University of Exeter

Faculty of Environment, Science and Economy

MTH3039 Computational Nonlinear Dynamics

Coursework 2 : Chaos in a controlled population model

Academic Year 2023 / 2024

Student Name: Kofi Barton-Byfield
Student ID: 710032265

Contents

1 Question 1: Brute force bifurcation diagrams and chaotic attractors

1.1 Part A : Brute Force Bifurcation Diagram

For the first part of this report, I will conduct a brute force bifurcation analysis for a system of equations describing the dynamics of pest and predator species. The analysis involves generating a set of equispaced points over the domain [0,10] × [0,10] and utilising them as initial conditions to plot a brute force diagram.
Then I computed trajectories using the recurrence relation defined by the system of equations for varying values of the parameter T in the interval [5, Tmax]. The goal is to observe the bifurcation behavior of the system by plotting the end points of these trajectories against T.
Finally plotting the end points of these trajectories against T after sufficiently many iterations. Below are the plots for Pest species shown by Figure 1 and by Figure 2 for the Predator species respectively.

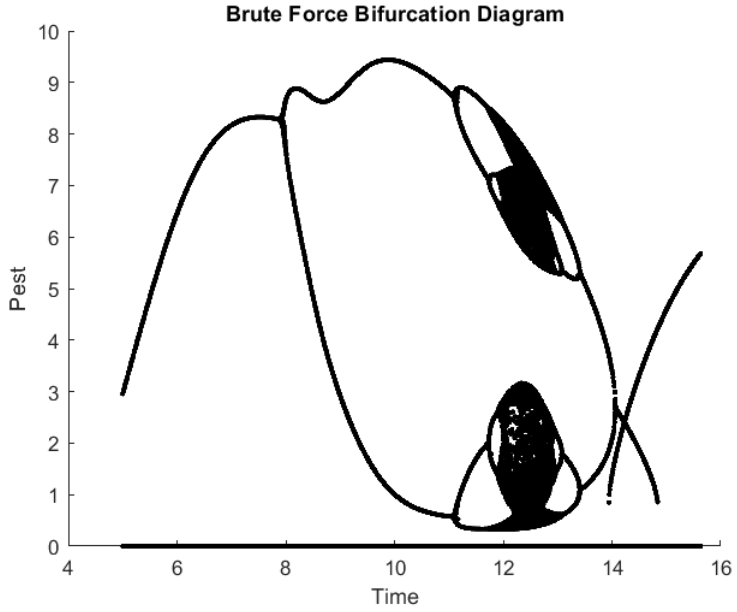

Figure 1: Brute Force Bifurcation Diagram for Pest Species

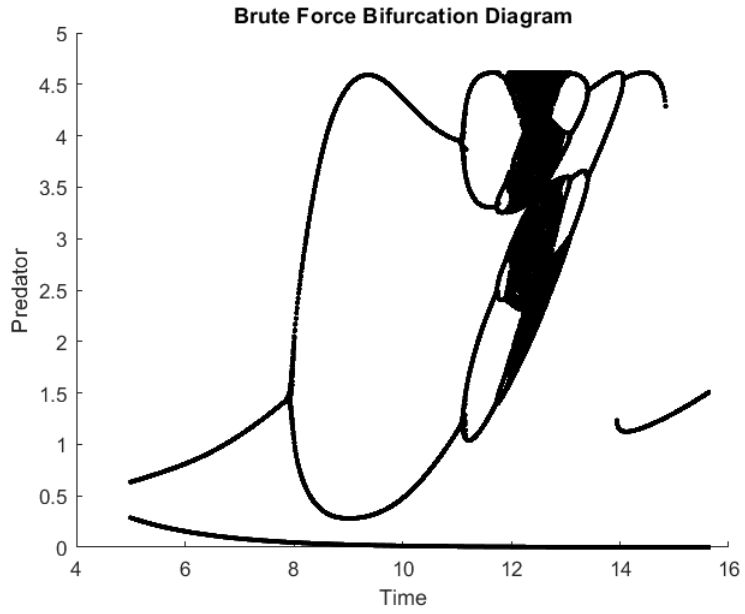

Figure 2: Brute Force Bifurcation Diagram for Predator Species

1.2 Part B : LyapunovQR

I proceeded with the report, by implementing the general-purpose function LyapunovQR, as outlined in the LyapunovQR.pdf instruction sheet. This can be found in the appendix with the rest of my code.

1.3 Part C : Computation of the Lyapunov Exponents

Then for each value of T used in the previous part, I utilised my function LyapunovQR to compute the Lyapunov exponents along one of my trajectories, which were chose arbitrarily. The following plot in Figure 3 shows the final value of both Lyapunov exponents against T for all values of $T \in [5, Tmax]$.

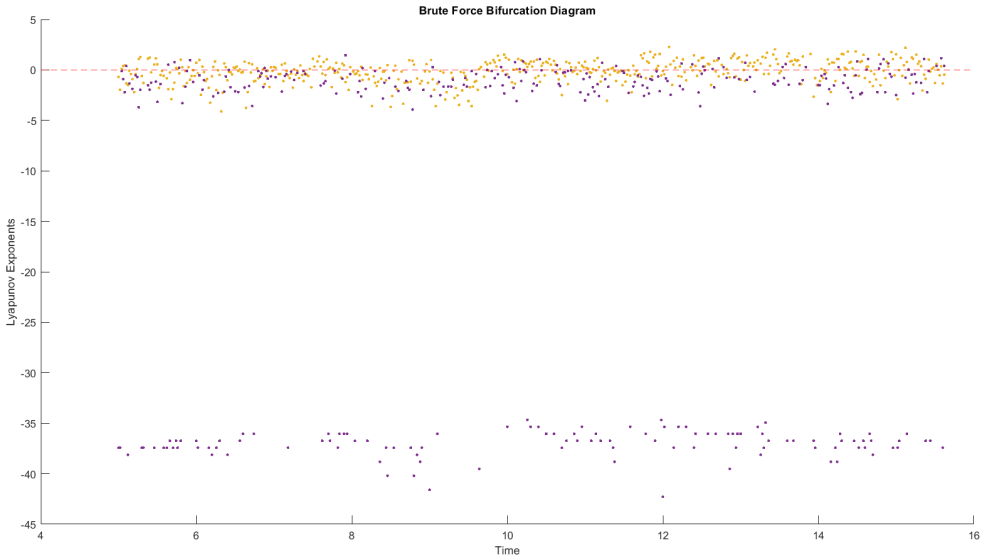

Figure 3: Plot of the final value of both Lyapunov exponents against T

1.4 Part D : Computation of the Lyapunov Exponents during Different Orbits

Now using the previous parts of this question, I can now identify intervals of T, to display the solution types below:
For each solution type (i)-(iv), I used my function LyapunovQR to evaluate the Lyapunov exponents at each step k = 1, . . . , K of one of the trajectory's, where K is sufficiently large that the Lyapnuov exponents (visually) converge. Producing a two-panel plots a shown below:

