School of Engineering and Applied Sciences

MATH-206

Numerical Method

Instructor: Dr. **Nasser Hassan Sweilam**

Theory and animation of pursuit problems

Ahmed Mohamed Fouad

Amr Khaled

Karim Mohamed Abouaish

Mohamed Ehab

Date

20/3/2020

ABSTRACT

Pursuit problems arise in mechanics when one particle (A), travelling on a given trajectory, is Pursued by another (B) in such a way that B’s velocity is always toward A. Examples include a missile homing in on a plane in flight, a robot ‘hand’ trying to pick up a moving object, or even a dog chasing a rabbit. The mathematical model of a pursuit problem is a system of differential equations that may, or may not, be amenable to analytic solution. This paper will introduce some path tracking techniques, convey the theory of pursuit problems and use MATLAB to arrive at numerical solutions to some example 2D and 3D problems as well as to produce animations.

TABLE OF CONTENTS

Contents

SECTION I: Introduction

Pursuit problems are when a dog chases a cat or when a rocket is sent to the moon. The pursuers or the aggressor in both cases are the dog and the rocket while the preys are the cat and the moon. The pursue problem consists of finding the path of the aggressor to catch the prey.

History:

This problem dates to zeno and da Vinci, with the first mathematical treatment by Pierre bouguer in his paper "lines of pursuit" (1732). However, when it comes for the term "pursuit curve" it was firstly identified by George Boole in his "Treatise on differential equations" in (1859). In the classical model, the speed of the prey is proportional to that of the pursuers. Also, the problem of the pursuit models to biology is important in ballistic and aviation. The mathematical model of a pursuit problem is a system of deferential equations that may, or may not, be amenable to analytic solution.

Report hypothesis

This project will convey the theory of pursuit problems and use MATLAB to arrive at numerical solutions to some example problems as well as to produce animations.

In this paper studied several generalizations of pursuit problem, with an emphasis on computer simulations:

- Discovery of relevant literature and theory. Coverage of pursuit theory in 2D and worked solutions to some example problems.
- Numerical solutions of the pursuit curves for 2D problems where A travels in a straight line.
- Extension of the above to the case where A travels in a non-straight line, perhaps even random, trajectory.

SECTION II: Methods

2.1 Derivation of the General Pursuit Curve:

Let’s assume that the rabbit’s position be given by the parametric function R(t) , and let the fox’s position be given by F(t) = x(t), y(t) . By using Property 2 in identifying the pursuit curve, the ratio of the rabbit’s and the fox’s speed is a constant, which we will call rabbit k = $\frac{fox'sspeed}{rabbit'sspeed}$ [1].

It can be more cleared by using property 1 and the accompanying diagram of the pursuit curve, at which the unit tangent vector to the fox’s curve is$\frac{F\prime}{||F\prime||} = \frac{R-F}{||R-F||}$. By the definition of k,$||F'|| = k||R'||$ .Substitute this into the previous equation to obtain the vector differential equation describing the general pursuit curve:

$$F' = k||R'||\frac{R-F}{||R-F||}$$

This system of nonlinear differential equations must be solved by numerical methods except for special cases. It is left to the reader as an exercise to use the chain rule to see that the parameterization of the rabbit’s path will not affect the shape of the fox’s path. Thus, we may parameterize the rabbit’s path using any convenient function, and it does not matter that the rabbit’s speed may not be constant [1].

2.2 Numerical methods for ordinary differential equations

The usage of the numerical methods is to calculate numerically the approximations to the answers of ordinary differential equations (ODEs). Numerical integration calculations are also considered from their usages. Analytical computation cannot solve all the differential equations but, numerical approximation to the solution can be suitable which can be done by using calculus techniques to get the series expansion of the solution. Some methods in numerical partial differential equations change the partial differential equation into an ODE, which then can be solved.

2.2.1 Euler Method

Euler method is considered in mathematics and computational science as a first-order numerical method to solve ODEs by a given initial value. It is also considered as an explicit method as the process is a direct computation of the dependent variables that can be produced in terms of known quantities. The relation between the local error and the square of the step size is proportional.

