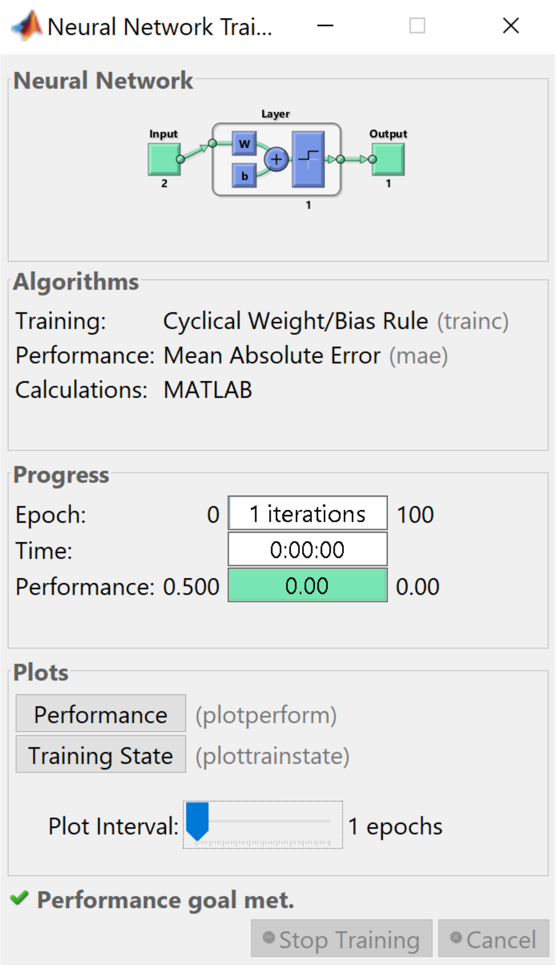
Problem – 1:

a. MATLAB Code

```
function neuralnet1
p(:,1)=[5;0];p(:,2)=[4;-1];p(:,3)=[6;0];p(:,4)=[5;-1];
p(:,5)=[2;-1];p(:,6)=[1;-2];p(:,7)=[2;-2];p(:,8)=[1;-3];

t(:,1)=[0];t(:,2)=[0];t(:,3)=[0];t(:,4)=[0];
t(:,5)=[1];t(:,6)=[1];t(:,7)=[1];t(:,8)=[1];

net1=newp(minmax(p),1);
out=sim(net1,p)
weight=net1.IW{1,1}
bias=net1.b{1}

net1.trainParam.epochs=100;
net1=train(net1,p,t);

out=sim(net1,p)
weight=net1.IW{1,1}
bias=net1.b{1}

figure
plotpv(p,t)
plotpc(weight,bias)
end
```



Neural Network

Algorithms

Training: Cyclical Weight/Bias Rule (trainc)
Performance: Mean Absolute Error (mae)
Calculations: MATLAB

