BITS F312

Neural Networks and Fuzzy Logic

Assignment - 2

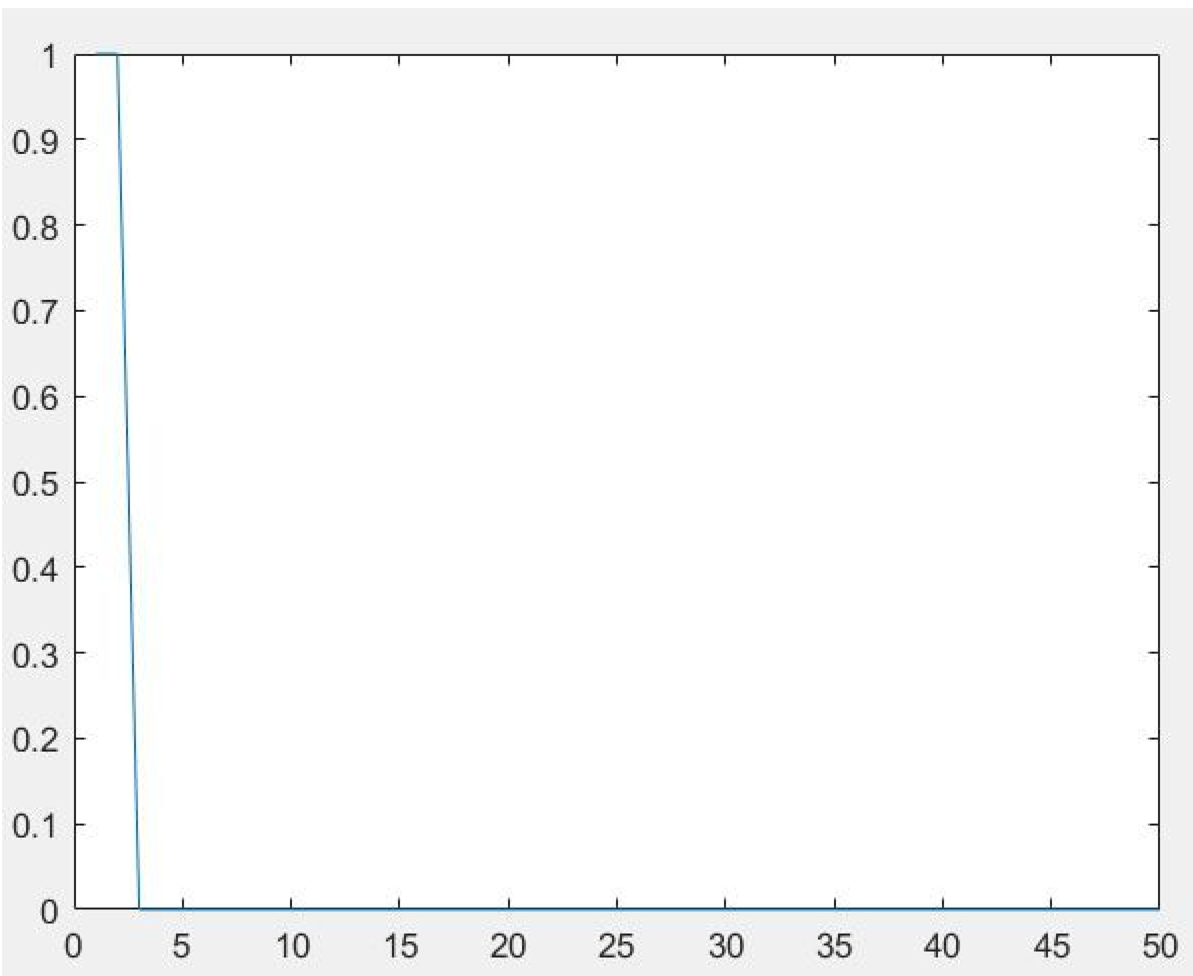
Name and ID :

Ravi Bharadwaj C

2016AAPS0244H

Question 1 :