We can find an approximation of an adjacent point from any point on a curve by moving a short distance along a line tangent to the curve.

Starting with the differential equation (1), we replace the derivative y' by the finite difference approximation:

$$y'(t) \approx \frac{y(t+h) - y(t)}{h} \qquad (1)$$

which when re-arranged yields the following formula

$$y(t+h) \approx \; y(t) + hy'(t) \qquad (2)$$

and using (1) gives:

$$y(t+h) \approx \; y(t) + hf(t, y(t)) \qquad (3)$$

This formula is usually applied in the following way. We choose a step size h, and we construct the sequence t0, t1 = t0 + h, t2 = t0 + 2h, … We denote by yn a numerical estimate of the exact solution y(tn). Motivated by (3), we compute these estimates by the following recursive scheme

$$y_{n+1} = y_n + hf(t_n, y_n) \qquad (4)$$

The Euler method is an example of an explicit method. This means that the new value yn+1 is defined in terms of things that are already known.

2.2.2 Runge-kutta 4th order method

Runge-Kutta 4th order method is a numerical technique used to solve ordinary differential equation of the form

$$\frac{dy}{dx} = f(x, y), y(0) = y_0 \qquad (5)$$

So only first order ordinary differential equations can be solved by using the Runge-Kutta 4th order method. In other sections, we have discussed how Euler and Runge-Kutta methods are used to solve higher order ordinary differential equations or coupled (simultaneous) differential equations.

$$k_1 = f(x_i, y_i) \qquad (5.1)$$

$$k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right) \qquad (5.2)$$

$$k_3 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h\right) \qquad (5.3)$$

$$k_4 = f(x_i + h, y_i + k_3 h) \qquad (5.4)$$

$$y_{i+1} = y_i + \frac{h}{6}(k_1 + k_2 + 2k_3 + k_4) \qquad (6)$$

Higher order Runge-Kutta methods are available (like RK5, RK6,….).. Higher order methods are more accurate but require more calculations. RK4 is a good choice. It offers good accuracy with a reasonable calculation effort.

2.2.3 Truncation error

The truncation error describes the error made by trimming an infinite sum and approximating it to a limited sum. For example, approximating the first non-zero factors in the sine function of its Taylor series, the concluded error is the truncation error. Also, discretization error is included from the truncation error which is the error resulted taking a finite number of steps to calculate the approximation of an infinite process. For instance, (ODEs) when it’s solved by the numerical methods, the answer of the differential equation of a variable function which is approximated by a process that analyses step by step. Also, round-off error is called truncation error.

2.3 Why Runge-Kutta Method:

Usually error in Euler method is higher than higher order RK method (RK2, RK3, etc.), because truncation error in higher order methods is less compared to Euler method.

In some of the beginner level literature in numerical methods, it is loosely mentioned that higher order methods (say, RK4) give less error than lower order method (say, Euler method). Most of the time this is true, but not all the time. This property depends on the mesh and initial condition and differential equations you have considered.

If the exact solution to the differential equation is a polynomial of order nn, it will be solved exactly by an n-th Runge-Kutta method. For example, forward Euler will be exact if the solution is a line. RK4 will be exact if the solution is a polynomial of degree 4 or less.

Initial "absolute maximum difference error" in RK4 method is equal (or) higher than Euler method for coarse grid and reduces with refining grid for problems with shorter waves relative to grid. Because convergence rate of RK4 method is more than Euler. Please note that coarseness or fineness of grid is completely based on differential equation, initial condition and numerical scheme. Please refer following link for more details. Though that is based on differentiation, we can make a relative comparison between time numerical integration and differentiation as long as the "numerical integration" is stable

<u>Methodology</u>

The objective of this project was to create 2 projections of animal populations and 1 chasing pursuit curve for lion and gazelle based on a simple predator-prey model and explore the trends visible. e. Each case began with a different set of initial conditions that in turn dictated how the populations changed over time.

<u>Application of pursuit curve in fox and rabbit population problem:</u>

Predator-prey models are useful and often used in the environmental science field because they allow researchers to both observe the dynamics of animal populations and make predictions as to how they will develop over time.

