Faculty of Environment, Science and Economy

MTH3039 Computational Nonlinear Dynamics

Coursework 1 An SIR system

Academic Year 2023 / 2024

Student Name: Kofi Barton-Byfield
Student ID: 710032265

Contents

This report analyses a simplified SIR (Susceptible-Infectious-Recovered) infectious disease model, represented by a set of nonlinear differential equations. The model focuses on the dynamics of the proportion of a population that is susceptible (S) and infected (I) by a particular infectious disease. The key parameter, β, influences the solutions' steady states and periodic behaviors. The study explores the impact of varying β on the number, stability, and nature of coexisting steady states and periodic orbits. The model operates under the constraint that the total population remains constant, expressed by the equation $S(t) + I(t) + R(t) = 100$, where each variable represents a percentage of the overall population.

$$\dot{S}(t) = rS(t)(1 - S(t)/k) - \beta\frac{S(t)I(t)}{1 + aI(t)}, 0 \leq S(t) \leq 100.$$

$$\dot{I}(t) = \beta\frac{S(t)I(t)}{1 + \alpha I(t)} - \theta I(t) - \lambda\frac{I(t)}{1 + \epsilon I(t)}, 0 \leq I(t) \leq 100.$$

The model has several parameters:
K: carrying capacity (default value K=100);
r: intrinsic growth rate (default value r=10);
β: maximum infection rate (varies over interval [0,2]);
α: inhibitory effect when number of infected is large (default value ɑ=0.01);
λ: maximum treatment rate (default value λ=20);
ϵ: reduction in treatment capacity when number of infected is large (varies according to Student ID No.);
θ: lumped parameter accounting for the natural death rate, the disease induced death rate, and the natural recovery rate (default value θ=2.3).
The nullclines for can be written as the S-values expressed in terms of I:

$$S = K - \frac{\beta KI}{r(1 + \alpha I}, andS = 0, [Snullclines]$$

.

$$S = \frac{1 + \alpha I}{\beta}(\theta + \frac{\lambda}{1 + \epsilon I}, andI = 0, [Inullclines].$$

For my report we will note that $(S, I) = (100, 0)$ is always an equilibrium as is $(0, 0)$

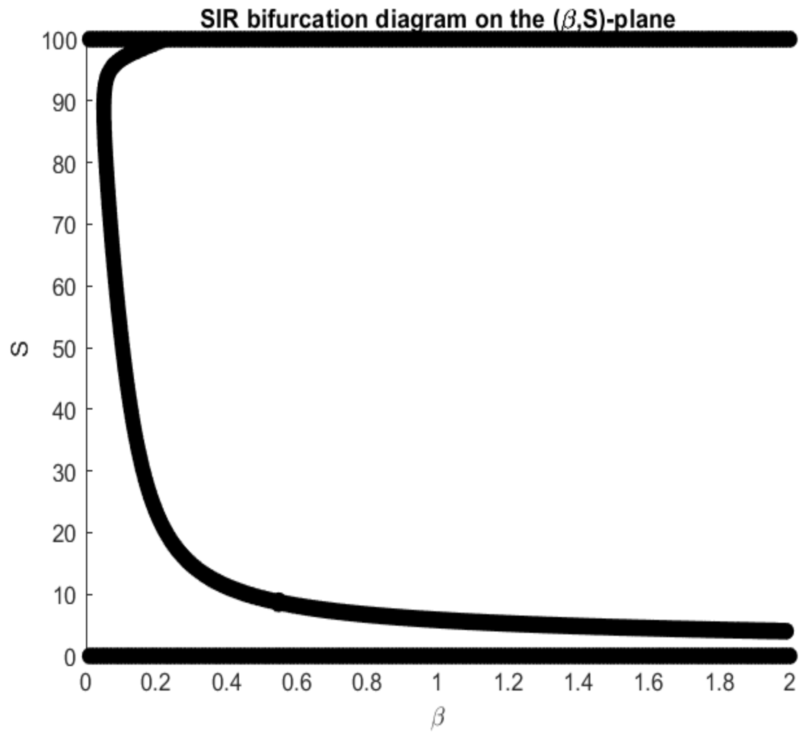
1 Question 1: Location of equilibria and their stability

1.1 Part A : Implement the functions

I have Implemented the functions MyJacobian, MySolve and MyTrackCurve throughout my code (as seen in the appendix), to complete the questions in the report.