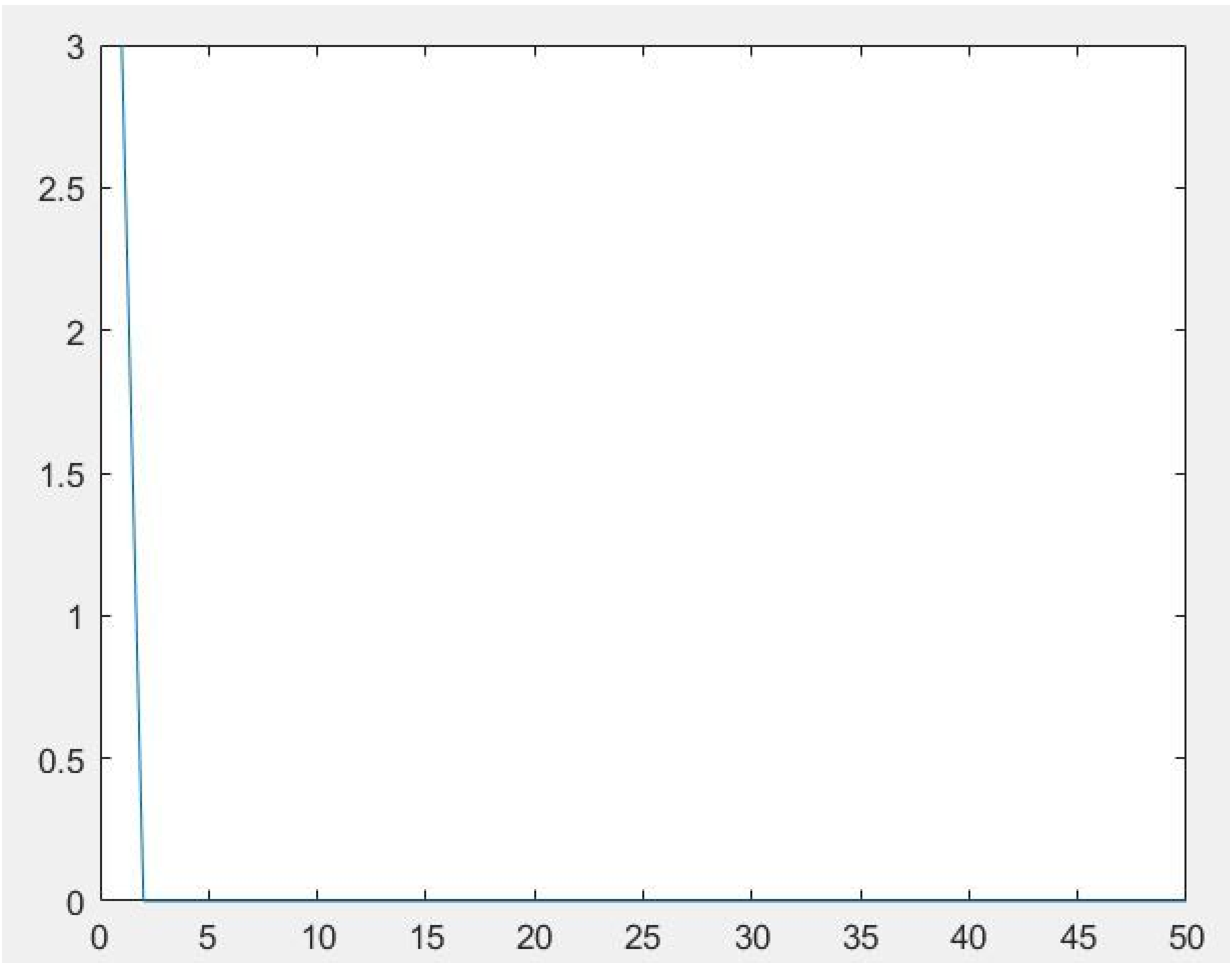
1. AND gate



w = 0.2,0.2

b = 0.2

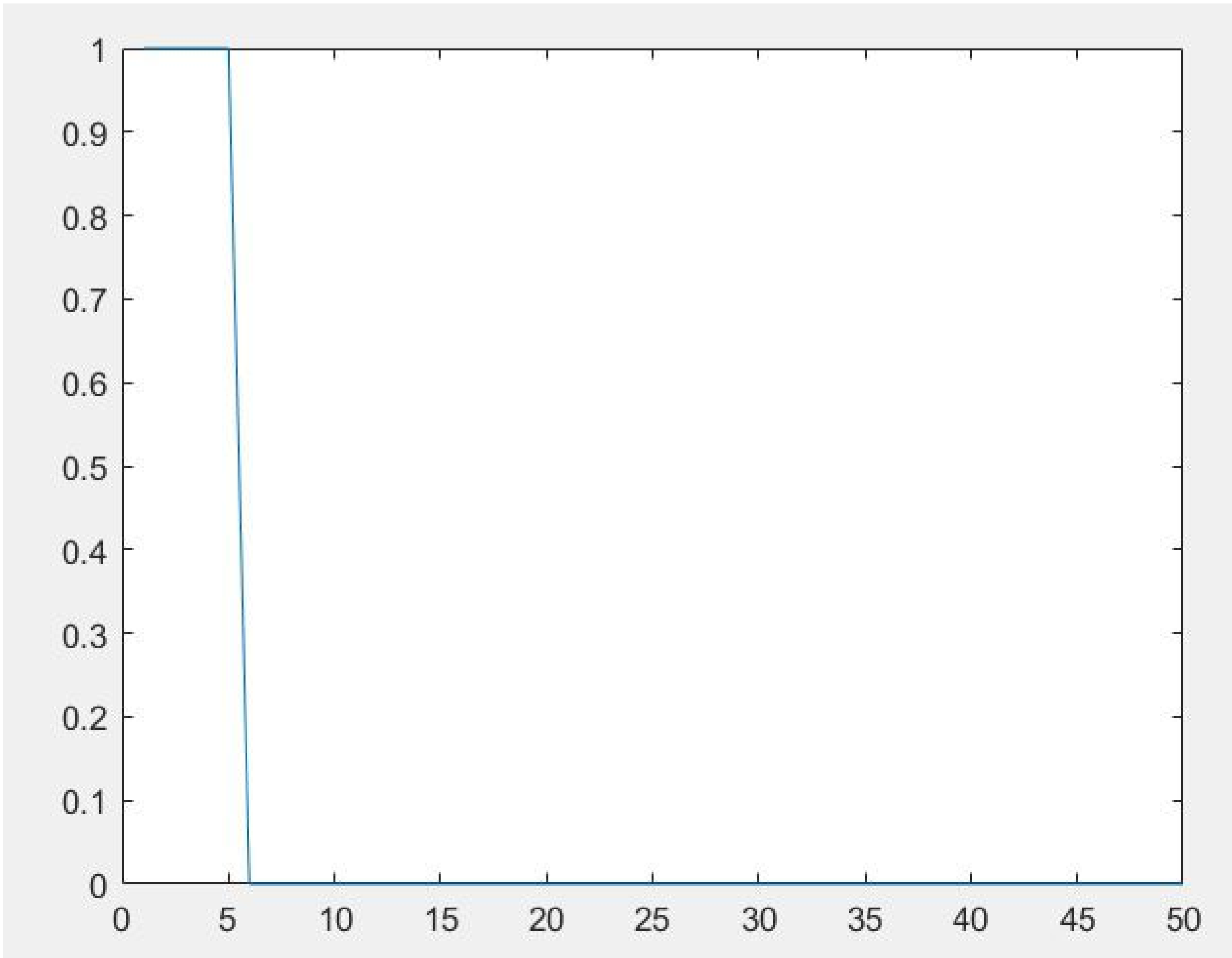
2. OR gate



w = 0.2,0.2

b = 0.3