Runge-Kutta 4th order method is a first order numerical procedure for approximating differential equations given an initial value. In this method the exact solution is unknown but the first order derivatives are known. In this case,

$$\frac{dR}{dt} = a_r R + b_r RF \qquad (1)$$

$$\frac{dF}{dt} = a_F RF + b_F F \qquad (2)$$

R and F are the number of rabbit and foxes, which are present at any given time t. The and values, are constants obtained from field data. Here we were given the values of these constants as listed below:

$$a_r = 0.2/year \qquad b_r = 0.001/year \quad a_F = 0.001/year \qquad b_F = 0.5/year \qquad (3)$$

We first explore if there are initial population values which lead to stable populations of both rabbits and foxes. This is determined by setting both equations (1) and (2) equal to zero and solving for the values of R and F, because a stable population will have a constant rate of change, i.e., and , which is equal to zero. This means.

$$a_r R + b_r RF = 0 \qquad \text{R=0 or F=200} \quad (4)$$

$$a_F RF + b_F F=0 \qquad \text{F=0 or R=500} \quad (5)$$

Note both equations are satisfied when either R=0 and F=0 OR R=500 and F=200. the first solution is trivial, we will focus on the second solution.

Application of pursuit curve in lion and gazelle (chasing):

Pursuit curves are used by fighter pilot to solve the geometry problems associated with combat situations. Any time you are chasing a target, you are applying pursuit curves. An understanding of when to use these things will improve your chances of arriving at the target in an offensive position

$$P(t)=\cos(t) \qquad \frac{dP}{dt} = -sint(t) \qquad (1)$$

$$q(t)=\sin(3t) \qquad \frac{dq}{dt} = 3\cos(3t) \qquad (2)$$

$$\frac{dx}{dt} = 0.7\sqrt{\left(\frac{dp}{dt}\right)^2 + \left(\frac{dq}{dt}\right)^2}\,\frac{(p-x)}{\sqrt{(p-x)^2+(q-y)^2}} \qquad (3)$$

$$\frac{dy}{dt} = 0.7\sqrt{\left(\frac{dp}{dt}\right)^2 + \left(\frac{dq}{dt}\right)^2}\,\frac{(q-x)}{\sqrt{(p-x)^2+(q-y)^2}} \qquad (4)$$

SECTION III: Results

Application of pursuit curve in fox and rabbit population problem:

.

In case 1, we started with a rabbit population of 1000 and a fox population of 100. Figure 1 shows that this model also provides a periodic population density where a decline in prey(rabbit) correlates with increase in predator (fox). Figure 1 shows an initially high population of rabbits that declines as the fox population increases. This trend seems plausible since the population of foxes in the first order differential is dependent on the population of its food It is also seen that the rabbit population depends on that of the fox and that is Reasonable.

