COMPSCI 4X03

Assignment 4

Mingzhe Wang

McMaster University

December 6, 2021

Problem 1

netbp2

```
function cost = netbp2(neurons, data, labels, niter, lr, file)

% Initialize weights and biases
rng(5000);
W2 = 0.5*randn(neurons(1),2); W3 = 0.5*randn(neurons(2),neurons(1)); W4
    = 0.5*randn(2,neurons(2));
b2 = 0.5*randn(neurons(1),1); b3 = 0.5*randn(neurons(2),1); b4 = 0.5*
   randn(2,1);

cost = zeros(niter,1); % value of cost function at each iteration
[~, n] = size(data);
for counter = 1:niter
    k = randi(n);
    x = data(:, k);
    % Forward pass
    a2 = activate(x,W2,b2);
    a3 = activate(a2,W3,b3);
    a4 = activate(a3,W4,b4);
    % Backward pass
    delta4 = a4.*(1-a4).*(a4-labels(:,k));
    delta3 = a3.*(1-a3).*(W4'*delta4);
    delta2 = a2.*(1-a2).*(W3'*delta3);
    % Gradient step
    W2 = W2 - lr*delta2*x';
    W3 = W3 - lr*delta3*a2';
    W4 = W4 - lr*delta4*a3';
    b2 = b2 - lr*delta2;
    b3 = b3 - lr*delta3;
    b4 = b4 - lr*delta4;
    % Monitor progress
    costval = costfunc(W2,W3,W4,b2,b3,b4) ;  % display cost to screen
```

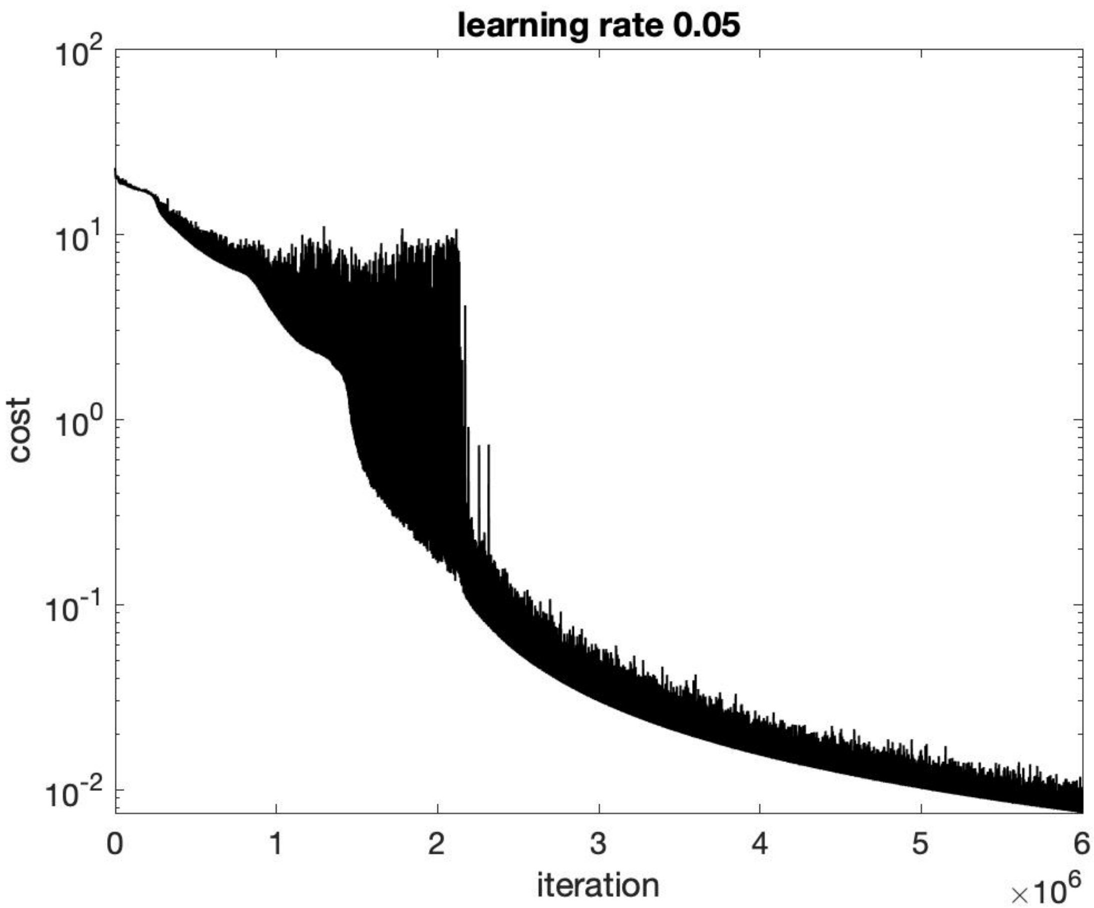
```
    cost(counter) = costval;
    % fprintf("i=%d  %e\n", counter, newcost);
end

% save file
save(file, 'W2', 'W3', 'W4', 'b2', 'b3', 'b4');

% nested cost function
    function costval = costfunc(W2,W3,W4,b2,b3,b4)
        costvec = zeros(n,1);
        for i = 1:n
            x = data(:, i);
            a2 = activate(x,W2,b2);
            a3 = activate(a2,W3,b3);
            a4 = activate(a3,W4,b4);
            costvec(i) = norm(labels(:,i) - a4,2);
        end
        costval = norm(costvec,2)^2;
    end % of nested function

end
```