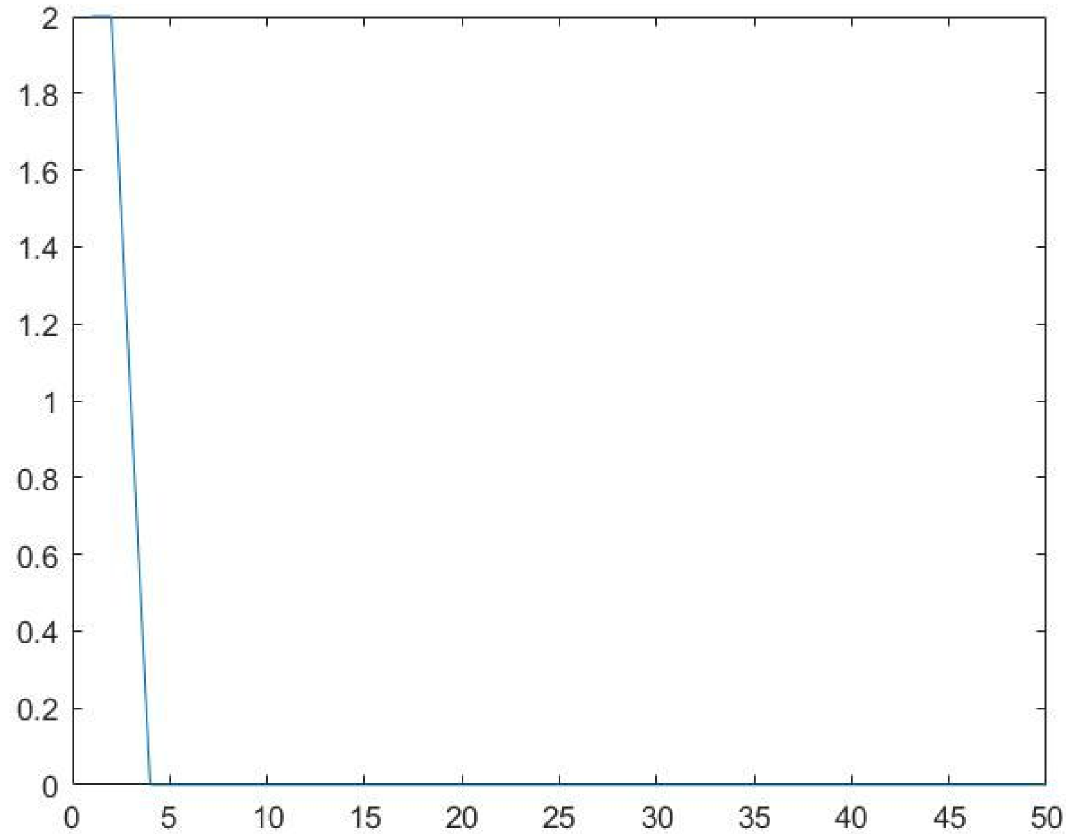
3. NOT gate



w = -0.4392

b = 0.5868

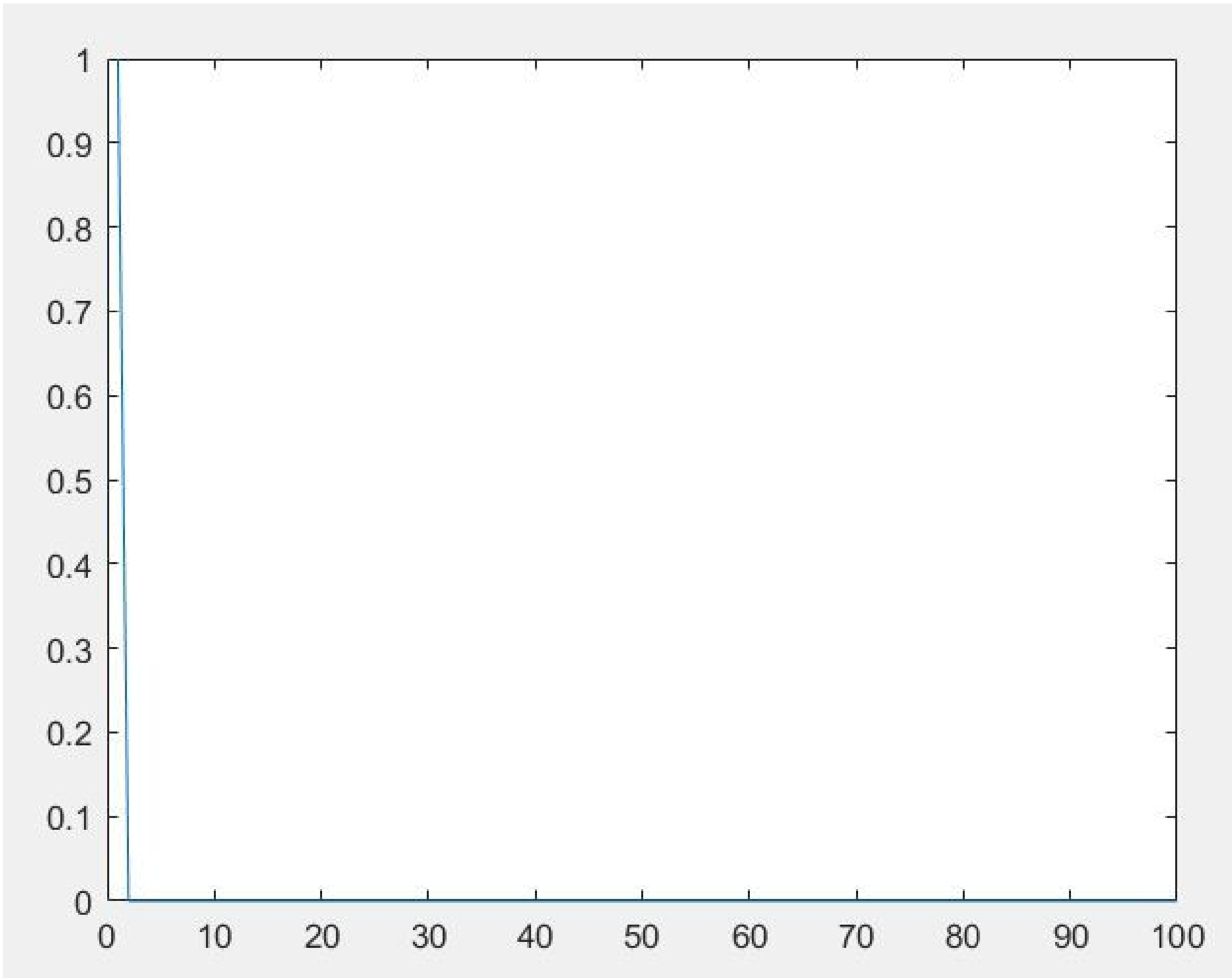
4. NAND gate



w=[-0.8878 -0.6993]