Progress

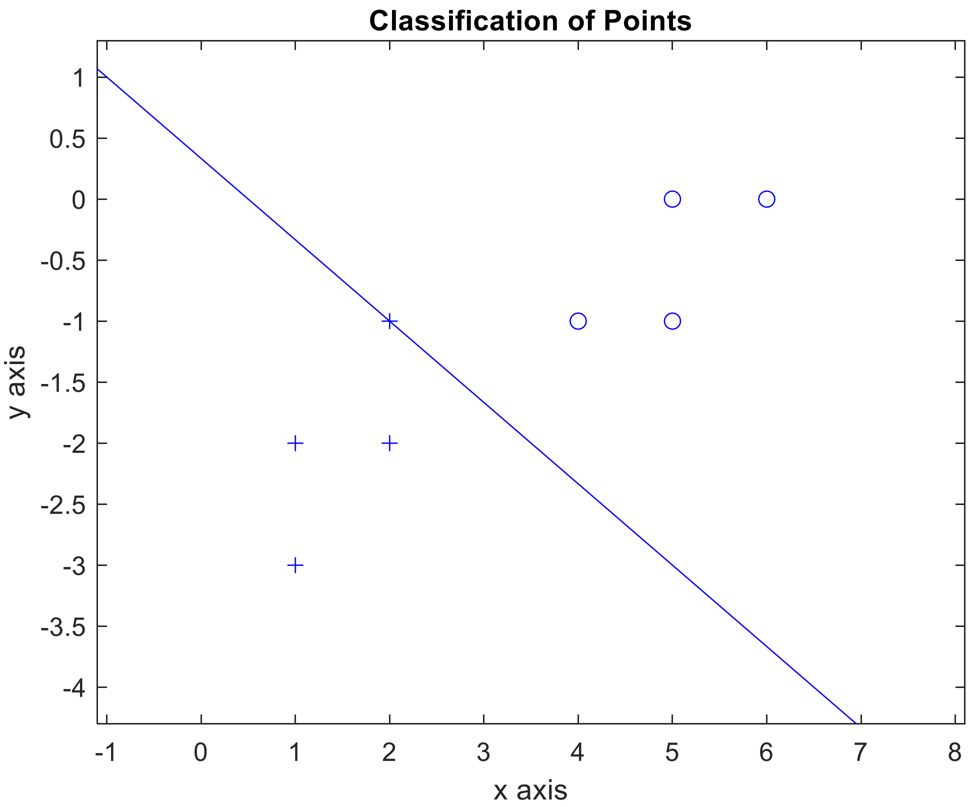
Epoch: 0 | 1 iterations | 100
Time: | 0:00:00 |
Performance: 0.500 | 0.00 | 0.00

Plots

(plotperform)
(plottrainstate)