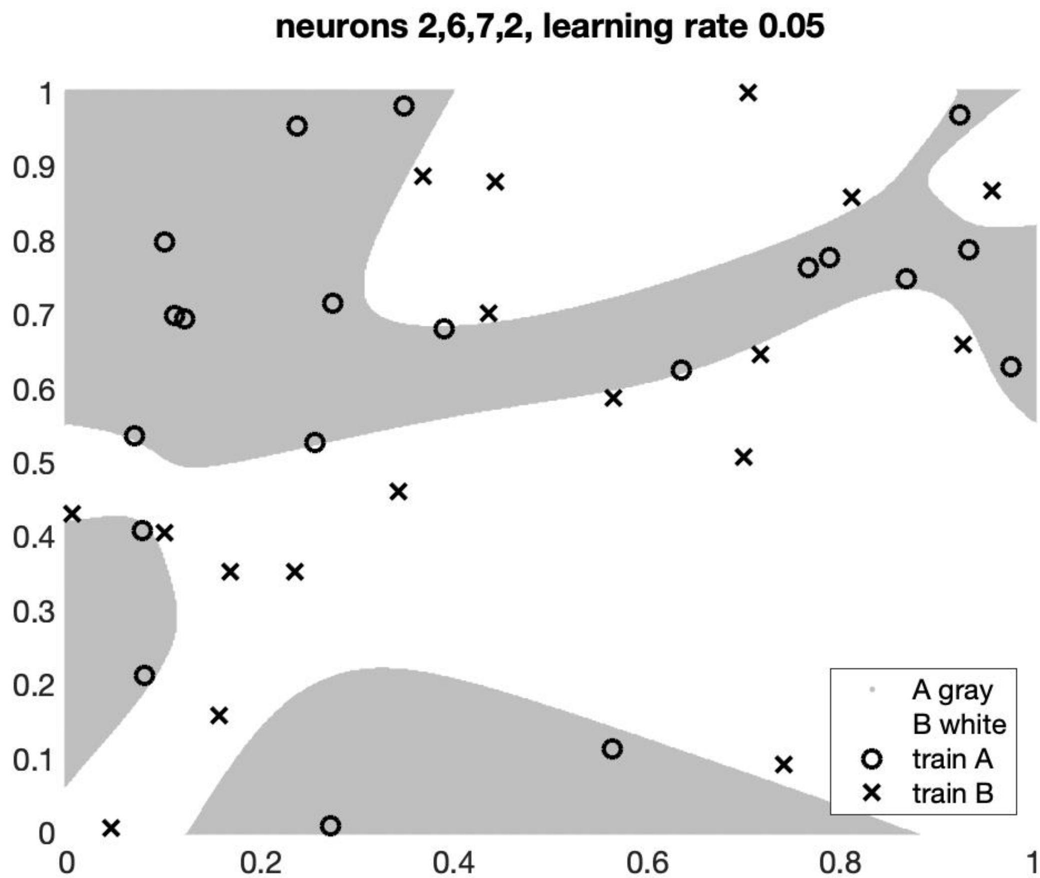
classifypoints

```
function category = classifypoints(file, points)

% load variables
load(file, 'W2', 'W3', 'W4', 'b2', 'b3', 'b4');
% get numbers of points
[~, n] = size(points);

% initial category with all -1, which means error.
category = -ones(n, 1);
for i = 1:n
    x = points(:, i);
    % Forward pass
    a2 = activate(x,W2,b2);
    a3 = activate(a2,W3,b3);
    a4 = activate(a3,W4,b4);
    % category based on a4's value
    if a4(1) >= a4(2)
        category(i) = 1;
    else
        category(i) = 0;
    end
end
```

```
end
```