b = 1.4786

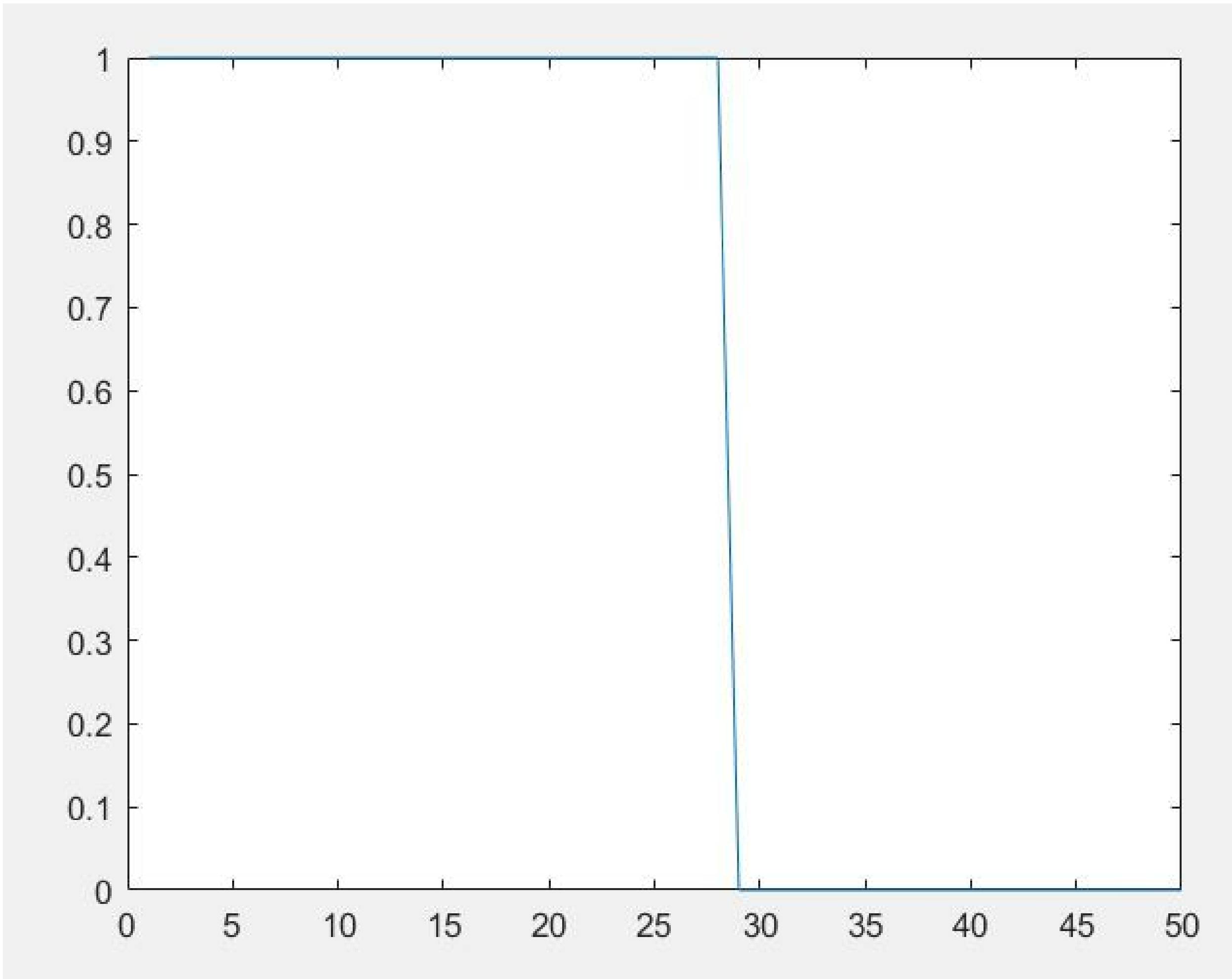
5. NOR gate



w = -0.1849 -0.2168

b = 0.5206

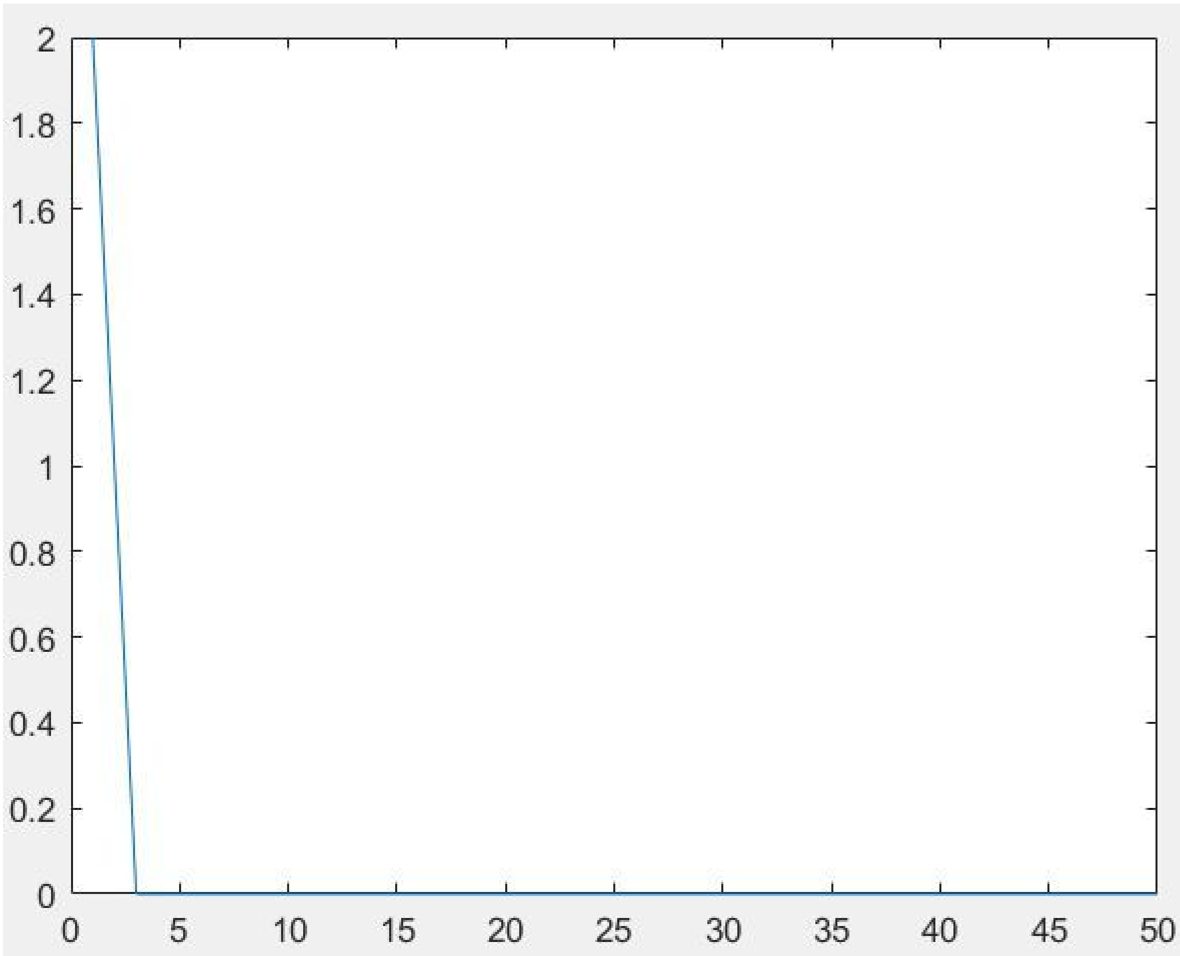
6. ANDNOT gate



w = 0.3233 -0.4336

b = 0.1899

7. XOR gate



w1 = 0.3115 0.7441

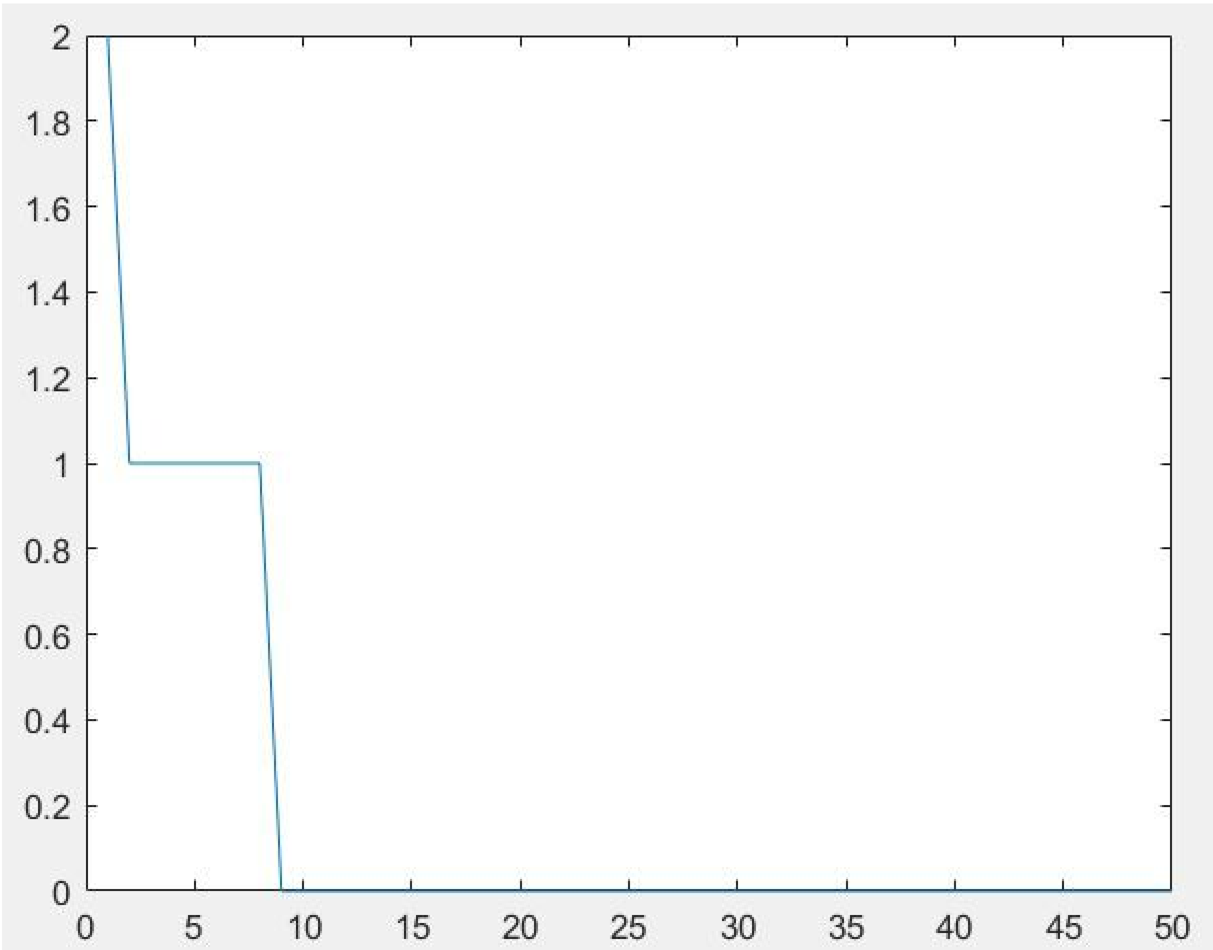
w2 = -0.6107 0.7121

w3 = 0.7368 0.6069

b1 = 0.3881

b2 = -0.0304

8. XNOR gate



w1 = 0.4312 0.4222

w2 = 0.4078 0.1276

w3 = 0.1814 1.3447

b1 = 0.5759

b2 = 0.4303

Question 2 :

The following code was used to find the maximum number of hidden neurons using the grid search approach. The same code is used for the other questions as well.

Editor - E:\Studies\3-1\NNFL\assignment2\assignment2\grid_search.m

mfnn.m | grid_search.m

```
% Grid search to find the best number of hidden neurons:
oa = [];
iter = 50;
for i = 1:iter
%      cm = rbfnn(iter);
    [cm,cost] = mfnn(iter);
    s = 0;
    for j = 1:3
        s = s + cm(j,j);
    end
    overall_accuracy = s/sum(sum(cm));
    oa = [oa; overall_accuracy];
end
[value, index] = max(oa);
fprintf("the max accuracy is achieved by using the number of hidden neurons to be : ")
disp(index)
fprintf("\n the max accuracy is : ")
disp(value)
```

For an MFNN using loops, the max accuracy achieved was 1 and the number of hidden neurons required for this was 36.

Command Window

```
the max accuracy is achieved by using the number of hidden neurons to be :     36


 the max accuracy is :      1
```

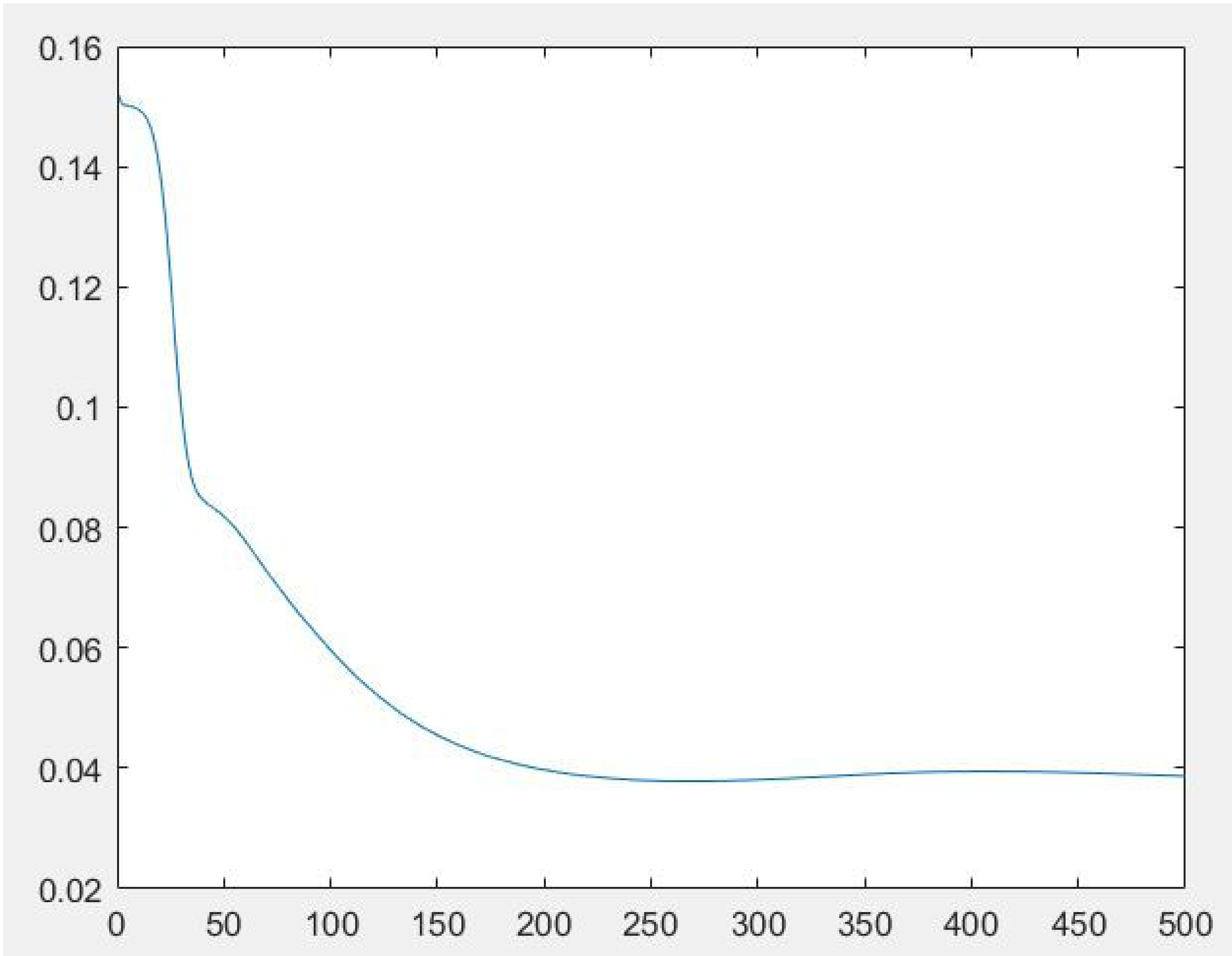
Code:

```
function [cm,cost] = mfnn(neurons)

% neurons = 10;
close all;
clc;
data = xlsread('dataset.xlsx');
data = data(randperm(size(data,1)),:);
X = data(:,(1:7));
X = normalize(X);
Y = data(:,8);
H = neurons; % number of hidden neurons
alpha = 0.01; % learning rate
iter = 500;
K = 3; % No. of output neurons = 3

