

## **BITS F312**

### **Neural Networks and Fuzzy Logic**

#### **Assignment - 2**

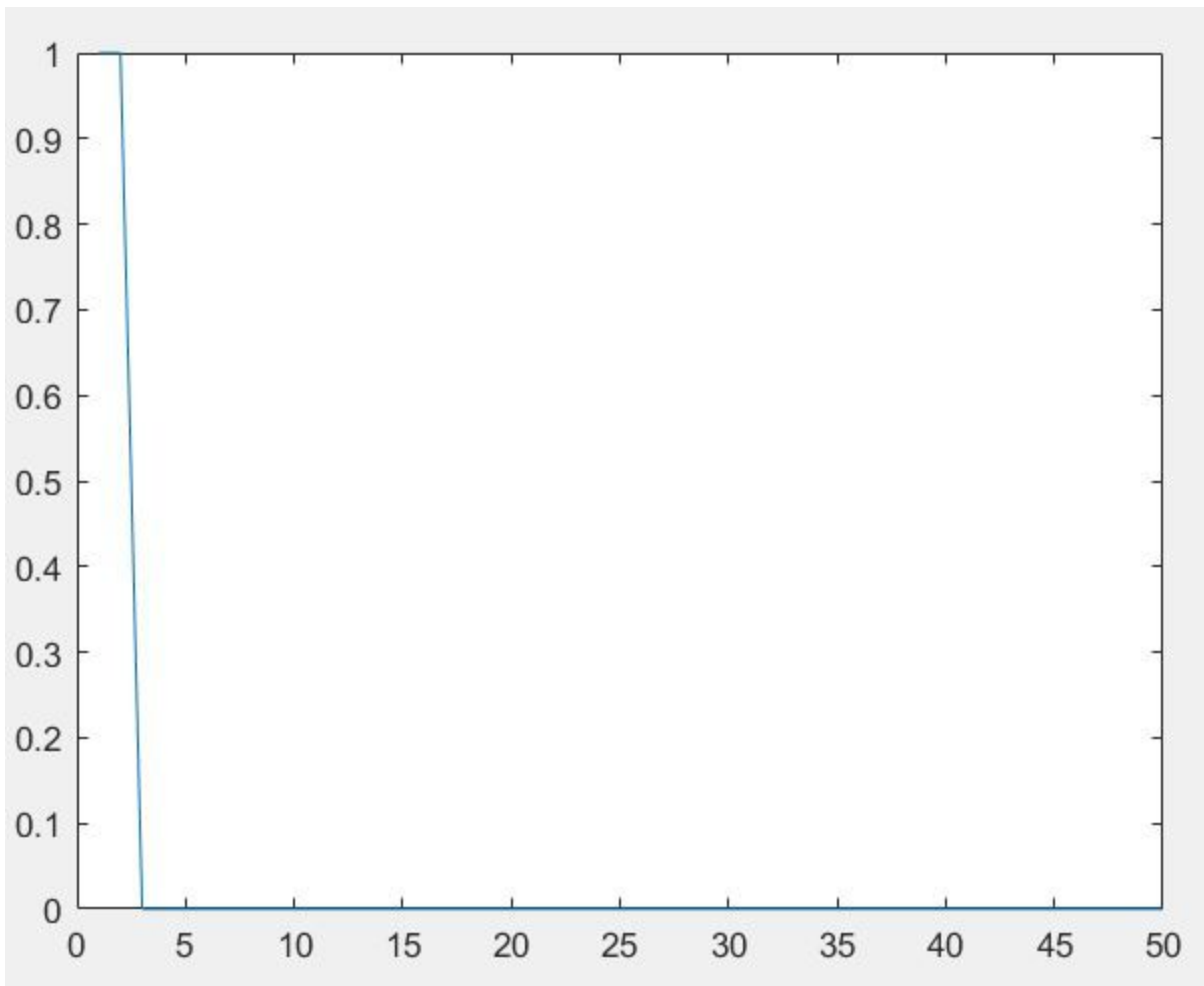
##### **Name and ID :**

Ravi Bharadwaj C

2016AAPS0244H

##### **Question 1 :**

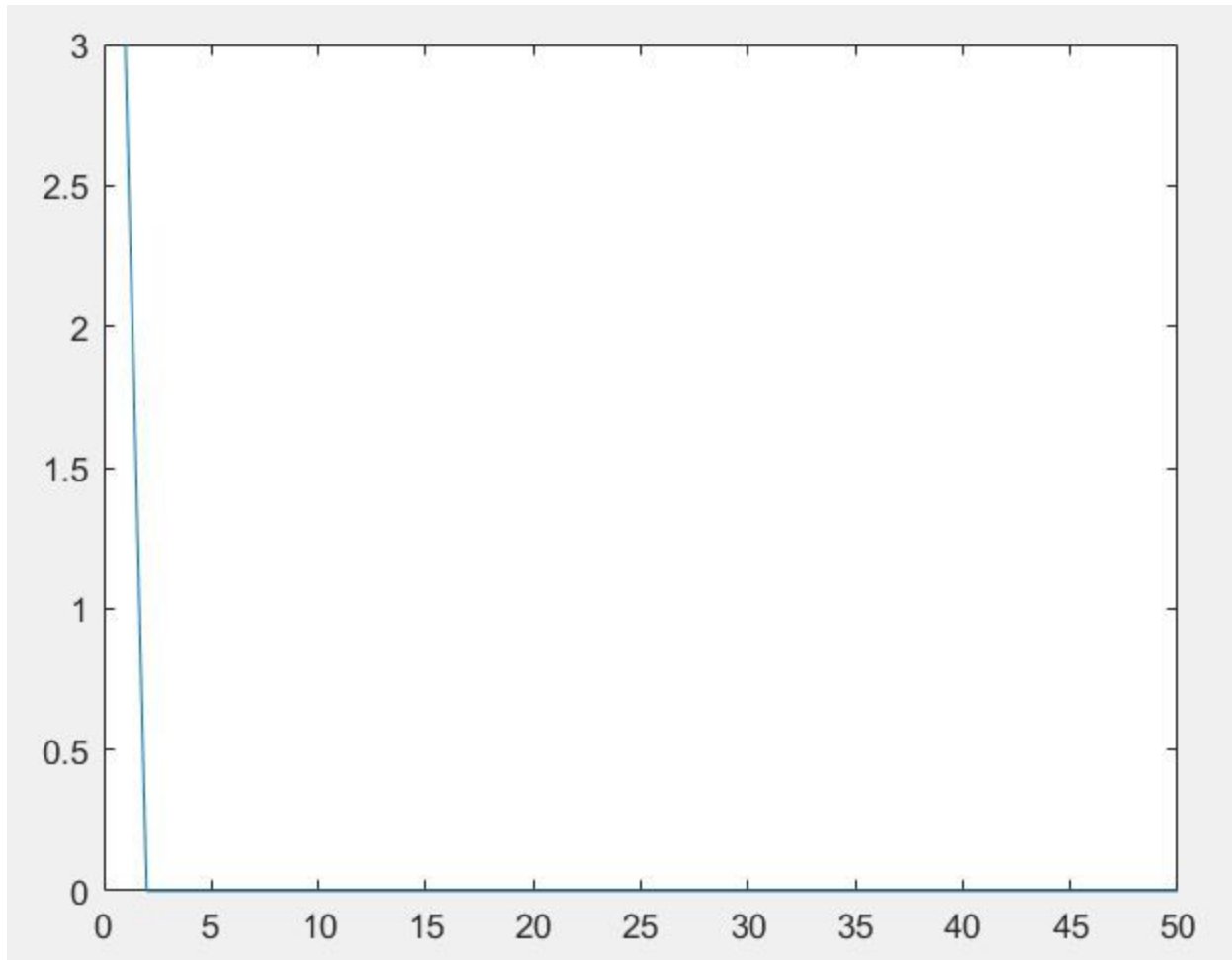
1. AND gate



$w = 0.2, 0.2$

$b = 0.2$

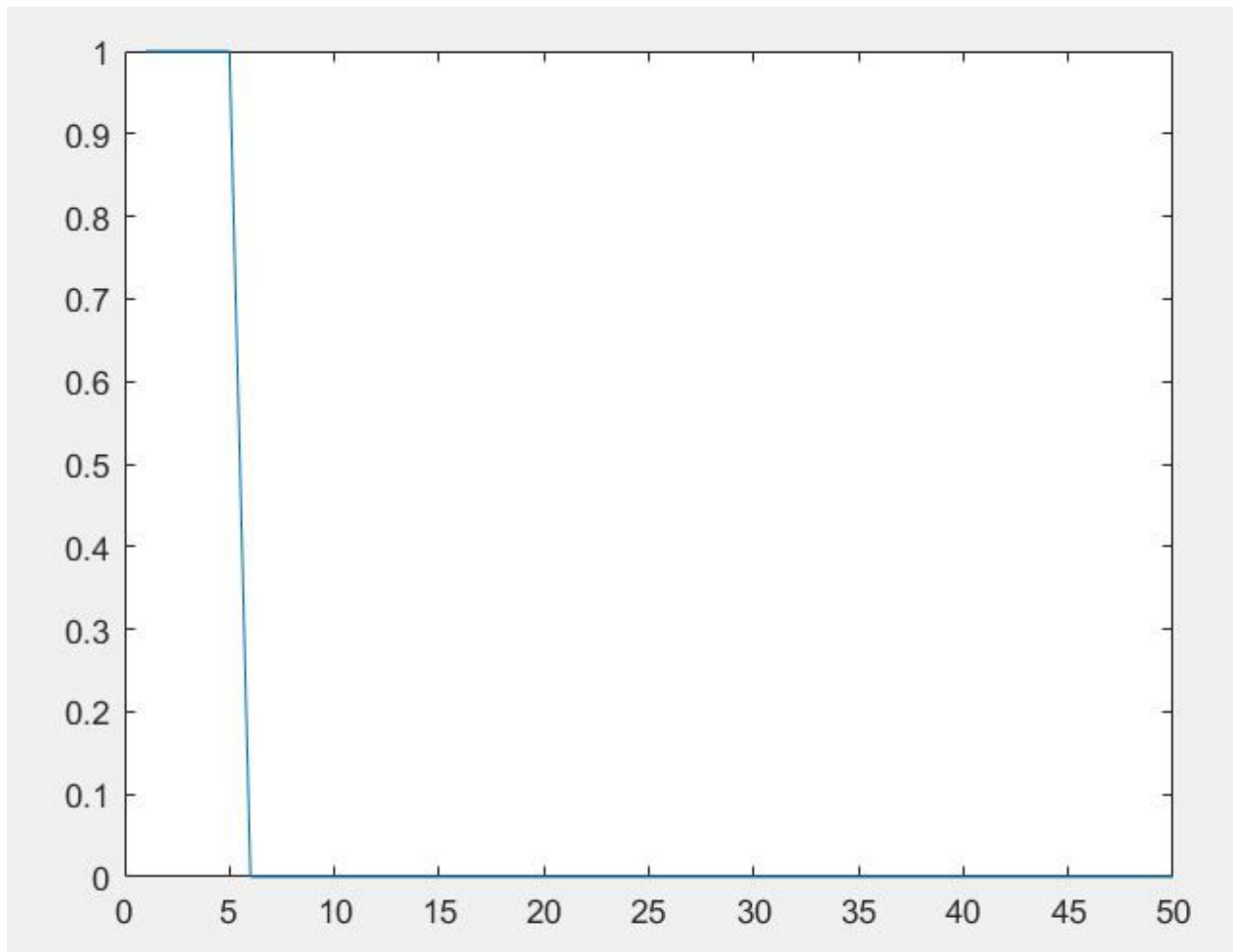
2. OR gate



$w = 0.2, 0.2$