1.4.1 (i) Stable period 1 orbits

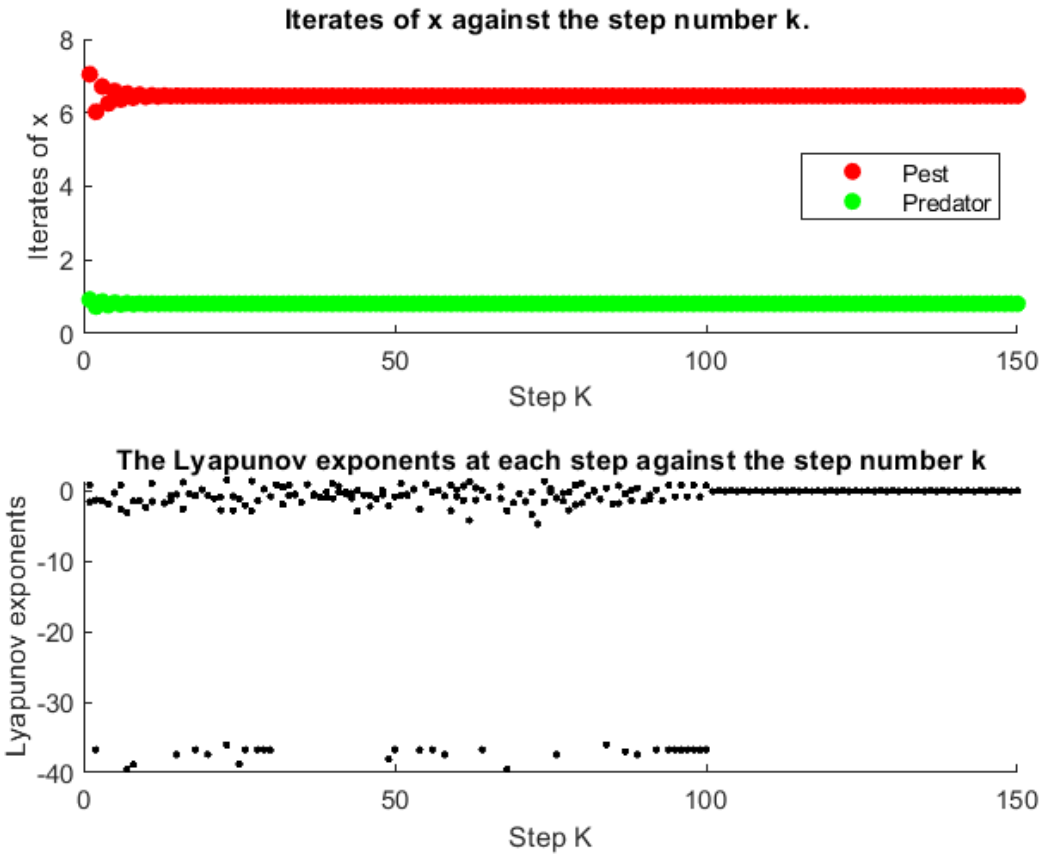

Figure 4: Plot of the Stable period 1 orbits

1.4.2 (ii) Stable period 2 orbits

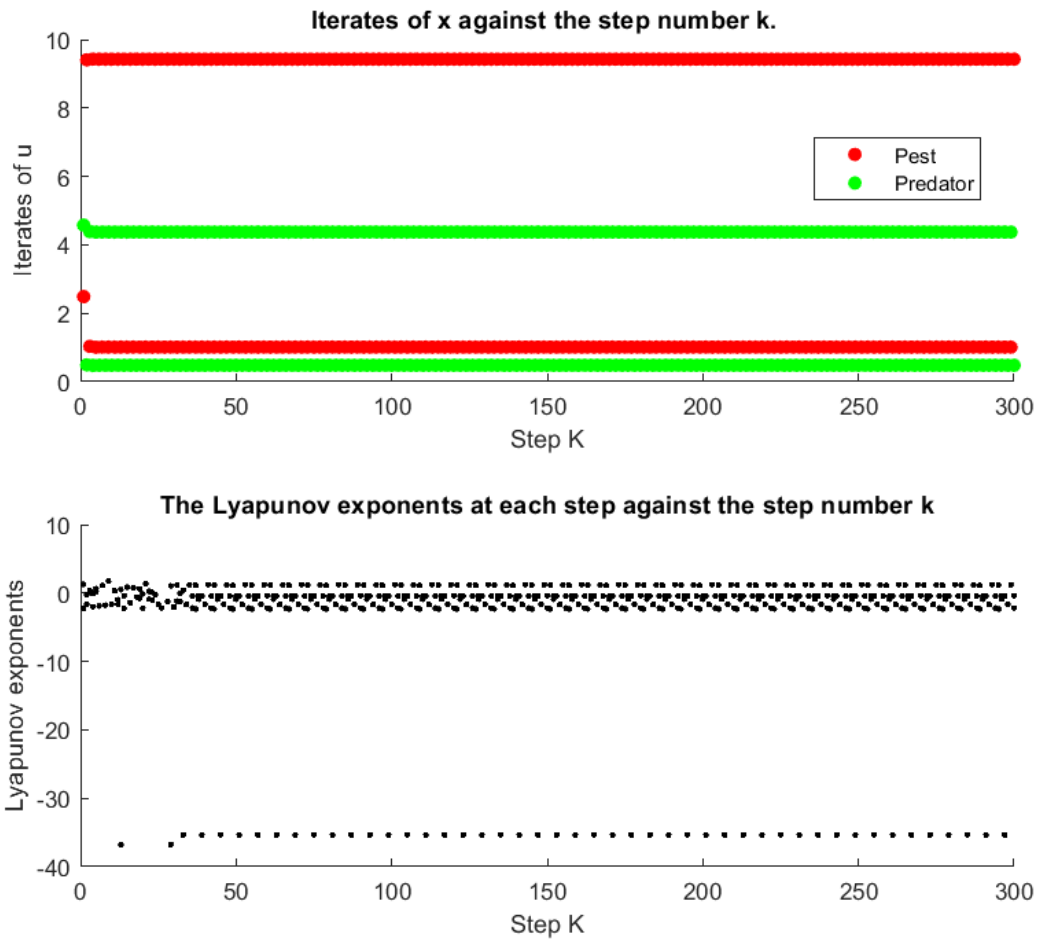

Figure 5: Plot of the Stable period 2 orbits

1.4.3 (iii) Stable period 4 orbits

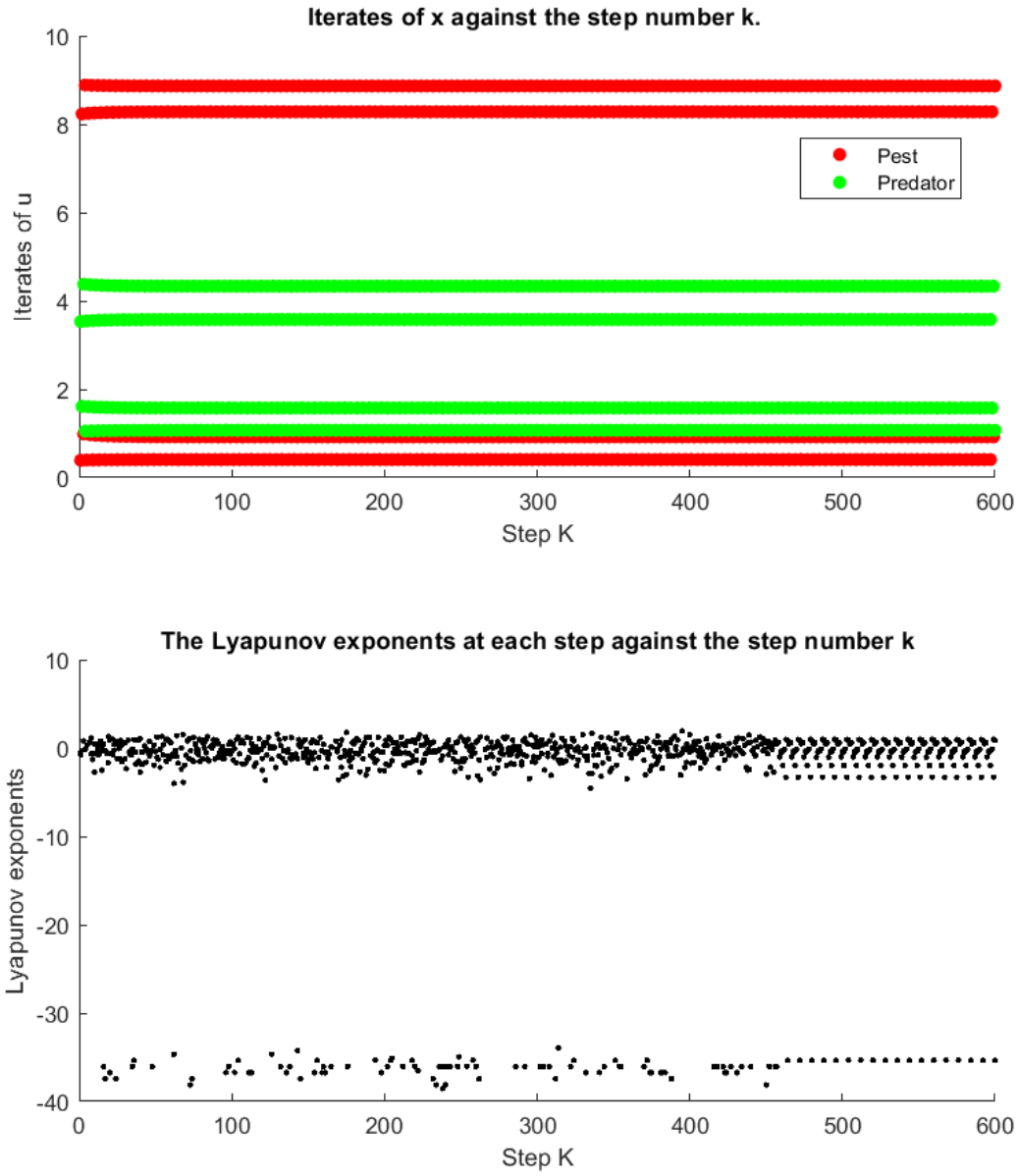

Figure 6: Plot of the Stable period 4 orbits

1.4.4 (iv) Chaotic orbits

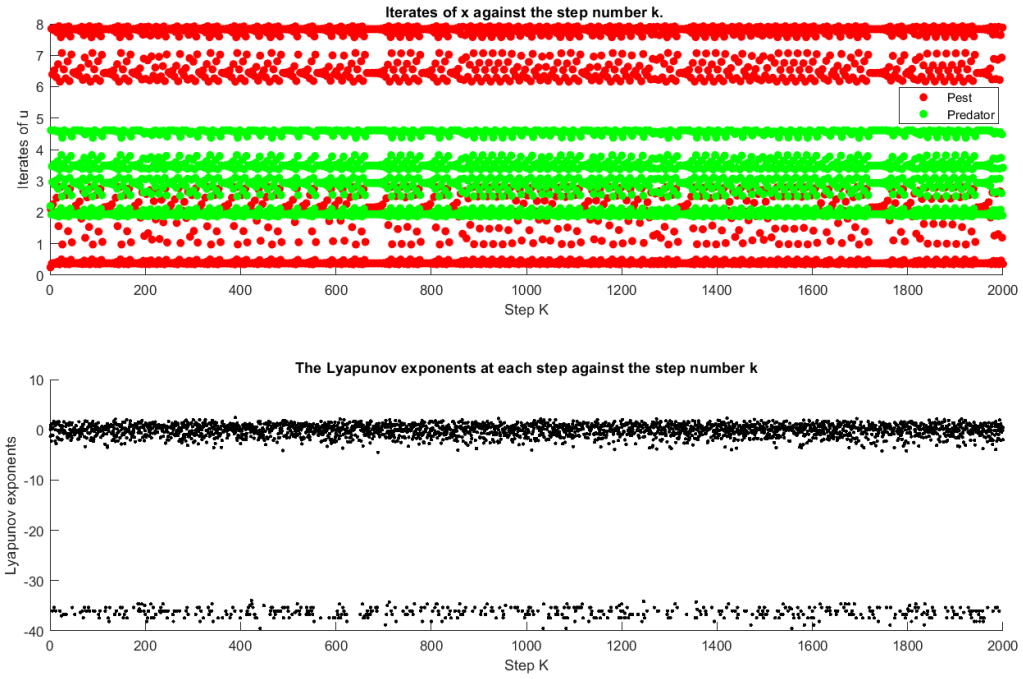

Figure 7: Plot of Chaotic orbits

2 Question 2: Bifurcation diagram

2.1 Part A : The Branch of Period-1 Solutions

For this section, I coded a function, 'ppn(u)' shown in the appendix, to represent a period-n orbit of the map M (4) as a regular root. Using the brute force bifurcation diagram, I obtained an initial guess for a period-1 orbit at a fixed T. Employing the 'MyTrackCurve' function, I traced a branch of period-1 solutions across $T \in [5, Tmax]$. These period-1 orbits were then integrated into the existing brute force bifurcation diagrams for the pest and predator species, with stability annotated using colors like red for sources, green for sinks, and orange for saddles. As demonstrated in the following figures:

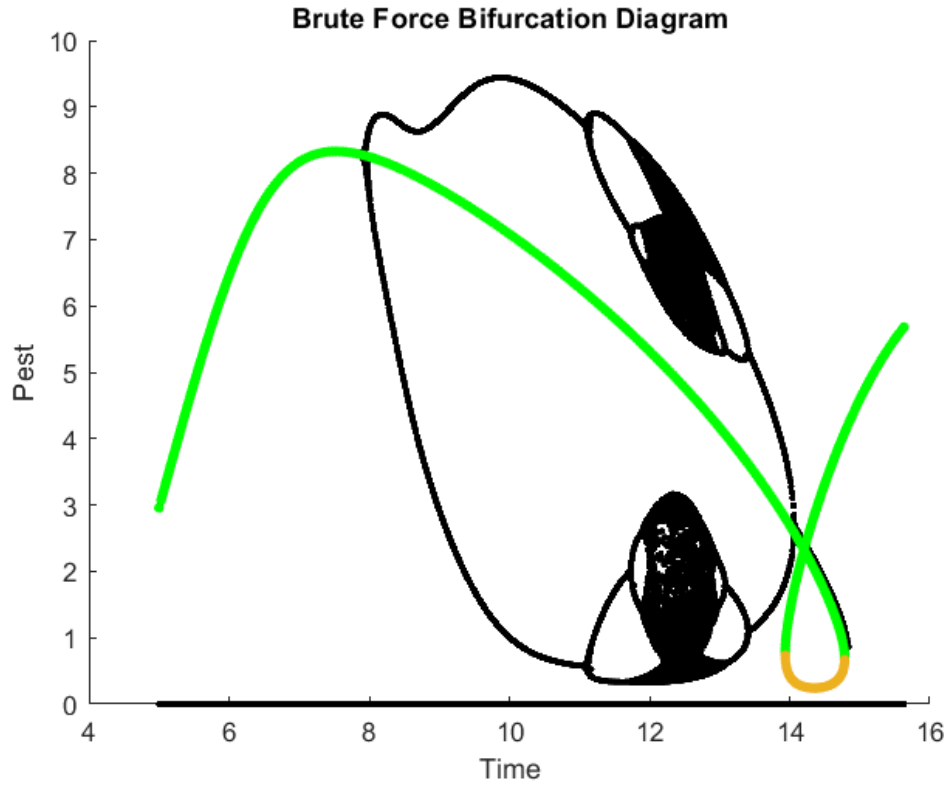

Figure 8: Plot of the branch of period-1 solutions for the pest species

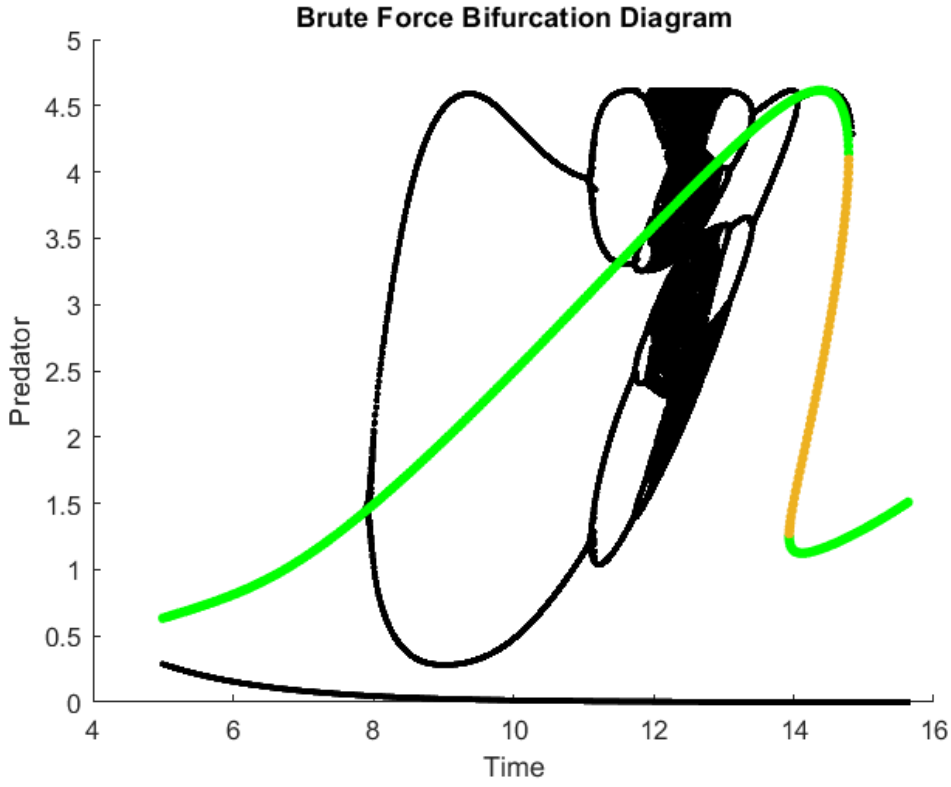

Figure 9: Plot of the branch of period-1 solutions for the pest species

2.2 Part B : The Periodic Doubling Bifurcation

Next, I implemented a function, ‘ppd(u)‘, designed to exhibit a general periodic doubling bifurcation, where period-2n+1 orbits branch from a period-2n orbit. This function is coded to represent such bifurcations as regular roots, and can be viewed in the appendix of this report.

2.3 Part C : The Fold of Period-1 Orbits

In addition, I developed a function, ‘pfold(u)‘, with the purpose of representing the point corresponding to a fold of period-1 orbits as a regular root. This function can also be viewed in the appendix of the report.