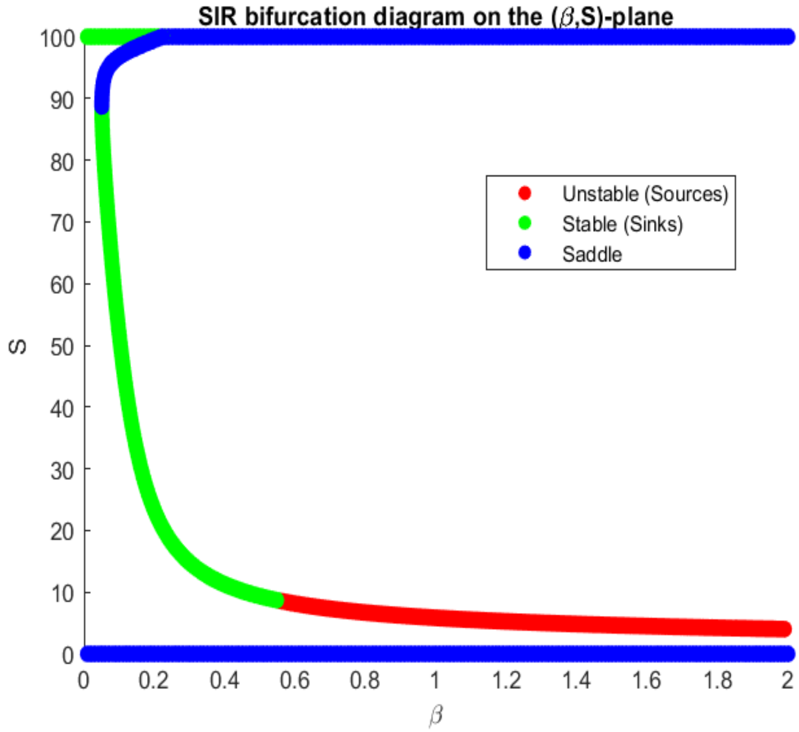
1.2 Part B : Equilibria

I have found all equilibria of the system for parameters $\beta \in [0, 2]$. Then in the diagram below I plotted the curves of equilibria (the bifurcation diagram) in the $(\beta, S)-plane$, in the specified range.



1.3 Part C : Stability

Next I will indicate the stability of each type of equilibrium along the curves I obtained in part (b). This can be seen in my plot below:



2 Question 2: Phase portraits

2.1 Part A

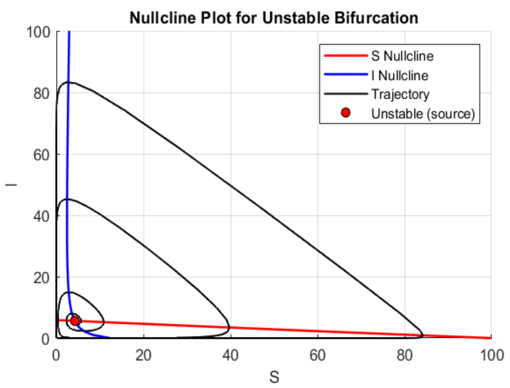
Next, I will implement MyIVP in my code to complete the following parts of question 2.

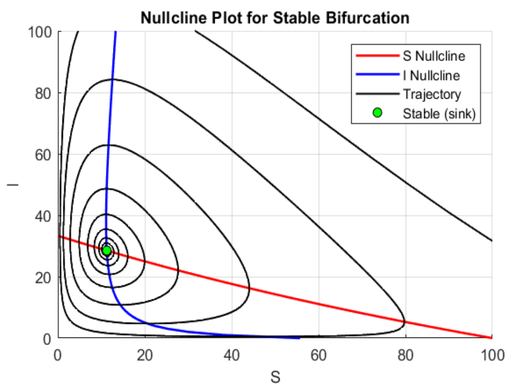
2.2 Part B

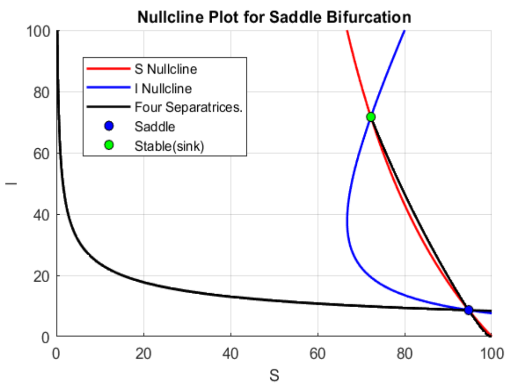
Below I have plotted the following robust phase portraits:

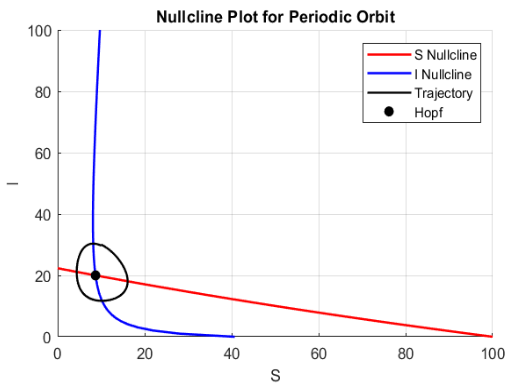
- nullclines.
- equilibria (indicating their type, ie sink, source or saddle).
- Periodic Orbits.
- For each sink, source and periodic orbit, I plotted a trajectory that approaches the sink/source/periodic orbit forward or backward in time.
- For each saddle, I plotted all four separatrices.

Then based on the stability computations from Q1, I found four parameter regions for β with qualitatively different phase portraits, these are restricted for $(S, I) \in [0, 100] \times [0, 100]$).





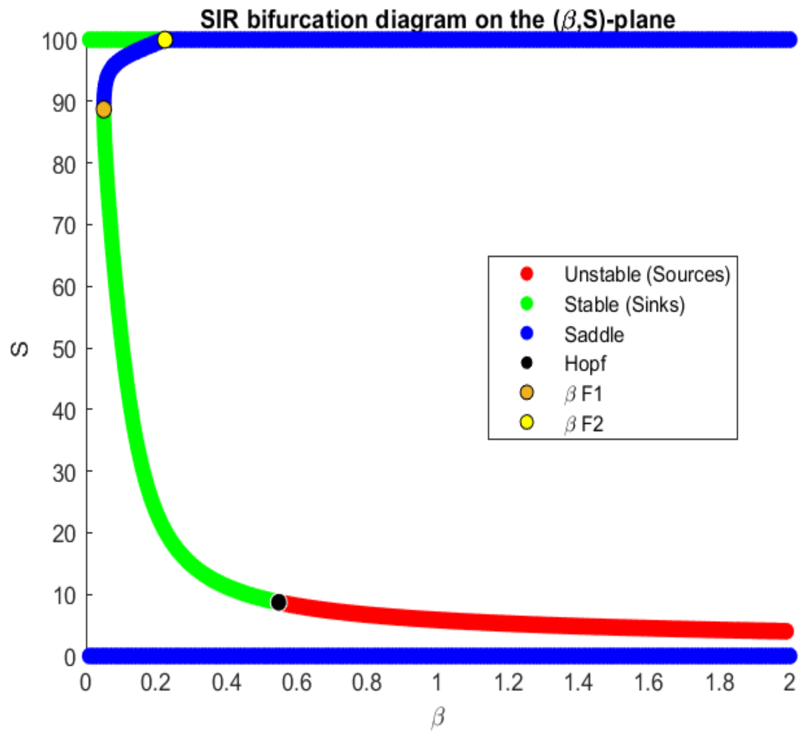




3 Question 3: Bifurcations of equilibria and the parameter plane

3.1 Part A

Furthermore, I calculated the following special equilibria and corresponding values of (β, S, I) to 4 significant digits. Then for each bifurcation, I coded a function res=fhopf(y) and res=ffold(y) that has the bifurcation point as a regular root, that I then solved with MySolve.
Below I have inserted my Hopf bifurcation and Fold & trans-critical into my bifurcation diagram from Question 1c. I computed the location of these points to 4 decimal paces and classified them as seen in the legend. I achieved this by solving a system of 5 equations in 5 unknowns, as given in the lecture notes and evident via my code.



