events in the run of ode45.
%   events(1) - fox reaches the gate
%   events(2) - fox stops/starts seeing the rabbit
%   events(3) - fox catches rabbit
%   events(4) - rabbit reaches the burrow
events(1) = r(4) - GATE(2);
events(2) = CanSeeRabbit(r(3:4), r(1:2), BARRIER);
events(3) = norm(r(1:2) - r(3:4)) - 0.1;
events(4) = r(1)-BURROW(1);
terminal = [0, 0, 1, 1];
dir = [1, 0, 0, 0];
end
```

3.2 CanSeeRabbit.m

This function (seen below) is used to detect whether the barrier blocks the view of the rabbit for the fox. It takes in positions of the fox, rabbit and the two endpoints of the barrier, then uses these to calculate two equations of lines (fox → rabbit and A → B). It then equals them and solves for x coordinate.

```
function [can_see_rabbit] = CanSeeRabbit(r_f, r_r, BARRIER)
%CheckIfSee Calculates whether the barrier blocks the view of the rabbit.
%   ARGUMENTS
%   r_f             : fox's position
%   r_r             : rabbit's position
%   BARRIER         : barrier's endpoints' positions
%
%   RETURNS
%   can_see_rabbit  : true if rabbit can be seen and false otherwise

if r_r(1) < BARRIER(1,1)
    can_see_rabbit = true;

elseif r_f(1) > BARRIER(1,1)
    can_see_rabbit = true;

else
    B_m = (BARRIER(2,2) - BARRIER (1,2)) / (BARRIER(2,1) - BARRIER(1,1));
    B_c = BARRIER(1,2) - B_m*BARRIER(1,1);
    FR_m = (r_r(2) - r_f(2))/(r_r(1) - r_f(1));
    FR_c = r_f(2) - FR_m*r_f(1);
    x = (FR_c - B_c)/(B_m - FR_m);

    if BARRIER(1,1) < x && x < BARRIER(2,1)
        can_see_rabbit = false;

    else
        can_see_rabbit = true;
    end
end
```

If $\mathbf{r}_A(1) < x < \mathbf{r}_B(1)$ is satisfied then the barrier blocks the view so the function returns false. Otherwise it returns true. Before doing this calculation we also checks whether $\mathbf{r}_r(1) < \mathbf{r}_A(1)$ or $\mathbf{r}_f(1) > \mathbf{r}_A(1)$, in which case it also returns true. This check was implemented to reduce the computing time in cases where the barrier is clearly not in the way.

3.3 VelocitySystem.m

This function is simply the combined system of equations for the motion of rabbit and fox together. It uses equations 1 through 6 as outlined in section 2. It is displayed below.

```
function drdt = VelocitySystem(t, r, s_f0, s_r0, mu_f, mu_r,...
                               GATE, BURROW, BARRIER)
%VelocitySystem A system of velocity ODEs for the scenario.
%   ARGUMENTS:
%   t           : time
%   r           : vector of rabbit's and fox'a positions and rabbit's
%                 angular position
%   s_f0        : initial speed of fox
%   s_r0        : initial speed of rabbit
%   mu_f        : decay coefficient for speed of fox
%   mu_r        : decay coefficient for speed of rabbit
%   GATE        : gate's position
%   BURROW      : burrow's position
%   BARRIER     : barrier's endpoints' positions
%
%   RETURNS
%   drdt(1:2)   : rabbit's velocity components
%   drdt(3:4)   : fox's velocity components
%   drdt(5)     : angular velocity of rabbit

A = BARRIER(1,:);
drdt = zeros(5,1);
speed = @(time, s_0, mu) 1 / (mu*time + (1/s_0));

s_r = speed(t, s_r0, mu_r);
s_f = speed(t, s_f0, mu_f);

drdt(5) = speed(t, s_r0, mu_r)/norm(BURROW);

drdt(1) = s_r * cos(r(5));
drdt(2) = -s_r * sin(r(5));

if r(4) < norm(GATE)
    drdt(3) = 0;
    drdt(4) = s_f;

elseif CanSeeRabbit(r(3:4), r(1:2), BARRIER)
    drdt(3) = s_f * (r(1)-r(3)) / norm(r(1:2) - r(3:4));
    drdt(4) = s_f * (r(2)-r(4)) / norm(r(1:2) - r(3:4));

else
    drdt(3) = s_f * (A(1)-r(3)) / norm(A(:) - r(3:4));
    drdt(4) = s_f * (A(2)-r(4)) / norm(A(:) - r(3:4));

end
```

3.4 main.m

This is the primary script which combines all above functions. The first 10 variables are parameters to be adjusted by the user to produce different scenarios or modify the precision of the ode45 solver. Later `VelocitySystem.m` is being inputted into ode45 with options as defined right above it. The resulting array is split into 3 more useful arrays: 1) path of the rabbit (`path_r`), 2) path of the fox (`path_f`) and 3) array of combined information about all events (`events`). A `for` loop is then used to identify every event and display the relevant information to the user. Finally, a plot of the

scenario is displayed to the user. It can be seen fully below but unfortunately does not fit a single page.