Plot Interval: 1 epochs

✓ **Performance goal met.**



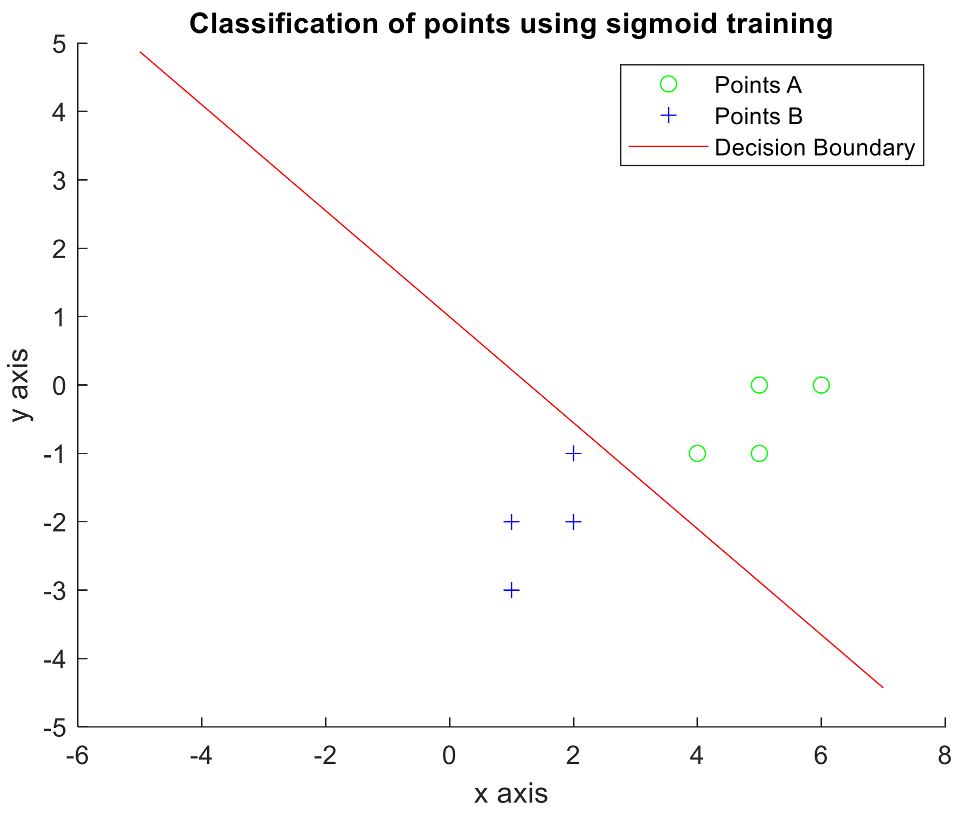
Classification of Points

b. Done using two similar methods

Method 1 MATLAB code:

```
function neuranlet2
p(:,1)=[5;0];p(:,2)=[4;-1];p(:,3)=[6;0];p(:,4)=[5;-1];
p(:,5)=[2;-1];p(:,6)=[1;-2];p(:,7)=[2;-2];p(:,8)=[1;-3];

t(:,1)=[0];t(:,2)=[0];t(:,3)=[0];t(:,4)=[0];
t(:,5)=[1];t(:,6)=[1];t(:,7)=[1];t(:,8)=[1];

weight=rand(3,1);
thresh=1;
error=ones(8,1);
e=0.5;
while norm(error)>1e-5
    for k=1:8
        P=[p(:,k);thresh];
        tnet=hardlim(weight'*P);
        error(k)=t(:,k)-tnet;
        if abs(error(k))>1e-6
            weight=weight-sign(weight'*P)*e*P;
        end
    end
end
it=-5:0.1:7;
out=(-weight(1)*it-weight(3))/weight(2);
hold on
plot(p(1,1:4),p(2,1:4),'go',p(1,5:8),p(2,5:8),'b+')
plot(it,out,'r')
end
```