With:
- $\beta_H \approx$ 0.5476
- $\beta_{F1} \approx$ 0.0508
- $\beta_{F2} \approx$ 0.2245
- $(S_h, I_h) \approx$ (8.7076 , 20.0078)
- $(S_{F1}, I_{F1}) \approx$ (88.6971 , 7.5234)
- $(S_{F1}, I_{F1}) \approx$ (99.9983 , 10.3416)

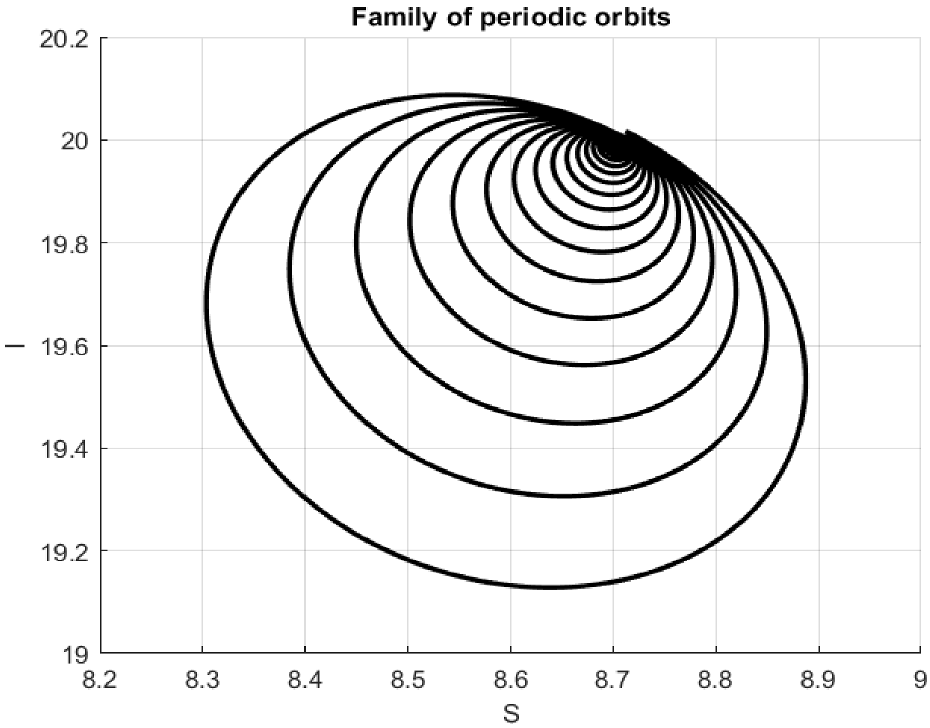
3.2 Part B

Here I explored how the model's dynamics change in the $(\beta, \lambda) - plane$ by tracking the fold and Hopf bifurcations in two parameters. Then I extended the defining systems fhopf and used it to locate the bifurcation points in part (a) to include one extra unknown (λ). Finally using MyTrackCurve with the extended defining systems to compute the Hopf curves under simultaneous variation of β and λ. Below are my investigation of the curves for $\beta \in [0, 2]$ and $\lambda \geq 0$ plotted in the $(\beta, \lambda) - plane.$ - Unfortunately, I was unable to finish Part B due to time constraints. However, I did attempt this as is evident from the code.

4 Question 4: Periodic orbits and their bifurcations

4.1 Part A : Periodic Orbits

Here I computed the family of periodic orbits that branch off from the hopf bifurcation by using my functions MyTrackCurve and MyIVP. To determine stability of the periodic orbits I needed to compute the eigenvalues of the linearised one-period map. 3



I inserted into the bifurcation diagram in the $(\beta, S) - plane$ the S-maxima and S-minima of the periodic orbits for each parameter value β, indicating their stability in the legends.
In addition, I plotted the phase portraits, including at least 10 periodic orbits approximately evenly spread along the branch, these are clearly labelled in the legends.
Then I plotted the period T of the periodic orbits along the family in the $(\beta, T) - plane$, again indicating the stability of the periodic orbits in the legend on my plot.

4.2 Part B : Fold of periodic orbits

- Unfortunately, I was unable to complete Part B due to time constraints.