```
% where applicable, values are in SI units
s_f0 = 16;                              % fox's initial sped
r_f = [0; 0];                           % fox's initial position
mu_f = 0.0002;                          % decay coefficient of fox's speed
s_r0 = 13;                              % rabbit's initial speed
r_r = [0; 800];                         % rabbit's initial position
mu_r = 0.0008;                          % decay coefficient of rabbit's speed
dt = 0.001;                             % default time step
GATE = 300*[0, 1];                      % gate's position
BURROW = 800*[sin(pi/3), cos(pi/3)];    % burrow's position
BARRIER = [350 620;550 300];            % barrier's endpoints' positions


ang = 2*pi*[0.01:0.0001:0.99];
fence = norm(GATE)*[sin(ang); cos(ang)];
t_final_max = 800*(pi/3) / (s_r0 * exp(-mu_r*800*(pi/3)));
t_span = [0:dt:t_final_max+1];
r0 = [r_r; r_f; 0];

options = odeset('AbsTol',1e-7,'RelTol',1e-6, 'Events',...
                 @(t,r)CatchEvents(t, r, GATE, BURROW, BARRIER));

[¬, r, te,ye,ie] =...
    ode45(@(t,r)VelocitySystem(t, r, s_f0, s_r0, mu_f,...
                               mu_r, GATE, BURROW, BARRIER),...
          t_span, r0, options);

path_r = r(:,1:2);
path_f = r(:,3:4);

events = [ie te ye];
stopped_seeing_rabbit = false;
for i = 1:size(events, 1)
    event = events(i,:);
    if event(1) == 1
        fprintf("The fox reached the gate at time=%.2fs.\n",event(2))
        fprintf("Rabbit's position at the time was (%.2f, %.2f)m.\n",...
                event(3), event(4))

    elseif event(1) == 2 && ¬stopped_seeing_rabbit
        fprintf("The fox stopped seeing the rabbit at time=%.2fs.\n",event(2))
        fprintf("Rabbit's position at the time was (%.2f, %.2f)m.\n",...
                event(3), event(4))
        fprintf("Fox's position at the time was (%.2f, %.2f)m.\n",...
                event(5), event(6))
        stopped_seeing_rabbit = true;

    elseif event(1) == 2 && stopped_seeing_rabbit
        fprintf("The fox could see the rabbit again at time=%.2fs.\n",event(2))
        fprintf("Rabbit's position at the time was (%.2f, %.2f)m.\n",...
                event(3), event(4))
        fprintf("Fox's position at the time was (%.2f, %.2f)m.\n",...
                event(5), event(6))
        stopped_seeing_rabbit = false;

    elseif event(1) == 3
        fprintf("The fox caught the rabbit at time=%.2fs.\n",event(2))
        fprintf("Rabbit's position at the time was (%.2f, %.2f)m.\n",...
                event(3), event(4))
        fox_rabbit_d = norm(event(3:4)-event(5:6));
        rabbit_burrow_d = norm(event(3:4) - BURROW(:)');
        fprintf("Fox was %.2fm away from rabbit at (%.2f, %.2f)m.\n",...
                fox_rabbit_d, event(5), event(6))
        fprintf("Burrow was %.2fm away from rabbit.\n",...
                rabbit_burrow_d)
```

```
66
67      else
68          fprintf("The rabbit managed to escape at time=%.2fs.\n",event(2))
69          fox_rabbit_d = norm(event(3:4)-event(5:6));
70          fprintf("Fox was %.2fm away from rabbit at (%.2f, %.2f)m.\n",...
71                  fox_rabbit_d, event(5), event(6))
72
73      end
74      fprintf('\n')
75  end
76
77  plot(path_r(:,1), path_r(:,2), 'r',...
78       path_f(:,1), path_f(:,2), 'b',...
79       BARRIER(:,1), BARRIER(:,2), 'k-', BURROW(1), BURROW(2), 'rx',...
80       fence(1,:), fence(2,:), 'k--');
81  legend("Rabbit's path", "Fox's path", 'Barrier', 'Burrow', 'Fence',...
82         'Location', 'Southeast')
```

When $\mu_r = \mu_f = 0$ as in the first task, the rabbit manages to escape this script's output is:

```
The fox reached the gate at time=18.75s.
Rabbit's position at the time was (240.00, 763.15)m.