2.4 Part D : The completed Bifurcation Diagram

For this section, I utilised the 'MySolve', 'ppn', 'ppd', 'pfold', and 'MyTrackCurve' functions to compute key bifurcation points and branches of solutions. The computed values of (x, y) and T at these bifurcation points, rounded to four decimal places, were printed to the Matlab Command Window. Additionally, markers indicating the location of these bifurcation points were added to the bifurcation diagram. To ensure clarity, all points along the branches of orbits, such as period-4 orbits, were plotted, with stability annotations following the convention established in part (a) of this question. This includes period doubling bifurcations along branches of period-1 and period-2 orbits, the branch of period-2 orbits branching off period-1 orbits, and the fold of period-1 orbit bifurcations.

2.4.1 (i) The period doubling bifurcations along the branch of period-1 solutions

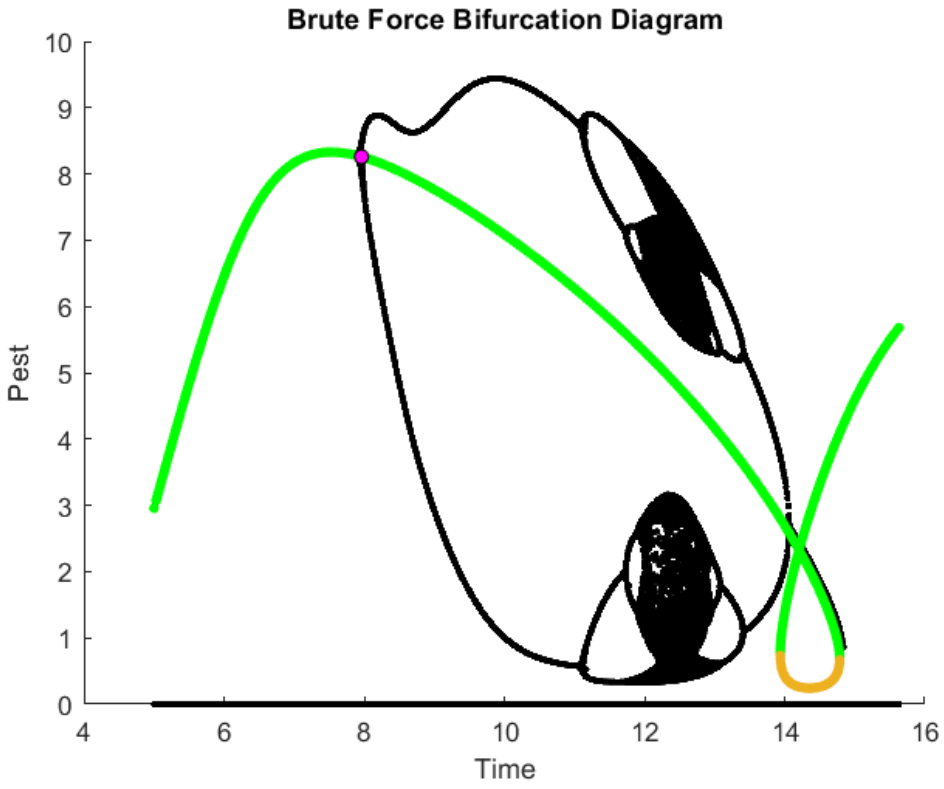

Figure 10: Plot of the period doubling bifurcations along the branch of period-1 solutions for the pest species

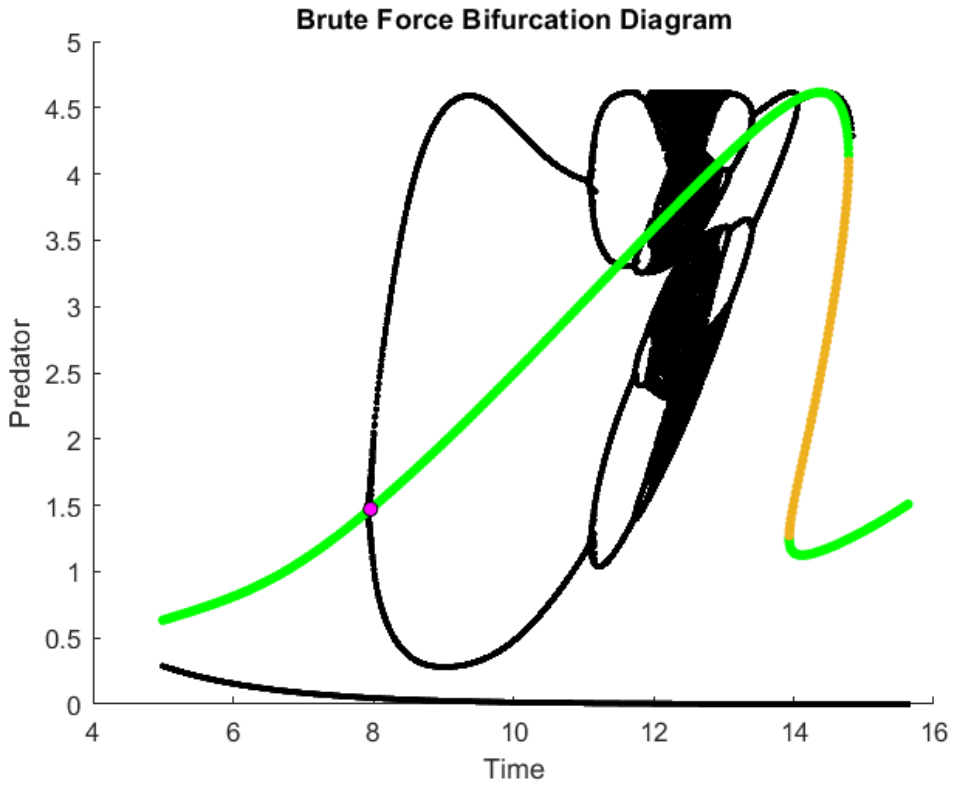

Figure 11: Plot of the period doubling bifurcations along the branch of period-1 solutions for the predator species

2.4.2 (ii) The branch of period-2 orbits that branches off the period-1 orbits at the above point.

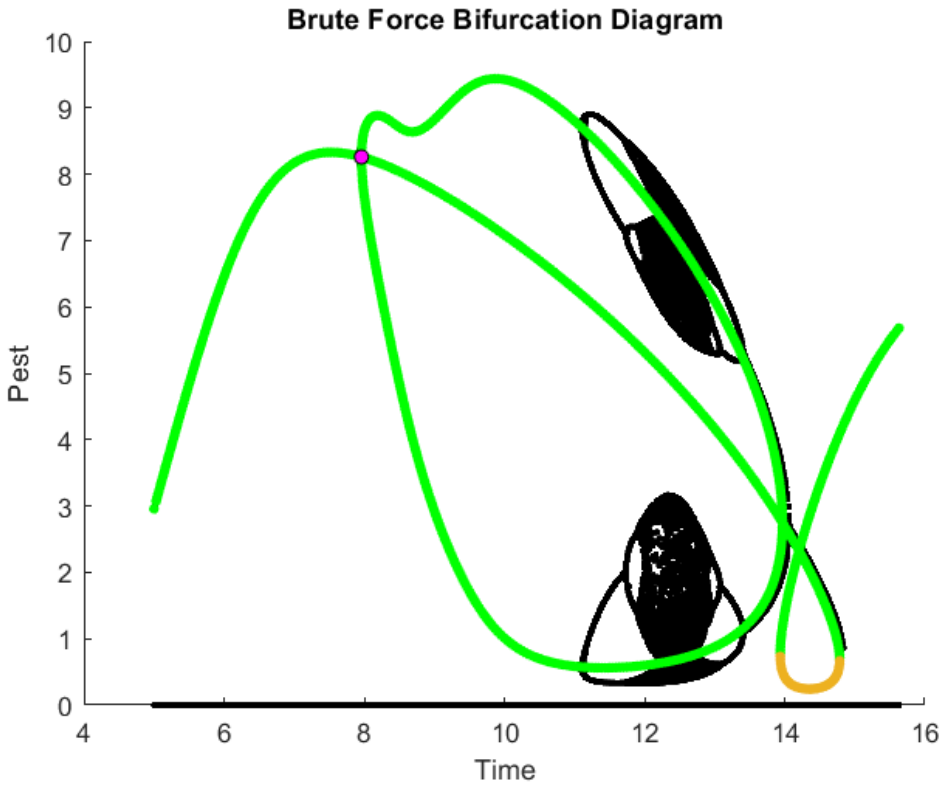

Figure 12: Plot of the branch of period-2 orbits that branches off the period-1 orbits at the above point for the Pest species

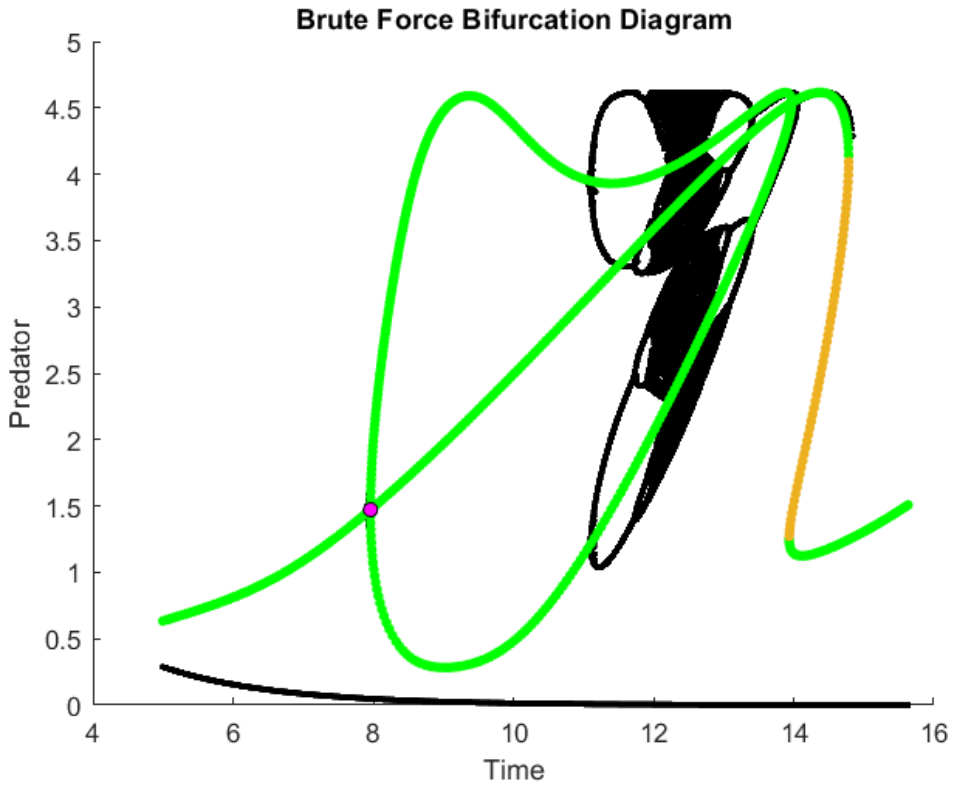

Figure 13: Plot of the branch of period-2 orbits that branches off the period-1 orbits at the above point for the Predator species

2.4.3 (iii) The period doubling bifurcations along the branch of period-2 orbits.