Note: The slope of the decision boundary can be altered using the it values

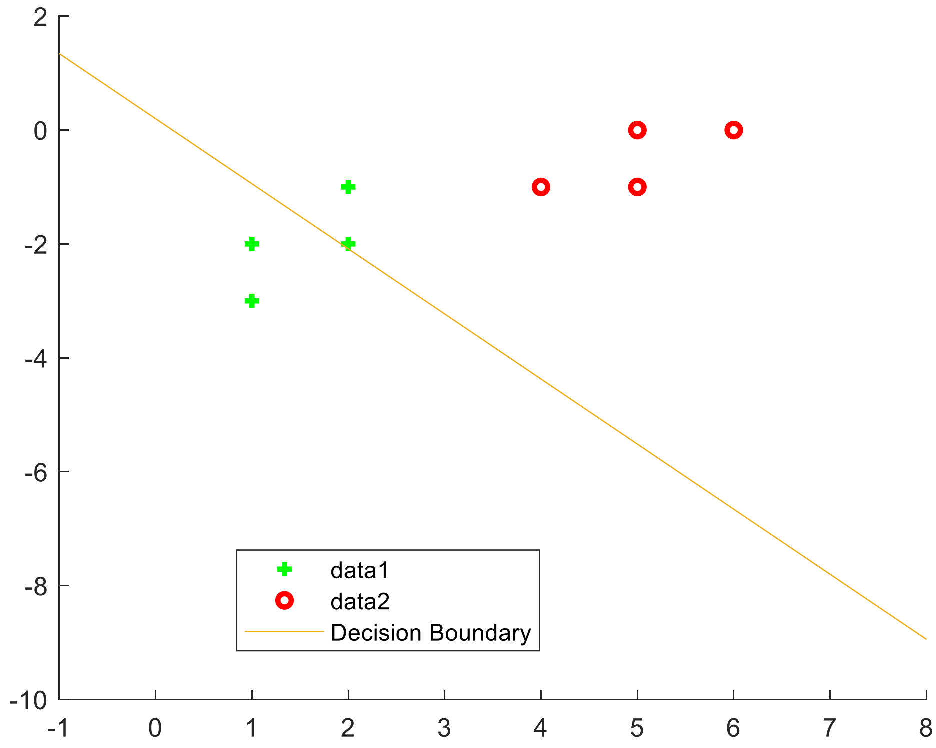


Method 2: MATLAB Code

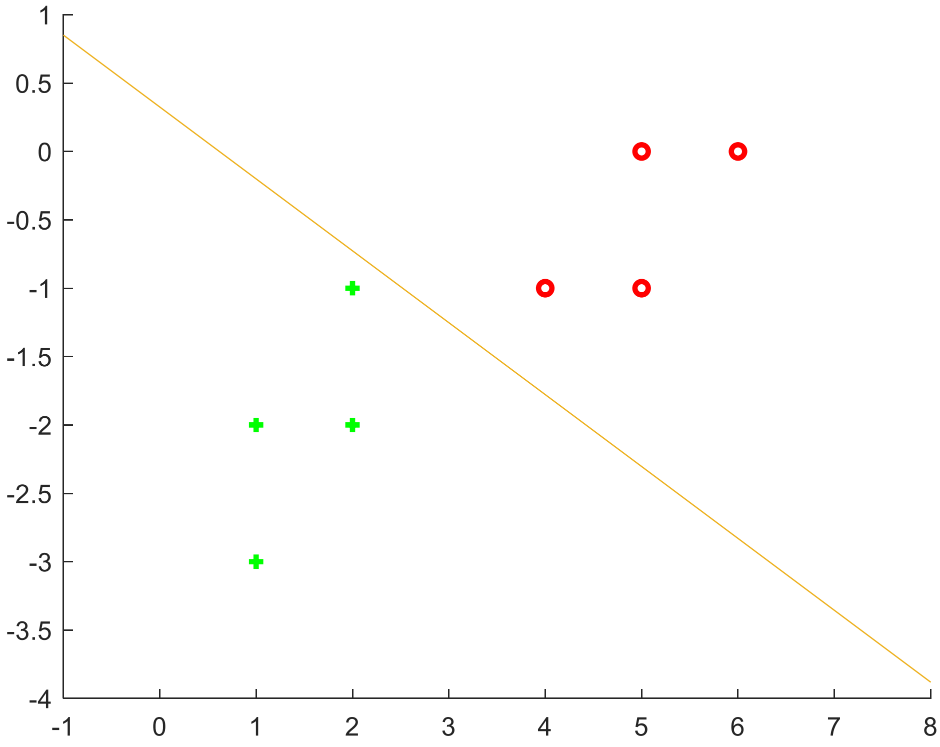
```
function neuralnet5

p=[5 4 6 5 2 1 2 1;0 -1 0 -1 -1 -2 -2 -3];

D=[5 4 6 5 2 1 2 1;0 -1 0 -1 -1 -2 -2 -3]';

target=[0 0 0 0 1 1 1 1]';

[m, n] = size(D);

D = [ones(m, 1) D];

Vin = zeros(n + 1, 1);

[cost, gradient] = costFn(Vin, D, target); %costFn defined in Appendix

%Initial cost and gradient

fprintf('Initial Cost %f', cost);

fprintf('\n Initial Gradient \n');

fprintf(' %f \n', gradient);

%Using fminunc for gradient descent

options = optimset('GradObj', 'on', 'MaxIter', 10);

[V, cost] = fminunc(@(t)(costFn(t, D, target)), Vin, options);
```

```
% Final cost
fprintf('Cost at final weights %f\n', cost);
fprintf('Weights and threshold: \n');
fprintf(' %f \n', V);
function plotDB(Vw, D, target)
    function plotD(x,y)
        figure;
    end
plotD(D(:,1:2), target);
hold on
plot(p(1,5:8),p(2,5:8),'g+','LineWidth', 2,'MarkerSize', 5)
plot(p(1,1:4),p(2,1:4),'ro', 'Linewidth',2,'MarkerSize', 5)
if size(D, 2) <= 3
%evaluating and plotting the boundary line
    xa = [min(D(:,2))-2,  max(D(:,2))+2];
    ya = (-1./Vw(3)).*(Vw(2).*xa + Vw(1));
    plot(xa, ya)
else
    u = linspace(-10, 0.1, 200);
    v = linspace(-10, 0.1, 200);
    z = zeros(length(u), length(v));
    for i = 1:length(u)
        for j = 1:length(v)
            z(i,j) = mapFeature(u(i), v(j))*Vw;
        end
    end
    z = z';
    contour(u, v, z, [0, 0], 'LineWidth', 2)
end
hold off
end
plotDB(V, D, target);
end
```