Appendix

4.3 Code for 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 Question 1

```
spiking_model_710032265

%parameters for MyTrackCurve
x0 = [30;64 ; 2];
triv1 = [100;0;0];
triv2 = [0;0;0];
xtan = [0;0;-1]; % backwards in beta
xtan_triv = [0;0;1]; % forwards in beta
nmax = 20000;
S = 0.01;

% parameters for MySolve:
tol = 1e-10;
maxit = 100;


% preallocate arrays for speed
stable_data = [];
saddle_data = [];
unstable_data = [];
stable_triv_data1 = [];
saddle_triv_data1 = [];
unstable_triv_data1 = [];
stable_triv_data2 = [];
saddle_triv_data2 = [];
unstable_triv_data2 = [];


% f and df/dt for MyTrackCurve
 f = @(x) [r0*x(1).*(1-x(1)/K0) - x(3)*x(1).*x(2)./(1+alpha0*x(2)) ;...
        x(3)*x(1).*x(2)./(1+alpha0*x(2))-Theta0*x(2)-lambda0*x(2)./(1+epsilon0*x(2)
    )];

df = @(x) MyJacobian(f,x,1e-10);      %eqn,x,1e-10

% compute MyTrackCurve for each initial guess
xlist      =    MyTrackCurve(f,df,x0,xtan , [nmax , S]);
triv_list1 =    MyTrackCurve(f,df,triv1,xtan_triv , [nmax , S]);
triv_list2 =    MyTrackCurve(f,df,triv2,xtan_triv , [nmax , S]);
```

```
% for loop assigning the stability to the data
for j = 2:length(xlist)

% identify each S, I, beta from xlist and compile into x0
S = xlist(1,j);
I = xlist(2,j);
beta = xlist(3,j);
x0 = [S;I];


% create an f and df/dt for MySolve
f = @(x,b) sirs(x(1,:),x(2,:),beta,lambda0,r0,K0,alpha0,Theta0,epsilon0);
df = @(x) MyJacobian(f,x,1e-10);      %eqn,x,1e-10

% implement Mysolve
[x, converged, J] = MySolve(f, x0, df , tol, maxit);

% check the eigen values
e_val = eig(J);

  % label the point based on the eigen values
    % unstable
    if all(real(e_val) > 0) % warning to be ignored due to unknown sizes of the
    future data sets
              unstable_data = [unstable_data; S, I , beta]; %#ok<AGROW>
    % stable
    elseif all(real(e_val) < 0)
              stable_data = [stable_data; S, I , beta]; %#ok<AGROW>
    % saddle
    elseif any(real(e_val)>0) && any(real(e_val)<0)
              saddle_data = [saddle_data; S, I , beta]; %#ok<AGROW>
    end
end




% trivial data for [100;0]
for j = 2:length(triv_list1)

% identify each S, I, beta from triv_list1 and compile into x0
S = triv_list1(1,j);
I = triv_list1(2,j);
beta = triv_list1(3,j);
x0 = [S;I];


% create an f and df/dt for MySolve
f = @(x,b) sirs(x(1,:),x(2,:),beta,lambda0,r0,K0,alpha0,Theta0,epsilon0);
```

```
df = @(x) MyJacobian(f,x,1e-10);

% implement Mysolve
[x, converged, J] = MySolve(f, x0, df , tol, maxit);

% check the eigen values
e_val = eig(J);

  % label the point based on the eigen values
    % unstable
    if all(real(e_val) > 0) % warning to be ignored due to unknown sizes of the
    future data sets
             unstable_triv_data1 = [unstable_triv_data1; S, I , beta]; %#ok<AGROW>
    % stable
    elseif all(real(e_val) < 0)
             stable_triv_data1 = [stable_triv_data1; S, I , beta]; %#ok<AGROW>
    % saddle
    elseif any(real(e_val)>0) && any(real(e_val)<0)
             saddle_triv_data1 = [saddle_triv_data1; S, I , beta]; %#ok<AGROW>
    end
end






% trivial data for [0;0]
for j = 2:length(triv_list2)

% identify each S, I, beta from triv_list2 and compile into x0
S = triv_list2(1,j);
I = triv_list2(2,j);
beta = triv_list2(3,j);
x0 = [S;I];


% create an f and df for MySolve
f = @(x,b) sirs(x(1,:),x(2,:),beta,lambda0,r0,K0,alpha0,Theta0,epsilon0);
df = @(x) MyJacobian(f,x,1e-10);

%implement Mysolve
[x, converged, J] = MySolve(f, x0, df , tol, maxit);

% check the eigen values
e_val = eig(J);

  % label the point based on it's eigen values
    % unstable
    if all(real(e_val) > 0) % warning to be ignored due to unknown sizes of the
    future data sets
             unstable_triv_data2 = [unstable_triv_data2; S, I , beta]; %#ok<AGROW>
```

```
    % stable
    elseif all(real(e_val) < 0)
             stable_triv_data2 = [stable_triv_data2; S, I , beta]; %#ok<AGROW>
    % saddle
    elseif any(real(e_val)>0) && any(real(e_val)<0)
             saddle_triv_data2 = [saddle_triv_data2; S, I , beta]; %#ok<AGROW>
    end
end



% data from Question 3
hopf_point =    [8.708,       0.5476]; % [ S_h , Beta_h ]
beta_fold =     [0.0508,      0.2245]; % [ beta_fold_1 ,beta_fold_2 ]
S_fold =        [88.69,       99.99];  % [ S_fold_1 ,S_fold_2 ]


% plot the data
hold on
        scatter(stable_triv_data1(:, 3)   , stable_triv_data1(:, 1)     , 50, 'g',
    'filled');
        scatter(saddle_triv_data1(:, 3)   , saddle_triv_data1(:, 1)     , 50, 'b',
    'filled');
        scatter(saddle_triv_data2(:, 3)   , saddle_triv_data2(:, 1)     , 50, 'b',
    'filled');
        scatter(unstable_data(:, 3) , unstable_data(:, 1)   , 50, 'r', 'filled');
        scatter(stable_data(:, 3)   , stable_data(:, 1)     , 50, 'g', 'filled');
        scatter(saddle_data(:, 3)   , saddle_data(:, 1)     , 50, 'b', 'filled');
        scatter(hopf_point(2) , hopf_point(1) , 60 , 'k' , 'filled' , '
    MarkerEdgeColor','w')

        % plot the fold point
        scatter(beta_fold(1) , S_fold(1) , 50 , [0.9290 0.6940 0.1250]   , 'filled'
     , 'MarkerEdgeColor','k')
        scatter(beta_fold(2) , S_fold(2) , 50 , 'y' , 'filled', 'MarkerEdgeColor','
    k')


hold off

% plot details
titles = { '','','','Unstable (Sources)','Stable (Sinks)', 'Saddle' , 'Hopf' , '\
    beta F1' , '\beta F2' };%, 'Hopf Birfurcation'};
title('SIR bifurcation diagram on the (\beta,S)-plane')
xlabel('\beta')
ylabel('S')
legend(titles, 'Location', 'best');

% axis limits
ylim([0, 100]);
xlim([0,2]);
```