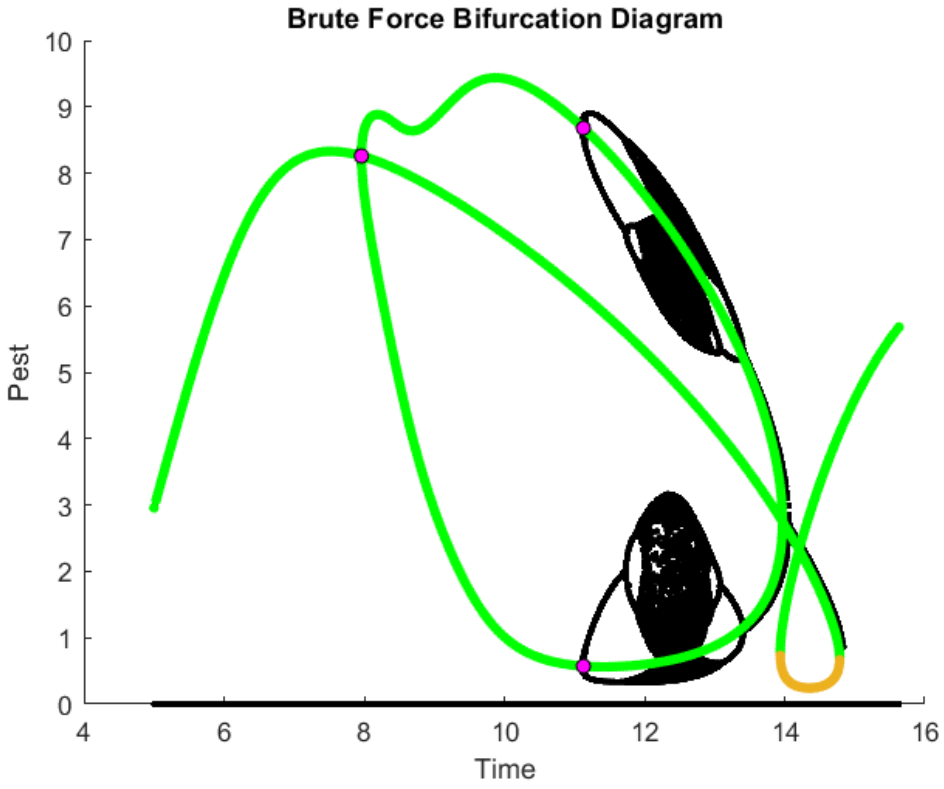

Figure 14: Plot of the period doubling bifurcations along the branch of period-2 orbits for the Pest species

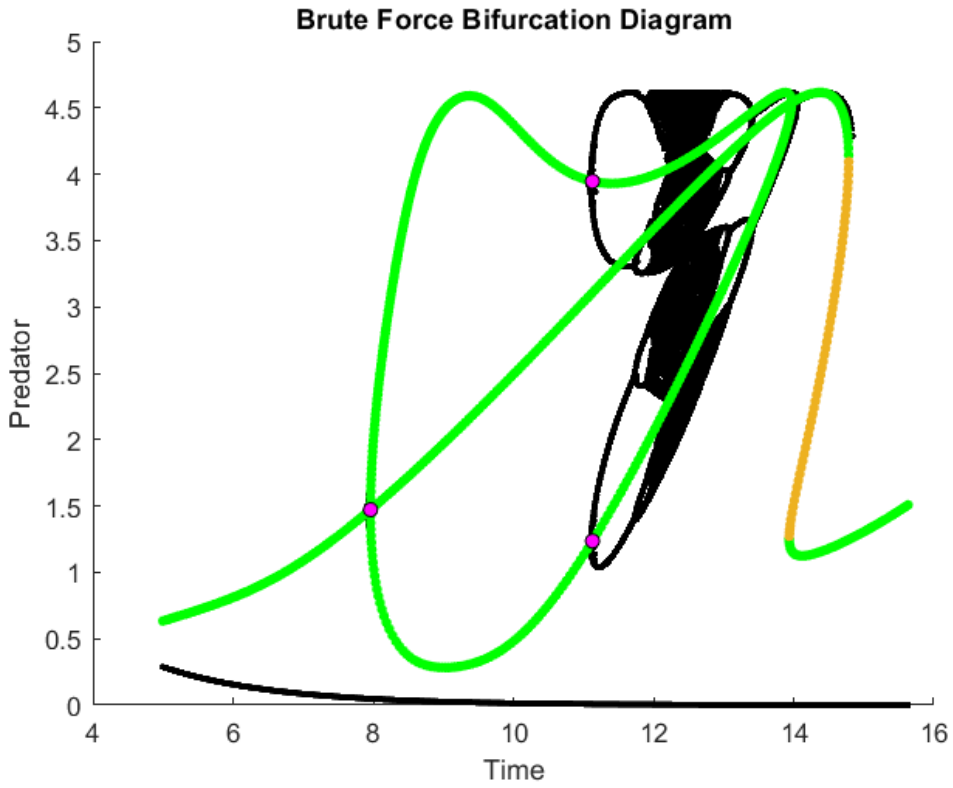

Figure 15: Plot of the period doubling bifurcations along the branch of period-2 orbits for the Predator species

2.4.4 (iv) The branch of period-4 orbits that branches off the period-2 orbits at the above point.

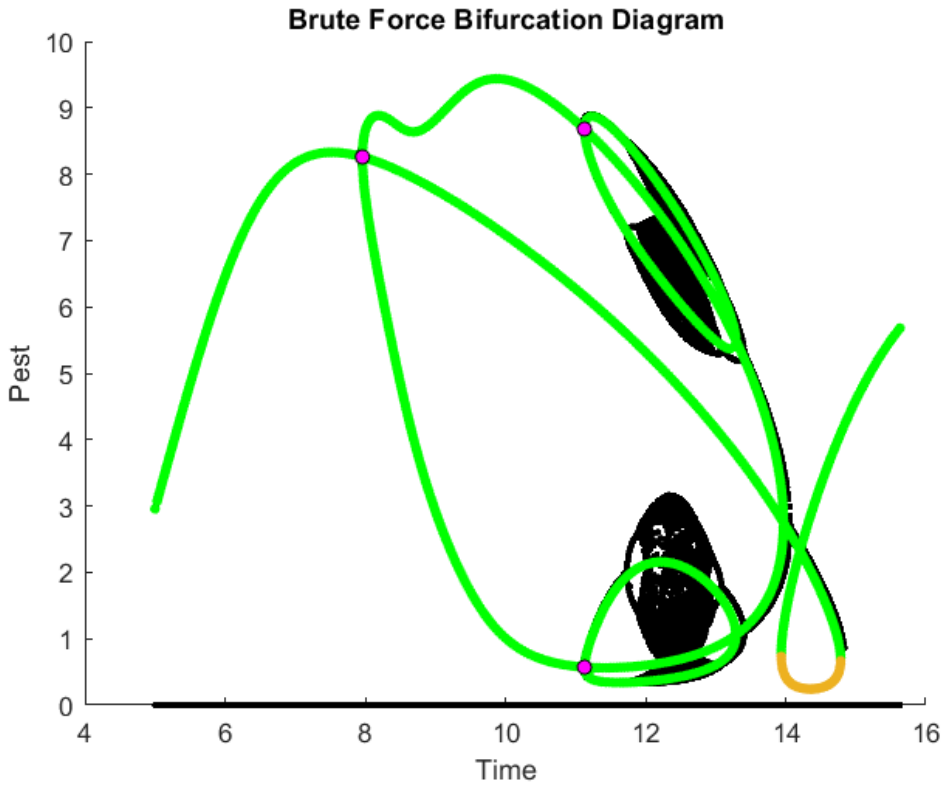

Figure 16: Plot of the branch of period-4 orbits that branches off the period-2 orbits at the above point for the Pest species

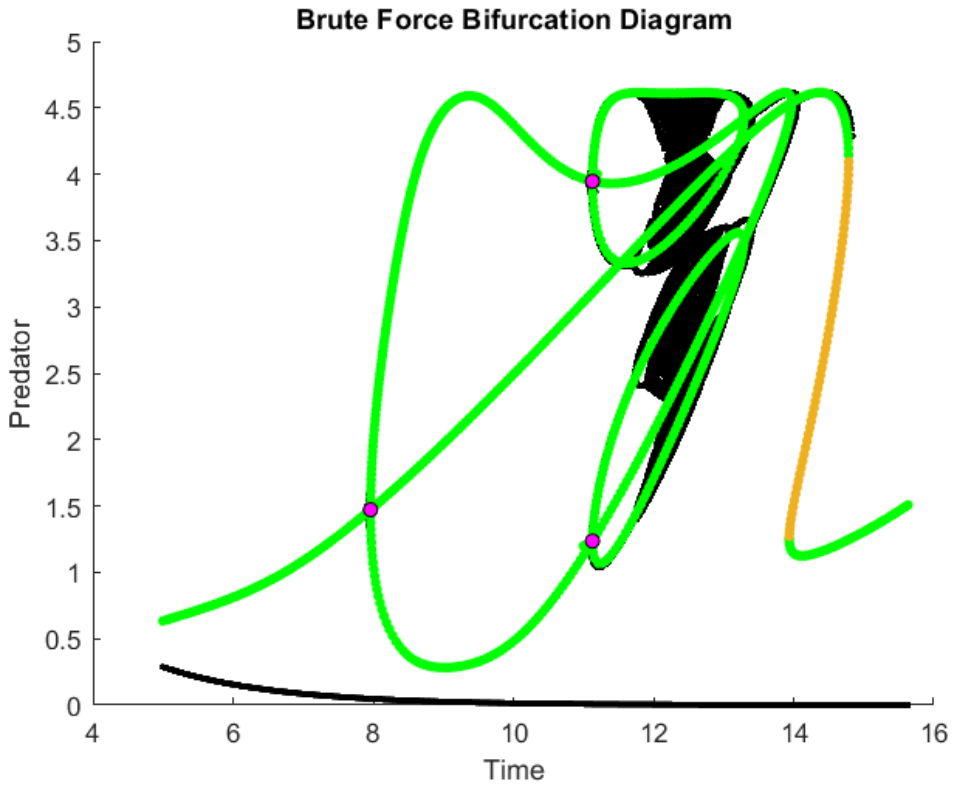

Figure 17: Plot of the branch of period-4 orbits that branches off the period-2 orbits at the above point for the Predator species

2.4.5 (v) The period doubling bifurcations along the branch of period-4 orbits.

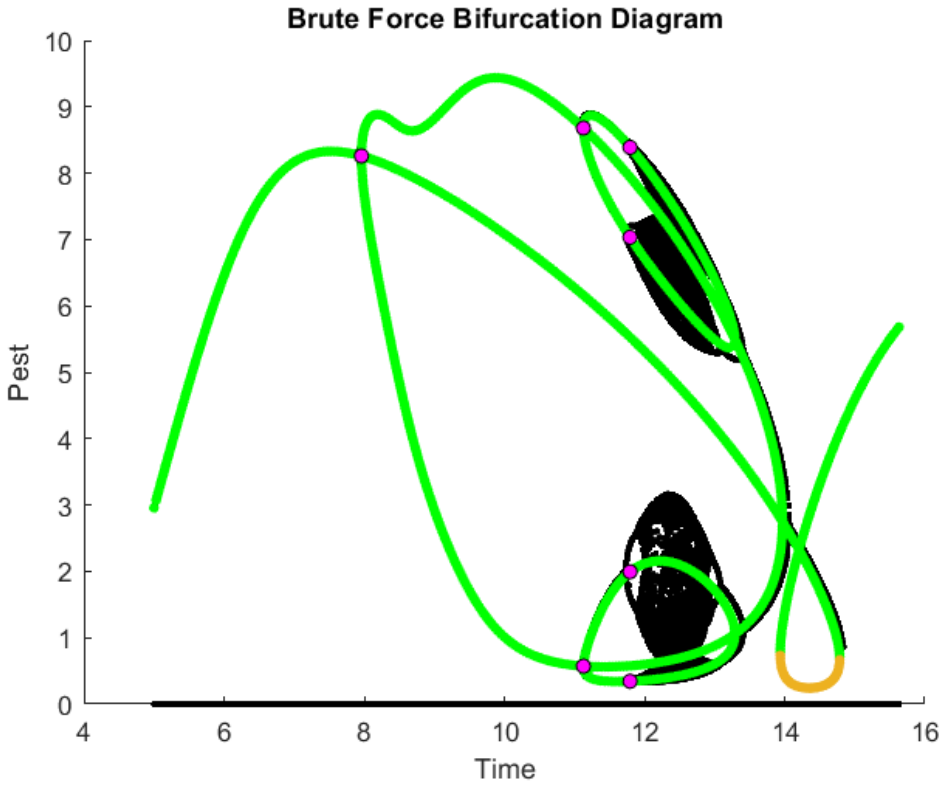

Figure 18: Plot of the period doubling bifurcations along the branch of period-4 orbits Pest species

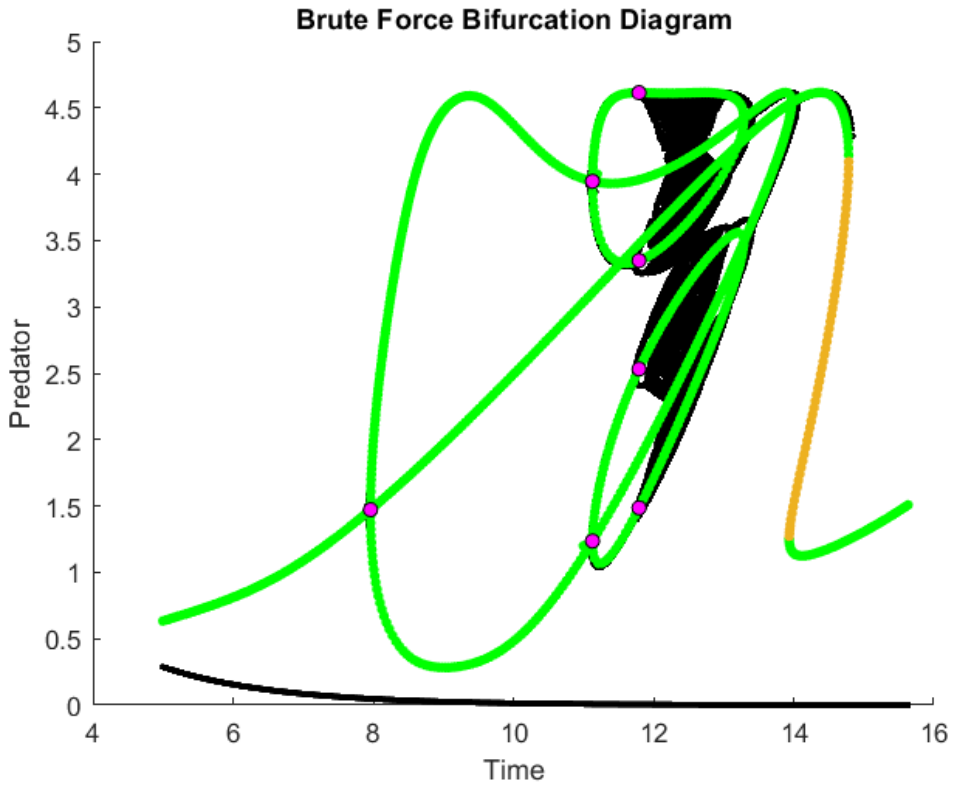

Figure 19: Plot of the period doubling bifurcations along the branch of period-4 orbits Predator species

2.4.6 (vi) The fold of period-1 orbit bifurcations.