% Sigmoid function definition
sigmoid = @(x) 1/(1 + exp(-x));

% Dividing data into test and training : 70-30 cross validation
train_x = X((1:105),:);
tr_y = Y((1:105),:);
train_y = zeros(105,3);
test_x = X((106:150),:);
test_y = Y((106:150),:);

for i = 1:length(tr_y)
    if (Y(i) == 1)
        train_y(i,:) = [1,0,0];
    elseif (Y(i) == 2)
        train_y(i,:) = [0,1,0];
    elseif (Y(i) == 3)
        train_y(i,:) = [0,0,1];
    end
end

% initializing random values of weight and bias between -0.01 and +0.01
a = -0.01;
b = 0.01;
w1 = a + rand(H,size(train_x,2))*(b-a);
% w1 = rand(H,size(train_x,2))*0.1;
b1 = a + (b-a)*rand();
```

```
b1 = a + (b-a)*rand();
w2 = a + (b-a)*rand(K,H);
b2 = a + (b-a)*rand();

z_train = zeros(1,H);
output = zeros(1,K);
cost = zeros(iter,1);
delta_w2 = zeros(K,H);
delta_w1 = zeros(H,size(train_x,2));

% Training :
for k = 1:iter
    for i = 1:size(train_x,1)
        % Forward propagation
        for j = 1:H
            z_train(j) = sigmoid(sum(w1(j,:).*train_x(i,:))+b1);
        end

        for j = 1:K
            output(i,j) = sigmoid(sum( w2(j,:).* z_train)+b2);
        end

        % Back propagation
        for j = 1:K
            cost(k) = cost(k) + ((output(i,j) - train_y(i,j))^2);
        end
        cost(k) = 0.5*sqrt(cost(k))/K;
        for j = 1:K
            for h = 1:H
                delta_w2(j,h) = -alpha*(train_y(i,j)-output(i,j))*output(i,j)*(1-output(i,j))*z_train(h);
            end
            delta_b2 = -alpha*(train_y(i,j)-output(i,j))*output(i,j)*(1-output(i,j));
        end

        for h = i:H
            for j = 1:size(train_x,2)
                s = 0;
                for g = 1:K
                    s = s + (train_y(i,g)-output(i,g))*w2(g,h);
                end
```