$b = 0.3$

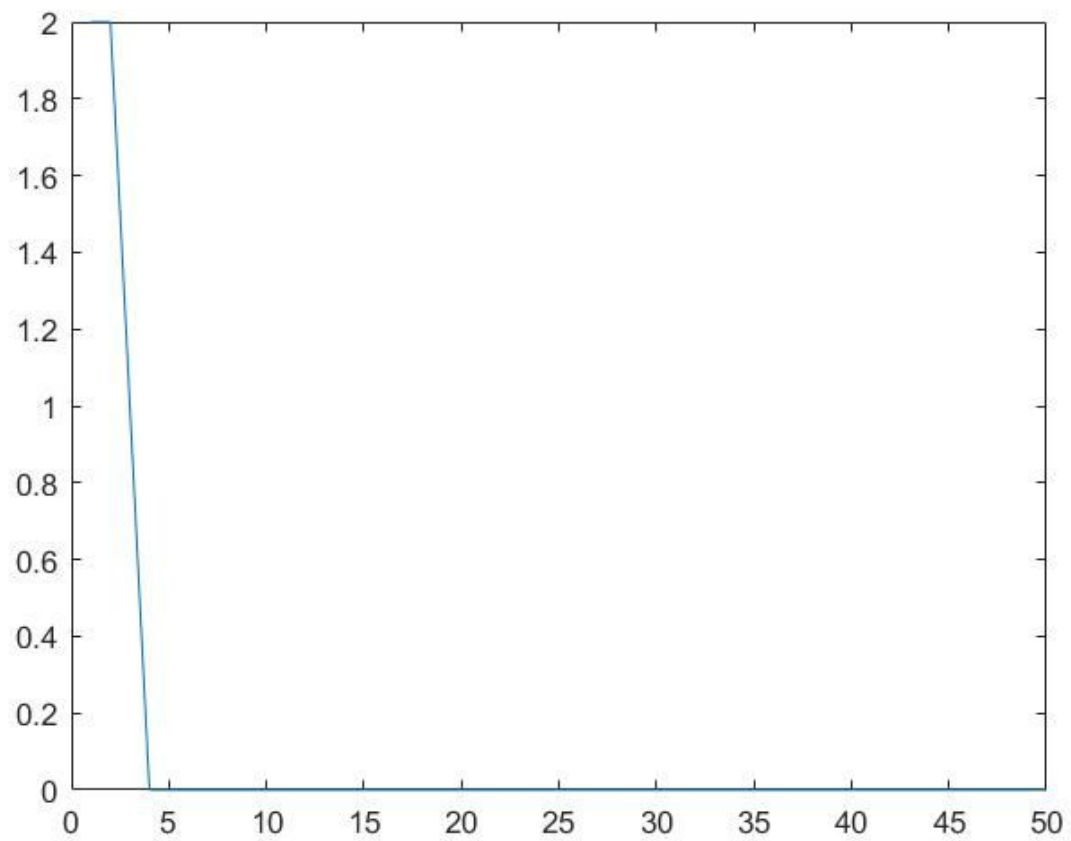
### 3. NOT gate



$w = -0.4392$

$b = 0.5868$

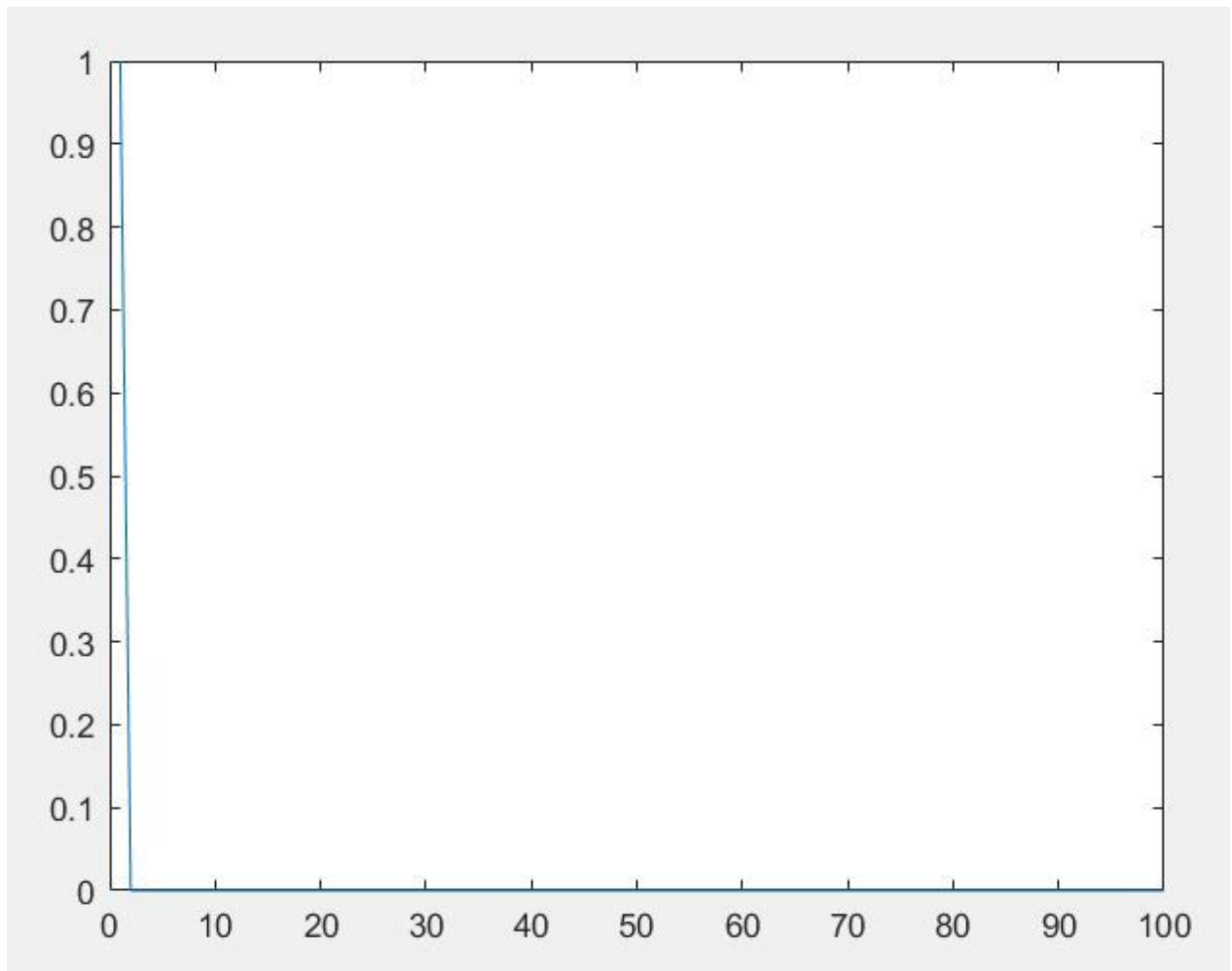
#### 4. NAND gate



$w = [-0.8878 \ -0.6993]$

$b = 1.4786$

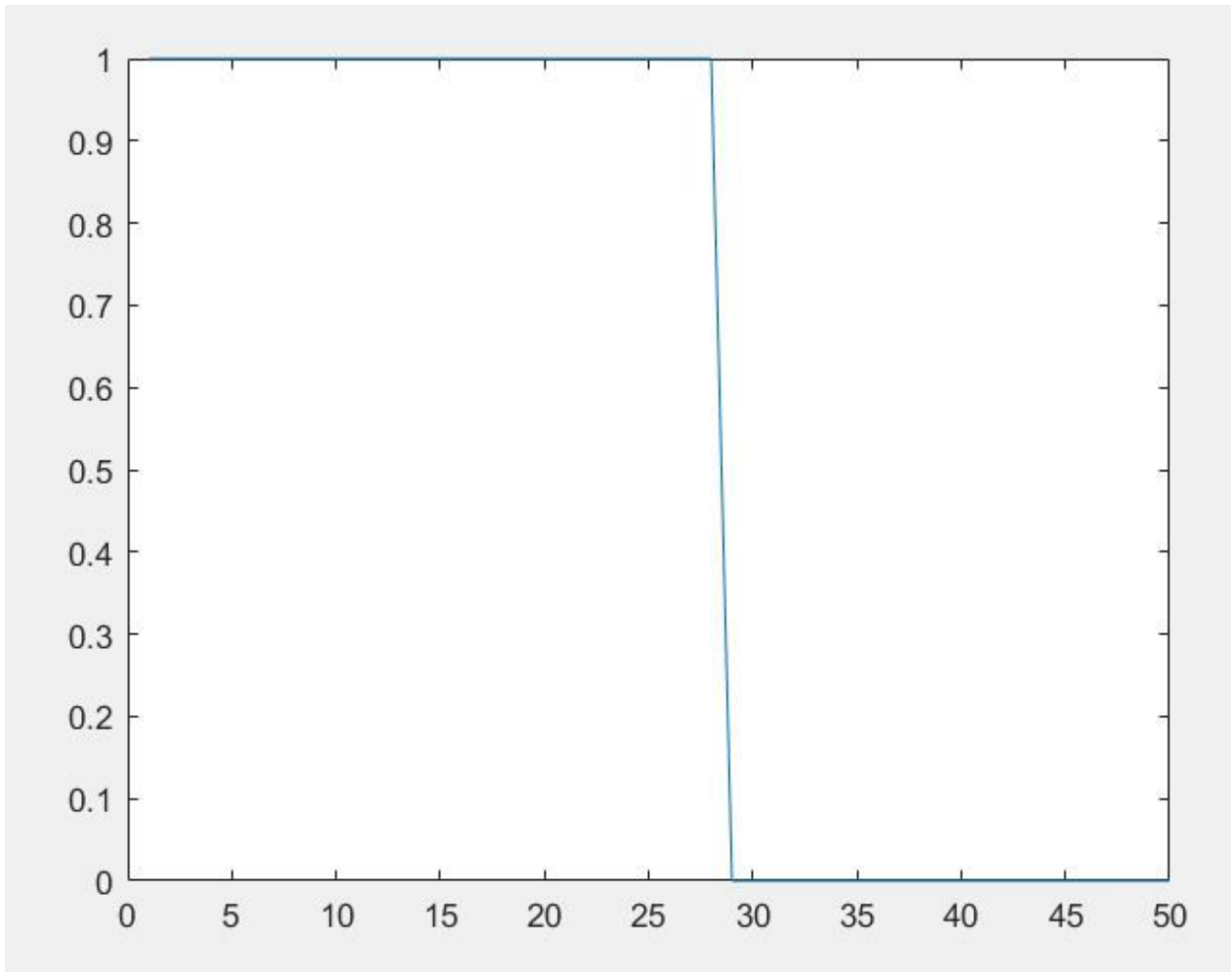
## 5. NOR gate



$w = -0.1849 \quad -0.2168$

$b = 0.5206$

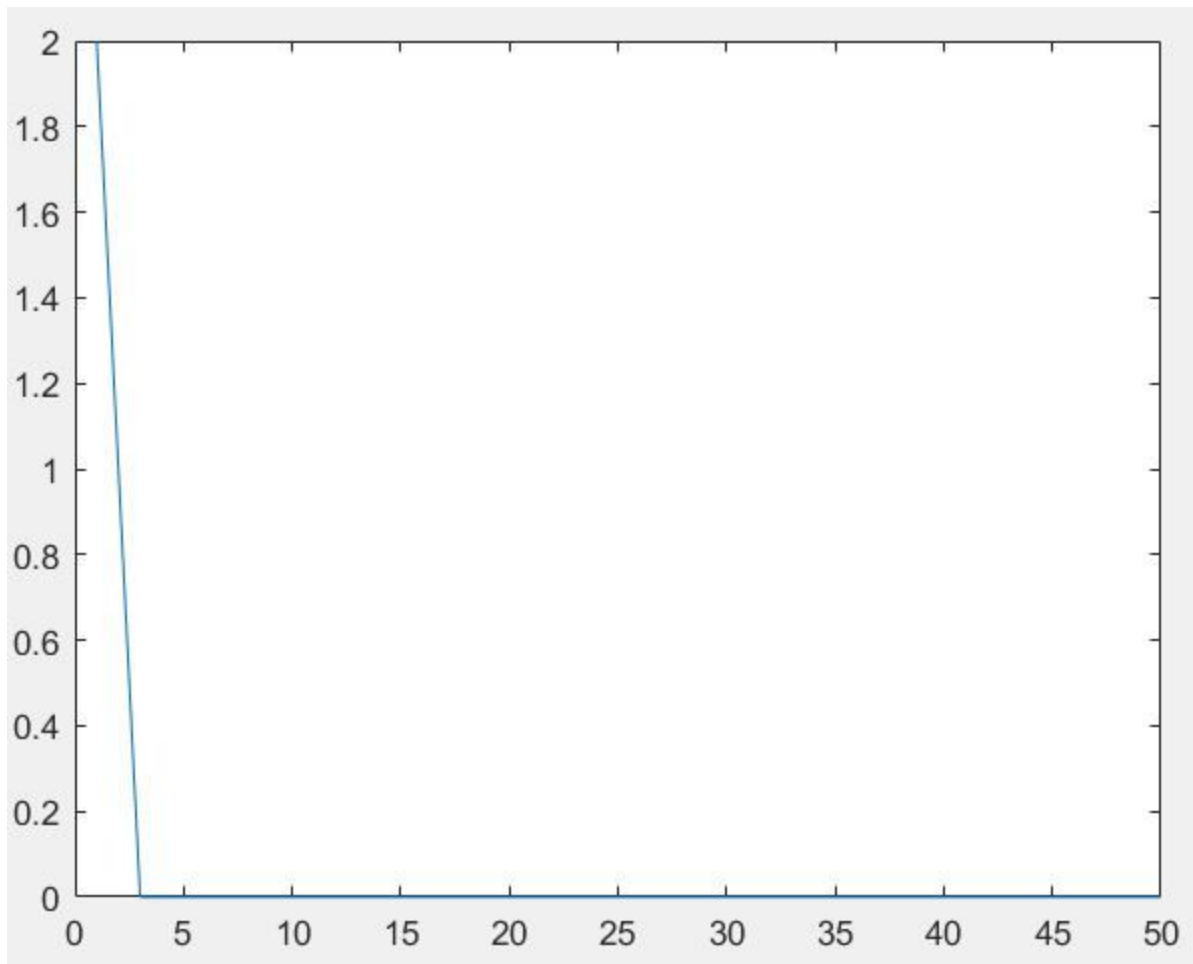
