PSBC Project 2

March 27, 2019

1 Introduction

This report details the approach and implementation relating to the second coursework project of MATH36032. Please note that all programming and testing was carried out using GNU Octave rather than MATLAB.

2 The Problem

The goal of this project is to investigate the dynamics between a fox and a rabbit in a specific scene. The goal of the rabbit in the scene is to reach it is a burrow safely. Meanwhile, the goal of the fox is to catch the rabbit. We will aim to answer the question of which outcome occurs based on a set of input parameters.

To do this, we will use differential equations, solving them numerically using Octave's `ode45` solver.

3 The Scene

The scene we will be working with is as follows. Its main feature is a warehouse defined by its two westerly corners which we will denote as NW and SW for northwest and southwest respectively. The scene will contain the rabbit's burrow (denoted B). Figure **??** shows an example configuration. We will denote the position of the rabbit by R and the fox by F.

The rabbit moves in a straightforward way, directly towards the burrow with constant speed s_{r0}.

The fox moves under the following rules;

- If the rabbit is within the fox's line of sight, the fox runs directly towards the rabbit.
- If a corner of the warehouse obscures the rabbit, then the fox runs towards that corner.
- If the fox reaches a corner and the rabbit still is not in sight, it follows the warehouse perimeter until it sees the rabbit.

Before continuing, we must make a few assumptions as to what our model can support. They are as follows.

- The rabbit is within the fox's line of sight initially.
- The burrow is within the rabbit's line of sight initially.

4 The General Approach

The approach we will use is to model the scene using Octave's built-in `ode45` function, which is a numerical solver for ODEs. We will take advantage of the fact that it is a numerical solver, not a symbolic one, allowing us will allow us to change the direction of the fox within the ODE function. These changes are based on the fox's line of sight, and the other factors referenced in the rules above.

We can do this because of the iterative nature of the solver, meaning that changing the target will not affect the values computed for previous timeframes.

5 The Solution

The first two inputs to our function defining the ODE are time t, and the vector z holding the solution at t, this is defined by

$$z = \begin{bmatrix} R_x \\ R_y \\ F_x \\ F_y \end{bmatrix}. \tag{1}$$

Within our ODE function, we must calculate the derivative of the elements of z with respect to time.

$$\dot{z} = \begin{bmatrix} \dot{R}_x \\ \dot{R}_y \\ \dot{F}_x \\ \dot{F}_y \end{bmatrix}. \tag{2}$$

In other words, we will calculate the velocity of the fox and rabbit in the x and y directions at time t.

We will break the problem up into smaller pieces and solve these small pieces first before looking at the solution as a whole. In practice, this means examining small helper functions which will contribute to the final solution.

5.1 Determining Line of Sight

hello

5.2 Determining the Fox's Target

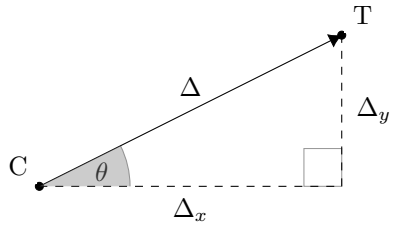
5.3 Velocities in the x and y directions

Now that we've seen in part **??** that the fox always runs towards some target. Furthermore, the rabbit perpetually runs towards its burrow.

Therefore, we can say that a creature C runs towards a target T with a speed of u. We seek it's velocity in the x direction (v_x) and in the y direction (v_y).

Figure 5.3 shows the scene. Using this we denote the angle between the creature and the target by θ and the distance between them by Δ. We can then easily deduce that

Figure 1: The scenario when a creature C runs towards a target T



$$v_x = s\cos\theta,$$
$$v_y = s\sin\theta.$$

Furthermore, we see there's no need to compute θ, we can observe that

$$\cos\theta = \frac{\Delta_x}{\Delta}$$
$$\sin\theta = \frac{\Delta_y}{\Delta}.$$

We can then combine these equations giving our final equations

$$v_x = s\frac{\Delta_x}{\Delta},$$
$$v_y = s\frac{\Delta_y}{\Delta}.$$

The advantage of using these equations rather than computing θ using trigonometry is that we can implement both components in one line using Octave's vector based arithmetic. This can be seen in our implementation in Listing **??**.

Listing 1: A function to compute the velocity of an entity facing a target with a given speed.

```
% For an entity directed at a target with a given speed
% return the x and y components of it's velocity
function velocity = computeVelocity(speed, pos, target)
  dist = distance(pos, target);
  velocity = speed .* ((target - pos) ./ dist);
end
```

5.4 Stopping the ODE solver

With this in place, we can now examine the ODE function itself, included in Listing: 1. We can see that we compute the rabbit's velocity directly, and compute the fox's velocity based on its trajectory.

Listing 2: The ODE function used to model the fox chasing the rabbit

```
function dzdt = modelODE(t, z, burrowPos, rabbitSpeed, ...
                         foxSpeed, warehouseNW, warehouseSW)
```

```
  Y = 2; % index of y in pos
  X = 1; % index of x in pos

  % z = [Rx, Ry, Fx, Fy]
  dzdt = zeros(size(z));

  % Converted into row vectors to match input parameters
  rabbitPos = z(1:2)';
  foxPos = z(3:4)';


  % rabbit velocity
  dzdt(1:2) = computeVelocity(rabbitSpeed, rabbitPos, burrowPos)';

  % fox velocity
  foxTarget = computeFoxTarget(rabbitPos, foxPos, warehouseNW, warehouseSW);
  dzdt(3:4) = computeVelocity(foxSpeed, foxPos, foxTarget)';

end
```

6 A more realistic scenario

References