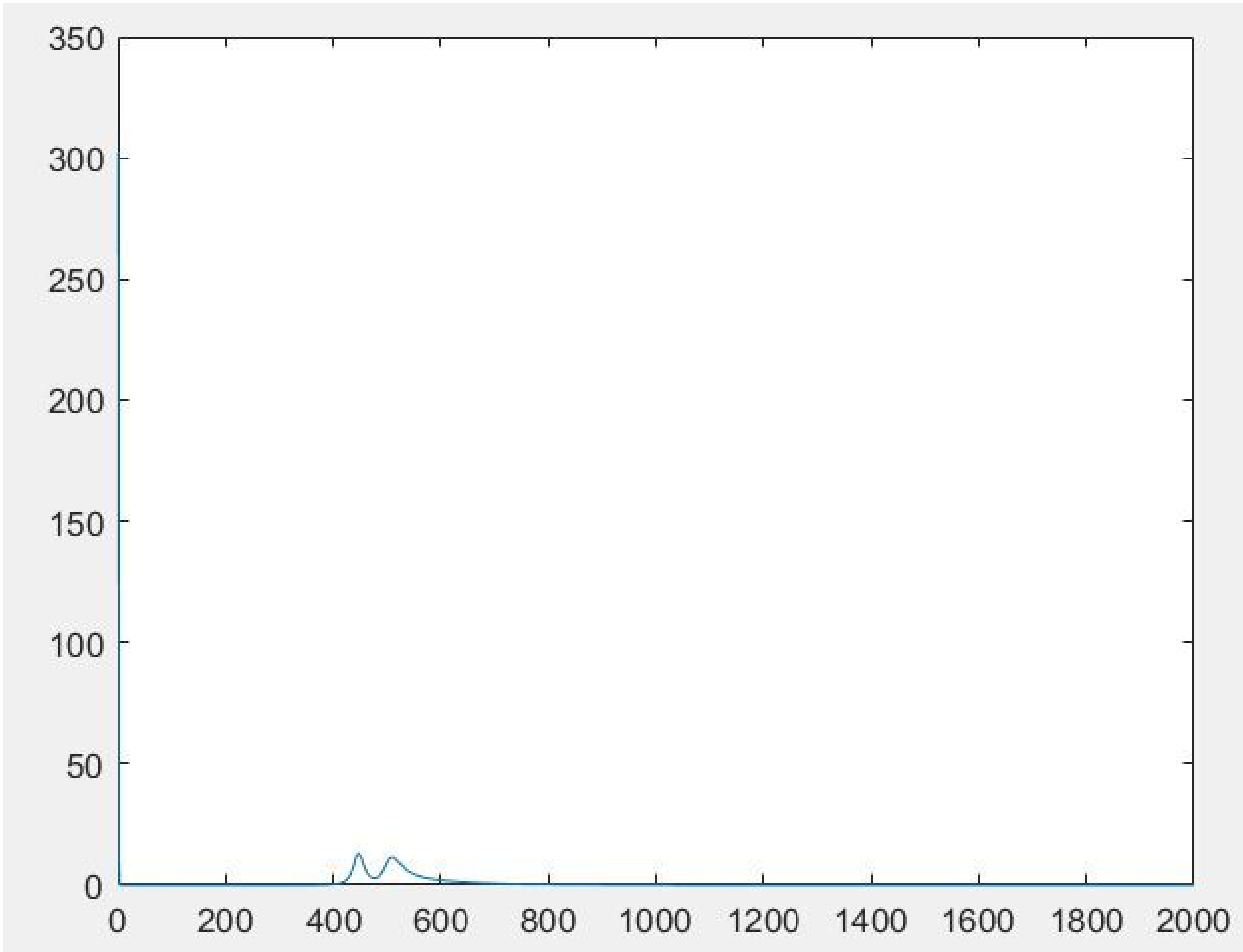
```
                end
                delta_w1(h,j) = -alpha*s*z_train(h)*(1-z_train(h))*train_x(i,j);
            end
            delta_b1 = -alpha*s*z_train(h)*(1-z_train(h));
        end
        % Weight update:
        for j = 1:K
            for h = 1:H
                w2(j,h) = w2(j,h) - delta_w2(j,h);
            end
            b2 = b2 - delta_b2;
        end

        for h = 1:H
            for j = 1:size(train_x,2)
                w1(h,j) = w1(h,j) - delta_w1(h,j);
            end
            b1 = b1 - delta_b1;
        end
    end
end

% Training :
z_test = zeros(1,H);
test_output = zeros(45,K);
 for p = 1:size(test_x,1)
        for h = 1:H
            z_test(h) = sigmoid(sum(w1(h, :).*test_x(p, :)) + b1);
        end
        for i = 1:K
            y_pred(p, i) = sigmoid(sum(w2(i, :).*z_test) + b2);
        end
end

    pl = zeros(1,size(y_pred,1));
    pa = zeros(1,size(y_pred,1));
    for i1 = 1:size(y_pred,1)
        [~,pl(i1)] = max(y_pred(i1,:));
        pa(i1) = test_y(i1,:);
    end
    [cm,~] = confusionmat(pa,pl);

    diagonal = 0;
    for i2 = 1:3
        diagonal = diagonal + cm(i2,i2);
    end
    accuracy = diagonal/sum(sum(cm));
    plot(cost)
end
```