## 6. ANDNOT gate



$w = 0.3233 \quad -0.4336$

$b = 0.1899$

## 7. XOR gate



$w1 = 0.3115 \ 0.7441$

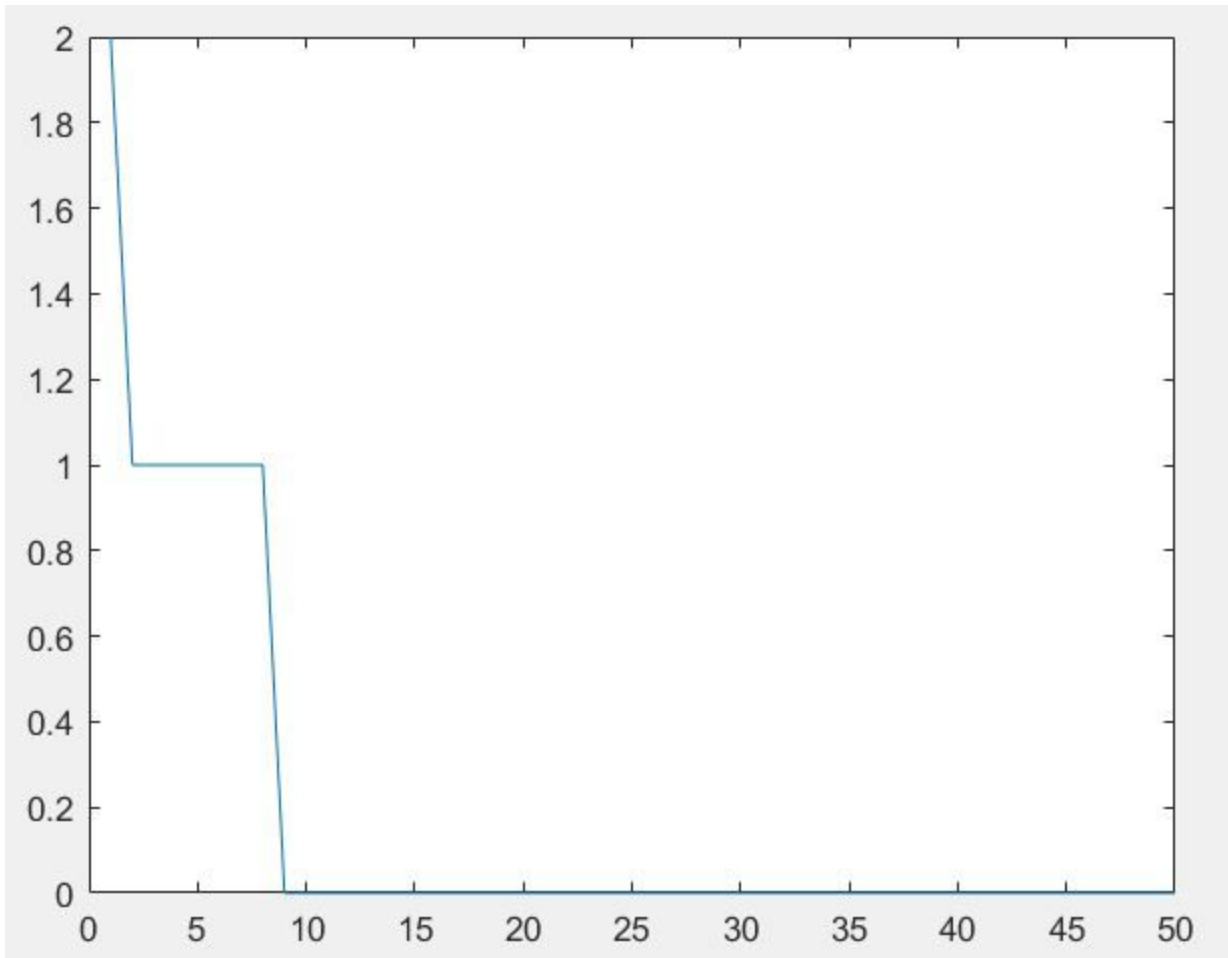
$w2 = -0.6107 \ 0.7121$

$w3 = 0.7368 \ 0.6069$

$b1 = 0.3881$

$b2 = -0.0304$

## 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  +
1  % Grid search to find the best number of hidden neurons:
2  oa = [];
3  iter = 50;