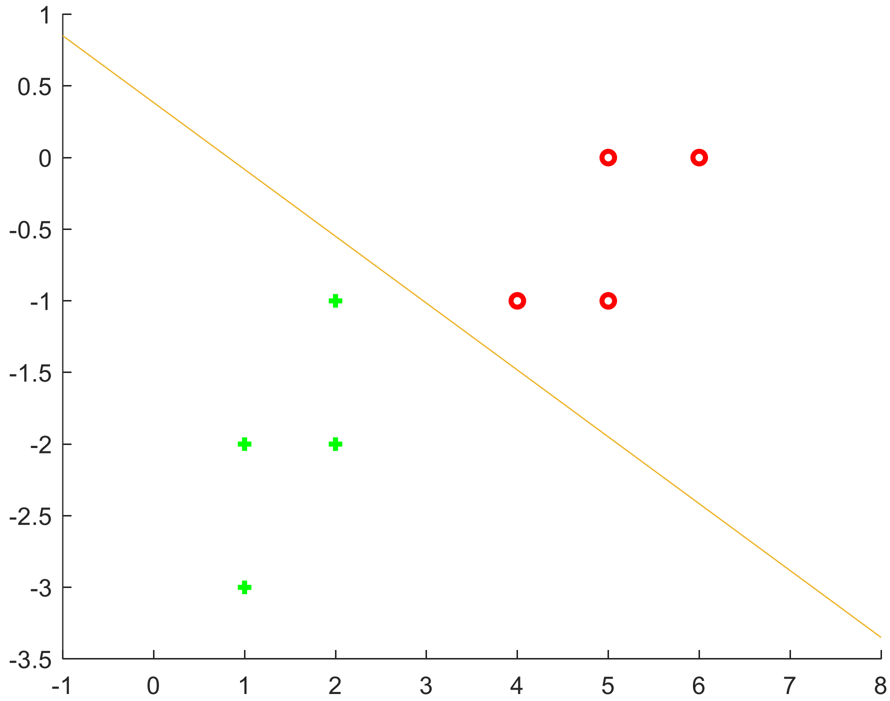
Plot of Decision Boundary after 1 iteration of fminunc



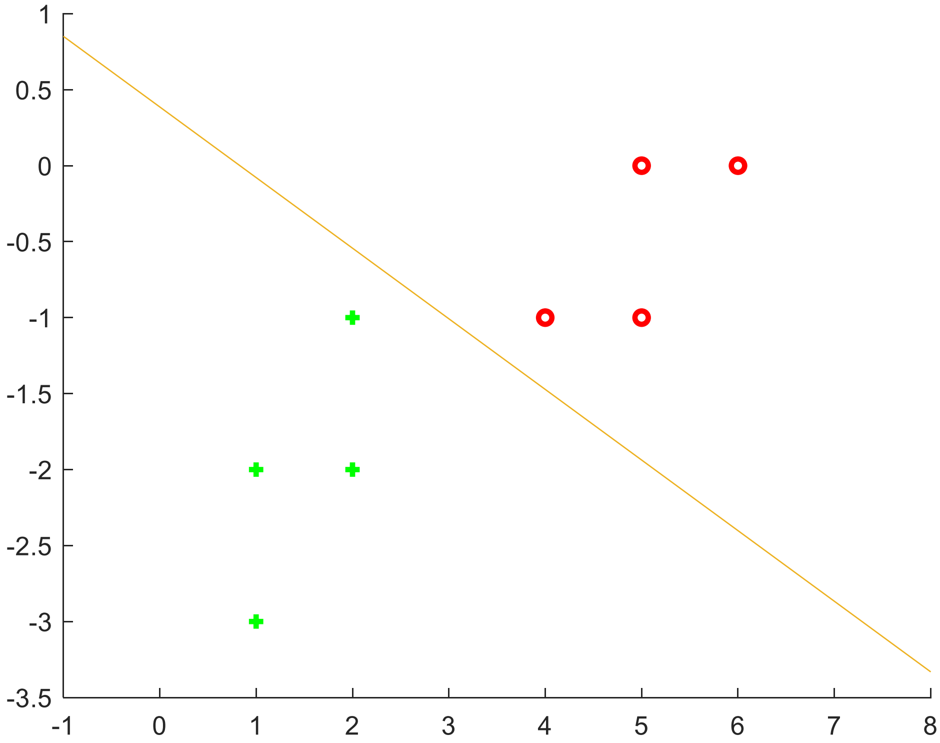
Plot of Decision Boundary after 3 iterations of fminunc