Cost Function vs iterations



<u>Question 3:</u>

Cost


350
300
250
200
150
100
50
0
0
200
400
600
800
1000
1200
1400
1600
1800
2000

Code

```
clc;
clear;
close all;
data = xlsread('dataset.xlsx');
data = data(randperm(size(data,1)),:);
X = data(:,(1:7));
X = normalize(X);
Y = data(:,8);

% number of hidden neurons
H1 = 5;
H2 = 3;

alpha = 0.5; % learning rate
mf = 0.001; % Momentum factor
iter = 2000;
K = 3; % No. of output neurons = 3

% Sigmoid function definition
sigmoid = @(x) 1./(1 + exp(-x));

% Dividing data into test and training : 70-30 cross validation
train_x = X((1:105),:);
tr_y = Y((1:105),:);
train_y = zeros(105,3);
test_x = X((106:150),:);
test_y = Y((106:150),:);
[M, N] = size(train_x);
[P, Q] = size(test_x);

for i = 1:length(tr_y)
    if (Y(i) == 1)
        train_y(i,:) = [1,0,0];
    elseif (Y(i) == 2)
        train_y(i,:) = [0,1,0];
    elseif (Y(i) == 3)
        train_y(i,:) = [0,0,1];
    end
end

% initializing random values of weight and bias between -0.01 and +0.01
rmin = -0.01;
rmax = 0.01;
w1 = rmin + rand(size(train_x,2)+1,H1)*(rmax-rmin);
w2 = rmin + (rmax-rmin)*rand(H1+1,H2);
w3 = rmin + (rmax-rmin)*rand(H2+1,K);
b = 1;
```

```
% Training :
train_x = [b*ones(M, 1) train_x];
Dw1 = zeros(N+1, H1);
Dw2 = zeros(H1+1, H2);
Dw3 = zeros(H2+1, 3);
cost = zeros(iter, 1);

for k = 1:iter
    % Forward Propagation :
    z1 = [ones(M,1) sigmoid(train_x*w1 + b)];
    z2 = [ones(M,1) sigmoid(z1*w2 + b)];
    y = sigmoid(z2*w3);

    % Back Propagation :
    cost(k) = mean(sum(train_y - y).^2);
    df = y.*(1-y);
    d3 = df.*(train_y - y);
    Dw3 = (alpha/N)*d3'*z2;
    w3 = (1+mf)*w3 + Dw3';

    df = z2.*(1-z2);
    d2 = df.*(d3*w3');
    d2 = d2(:, 2:end);
    Dw2 = (alpha/N)*d2'*z1;
    w2 = (1+mf)*w2+Dw2';

    df = z1.*(1-z1);
    d1 = df.*(d2*w2');
    d1 = d1(:, 2:end);
    Dw1 = (alpha/N)*d1'*train_x;
    w1 = (1+mf)*w1 + Dw1';
end

% Testing :
test_x = [ones(size(test_x,1),1) test_x];
z1_test = [ones(size(test_x,1),1) sigmoid(test_x*w1 + b)];
z2_test = [ones(size(test_x,1),1) sigmoid(z1_test*w2 + b)];
% z1_test = sigmoid(test_x*w1 + b);
% z2_test = sigmoid(z1_test*w2 + b);
y_output = sigmoid(z2_test*w3);

pl = zeros(1,size(y_output,1));
pa = zeros(1,size(y_output,1));
for i1 = 1:size(y_output,1)
    [~,pl(i1)] = max(y_output(i1,:));
    pa(i1) = test_y(i1,:);
end
[cm,~] = confusionmat(pa,pl);

diagonal = 0;
for i2 = 1:3
    diagonal = diagonal + cm(i2,i2);
```

```
    diagonal = diagonal + cm(i2,i2);
end
accuracy = diagonal/sum(sum(cm));
plot(cost)
```

The accuracy is 0.933 and the best combination of hidden neurons was found to be:

H1 = 5; H2 = 3;

cm

3x3 double

	1	2	3
1	13	0	0
2	1	12	2
3	0	0	17
4			

<u>Question 4:</u>

Code

```matlab
% The input n is the number of neurons. To find n, Run 'grid_search.m'

function cm = rbfnn(n)
clc;
% clear;
close all;
data = xlsread('dataset.xlsx');
X = data(:,(1:7));
Y = data(:,8);
z = zeros(150,3);
% No. of output neurons = 3

for i = 1:length(Y)
    if (Y(i) == 1)
        z(i,:) = [1,0,0];
    elseif (Y(i) == 2)
        z(i,:) = [0,1,0];
    elseif (Y(i) == 3)
        z(i,:) = [0,0,1];
    end
end

% Dividing data into test and training : 70-30 cross validation
traininput = [];
trainoutput = [];
testinput = [];
testoutput = [];

for j = 1:size(X,1)
    if rand < 0.7
        traininput = [traininput; X(j,:)];
        trainoutput = [trainoutput;z(j,:)];
    else
        testinput = [testinput; X(j,:)];
        testoutput = [testoutput;z(j,:)];
    end
end

x = traininput;
y = trainoutput;
xt = testinput;
yt = testoutput;

[~ , mu] = kmeans(x,n);

% Hidden layer eval
for i = 1:size(x,1)
    for j = 1:size(mu,1)
        h(i,j) = (norm( x(i,:) - mu(j,:)))^3;
    end
end

% Weight eval
```

```
% Weight eval
W = pinv(h)*y;

% Test data eval

for i1 = 1:size(xt,1)
    for j1 = 1:size(mu,1)
        H(i1,j1) = (norm( xt(i1,:) - mu(j1,:)))^3;
    end
end
output = H*W;
pl = zeros(1,40);
pa = zeros(1,40);

for i1 = 1:size(output,1)
    [~,pl(i1)] = max(output(i1,:));
    [~,pa(i1)] = max(yt(i1,:));
end

[cm,~] = confusionmat(pa,pl);

% fprintf("The confusion matrix is : \n");
% disp(cm)
end
```

Optimal number of hidden neurons :

Editor - E:\Studies\3-1\NNFL\assignment2\assignment2\grid_search.m

grid_search.m | rbfnn.m

```
% Grid search to find the best number of hidden neurons:
oa = [];
iter = 50;
for i = 1:iter
    cm = rbfnn(iter);
%     [cm,cost] = mfnn(iter);
    s = 0;
    for j = 1:3
        s = s + cm(j,j);
    end
    overall_accuracy = s/sum(sum(cm));
    oa = [oa; overall_accuracy];
end
[value, index] = max(oa);
fprintf("the max accuracy is achieved by using the number of hidden neurons to be : ")
disp(index)
fprintf("\n the max accuracy is : ")
disp(value)
```

Command Window

```
the max accuracy is achieved by using the number of hidden neurons to be :      42


 the max accuracy is :      0.9268
```

Question 5:

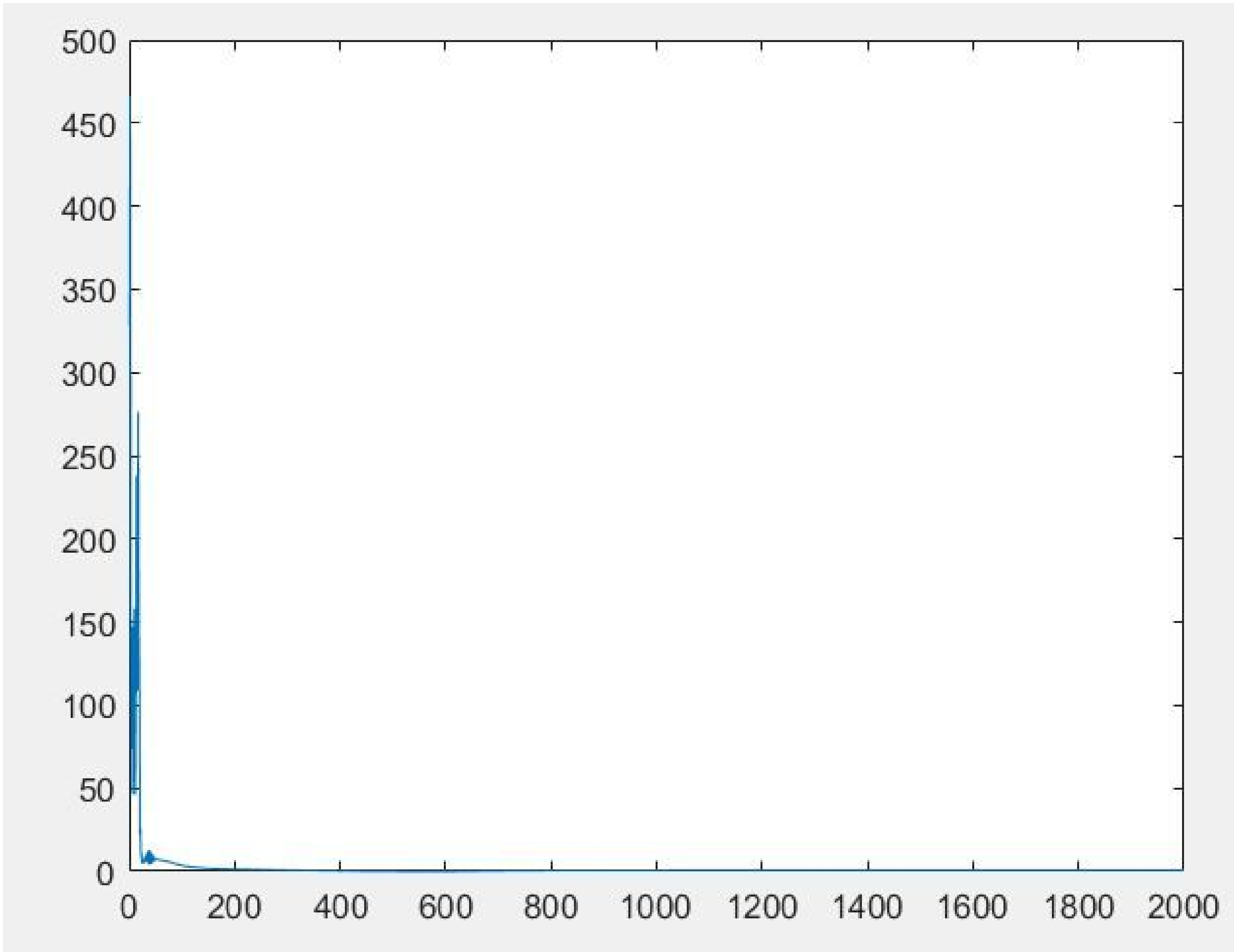
Code

```
clc;
clear;
close all;
data = xlsread('dataset.xlsx');
data = data(randperm(size(data,1)),:);
X = data(:,(1:7));
X = normalize(X);
Y = data(:,8);

alpha = 0.5; % learning rate
mf = 0.001; % Momentum factor
iter = 2000;
K = 3; % No. of output neurons = 3

% Sigmoid function definition
sigmoid = @(x) 1./(1 + exp(-x));

% Dividing data into test and training : 70-30 cross validation
train_x = X((1:105),:);
tr_y = Y((1:105),:);
train_y = zeros(105,3);
test_x = X((106:150),:);
test_y = Y((106:150),:);
[M, N] = size(train_x);
[P, Q] = size(test_x);

H1 = 10; % Number of hidden neurons in MFNN

for i = 1:length(tr_y)
    if (Y(i) == 1)
        train_y(i,:) = [1,0,0];
    elseif (Y(i) == 2)
        train_y(i,:) = [0,1,0];
    elseif (Y(i) == 3)
        train_y(i,:) = [0,0,1];
    end
end

% initializing random values of weight and bias between -0.01 and +0.01
rmin = -0.01;
rmax = 0.01;
w1 = rmin + rand(size(train_x,2)+1,H1)*(rmax-rmin);
w2 = rmin + (rmax-rmin)*rand(H1+1,K);
b = 1;


% --------TRAINING---------
train_x = [b*ones(M, 1) train_x];
Dw1 = zeros(N+1, H1);
Dw2 = zeros(H1+1, K);
cost = zeros(iter, 1);
```

```
% MFNN Part
for k = 1:iter
    % Forward Propagation
    z = [ones(M,1) sigmoid(train_x*w1 + b)];
    y = sigmoid(z*w2);

    % Backward Propagation
    cost(k) = mean(sum(train_y - y).^2);
    df = y.*(1-y);
    d2 = df.*(train_y - y);
    Dw2 = (alpha/N)*d2'*z;
    w2 = (1+mf)*w2 + Dw2';

    df = z.*(1-z);
    d1 = df.*(d2*w2');
    d1 = d1(:, 2:end);
    Dw1 = (alpha/N)*d1'*train_x;
    w1 = (1+mf)*w1 + Dw1';
end

% RBFNN Part

n = 10; % Number of hidden neurons in RBFNN(no. of cluster centers)
[~ , mu] = kmeans(y,n);

% Hidden layer eval
for i = 1:size(y,1)
    for j = 1:size(mu,1)
        h(i,j) = (norm( y(i,:) - mu(j,:)))^3;
    end
end

% Weight eval
W = pinv(h)*train_y;

% ------TESTING--------

% MFNN Forward prop:
test_x = [ones(size(test_x,1),1) test_x];
z_test = [ones(size(test_x,1),1) sigmoid(test_x*w1 + b)];
mfnn_output = sigmoid(z_test*w2);

% RBFNN Forward prop
for i1 = 1:size(mfnn_output,1)
    for j1 = 1:size(mu,1)
        H(i1,j1) = (norm( mfnn_output(i1,:) - mu(j1,:)))^3;
    end
end
final_output = H*W;
```

```
% Accuracy and confusion matrix eval
pl = zeros(1,size(final_output,1));
pa = zeros(1,size(final_output,1));
for i1 = 1:size(final_output,1)
    [~,pl(i1)] = max(final_output(i1,:));
    pa(i1) = test_y(i1,:);
end
[cm,~] = confusionmat(pa,pl);

diagonal = 0;
for i2 = 1:3
    diagonal = diagonal + cm(i2,i2);
end
accuracy = diagonal/sum(sum(cm));
plot(cost)
```

Cost



Accuracy and Confusion matrix:

Accuracy = 0.9556

CM :

cm

3x3 double

	1	2	3
1	10	0	0
2	0	20	1
3	0	1	13
4			