two plots



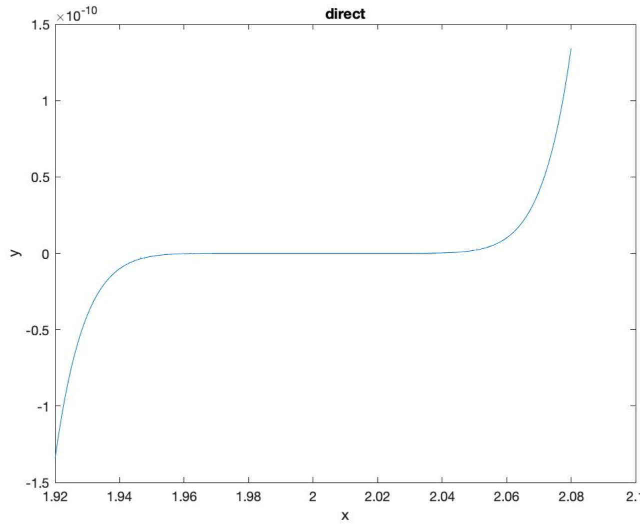


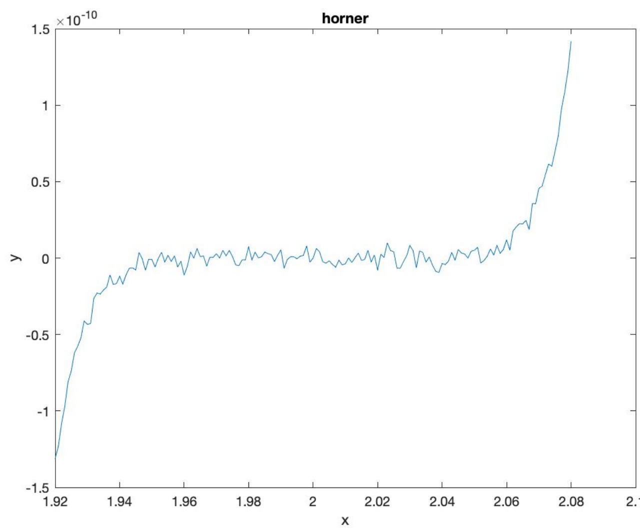
Summarize

- The more neurons the hidden layers have, the more accuracy of this output of this system.
- When iteration number = 1e5, there is no useful output.
- It seems the accuracy stop increasing on some thershold of some parameter.
- The time consumed by this module highly depends on the iterations steps.

Problem 2

(a)





The figure evaluated by Horner's method have more small waves, which means the evaluation using Horner's method can preserve more accuracy.

(b)

```
root = bisection(g, 1.92, 2.08, 1e-6);
fprintf("root is %6.10f\n", root);
```

output is:
root is 2.0000000000. r = 1.999999.

(c)

Because we loose some digits of accuracy in the intermediate steps, so the bisection function could iterate forever, then there will be no such a root.