Plot of Decision Boundary after 10 iterations of fminunc



Plot after 15 iterations, mimima reached

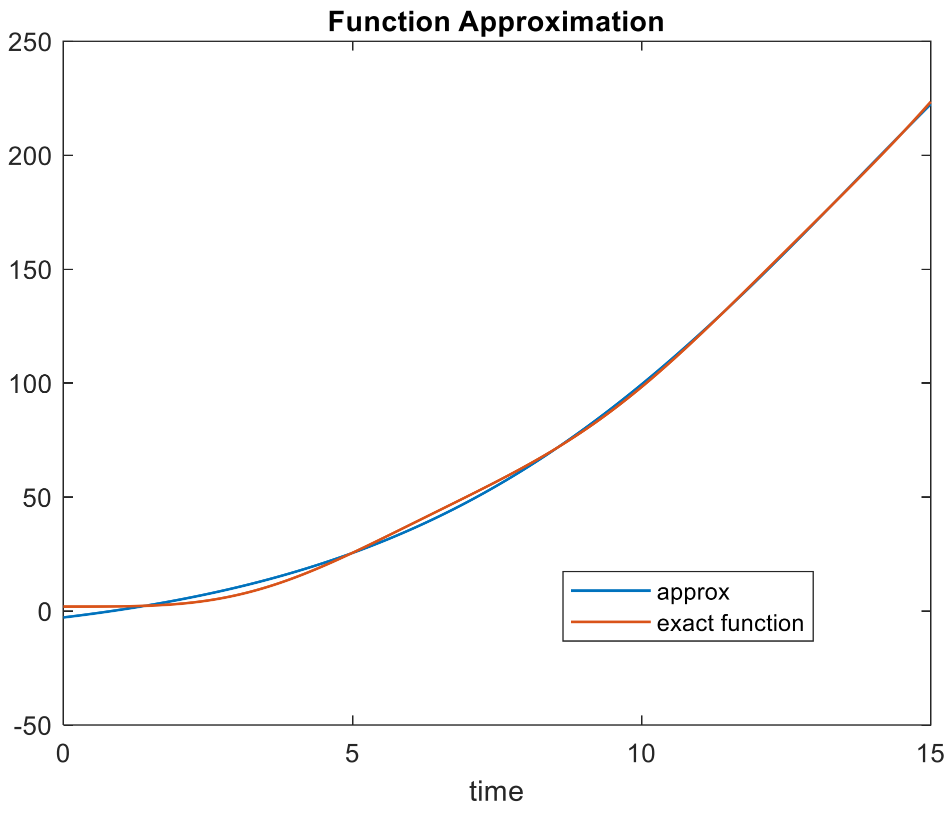


Conclusion – It is apparent that the second method is better at classification because one point in part a almost touches the decision boundary. This does not happen in part b once the appropriate minima location is reached. Thus, part b provides better results.