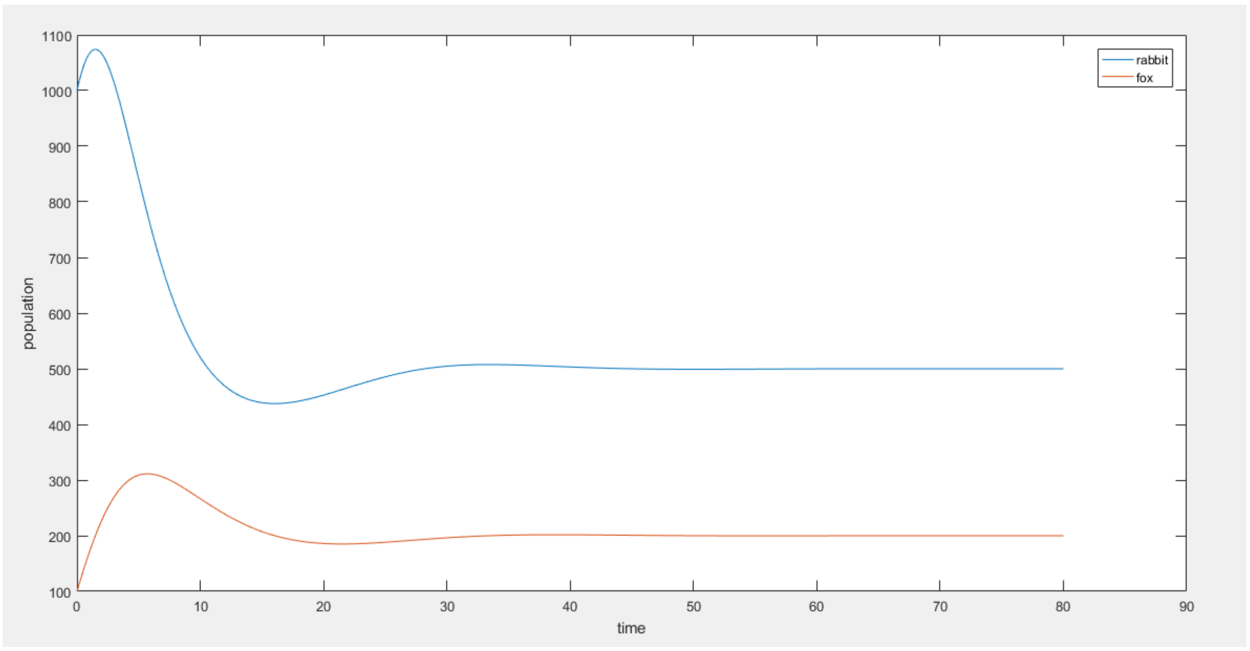


Figure 1: Rabbit and fox populations over an 80 year span starting with 1000 rabbits and 100 foxes.

In case 2, we used the values to calculate the part in which to prove that a stable population is in fact possible with those values provide fully stable populations that are straight lights at those particular values ($R = 500$ and $F = 200$). Figure 2 shows that there are theoretically no changes in the population of either the rabbits or foxes over the 80 year time span.

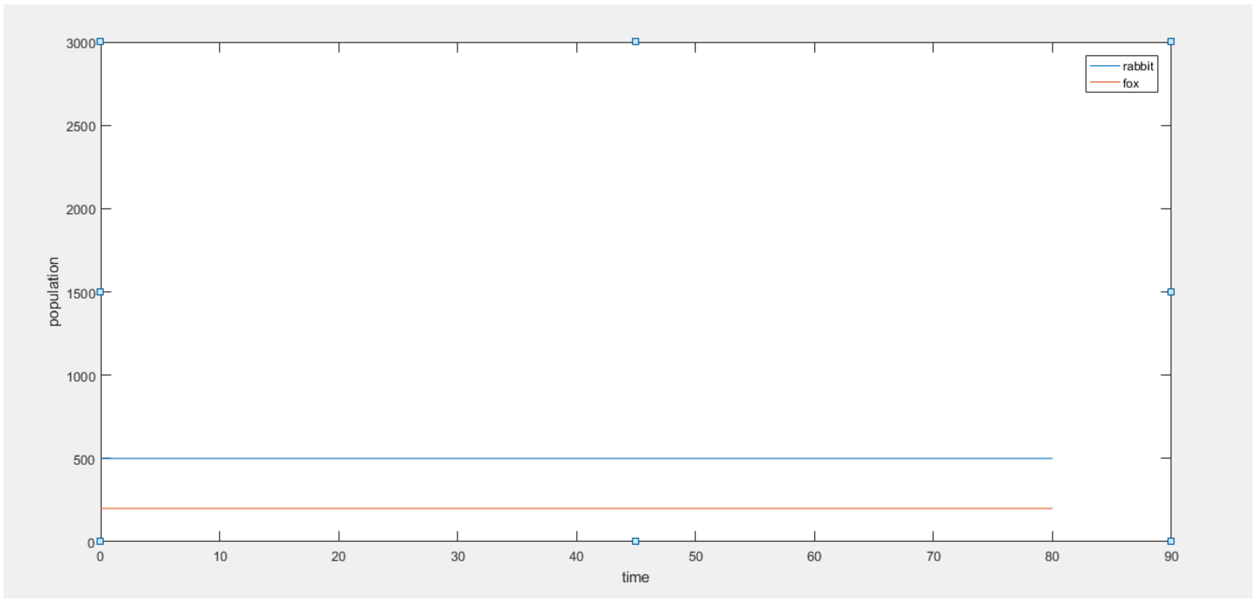


Figure 2: Rabbit and fox populations over an 80 year span starting with 500 rabbits and 200 foxes.