(d)

```
r1 = fsolve(z, 1.9)
r2 = fsolve(f, 1.9)
```

output is:
r1 = 1.9000 r2 = 1.9000

Problem 3

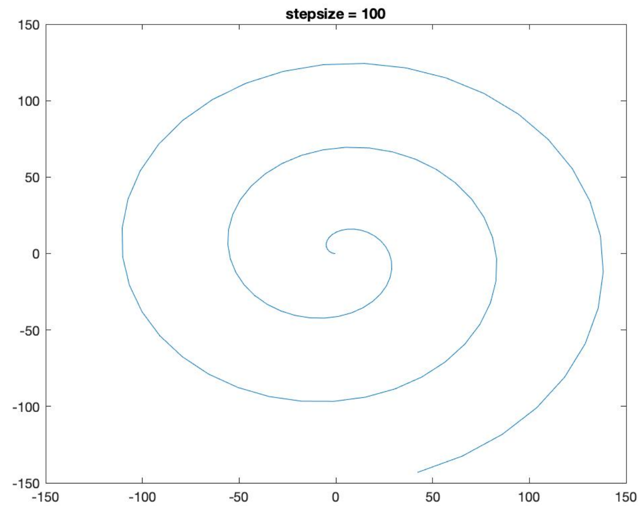
For system(a): Newton method: x1 = 5.000000 x2 = 4.000000
For system(a): Matlab solver: x1 = 11.412779 x2 = -0.896805
For this system, Newton's result is NOT correct, the reason is "Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 4.344298e-18. ". Because the implementations uses "/" divide to solve the system, if the matrix is singular, then the matrix is too dependent, then the inverse could be bad or not existed.

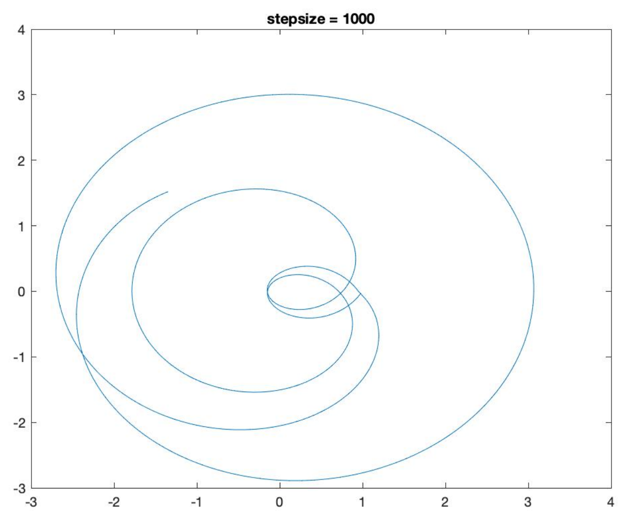
For system(b): Newton method: x1 = 1.666667 x2 = -0.666667 x3 = 1.333333
For system(b): Matlab solver: x1 = 1.000000 x2 = 0.000000 x3 = 2.000000
For this system, Newton's result is NOT correct, the reason is "Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 4.344298e-18. ". Because the implementations uses "/" divide to solve the system, if the matrix is singular, then the matrix is too dependent, then the inverse could be bad or not existed.

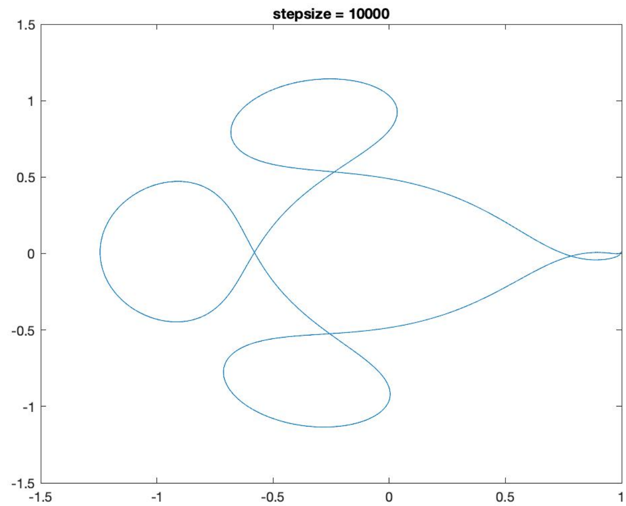
For system(c): Newton method: x1 = NaN x2 = NaN x3 = NaN x4 = NaN
For system(c): Matlab solver: x1 = -0.002673 x2 = 0.000267 x3 = 0.000407 x4 = 0.000407
For this system, Newton's method cannot provide a solution due to the matrix is singular. Because the implementations uses "/" divide to solve the system, if the matrix is singular, then the matrix is too dependent, then the inverse could be bad or not existed.