Problem – 2:

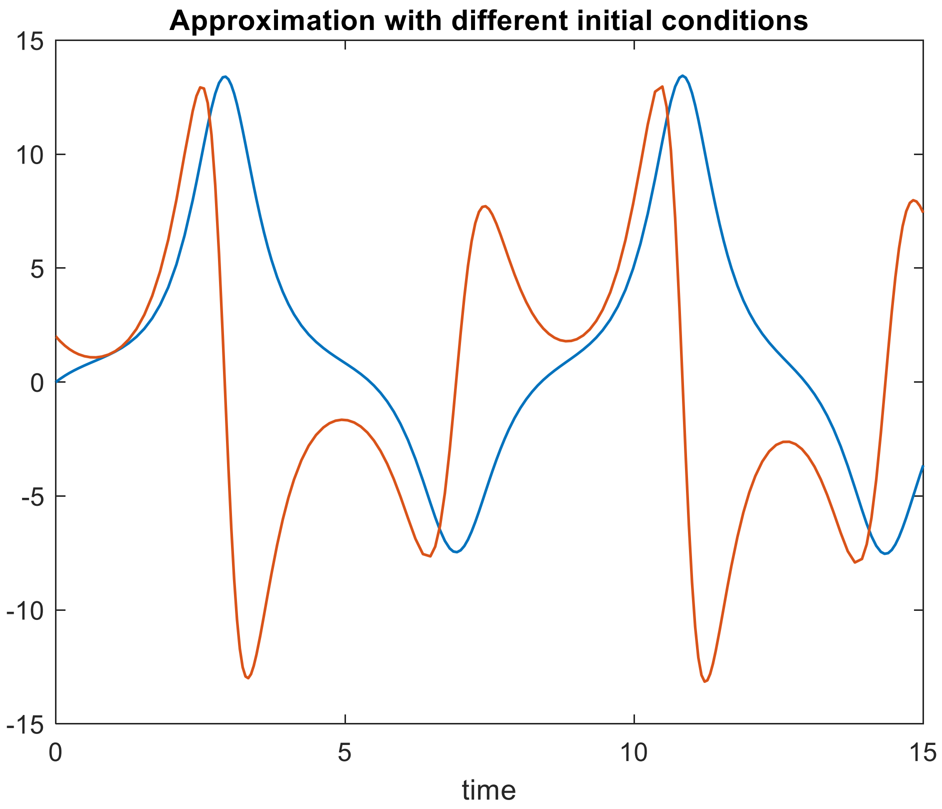
a. MATLAB Code:

```
function approx1

x=0:0.01:15;

fun=(x.^2)+2*cos(x);

approx=feedforwardnet(1);

approx.trainParam.epochs=500;

approx.layers{1}.transferFcn='logsig';

approx.trainParam.goal=0.01;

approx=train(approx,x,fun);

nefn=approx(x);

plot(x,nefn,x,fun)

end
```



b. MATLAB Code

```
function approx1

x=0:0.01:15;

fun=(x.^2)+2*cos(x);

approx=feedforwardnet(1);

approx.trainParam.epochs=500;

approx.layers{1}.transferFcn='logsig';

approx.trainParam.goal=0.01;

approx=train(approx,x,fun);

nefn=approx(x);

plot(x,nefn,x,fun)

approximate=@(t,x)[x(2);(-0.1*x(1)^3+sim(approx,x(1)))];

    [time,fn]=ode45(approximate,[0 15],[0 2]);

    plot(time,fn);

end
```



Appendix

CostFn.m

```
function [J, gradient] = costFn(angle, X, target)

m = length(target);

J = 0;

gradient = zeros(size(angle));

J=1./m*sum((-target'.*log(1./(1+exp(-angle'*X'))))-((1-target)'.*log(1-1./(1+exp(-angle'*X')))));

gradient=1./m*(1./(1+exp(-angle'*X')) -target')*X;

end
```

References

1. Coursera Machine Learning by Andrew NG