In case 3, A lion chasing a gazelle uses pursuit curves to determine how it can place itself to pounce or attack on the target when it catches up. You need to understand their path to determine how to pursue them.

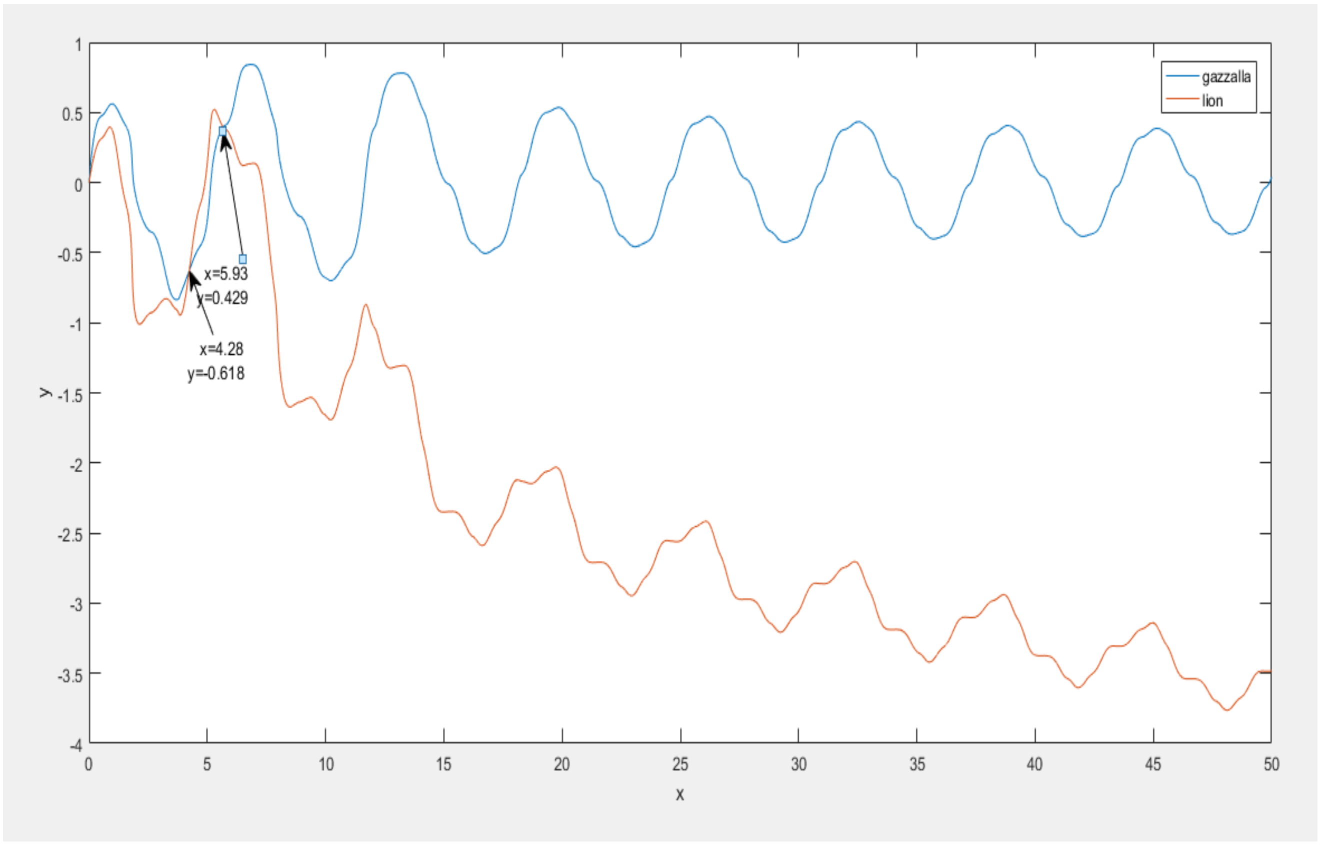
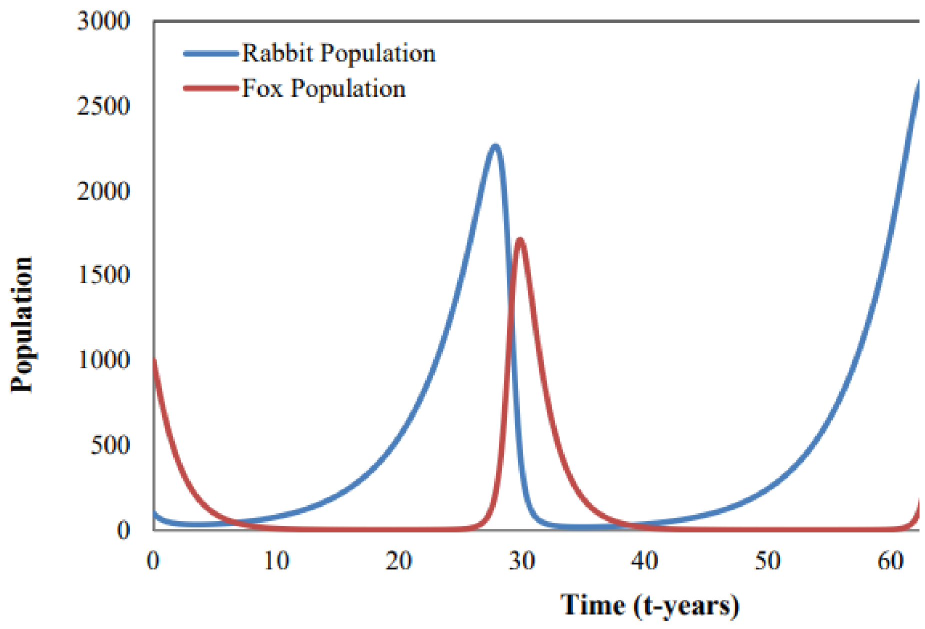