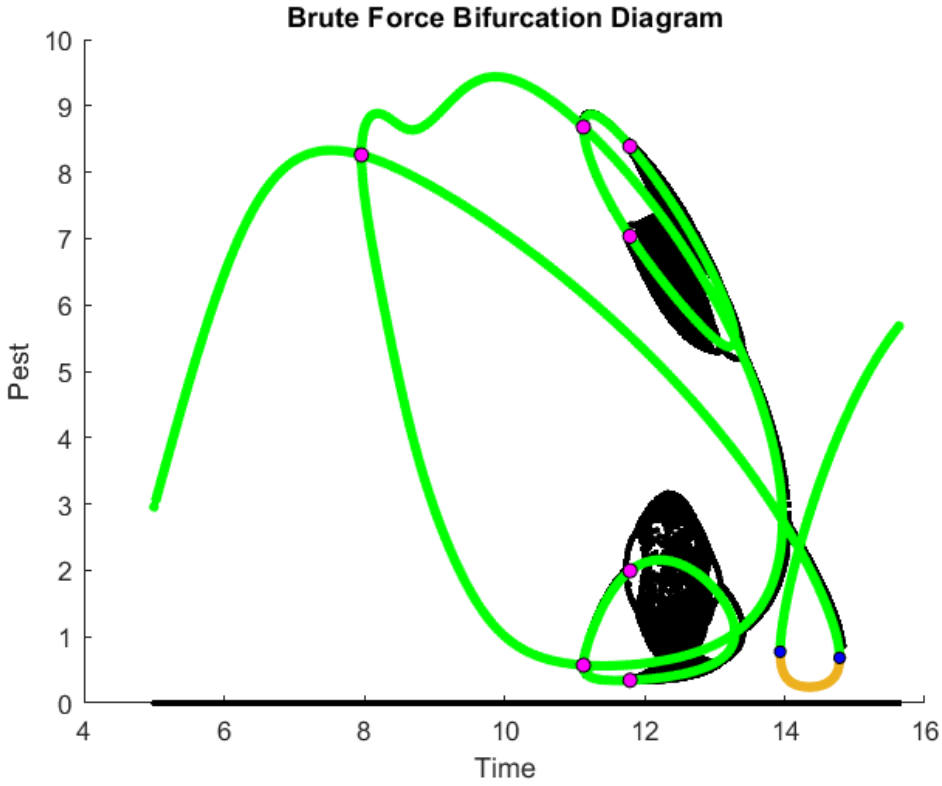

Figure 20: Plot of the fold of period-1 orbit bifurcations Pest species

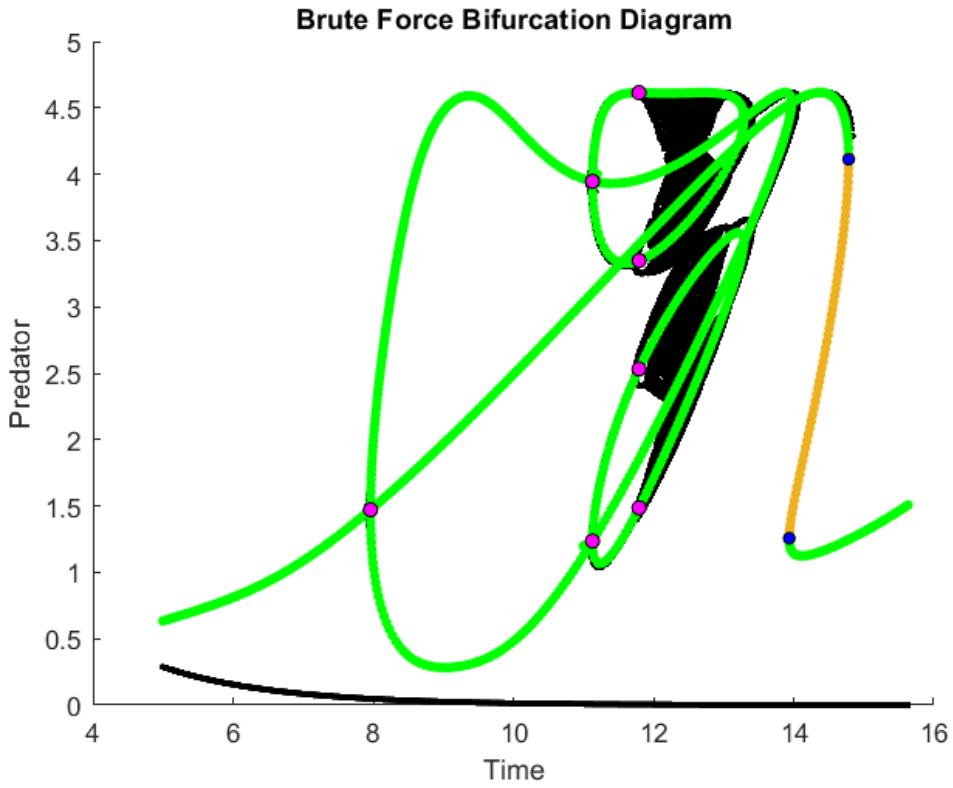

Figure 21: Plot of the fold of period-1 orbit bifurcations Predator species

3 Question 3: Basins of attraction and manifolds

3.1 Part A

Here, I implemented the general-purpose function 'MapLine', which calculates the image of a curve under a map, incorporating refinement and reduction of the curve's point mesh. My function Mapline is available to view in the appendix.

3.2 Part B

Next, I identified a specific value of T in my bifurcation diagram where the map M exhibits three period-1 orbits, including two stable ones and one of saddle type. For T=14.6, I plotted the basins of attraction for the two stable period-1 orbits over the domain (x, y) Lin [0,10] × [0,10]. The accurate locations of the period-1 orbits were determined using MySolve and annotated based on their stability.

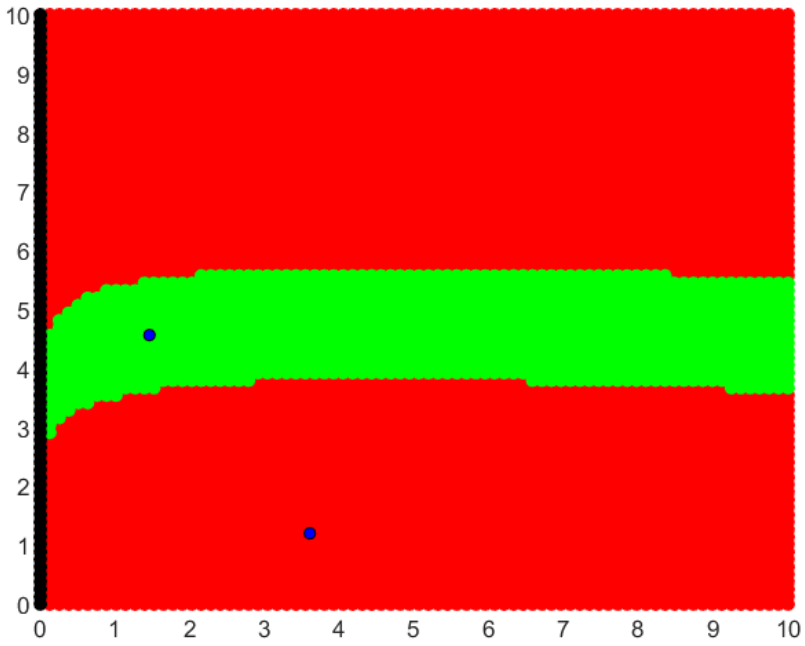

Figure 22: Plot of the basins of attraction for the two stable period-1 orbits

To enhance the plot, I employed the 'MapLine' function to compute the stable and unstable manifolds of the saddle orbit. Additionally, two trajectories were computed from initial conditions near the saddle orbit on either side of its stable manifold. These manifolds and trajectories were added to the basin of attraction plot, with distinct colors or symbols to differentiate between the two trajectories. Importantly, no lines connecting iterates were plotted to maintain clarity in the visualization.

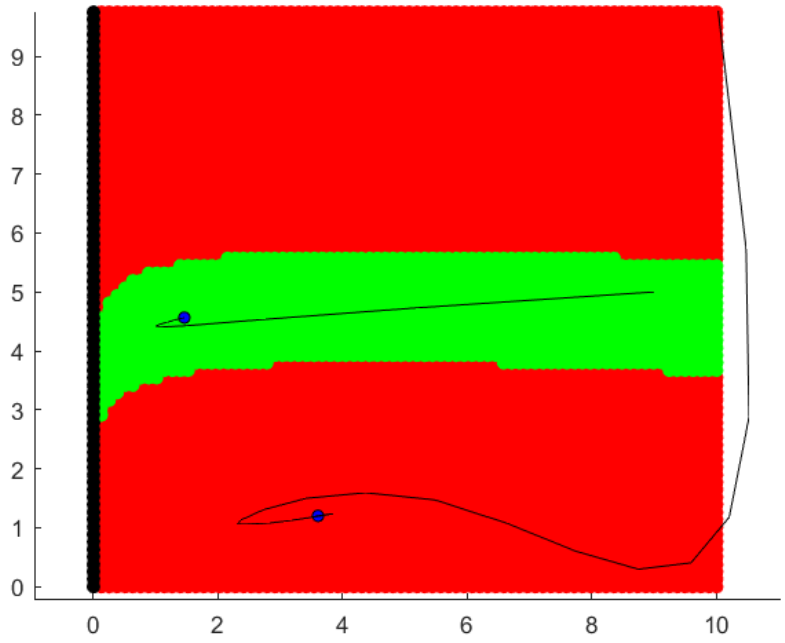

Figure 23: Plot of the manifolds and trajectories added to the basin of attraction plot

3.3 Part C

For part C, I selected a specific value of T in my bifurcation diagram where the largest Lyapunov exponent, computed in Question 1(c), is positive. At this T value, the map M has one period-1 orbit of saddle type. I computed and plotted the location of this saddle orbit over the domain (x, y) in [0, 10] × [0, 10], along with its stable and unstable manifolds. Additionally, a large-time trajectory approximating the chaotic attractor was computed and added to the plot. This visualisation provides insights into the behavior of the system at a parameter value associated with a positive largest Lyapunov exponent, indicating chaotic dynamics.

4 Question 4: Continuation in two parameters

4.1 Part A :

In the final part of this report, I utilised 'MyTrackCurve' with 'ppd' to trace out the period-doubling bifurcations identified in Question 2(b) across the (T,δ) parameter plane. Three distinct curves were generated, each corresponding to the following points:

1. Period-1 orbits bifurcate to period-2 orbits.
2. Period-2 orbits bifurcate to period-4 orbits.
3. Period-4 orbits bifurcate to period-8 orbits.