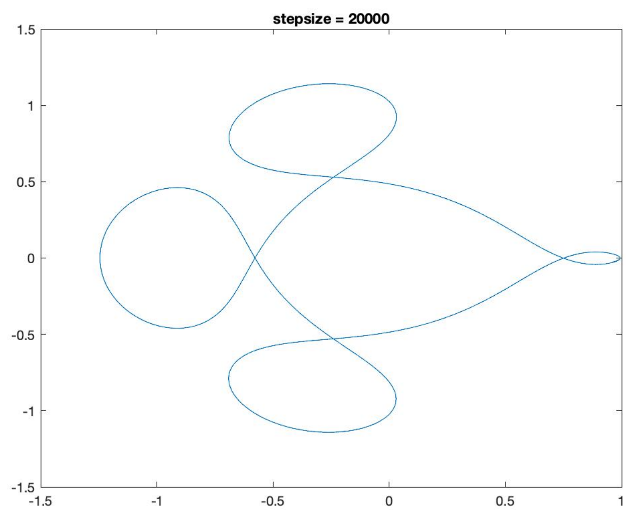
For system(d): Newton method: x1 = NaN x2 = NaN
For system(d): Matlab solver: x1 = 0.010048 x2 = 0.010048
For this system, Newton's method cannot provide a solution due to the matrix is singular. Because the implementations uses "/" divide to solve the system, if the matrix is singular, then the matrix is too dependent, then the inverse could be bad or not existed.

Problem 4


stepsize = 100
150
100
50
0
-50
-100
-150
-150
-100
-50
0
50
100
150


stepsize = 1000
4
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3
4





How many uniform steps are needed before the orbit appears to be qualitatively correct?
Bases on the experiment, "stepsize = 10000" is needed before the orbit appears to be qualitatively correct.

Problem 5

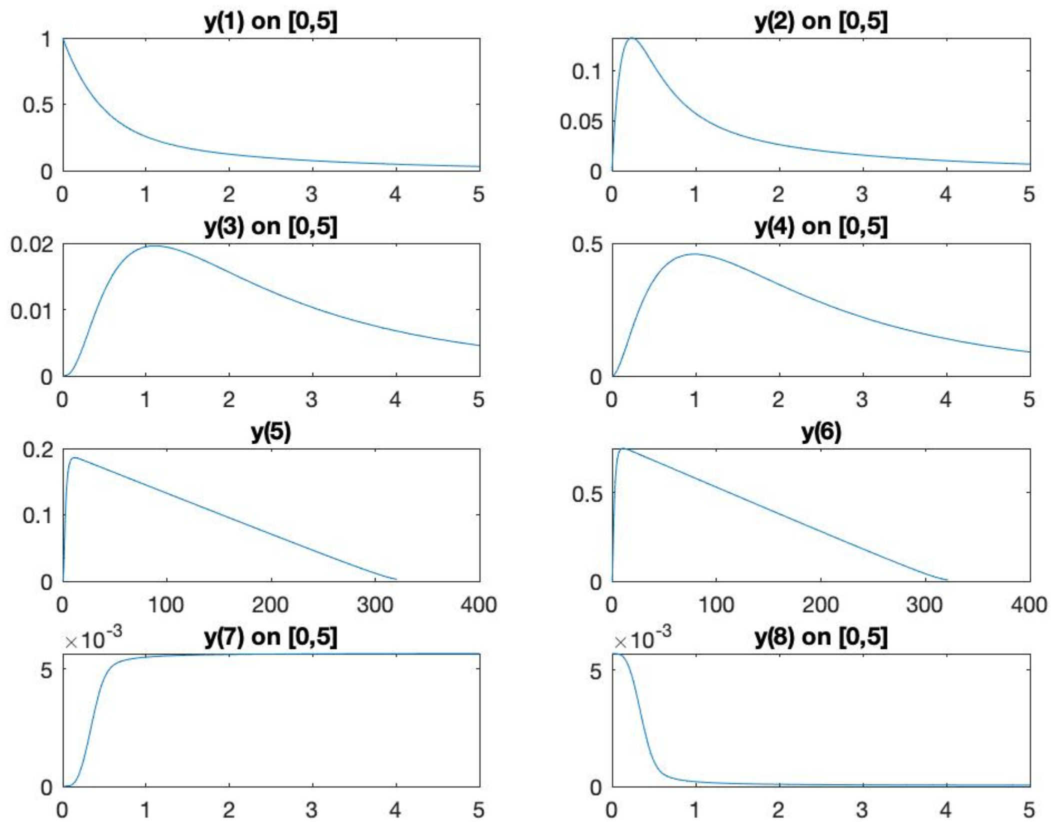
		number of		
solver	CPU time	steps	failed steps	function evaluations
ode23	8.8339	1.2156e+06	0.0000e+00	3.6467e+06
ode45	0.5089	3.9039e+04	4.7000e+01	2.3452e+05
ode78	0.4882	9.9650e+03	2.9300e+02	1.7292e+05