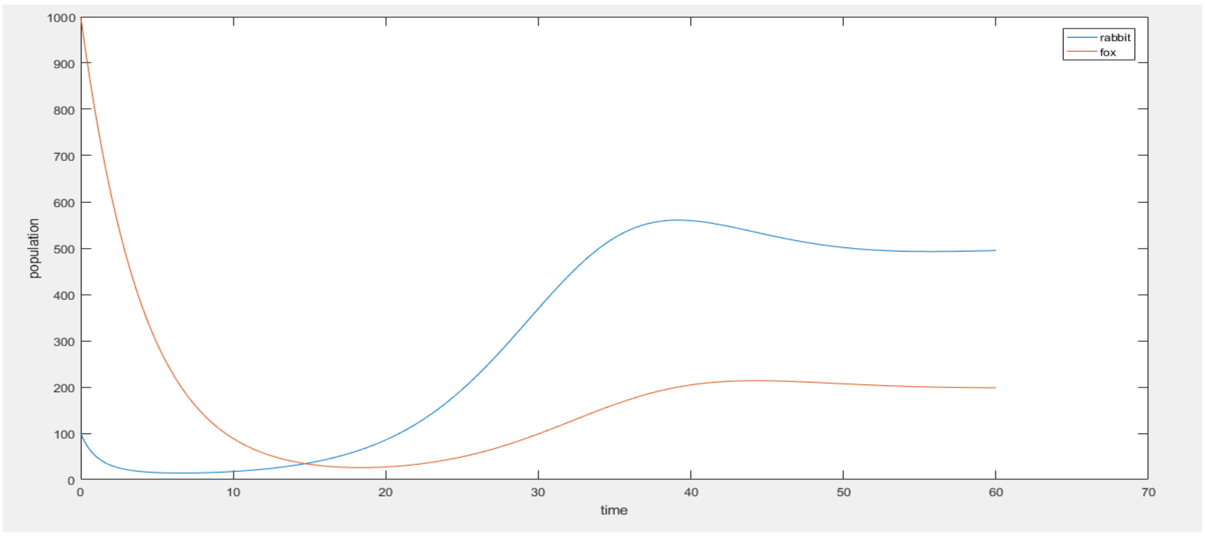
Figure 3: Lion catches a gazelle in two points 4.28 and 5.93

SECTION IV: Discussion

We compare our result with paper result of animal population and we found that graphs are most like each other. But there is a small difference which refers to the difference in methods. In the paper they use Euler methods but we use Runge kutta fourth order (RK4).Rk4 is more accurate than Euler.