These bifurcation curves were plotted over the parameter domain (T,δ) ☐ [5, Tmax] × [0,inf), providing a visual representation of the system's behavior during the period-doubling bifurcations. These are shown in the figures below:

4.2 Part B :

Moreover, I applied 'MyTrackCurve' with 'pfold' to trace out the two fold points identified in Question 2(d) across the (T,δ) parameter plane. The resulting curves of these bifurcations were added to the bifurcation diagram created in part (a). This can be demonstrated by the figure below:

Appendix

4.3 Code for old functions : MyJacobian, MySolve, MyTrackCurve, MyIVP

```
function df=MyJacobian(f,x,h)

%imputs
%f = function
%x = point to calculate @
%h = stepsize

% define the size of the jacobian from the f(x) and x
m = length(f(x));
n = length(x);
df = zeros(m,n);

for i = 1:n

    %define the initial values of x+h & x-h as x
    xplus_h = x;
    xminus_h = x;

    % calc each value of x+h & x-h for each given n
    xplus_h(i) = xplus_h(i) + h;
    xminus_h(i) = xminus_h(i) - h;

    % very simple Backward Euler Time stepping [FIRST ORDER]
    %df(:,i) = 1/h * (f(xplus_h) -f(x)) ;


    % more precise Centered Difference Formula [SECOND ORDER]
    df(:, i) = 1/(2*h) * (f(xplus_h) - f(xminus_h));

end
end
```

```
function [x, converged, J] = MySolve(f, x0, df, tol, maxit)
    % Initial Conditions
    x = x0;
    format long

    % for loop of the iterations
    for k = 1:maxit

        % Evaluate the function and its Jacobian and Calculate the Newton update
        dx = df(x) \ f(x);

        % Calculate the norms
        ek = norm(dx); % correction
        rk = norm(f(x)); % residual
```

```
        % Print the Correction and Residual Norm for each iteration
        % - uncomment to use!
        %fprintf('Iteration %d - Correction Norm: %e, Residual Norm: %e\n', k, ek,
    rk);

        % Update the solution
        x = x - dx;

        %Check for convergence
        if rk < tol && ek < tol % rk and ek must be less than the tolerance for
    convergence

            % Update Convergence Status
            converged = 1;

            break;
        else
            % Update Convergence Status
            converged = 0;
        end
    end

    % Output the most recent Jacobian
    J = df(x);

    % Tell user if the function has converged
    % - uncomment to use!

    % if converged == 1
    % disp('The Function Has Converged')
    % else
    %     disp('The Function Has Not Converged')
    % end
end
```

```
function ylist=MyTrackCurve(userf,userdf,y0,ytan,varargin)

    % optional parameters imputed by user
    % - if user provides no additional inputs
    if isempty(varargin)

    % Set default values for internal parameters
        nmax = 100;
        s = 0.01;

    else

        % take the users input arguments
        params = varargin{1};
        nmax = params(1);
        s = params(2);
    end
```

```
    % Initialize variables
    ylist = zeros(length(y0), nmax);
    ylist(:, 1) = y0;
    tol = 1e-10;



    for k = 2:nmax


        % Step 1: Compute the new initial guess ypred
        ypred = ylist(:, k-1) + s * ytan;

        % Step 2: Solve the system using MySolve
        f=@(y)[(userf(y)) ; (ytan' * (y - ypred))] ;
        df = @(y)[userdf(y); ytan'];

        [yk, ~, J] = MySolve(f, ypred, df, tol ,10);


        % Store the solution
        ylist(:, k) = yk;

        % Compute the next ytan direction
        z = J \ [zeros(size(userf(yk))); 1];
        ytan = z / norm(z, inf) * sign(z' * ytan);


    end
end
```

```
function [xend, t, xt] = MyIVP(f, x0, tspan, N)

    % Initialize variables
    t0 = tspan(1);
    tend = tspan(2);
    h = (tend - t0) / N;
    t = zeros(N + 1, 1);
    xt = zeros(N + 1, length(x0));
    x = x0;

    % Store initial values
    t(1) = t0;
    xt(1, :) = x0;

    % RK4
    for i = 1:N
        % RK4 coefficients
        k1 = h * f(t(i), x);
        k2 = h * f(t(i) + 0.5 * h, x + 0.5 * k1);
```

```
        k3 = h * f(t(i) + 0.5 * h, x + 0.5 * k2);
        k4 = h * f(t(i) + h, x + k3);

        % Update values
        t(i + 1) = t(i) + h;
        x = x + (k1 + 2 * k2 + 2 * k3 + k4) / 6;
        xt(i + 1, :) = x;
    end

    % Output
    xend = x;
end
```

4.4 Code for new functions : LyapunovQR, Mapline

```
function [lambda, Rdiag, x] = LyapunovQR(M, xini, N)

    % information for preallocation
    n = length(xini);

    % preallocating
    x = zeros(n, N);
    Rdiag = zeros(n, N);
    Q = eye(n);

    % Iterate through the trajectory
    for i = 1:N
        % Update the initial point
        x(:, i) = xini;

        % Calculate the Jacobian
        A = MyJacobian(M, xini , 1e-15);

        % QR decomposition
        [Q, R] = qr(A * Q);

        % Flip signs if necessary
        for k = 1:n
            if R(k, k) < 0
                R(:, k) = -R(:, k);
                Q(:, k) = -Q(:, k);
            end
        end

        % Store diagonal entries of R
        Rdiag(:, i) = diag(R); % written as r_i in the notes)

        % Update xini for the next iteration
        xini = M(xini);
    end

    % Compute Lyapunov exponents
    lambda = (1 / N) * sum(log(Rdiag));
```

```
end
```

```
function [ynew,snew,xnew]=MapLine(M,x,s,maxdist,maxangle)

[nstar,N]=size(x);

if nstar ~= 2
    disp('x is not two dimensional')
end

% define snew and s_star
snew=s;
s_star = s;

% collect the map data
for i = 1:length(x)
  y(:,i) = M(x(:,i));
end


% Check the max distance
for j = 1:100
    for i = 2:N

        n = i + length(s)-N;

        % check the distance
        if norm(y(:,i)-y(:,i-1)) > maxdist

            % insert a new point
            snew = [s(1:n-1), (s(n)+s(n-1))/2, s(n:end)];

        end
        % update s
        s = snew;

    end

% gain new points using interp1
xnew = interp1(s_star,x',snew,'spline');


% collect the new map data
for i = 1:length(x)
  ynew(:,i) = M(xnew(i,:));
end

N = length(ynew);
y = ynew;


% break if not functioning
    if length(s)-N  < 1 && j > 1
```

```
            break
    end
end

% check against maxangle
if N > 2
    for j = 1:100

        for i = 2:N-1

            n = i + length(s)-N;

            ang1 = (y(:,i) - y(:,i-1))/norm(y(:,i) - y(:,i-1));
            ang2 = (y(:,i) - y(:,i+1))/norm(y(:,i) - y(:,i+1));

            % check the angle is less than maxangle
            if (abs(acos(dot(ang1,ang2)) - pi)) > maxangle

            % insert 2 new points
            snew = [s(1:n-1), (s(n) + s(n-1))/2, s(n), (s(n) + s(n+1))/2, s(n+1:end
    )];

            end
            % update s
            s = snew;

        end

        % gain new points using interp1
        xnew = interp1(s_star,x',snew,'spline');

% collect the new map data
for i = 1:length(x)
  ynew(:,i) = M(xnew(i,:));
end

N = length(ynew);
y = ynew;

% break if not functioning
        if length(s)-N < 1 && j > 1
                break;
        end

    end
end
end
```

4.5 Code for functions with regular roots

```
function [res] =ppn(U,M,j)
pp_model_710032265;
```

```
% store the data
u1 = [U(1) ; U(2)];
T = U(3);

% allocate the iterate
iterate = u1;

% iterate j times
for i = 1:j
    iterate = M(iterate,T,delta);
end

% construct res
res = iterate - u1;

end
```

```
function [res] =ppd(U,M,j)
pp_model_710032265;

% store the data
u1 = [U(1) ; U(2)];
T = U(3);

% initialise point
iterate = u1;

% loop j times
for i = 1:j
    iterate = M(iterate,T,delta);
end


% sorry could not get to work in a for loop
if j == 1
jac = MyJacobian(@(u)M(u,T,delta) , u1 , 1e-8);
elseif j ==2
jac = MyJacobian(@(u)M(M(u,T,delta),T,delta) , u1 , 1e-8);
elseif j ==4
jac = MyJacobian(@(u)M(M(u,T,delta),T,delta) , u1 , 1e-8);
end



% calcuate the eigenvalues and vectors
[V,D] = eig(jac);

eval_1 = D(1,1);
eval_2 = D(2,2);
v1 = V(:,1);
v2 = V(:,2);

% select the closest eigenvalue to -1
```

```
if abs(eval_1 - -1) > abs(eval_2 - -1)
    v = v2;
else
    v = v1;
end

% construct res
eqn1 = iterate - u1;
eqn2 = jac*v + v;
eqn3 = v' * v - 1;




res = [eqn1;eqn2;eqn3];
```

```
function res=pfold(U,M)
pp_model_710032265;

% store the data
T = U(3);
u1 = [U(1) ; U(2)];


% calculate the derivative
jac = MyJacobian(@(u)M(u,T,delta) , u1 , 1e-8);
[V,D] = eig(jac);

eval_1 = D(1,1);
eval_2 = D(2,2);
v1 = V(:,1);
v2 = V(:,2);


% select the closest eigenvalue to -1
if eval_1 > eval_2
    v = v1;
else
    v = v2;
end

% construct res
eqn1 = M(u1,T,delta) - u1;
eqn2 = jac*v-v;
eqn3 = v'*v-1;


res = [eqn1;eqn2;eqn3];
end
```

4.6 Code for PP model information

```
%% 2 species predator-pest model
load('Tmax_fun.mat')

%% Set your personal parameter value for gamma
% insert your student number xxxyyyzzz (usually starts with a 6 or 7)
% rename as pp_model_xxxyyyzzz.m and call this in your scripts for
% each questions
SNumber = 710032265;
if SNumber == 0
   error(['Enter your 9 digit student number in your local copy of pp_model.m   '
    ,...
            'See instruction sheet or contact Kyle Wedgwood at k.c.a.
    wedgwood@exeter.ac.uk if unsure.']);
end
rng(SNumber);

gamma = 0.2+0.1*rand;

% fprintf('Your personal value of gamma is %0.4f \n', gamma);

% Predator-pest intrinsic parameters
r = 2.79;
K  = 10.0;
alpha = 1.2;
c = 0.37;
d = 0.59;

% Management intervetion parameters
delta = 0.69;
h = 0.27;
lam1 = 1.25;
lam2 = 5.2;
theta = 12.5;

% Maximum management intervention period
Tmax = Tmax_interp(gamma);

% fprintf('Your personal value of T_max is %0.4f \n', Tmax);

% System of ODEs
PP_eq = @(t,x,y) [ r*x.*(1-x/K)-alpha*x./(1+gamma*x).*y;
                      c*alpha*x./(1+gamma*x).*y-d*y];

% Rewrite function PP model as f:R^2->R^2
rhs = @(x,t) PP_eq(t,x(1,:),x(2,:));

% Management intervention map
man_map = @(x,delta) [(1-delta*x(1,:)./(x(1,:)+h)).*x(1,:);
                     x(2,:)+lam1*x(1,:)./(1+theta*x(1,:))+lam2];
```

```
% Inverse of management intervention map
man_map_invX = @(x,delta) ...
  (x(1,:)-h+sqrt((x(1,:)-h).^2+4*(1-delta)*x(1,:)*h))/(2*(1-delta));
man_map_inv = @(x,delta) [ man_map_invX(x,delta);
                  x(2,:)-lam2-lam1*man_map_invX(x,delta)...
                  ./(1+theta*man_map_invX(x,delta))];
```