4.5 Code for Question 2

```
spiking_model_710032265

% Parameters for MyTrackCurve:
x0 = [30;64 ; 2];
xtan = [0;0;-1]; % backwards in beta
nmax = 20000;
S = 0.01;

% Parameters for MySolve:
tol = 1e-10;
maxit = 100;

% f and df/dt for MyTrackCurve
 f = @(x) [r0*x(1).*(1-x(1)/K0) - x(3)*x(1).*x(2)./(1+alpha0*x(2)) ;...
        x(3)*x(1).*x(2)./(1+alpha0*x(2))-Theta0*x(2)-lambda0*x(2)./(1+epsilon0*x(2)
    )];

df = @(x) MyJacobian(f,x,1e-10);

% compute MyTrackCurve for each initial guess
xlist = MyTrackCurve(f,df,x0,xtan , [nmax , S]);

% Define a range of values for I
I = linspace(0, 100, 100);


% unstable, stable , saddle  ,hopf [1496/1495]
% values from xlist

graph_num = [57 ; 2346 ; 18410 ; 1495];
%graph_num = [16815];

% iterate for each phase portrait
for j = 1:length(graph_num)

beta = xlist(3,graph_num(j));
S_eq = xlist(1,graph_num(j));
I_eq = xlist(2,graph_num(j));

x0 = [S_eq ; I_eq];


% S nullcline and I nullcline
S_nullcline = K0 - ((beta * K0 * I) ./ (r0 * (1 + alpha0 * I)));
I_nullcline = (1 + alpha0 * I) ./ beta .* ( Theta0 + lambda0 ./ (1 + epsilon0 * I))
    ;


% equations for MySolve
n = @(x) [r0*x(1).*(1-x(1)/K0) - beta*x(1).*x(2)./(1+alpha0*x(2)) ;...
```

```
        beta*x(1).*x(2)./(1+alpha0*x(2))-Theta0*x(2)-lambda0*x(2)./(1+epsilon0*x
    (2))];

dN = @(x) MyJacobian(n,x,1e-10);

% implement MySolve:
[x, converged, J] = MySolve(n, x0, dN , tol, maxit);

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

        % label the point based on it's eigen values
            if all(real(e_val) > 0)
               colour = 'r' ;
               stability = 'Unstable (source)';

            elseif all(real(e_val) < 0)
                colour = 'g' ;
                stability = 'Stable (sink)';

            elseif any(real(e_val)>0) && any(real(e_val)<0)
                colour = 'b' ;
                stability = 'Saddle';
            end




%plot the nullclines
nexttile
hold on
    plot(S_nullcline,I ,'r', 'LineWidth', 1.5); % S nullcline in red
    plot(I_nullcline,I, 'b', 'LineWidth', 1.5); % I nullcline in blue


% conditions for MyIVP
N = 1000;
T = 10;

% time dependant functionfor MyIVP
f = @(t,x) [r0*x(1).*(1-x(1)/K0) - beta*x(1).*x(2)./(1+alpha0*x(2)) ;...
        beta*x(1).*x(2)./(1+alpha0*x(2))-Theta0*x(2)-lambda0*x(2)./(1+epsilon0*x(2)
    )];

if j == 1 % unstable trajectory

    % conditions for MyIVP
    tspan = [1;T];
    x0 = [4;5.6];

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