In case 1: this is the difference between the two graphs.





In case 2: this is the difference between the two graphs.

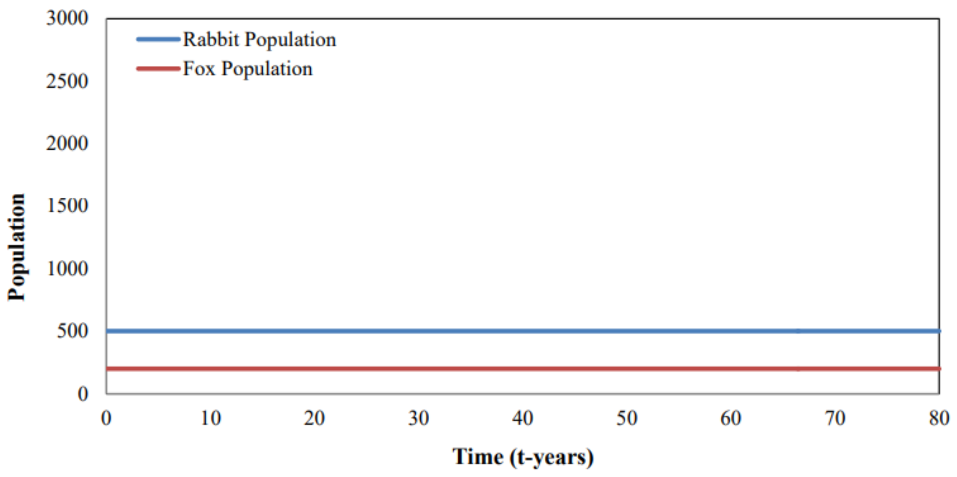
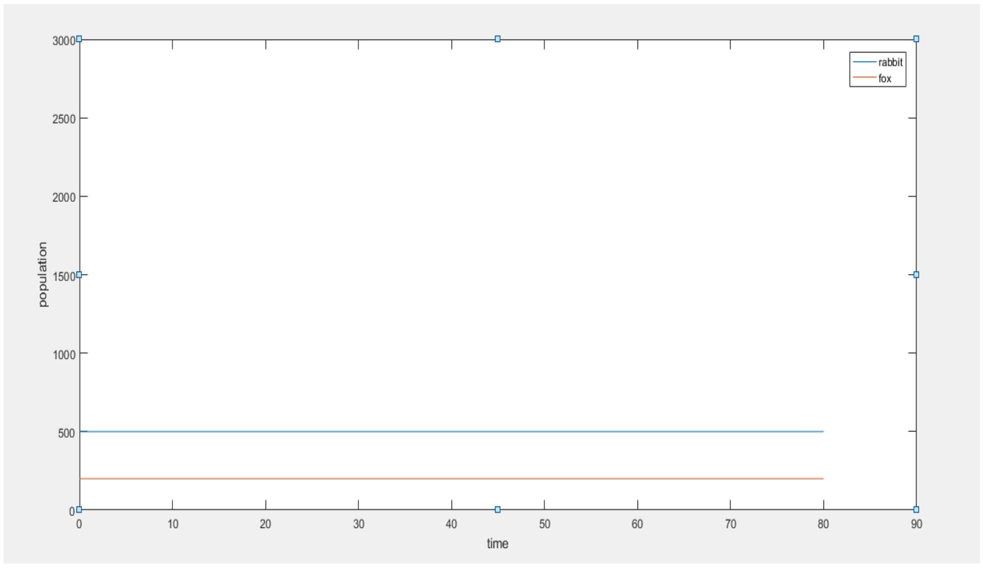


Figure 5: Rabbit and fox populations over an 80 year span starting with 500 rabbits and 200 foxes.



SECTION V: References

[1]M. Lloyd, *Pursuit Curves*. 2006.