4.7 Code for Question 1

```
clear;close all; % clear and close everything to start
pp_model_710032265

N = 100; % steps for MyIVP in function M
numpoints = 1;  % number of grid points
k = 60;         % number of map itterations
buff = 0.02;    % space between values
time_interval = 5:buff:Tmax; % time interval

% Values used to produce the plot in the report
% THIS TAKE A VERY LONG TIME TO RUN!!! (apologies)
% N = 100;
% numpoints = 4;
% K = 100;
% buff = 0.05;
% time_interval = 5:buff:Tmax;

% LyapunovQR variables
N_L = 300;

% define the Map to iterate
M_T = @(u,T)[MyIVP(@(u,t) rhs(t,u), u, [0;T], N)];
M = @(u,T,delta)(M_T(man_map(u,delta) ,T));

% create a grid of initial conditions
xvalues = linspace(0, 10, numpoints);
yvalues = linspace(0, 10, numpoints);
[x, y] = meshgrid(xvalues, yvalues);

% reshape to a vector
X = reshape(x, [], 1);
Y = reshape(y, [], 1);

% preallocate arrays
u1 = zeros(1,length(X));
u2 = zeros(1,length(Y));
x1 = zeros(length(X),length(time_interval));
y1 = zeros(length(Y),length(time_interval));
l1 = zeros(1,length(time_interval));
l2 = zeros(1,length(time_interval));


% initialise counting variable
p=0;
```

```
% time the for loop
tic
% iterate for 5:Tmax
for T = time_interval
    p = p+1;
    lpyap=0;

        % iterate each initial condition
        for j = 1:length(X)
        u0 = [X(j);Y(j)];

            % collect the last Lyapunov components for each time interval
            % with u0!=0
            if u0(1) ~= 0 && u0(2) ~= 0 && lpyap==0

                [lyp_exp, Rdiag, ~] = LyapunovQR(@(u)M(u,T,delta), u0, N_L);

                l1(p) = Rdiag(1,N_L);
                l2(p) = Rdiag(2,N_L);


                lpyap=1;
            end

                % loop the map onto itself K times
                for i = 1:k
                    u0 = M(u0,T,delta);
                end

        % collect the new u values
        u1(j) = u0(1);
        u2(j) = u0(2);

        end

    % collect all the new u values for each time interval
    x1(:,p) = u1;
    y1(:,p) = u2;

    % display the T interval iteration for user ease
    disp(T)

end
% end the timer
toc

% reset value of T for ease
T = time_interval;


% plot the data
```

```
% pest
figure(1);
scatter(T,x1,'.' ,'k' )
title('Brute Force Bifurcation Diagram')
ylabel('Pest')
xlabel('Time')

% predators
figure(2);
scatter(T,y1,'.' ,'k' )
title('Brute Force Bifurcation Diagram')
ylabel('Predator')
xlabel('Time')


%% C

% plot the lyapunov exponents
figure(3)
hold on
scatter((T) , [(log(l1));(log(l2))] , '.' )
yline(0,'r--')
title('Brute Force Bifurcation Diagram')
xlabel('Time')
ylabel('Lyapunov Exponents')


%% D)

%period 1 orbit

% initial conditions
u0 = [4.5;0.7];
T = 6;
k = 150;
fit={'linewidth',2};
N_L = 10;

% preallocate
p1_lup1 = zeros(1,k);
p1_lup2= zeros(1,k);
p1_u1= zeros(1,k);
p1_u2= zeros(1,k);

% loop the Lyapunov and save the data
for i = 1:k
    u0 = M(u0,T,delta);

    [lambda, Rdiag, u] = LyapunovQR(@(u)M(u,T,delta), u0, N_L);

    p1_lup1(:,i) = Rdiag(1,N_L);
    p1_lup2(:,i) = Rdiag(2,N_L);
```

```
    p1_u1(i) = u(1);
    p1_u2(i) = u(2);
end

% plot the data
figure(4)
nexttile
hold on

scatter(1:k,p1_u1,'filled','r',fit{:})
scatter(1:k,p1_u2,'filled','g',fit{:})


xlabel('Step K')
ylabel('Iterates of x')
legend('Pest' , 'Predator' , Location='best')
title('Iterates of x against the step number k. ')

nexttile
scatter(1:k , [(log(p1_lup1));(log(p1_lup2))] , '.' , 'k')
xlabel('Step K')
ylabel('Lyapunov exponents')
title('The Lyapunov exponents at each step against the step number k')




%% D)

%period 2 orbit

% initial conditions
u0 = [1;1];
T =10;
k = 300;
N_L = 10;

% preallocate
p2_lup1 = zeros(1,k);
p2_lup2= zeros(1,k);
p2_u1= zeros(1,k);
p2_u2= zeros(1,k);

% loop lyapunov and save the data
for i = 1:k
    u0 = M(u0,T,delta);

[lambda, Rdiag, u] = LyapunovQR(@(u)M(u,T,delta), u0, N_L);

    p2_lup1(:,i) = Rdiag(1,N_L);
    p2_lup2(:,i) = Rdiag(2,N_L);
```

```
    p2_u1(i) = u(1);
    p2_u2(i) = u(2);
end

% plot the data
figure(5)
nexttile
hold on
scatter(1:k,p2_u1,'filled','r',fit{:})
scatter(1:k,p2_u2,'filled','g',fit{:})

xlabel('Step K')
ylabel('Iterates of u')
legend('Pest' , 'Predator' , Location='best')
title('Iterates of x against the step number k. ')

nexttile
scatter(1:k , [(log(p2_lup1));(log(p2_lup2))] , '.' , 'k')
xlabel('Step K')
ylabel('Lyapunov exponents')
title('The Lyapunov exponents at each step against the step number k')


%% D)

%period 4 orbit

% initial conditions
u0 = [1;1];
T =11.2;
k = 600;
N_L = 15;

% preallocate the data
p4_lup1 = zeros(1,k);
p4_lup2= zeros(1,k);
p4_u1= zeros(1,k);
p4_u2= zeros(1,k);

% loop lyapunov and store the data
for i = 1:k
    u0 = M(u0,T,delta);

[lambda, Rdiag, u] = LyapunovQR(@(u)M(u,T,delta), u0, N_L);

    p4_lup1(:,i) = Rdiag(1,N_L);
    p4_lup2(:,i) = Rdiag(2,N_L);
    p4_u1(i) = u(1);
    p4_u2(i) = u(2);
end

% plot the data
```

```
figure(6)
nexttile
hold on
scatter(1:k,p4_u1,'filled','r',fit{:})
scatter(1:k,p4_u2,'filled','g',fit{:})

xlabel('Step K')
ylabel('Iterates of u')
legend('Pest' , 'Predator' , Location='best')
title('Iterates of x against the step number k. ')

nexttile
scatter(1:k , [(log(p4_lup1));(log(p4_lup2))] , '.' , 'k')
xlabel('Step K')
ylabel('Lyapunov exponents')
title('The Lyapunov exponents at each step against the step number k')

%% D)

% CHAOTIC ORBIT

% initial conditions
u0 = [1;1];
T =12.2;
k = 2000;
N_L = 40;

% preallocate
c_lup1 = zeros(1,k);
c_lup2= zeros(1,k);
c_u1= zeros(1,k);
c_u2= zeros(1,k);

% loop the lyapunov and store the data
for i = 1:k
    u0 = M(u0,T,delta);

[lambda, Rdiag, u] = LyapunovQR(@(u)M(u,T,delta), u0, N_L);

    c_lup1(:,i) = Rdiag(1,N_L);
    c_lup2(:,i) = Rdiag(2,N_L);
    c_u1(i) = u(1);
    c_u2(i) = u(2);
end

% plot the data
figure(7)
nexttile
hold on
scatter(1:k,c_u1,'filled','r',fit{:})
scatter(1:k,c_u2,'filled','g',fit{:})
```

```
xlabel('Step K')
ylabel('Iterates of u')
legend('Pest' , 'Predator' , Location='best')
title('Iterates of x against the step number k. ')

nexttile
scatter(1:k , [(log(c_lup1));(log(c_lup2))] , '.' , 'k')
xlabel('Step K')
ylabel('Lyapunov exponents')
title('The Lyapunov exponents at each step against the step number k')
```