ode89	0.2598	3.6840e+03	5.5000e+01	7.8189e+04
ode113	0.3613	1.7789e+04	2.2600e+02	3.5805e+04

ode89 is the most efficient solver on this problem.

Problem 6

a



b

Note: the table' width is too large, so I include the screenshot here.

		number of				
solver	CPU time	steps	failed steps	function evaluations	LU decompositions	nonlinear solves
ode23s	0.0702	3.0300e+02	0.0000e+00	3.3360e+03	3.0300e+02	9.0900e+02
ode15s	0.0114	2.0000e+02	2.2000e+01	4.3500e+02	5.5000e+01	3.7000e+02
ode45	0.1164	1.0381e+04	6.4100e+02	6.6133e+04	0.0000e+00	---------

c

For solving stiff and non-linear differential equations, the ode15s solver is the most efficient one.

Problem 7

(a)

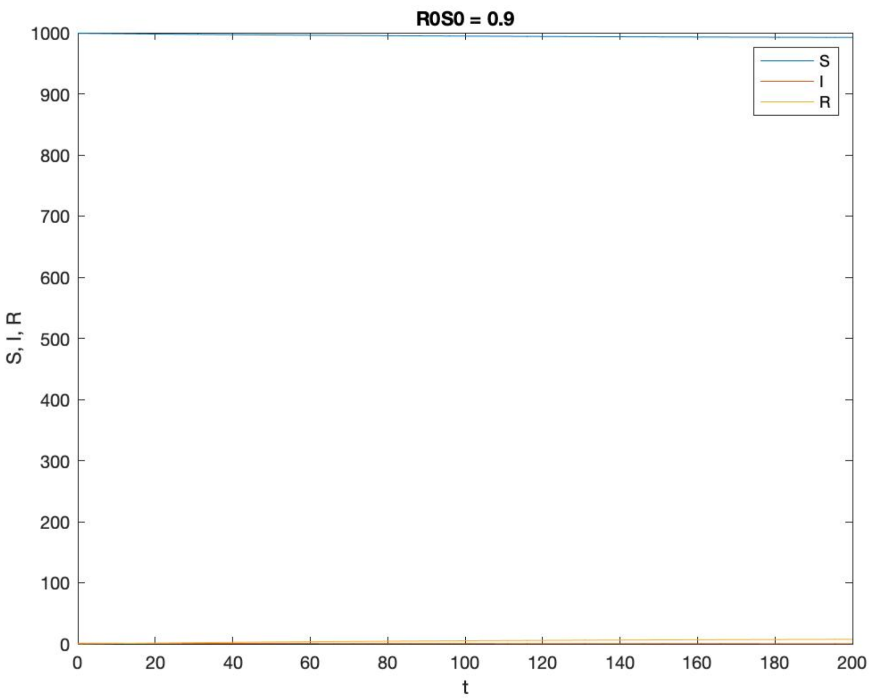
translate the data to a time series with t is the time array and xyz is the matrix constituted of [x, y, z].
use matlab's time series function to interpolate this time series by a day.
while T ¡ maxtime, check $(x[T]-x[0])^2+(y[T]-y[0])^2+(z[T]-z[0])^2 <= tolerant.$, then increment T by 10 days.
return the minimum T.