```
    % plot the data
    plot(xt(:,1),xt(:,2) , 'k' , 'LineWidth',1.2)
    scatter(S_eq,I_eq , colour ,'filled', 'MarkerEdgeColor','k')
    title('Nullcline Plot for Unstable Bifurcation');
    legend('S Nullcline', 'I Nullcline' , 'Trajectory' ,stability, 'Location', '
    northeast');

end

if j == 2 % stable trajectory

    % conditions for MyIVP
    N = 10000;
    T =  100;
    tspan = [T;1];
    x0 = [11.2;28.5];

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

    % plot the data
    plot(xt(:,1),xt(:,2) , 'k' , 'LineWidth',1.2)
    scatter(S_eq,I_eq , colour ,'filled', 'MarkerEdgeColor','k')
    title('Nullcline Plot for Stable Bifurcation');
    legend('S Nullcline', 'I Nullcline' , 'Trajectory' ,stability, 'Location', '
    northeast');

end

if j == 3 %saddle point

    % conditions for MyIVP
    tspan = [T;1];
    x0_one = [94.7;8.6];
    [~,~, xt_one] = MyIVP(f, x0_one, tspan, N);

    % conditions for MyIVP
    tspan = [T;1];
    x0_two =  [94.8;8.6];
    [~,~, xt_two] = MyIVP(f, x0_two, tspan, N);

    % conditions for MyIVP
    tspan = [1;T];
    x0_three =  [94.8;8.5];
    [~,~, xt_three] = MyIVP(f, x0_three, tspan, N);

    % conditions for MyIVP
    x0_four = [94.7;9];
    [xend,t, xt_four] = MyIVP(f, x0_four, tspan, N);

    %plot the separatricies:
    plot(xt_one(:,1) , xt_one(:,2) ,'k' , 'LineWidth',1.5)
    plot(xt_two(:,1) , xt_two(:,2) , 'k' , 'LineWidth',1.5)
```