4.8 Code for Question 2

```
% clear workspace and load pp_model
clear;
pp_model_710032265;
N = 100;
% define M
M_T = @(u,T)[MyIVP(@(u,t) rhs(t,u), u, [0;T], N)];
M = @(u,T,delta)(M_T(man_map(u,delta) ,T));

%% 2a] - WORKING!!!
% preallocate data arrays
stable_data = [];
unstable_data = [];
saddle_data = [];

% pick a time a (x,y) from bifurcation diagram
T = 5;
u = [5;5];

% ensure point picked is accurate
for i = 1:200
u = M(u,T,delta);
end

% add time value to the point
u = [u;T];
period = 1;

% define varialbes for MyTrackCurve
ytan = [1;1;1];
nmax = 500;
s = 0.05;

% MyJacobian for MyTrackCurve
dppn = @(u)MyJacobian(@(u)ppn(u,M,period) , u , 1e-8); % 1e-8
ylist = MyTrackCurve( @(u)ppn(u,M,period) , @(u)dppn(u) , u , ytan, [nmax,s]);

% plotting in colours depending on stability
for i = 1:length(ylist)

    % stop when tmax reached
```

```
    if ylist(3,i) > Tmax
        break;
    end

% inspect each element of u
u = [ylist(1,i) ; ylist(2,i); ylist(3,i)];

% check the jacobian
jac = MyJacobian(@(u)ppn(u,M,period) , u , 1e-12);

% ensure it is square
jac = jac(1:2,1:2);

%check eigenvalues
e_val = eig(jac);

    if all(e_val < 0)                                    % STABLE
             stable_data = [stable_data; u(1), u(2) , u(3)];%#ok<AGROW>
    elseif all(e_val > 0)                               % UNSTABLE
             unstable_data = [unstable_data; u(1), u(2) , u(3)];%#ok<AGROW>
    else                                                 % SADDLE %
             saddle_data = [saddle_data; u(1), u(2) , u(3)];%#ok<AGROW>
    end
end

% if the arrays are not empty the produce plot
if isempty(stable_data) == 0
    col ={15,'filled','MarkerFaceColor', ' g'};

    figure(1)
    hold on
    scatter(stable_data(:,3) , stable_data(:,2),col{:})
    figure(2)
    hold on
    scatter(stable_data(:,3) , stable_data(:,1),col{:})
end
if isempty(unstable_data) == 0
    col ={15,'filled','MarkerFaceColor', ' r'};

    figure(1)
    hold on
    scatter(unstable_data(:,3) , unstable_data(:,2),col{:})
    figure(2)
    hold on
    scatter(unstable_data(:,3) , unstable_data(:,1),col{:})
end
if isempty(saddle_data) == 0

    col ={15,'filled','MarkerFaceColor', '[0.9290 0.6940 0.1250]'};

    figure(1) % -pest
    hold on
```

```
    scatter(saddle_data(:,3) , saddle_data(:,2),col{:})
    figure(2) % -predator
    hold on
    scatter(saddle_data(:,3) , saddle_data(:,1),col{:})

end

%% 2)B] - WORKING

%% 2)C] - WORKING

%% 2)D]i} WORKING!! - put label on the plot??
%The period doubling bifurcations along the branch of period-1 solutions
% initial guess
u = [8; 1 ; 7];

% state what period we are looking for
period = 1;

% apply MyJacobian
dppd = @(u)MyJacobian(@(u)ppd(u,M,period) , u , 1e-6);

% apply MySolve
[x, converged, ~] = MySolve(@(u)ppd(u,M,period) ,u, dppd , 1e-6 , 20 );


% display the values rounded as asked
    disp('Function Has Converged!')
    disp('The period doubling bifurcations along the branch of period-1 solutions:'
    )

    u_p1_PDB = round( [x(1); x(2)], 4);
    T_p1_PDB = round(x(3),4);

    fprintf('(x, y) = \n(%0.6f, %0.6f)\n',u_p1_PDB);
    fprintf('T = \n%0.6f\n',T_p1_PDB);

% details for scatter
col ={30,'filled', 'MarkerFaceColor', 'm' , 'MarkerEdgeColor' , 'k' , 'LineWidth'
    ,0.6};

% plot predator data
figure(2)
hold on
scatter(T_p1_PDB,u_p1_PDB(2) ,col{:} );

% plot pest data
figure(1)
hold on
scatter(T_p1_PDB,u_p1_PDB(1) ,col{:});

%% 2)D]ii}
```

```
% The branch of period-2 orbits that branches off the period-1 orbits at the above
    point.

% preallocate data arrays
stable_data = [];
unstable_data = [];
saddle_data = [];

% give the period we are looking for
period = 2;

% variables for MyTrackCurve
ytan = [1;1;1];
T = T_p1_PDB;
u = [7.69;1.96;T];
nmax = 450;
s = 0.05;

% apply MyJacobian
dppn = @(u)MyJacobian(@(u)ppn(u,M,period) , u , 1e-7);

% apply MySolve
ylist = MyTrackCurve( @(u)ppn(u,M,period) , @(u)dppn(u) , u , ytan, [nmax,s]);


% plotting in colours
for i = 1:length(ylist)

    % end if tmax reached
    if ylist(3,i) > Tmax
        break;
    end

    % identify the data
    u = [ylist(1,i) ; ylist(2,i); ylist(3,i)];

    % apply MyJacobian
    jac = MyJacobian(@(u)ppn(u,M,period) , u , 1e-8);

    % ensure it is square
    e_val = eig(jac(1:2,1:2));

    if all(e_val < 0)                                    % STABLE %
             stable_data = [stable_data; u(1), u(2) , u(3)]; %#ok<AGROW>
    elseif all(e_val > 0)                               % UNSTABLE %
             unstable_data = [unstable_data; u(1), u(2) , u(3)];%#ok<AGROW>
    else                                                 % SADDLE %
             saddle_data = [saddle_data; u(1), u(2) , u(3)]; %#ok<AGROW>
    end
end

% igonre the first few inaccurate data
```

```
exclude_first = 3:nmax;

% plot the non empty data sets
if isempty(stable_data) == 0
    col ={15,'filled','MarkerFaceColor', ' g'};

    figure(2)
    hold on
    scatter(stable_data(exclude_first,3) , stable_data(exclude_first,2),col{:})
    figure(1)
    hold on
    scatter(stable_data(exclude_first,3) , stable_data(exclude_first,1),col{:})
end
if isempty(unstable_data) == 0
    col ={15,'filled','MarkerFaceColor', ' r'};

    figure(2)
    hold on
    scatter(unstable_data(exclude_first,3) , unstable_data(exclude_first,2),col{:})
    figure(1)
    hold on
    scatter(unstable_data(exclude_first,3) , unstable_data(exclude_first,1),col{:})
end
if isempty(saddle_data) == 0

    col ={15,'filled','MarkerFaceColor', '[0.9290 0.6940 0.1250]'};

    figure(2) % -pred
    hold on
    scatter(saddle_data(exclude_first,3) , saddle_data(exclude_first,2),col{:})
    figure(1) % -pest
    hold on
    scatter(saddle_data(exclude_first,3) , saddle_data(exclude_first,1),col{:})

end

%% 2)D]iii}
% The period doubling bifurcations along the branch of period-2 orbits.

% identify the period we are looking for
period = 2;
for i = 1:2

    % perform actions for each initial condition.
    if i ==1
        u = [8;1;10];
    elseif i == 2
        u = [1;5;10];
    end

    % apply MyJacobian
    dppd = @(u)MyJacobian(@(u)ppd(u,M,period) , u , 1e-6);
```

```
    % apply MySolve
    [x, converged, ~] = MySolve(@(u)ppd(u,M,period) ,u, dppd , 1e-6 , 20 );

    % display the values rounded as asked
    disp('Function Has Converged!')
    disp('The fold of period-1 orbit bifurcations:')

    u_p2_PDB(i,:) = [x(1); x(2)]; %#ok<SAGROW>
    T_p2_PDB = x(3);

    fprintf('(x, y) = \n(%0.6f, %0.6f)\n',round(u_p2_PDB(i,:),4));
    fprintf('T = \n%0.6f\n',round(T_p2_PDB,4));

 % scatter information
col ={30,'filled', 'MarkerFaceColor', 'm' , 'MarkerEdgeColor' , 'k' , 'LineWidth'
    ,0.6};

% plot predator data
figure(2)
hold on
scatter(T_p2_PDB,u_p2_PDB(i,2) ,col{:});

% plot prey data
figure(1)
hold on
scatter(T_p2_PDB,u_p2_PDB(i,1) ,col{:});
 end
%% 2)D]iv}
%The branch of period-4 orbits that branches off the period-2 orbits at the above
    point.

% perallocate the data
stable_data = [];
unstable_data = [];
saddle_data = [];

% initialise values for T and the period we are looking for
T = T_p2_PDB;
period = 4;


for j = 1:2

    % iterate for both initial conditions
    if j == 1
        u = [8.8;1.2;11];

    elseif j == 2
        u = [0.5;4;11.2];

    end
```

```
% data for MyTrackCurve
nmax = 180;
ytan = [1;1;1];
s = 0.05;

% use MyJacobian and MyTrackCurve
dppn = @(u)MyJacobian(@(u)ppn(u,M,period) , u , 1e-8);
ylist = MyTrackCurve( @(u)ppn(u,M,period) , @(u)dppn(u) , u , ytan, [nmax,s]);


%plotting in colours
for i = 1:length(ylist)

    % stop if Tmax is reached
    if ylist(3,i) > Tmax
        break;
    end

% initialise each value
u = [ylist(1,i) ; ylist(2,i); ylist(3,i)];

%find the jacobian
jac = MyJacobian(@(u)ppn(u,M,period) , u , 1e-8);

% ensure it is square
jac = jac(1:2,1:2);

% find the eigen values
e_val = eig(jac);

    if all(e_val < 0)                                    % STABLE %
             stable_data = [stable_data; u(1), u(2) , u(3)]; %#ok<AGROW>
    elseif all(e_val > 0)                               % UNSTABLE %
             unstable_data = [unstable_data; u(1), u(2) , u(3)]; %#ok<AGROW>
    else                                                 % SADDLE %
             saddle_data = [saddle_data; u(1), u(2) , u(3)]; %#ok<AGROW>
    end
end

% plot the non empty data sets
if isempty(stable_data) == 0
    col ={15,'filled','MarkerFaceColor', ' g'};

    figure(2)
    hold on
    scatter(stable_data(:,3) , stable_data(:,2),col{:})
    figure(1)
    hold on
    scatter(stable_data(:,3) , stable_data(:,1),col{:})
end
if isempty(unstable_data) == 0
```

```
    col ={15,'filled','MarkerFaceColor', ' r'};

    figure(2)
    hold on
    scatter(unstable_data(:,3) , unstable_data(:,2),col{:})
    figure(1)
    hold on
    scatter(unstable_data(:,3) , unstable_data(:,1),col{:})
end
if isempty(saddle_data) == 0

    col ={15,'filled','MarkerFaceColor', '[0.9290 0.6940 0.1250]'};

    figure(2) % -predator
    hold on
    scatter(saddle_data(:,3) , saddle_data(:,2),col{:})
    figure(1) % -pest
    hold on
    scatter(saddle_data(:,3) , saddle_data(:,1),col{:})
end
end

%% 2)D]v}
% The period doubling bifurcations along the branch of period-4 orbits.

% identify the period we are looking for
period = 4;

% iterate for all 4 initial conditions
for i = 1:4

if i ==1
    u = [2;4;11];
elseif i == 2
    u = [1.99;4.614; 11];
elseif i == 3
    u = [10;1;11.7];
elseif i == 4
    u = [8;2;11];
end

% use MyJacobian and MyTrackCurve
    dppd = @(u)MyJacobian(@(u)ppd(u,M,4) , u , 1e-6);
    [x, converged, ~] = MySolve(@(u)ppd(u,M,4) ,u, dppd , 1e-6 , 20 );

    % display the information
    disp('Function Has Converged!')
    disp('The fold of period-1 orbit bifurcations:')

    u_p4_PDB(i,:) = [x(1); x(2)]; %#ok<SAGROW>
    T_p4_PDB(i) = x(3); %#ok<SAGROW>
```

```
    fprintf('(x, y) = \n(%0.6f, %0.6f)\n',round(u_p4_PDB,4));
    fprintf('T = \n%0.6f\n',round(T_p4_PDB,4));

% scatter information
col = {30,'filled','MarkerFaceColor', ' m', 'MarkerEdgeColor' , 'k'};

% plot the predator data
figure(2)
hold on
scatter(T_p4_PDB(i),u_p4_PDB(i,2) ,col{:});

% plot the pest data
figure(1)
hold on
scatter(T_p4_PDB(i),u_p4_PDB(i,1) ,col{:});

end
%% 2)D]vi}
% The fold of period-1 orbit bifurcations

% iterate for both folds
for i = 1:2
if i ==1
    u = [1;4;15];
elseif i == 2
    u = [1;1;14];
end

% use MyJacobian and MySolve
dpfold = @(u,T)MyJacobian(@(u)pfold(u,M) , u , 1e-6);

 [x, converged, ~] = MySolve(@(u)pfold(u,M) ,u, dpfold , 1e-6 , 20);


    disp('Function Has Converged!')
    disp('The fold of period-1 orbit bifurcations:')

    u_fold = [x(1); x(2)];
    T_fold = x(3);

    T_fold_all(i) = T_fold;

    fprintf('(x, y) = \n(%0.6f, %0.6f)\n',round(u_fold,4));
    fprintf('T = \n%0.6f\n',round(T_fold,4));


% scatter data
col = {20,'filled','MarkerFaceColor', ' b', 'MarkerEdgeColor' , 'k'};

% plot predator
figure(2)
hold on
```

```
scatter(T_fold,u_fold(2) ,col{:});

% plot pest data
figure(1)
hold on
scatter(T_fold,u_fold(1) ,col{:});
end
%% TIDY THE PLOT UP A BIT
% scatter colours
col = {20,'filled','MarkerFaceColor', ' m', 'MarkerEdgeColor' , 'k'};

% plot all the points on top again
% pred
figure(1)
hold on
scatter(T_p1_PDB,u_p1_PDB(2) ,col{:} );

% prey
figure(2)
hold on
scatter(T_p1_PDB,u_p1_PDB(1) ,col{:});



for i = 1:2
    % pred
    figure(1)
    hold on
    scatter(T_p2_PDB,u_p2_PDB(i,2) ,col{:});

    % prey
    figure(2)
    hold on
    scatter(T_p2_PDB,u_p2_PDB(i,1) ,col{:});
end

for i = 1:4
    % pred
    figure(1)
    hold on
    scatter(T_p4_PDB(i),u_p4_PDB(2,i) ,col{:});

    % prey
    figure(2)
    hold on
    scatter(T_p4_PDB(i),u_p4_PDB(1,i) ,col{:});
end


%% Display ALL the information

fprintf('Period 1 Doubling Bifurcation \n(%0.6f, %0.6f)\n',u_p1_PDB);
```

```
fprintf('T = \n%0.6f\n',T_p1_PDB);

fprintf('Period 2 Doubling Bifurcation \n(%0.6f, %0.6f)\n',round(u_p2_PDB,4)) ;
fprintf('T = \n%0.6f\n',round(T_p2_PDB,4));

fprintf('Period 4 Doubling Bifurcation\n(%0.6f, %0.6f)\n',round(u_p4_PDB,4));
fprintf('T = \n%0.6f\n',round(T_p4_PDB(1),4));

fprintf('Fold of period-1 orbit bifurcations \n(%0.6f, %0.6f)\n',round(u_fold,4));
fprintf('T = \n%0.6f\n',round(T_fold,4));
```

4.9 Code for Question 3

```
pp_model_710032265;

N = 100;
M_T = @(u,T)[MyIVP(@(u,t) rhs(t,u), u, [0;T], N)];
M = @(u,T,delta)(M_T(man_map(u,delta) ,T));


%% 3)a]

%% 3)b]

% loop 2 sets of initial conditions to find stable points
for j = 1:2
u0 = [2;4];
    if j == 2
    u0 = [1;10];
    end
T = 14.6;

for i = 1:200
u0 = M(u0,T,delta);
end

% store the information
eq(:,j) = round(u0,4); %#ok<SAGROW>
end

%% BASINS OF ATTRACTION

% data for the plots
numpoints = 80;
k = 200;

% preallocate data
equil_1 = [];
equil_2 = [];
saddle1 = [];

% create the grid of data points
xvalues = linspace(0, 10, numpoints);
```

```
yvalues = linspace(0, 10, numpoints);
[x, y] = meshgrid(xvalues, yvalues);

X = reshape(x, [], 1);
Y = reshape(y, [], 1);

% iterate for each initial condition
for j = 1:length(X)

u0 = [X(j);Y(j)];

% loop the map until convergence (or large k)
for i = 1:k
u0 = M(u0,T,delta);
end



if abs(round(u0,4) - eq(:,1))  < 0.1 % stable point 1
equil_1 = [equil_1; X(j), Y(j)]; %#ok<AGROW>

elseif abs(round(u0,4) - eq(:,2)) < 0.1 % stable point 2
equil_2 = [equil_2; X(j), Y(j)]; %#ok<AGROW>

else                                  % saddle point
saddle1 = [saddle1; X(j), Y(j)]; %#ok<AGROW>

end
end

figure(1)
hold on
scatter(equil_1(:,1),equil_1(:,2) , 'r' , 'filled')
scatter(equil_2(:,1),equil_2(:,2) , 'g', 'filled')
scatter(saddle1(:,1),saddle1(:,2) , 'k', 'filled')

%% plot the 2 stable points

col ={25,'filled','MarkerFaceColor', 'b' , 'MarkerEdgeColor' , 'k'};

figure(1)
hold on
scatter(eq(:,1),eq(:,2) , col{:})


%% TRAJECTORIES

% define M for a fixed value of T
Mm = @(u)(M_T(man_map([u(1);u(2)],delta) ,T));

% loop the initial conditions
for j = 1:2
```

```
% reset the data each loop
clear data;

u0 = [10;10];

if j == 2
u0 = [9;5];
end


k = 15;

% iterate the map to get the data
    for i = 1:k

    data(i,:) = u0; %#ok<SAGROW>

    u0 = Mm(u0);

    end

% transpose this for use of Mapline
x=data';

% define the imputs for mapline
s = 1:k;
maxdist = 0.01;
maxangle = 5*pi/180;

% implement MapLine
[ynew, snew, xnew] = MapLine(Mm, x, s, maxdist, maxangle);

% if using the first initial condition repeat mapline
if j ==1
s = 1:length(xnew);
[ynew, snew, xnew] = MapLine(Mm, xnew', s, maxdist, maxangle);
end

% plot both trajectories
figure(1)
hold on
plot(xnew(:,1), xnew(:,2) ,'k' )

end
```

4.10 Code for Question 4