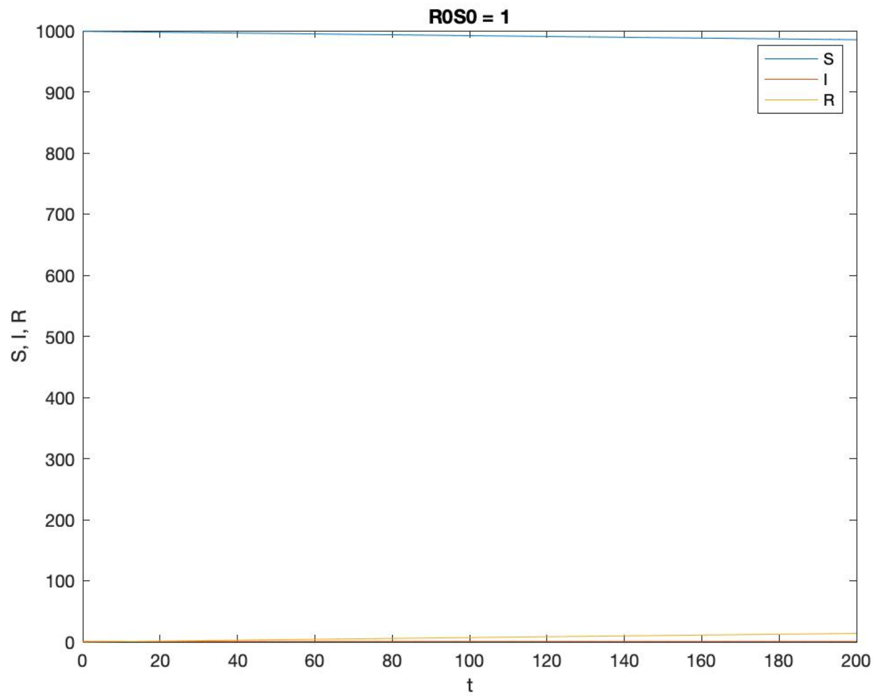
```
function period = findPeriod(t, x, y, z)
% Using matlab's timeseries to interpolate the data
xyz = [x, y, z];
ts = timeseries(xyz, t, 'Name', 't-xyz');
ts.TimeInfo.Units = '100 Days';
t_interp = ts.Time(1) : 1/100 : ts.Time(end); % interpolate by a day
tsInt = resample(ts, t_interp);
xyzInt = tsInt.Data;

% test: plot to visualize the interpolation
% figure()
% plot3(xyz(:,1),xyz(:,2),xyz(:,3),'bo','LineWidth',2,'DisplayName','
   Original (2-min)')
% hold on
% plot3(xyzInt(:,1),xyzInt(:,2),xyzInt(:,3),'r.','MarkerSize',3,'
   DisplayName','Interpolated (1-sec)')
% grid on
% legend('Location','SouthOutside');

TT = array2timetable(tsInt.Data,...
     'RowTimes',days(tsInt.Time*100), ...
     'VariableNames', {'x','y','z'});
% head(TT)
% transform timetable to matrix
tt = [days(TT.Time), TT.x, TT.y, TT.z];

[n, ~] = size(tt);
% finding the minimum period
stepsize = 30;
base_case = tt(1,:);
for T = stepsize : stepsize : n - 1
```

```
    cur_case = tt(1 + T, :);
    if (cur_base(2)^2 - base_case(2)^2 = )
end


end
```