[2]K. Steiner and J. Franchi, *Pursuit Curves*. 2011.

[3]M. Samuel, M. Hussein and M. Binti Mohamad, *A Review of some Pure-Pursuit based Path Tracking Techniques for Control of Autonomous Vehicle*. 2016.

[4]P. Zhao, J. Chen, Y. Song and X. Tao, *Design of a Control System for an Autonomous Vehicle Based on Adaptive-PID*. 2012.

[5]J. Morales, J. L. Martínez, M. A. Martínez and A. Martínez, *Pure-Pursuit Reactive Path Tracking for Nonholonomic Mobile Robots with a 2D Laser Scanne*. 2009.

[6]R. Craig Conlter, *Implementation of the Pure Pursuit Path 'hcking Algorithm*. 1992.

SECTION VI: Appendx

Matlab code

```
ar=0.2;
af=0.001;
br=0.001;
bf=0.5;
fr=@(t,R,F)   ar*R-br*R*F;
ff=@(t,R,F)   af*R*F-bf*F;
R(1)=100;
F(1)=1000;
t(1)=0;
h=0.0001;
tfinal=60;
n=ceil(tfinal/h);
for i=1:n
     t(i+1)=t(i)+h;
     %runge kutta method
     kR1=fr(t(i),    R(i),         F(i));
     KF1=ff(t(i),    R(i),         F(i));
     kR2=fr(t(i)+h/2,R(i)+kR1*h/2,F(i)+KF1*h/2);
     kF2=ff(t(i)+h/2,R(i)+kR1*h/2,F(i)+KF1*h/2);
     kR3=fr(t(i)+h/2,R(i)+kR2*h/2,F(i)+kF2*h/2);
     kF3=fr(t(i)+h/2,R(i)+kR2*h/2,F(i)+kF2*h/2);
     kR4=fr(t(i)+h,  R(i)+kR3*h,  F(i)+kF3*h);
     kF4=fr(t(i)+h/2,R(i)+kR3*h,  F(i)+kF3*h);
     R(i+1)=R(i)+(h/6)*(kR1+2*kR2+2*kR3+kR4);
     F(i+1)=F(i)+(h/6)*(KF1+2*kF2+2*kF3+kF4);

end
plot(t,R)
hold on
plot(t,F)
xlabel('time')
ylabel('population')
legend('rabbit','fox')
set(gca,'font size',16)
```

second example

Matlab code

```
fr=@(t,R,F)  0.7*sqrt((sin(t).^2)+((3*cos(3*t)).^2))*(cos(t)-R)/sqrt((cos(t-R).^2)+(sin(3*t)-F).^2) ;
ff=@(t,R,F)
0.7*sqrt((sin(t).^2)+((3*cos(3*t)).^2))*(sin(3*t)-R)/sqrt((cos(t-R).^2)+(sin(3*t)-F).^2)  ;
R(1)=0;
F(1)=0;
t(1)=0;
h=0.0001;
tfinal=20;
n=ceil(tfinal/h);
for i=1:n
     t(i+1)=t(i)+h;
     %runge kutta method
     kR1=fr(t(i),    R(i),         F(i));
     KF1=ff(t(i),    R(i),         F(i));
     kR2=fr(t(i)+h/2,R(i)+kR1*h/2,F(i)+KF1*h/2);
     kF2=ff(t(i)+h/2,R(i)+kR1*h/2,F(i)+KF1*h/2);
     kR3=fr(t(i)+h/2,R(i)+kR2*h/2,F(i)+kF2*h/2);
     kF3=fr(t(i)+h/2,R(i)+kR2*h/2,F(i)+kF2*h/2);
     kR4=fr(t(i)+h,  R(i)+kR3*h,  F(i)+kF3*h);
     kF4=fr(t(i)+h/2,R(i)+kR3*h,  F(i)+kF3*h);
     R(i+1)=R(i)+(h/6)*(kR1+2*kR2+2*kR3+kR4);
     F(i+1)=F(i)+(h/6)*(KF1+2*kF2+2*kF3+kF4);

end
plot(t,R)
hold on
plot(t,F)
xlabel('x')
ylabel('y')
legend('gazzalla','lion')
```