The fox stopped seeing the rabbit at time=34.92s.
Rabbit's position at the time was (429.99, 674.62)m.
Fox's position at the time was (166.43, 494.66)m.

The fox could see the rabbit again at time=48.81s.
Rabbit's position at the time was (570.08, 561.26)m.
Fox's position at the time was (349.99, 620.00)m.

The rabbit managed to escape at time=64.44s.
Fox was 161.65m away from rabbit at (569.50, 504.52)m.}
```

When $\mu_r = 0.0008\text{m}^{-1}$ and $\mu_f = 0.0002\text{m}^{-1}$ as in the second task, the fox catches the rabbit and the output is:

```
The fox reached the gate at time=19.32s.
Rabbit's position at the time was (225.80, 767.47)m.

The fox stopped seeing the rabbit at time=43.39s.
Rabbit's position at the time was (439.68, 668.34)m.
Fox's position at the time was (230.73, 555.71)m.

The fox could see the rabbit again at time=53.16s.
Rabbit's position at the time was (507.80, 618.17)m.
Fox's position at the time was (350.00, 619.98)m.

The fox caught the rabbit at time=78.93s.
Rabbit's position at the time was (644.35, 474.15)m.
Fox was 0.10m away from rabbit at (644.29, 474.23)m.
Burrow was 88.58m away from rabbit.
```

References

[1] J.R. Forshaw and A.G. Smith. *Dynamics and Relativity*. Wiley, The University of Manchester, 2009.

Appendix A - Plots for tasks 1 and 2

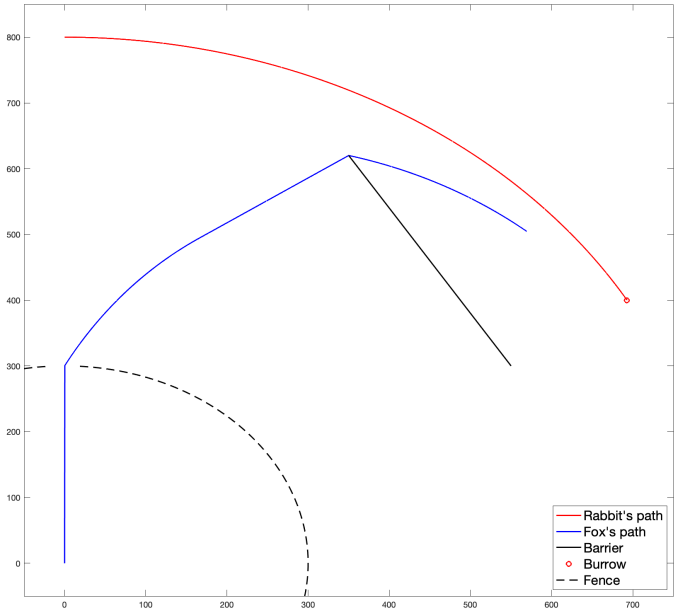

Figure 2: Plot of paths taken by rabbit and fox in task 1.

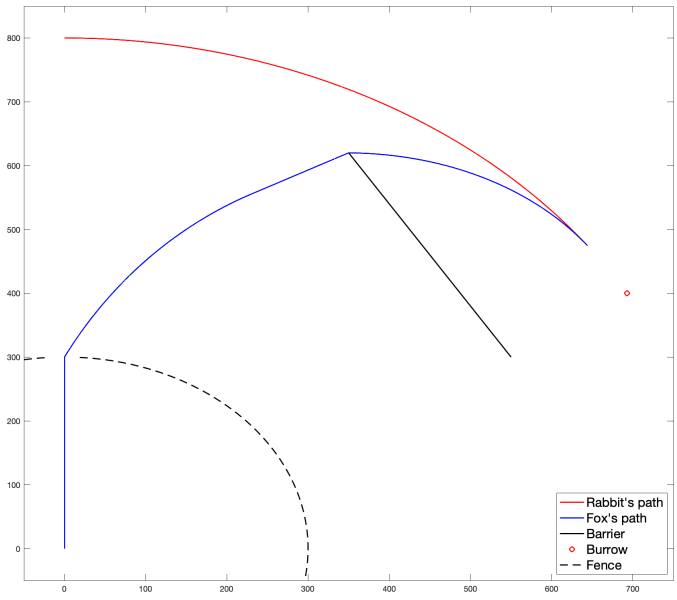

Figure 3: Plot of paths taken by rabbit and fox in task 2.