```
    plot(xt_three(:,1) , xt_three(:,2) , 'k' , 'LineWidth',1.5)
    plot(xt_four(:,1) , xt_four(:,2) , 'k' , 'LineWidth',1.5)
    scatter(S_eq,I_eq , colour ,'filled', 'MarkerEdgeColor','k')
    scatter(72.255 , 71.672, 'g' ,'filled' ,  'MarkerEdgeColor','k')
    title('Nullcline Plot for Saddle Bifurcation');
    legend('S Nullcline', 'I Nullcline' , '','','Four Separatrices.','' ,stability
    , 'Stable(sink)', 'Location', 'best');
end

if j == 4 % periodic orbit

    % conditions for MyIVP
    N = 100;
    T = 2.18;
    tspan = [T;1];
    x0 = [10;30];

    % Hopf point identified in Q3
    beta =   0.547577513629019;
    S_eq =   8.707616812821446;
    I_eq =   20.007748893687562;

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

    % plot the data
    plot(xt(:,1),xt(:,2) , 'k' , 'LineWidth',1.4)
    scatter(S_eq,I_eq , 'k' ,'filled', 'MarkerEdgeColor','k')
    title('Nullcline Plot for Periodic Orbit');
    legend('S Nullcline', 'I Nullcline' , 'Trajectory' ,'Hopf', 'Location', '
    northeast');

end
hold off

% Label the axes and add a legend
xlabel('S');
ylabel('I');

% Set axis limits
xlim([0, 100]);
ylim([0, 100]);
grid on;
end
```

4.6 Code for Question 3

```
spiking_model_710032265

% FINDING THE HOPF:

% parameters for MySolve
maxit=250;
tolerance=1e-12;
```

```
h1=1e-3;

% initial guess
x0 = [10;25;1];

% equations for MySolve
f =@(x) fhopf(x);
df=@(x)MyJacobian(f,x,h1);

% implement MySolve
[x_hopf,conv1]=MySolve(f,x0,df,tolerance,maxit);

% assign hopf data points
S_h = x_hopf(1);
I_h = x_hopf(2);
Beta_h = x_hopf(3);

% % Display Hopf point
format short
disp('hopf:')
disp(x_hopf)
disp('converged = ')
disp(conv1)

%-------------------------------------

% FINDING THE FOLD:

% parameters for MySolve
maxit1=800;
tolerance1=1e2;
h2 = 1e-10;
maxit2=600;
tolerance2=1e-2;
h22 = 1e-1;

% initial guesses
x1 = [90;10;0.05];
x2 = [80;50;0.2];

% equations for MySolve
g = @(x)ffold(x);
dg = @(x)MyJacobian(g,x,h2);

% implement MySolve
[x_fold1,conv2] = MySolve(g,x1,dg,tolerance1,maxit1);
[x_fold2,conv3] = MySolve(g,x2,dg,tolerance2,maxit2);

%remove any Imaginary Values
x_fold2 = real(x_fold2);

% assign fold data points
```

```
S_fold_1 = x_fold1(1);
beta_fold_1 =  x_fold1(3);

S_fold_2 = x_fold2(1);
beta_fold_2 =  x_fold2(3);


% Display Fold point
format short
disp('fold beta values:')
disp(beta_fold_1)
disp(' ')
disp(beta_fold_2)
```

4.7 Code for Question 4

```
spiking_model_710032265


% define values for the hopf point
 beta = 0.5476;
x0 = [8.708 ;    20.0078];


% parameters used for MyIVP
N = 1000;
T = 2.15;
tspan = [1;T];

% iterations of periodic orbits
for j = 2:.1:10


% define a 'nudge' value
nug = 1*10^-j;

% appply 'nudge' to beta
beta = 0.5476 + nug;

% definet the equation for the orbit
f = @(t,x) [r0*x(1).*(1-x(1)/K0) - beta*x(1).*x(2)./(1+alpha0*x(2)) ;...
        beta*x(1).*x(2)./(1+alpha0*x(2))-Theta0*x(2)-lambda0*x(2)./(1+epsilon0*x(2)
    )];

% implement MyIVP
[xend, t,xt] = MyIVP(f, x0, tspan, N);

% Plot the periodic orbits
hold on
plot(xt(:,1),xt(:,2) , 'k' , 'LineWidth',2)

end
```

```
% Label the axes
xlabel('S');
ylabel('I');
xlim([8.2,9])
ylim([19,20.2])
title('Family of periodic orbits')
grid on;
```