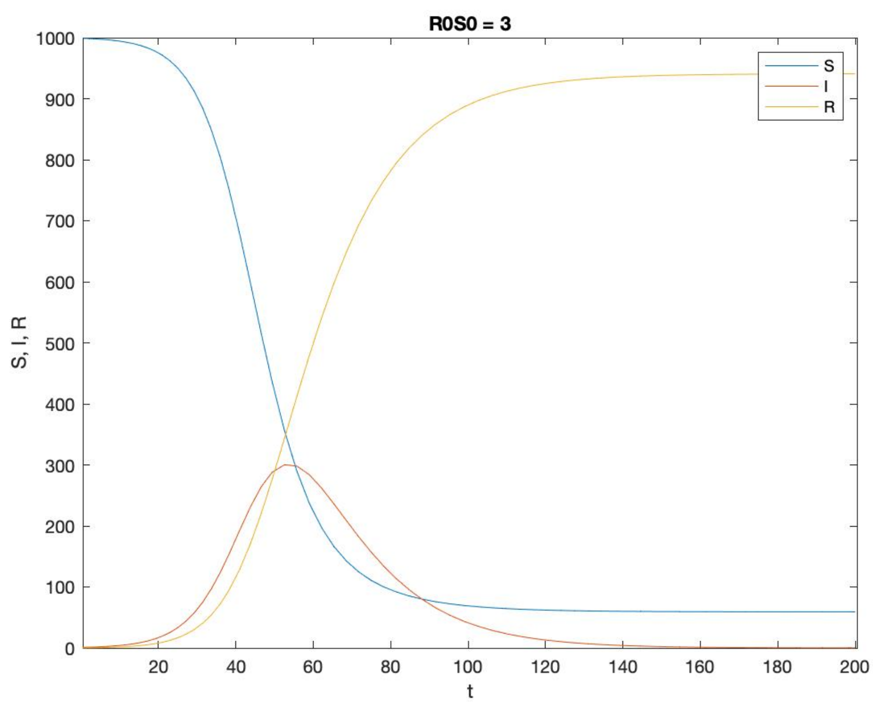
(b)

Problem 8

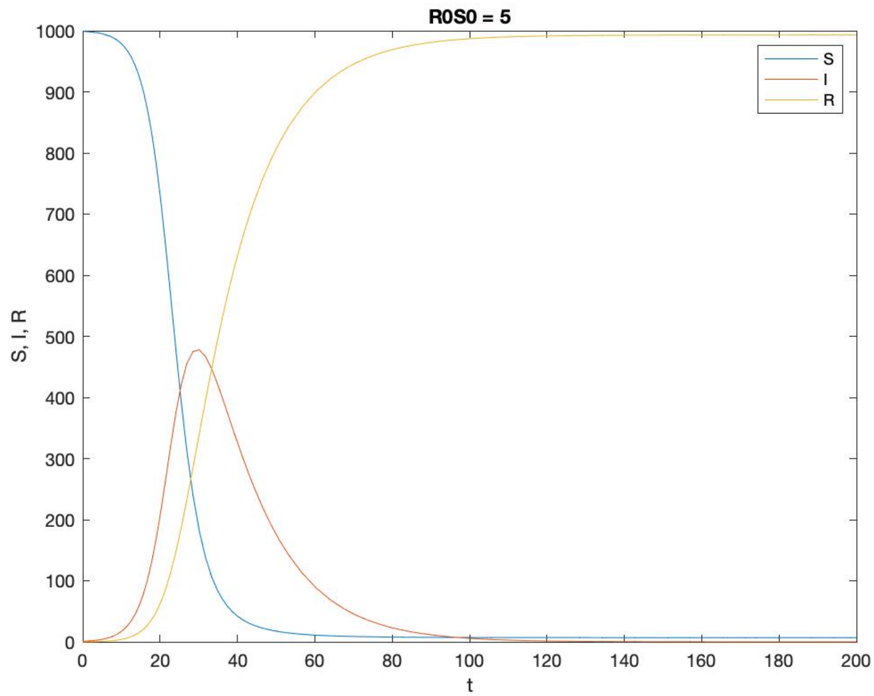




R0S0 = 1
S
I
R
S, I, R
t



R0S0 = 3
S
I
R
S, I, R
t


R0S0 = 5
S
I
R
S, I, R
t
0
100
200
300
400
500
600
700
800
900
1000
20
40
60
80
100
120
140
160
180
200