4  for i = 1:iter
5      % cm = rbfnn(iter);
6      [cm, cost] = mfnn(iter);
7      s = 0;
8      for j = 1:3
9          s = s + cm(j,j);
10     end
11     overall_accuracy = s/sum(sum(cm));
12     oa = [oa; overall_accuracy];
13 end
14 [value, index] = max(oa);
15 fprintf("the max accuracy is achieved by using the number of hidden neurons to be : ")
16 disp(index)
17 fprintf("\n the max accuracy is : ")
18 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:**

```

1  function [cm,cost] = mfnn(neurons)
2
3  % neurons = 10;
4  close all;
5  clc;
6  data = xlsread('dataset.xlsx');
7  data = data(randperm(size(data,1)),:);
8  X = data(:,(1:7));
9  X = normalize(X);
10 Y = data(:,8);
11 H = neurons; % number of hidden neurons
12 alpha = 0.01; % learning rate
13 iter = 500;
14 K = 3; % No. of output neurons = 3
15
16 % Sigmoid function definition
17 sigmoid = @(x) 1/(1 + exp(-x));
18
19 % Dividing data into test and training : 70-30 cross validation
20 train_x = X((1:105),:);
21 tr_y = Y((1:105),:);
22 train_y = zeros(105,3);
23 test_x = X((106:150),:);
24 test_y = Y((106:150),:);
25
26 for i = 1:length(tr_y)
27     if (Y(i) == 1)
28         train_y(i,:) = [1,0,0];
29     elseif (Y(i) == 2)
30         train_y(i,:) = [0,1,0];
31     elseif (Y(i) == 3)
32         train_y(i,:) = [0,0,1];
33     end
34 end
35
36 % initializing random values of weight and bias between -0.01 and +0.01
37 a = -0.01;
38 b = 0.01;
39 w1 = a + rand(H,size(train_x,2))*(b-a);
40 % w1 = rand(H,size(train_x,2))*0.1;
41 b1 = a + (b-a)*rand();

```

```

41 b1 = a + (b-a)*rand();
42 w2 = a + (b-a)*rand(K,H);
43 b2 = a + (b-a)*rand();
44
45 z_train = zeros(1,H);
46 output = zeros(1,K);
47 cost = zeros(iter,1);
48 delta_w2 = zeros(K,H);
49 delta_w1 = zeros(H,size(train_x,2));
50
51 % Training :
52 for k = 1:iter
53     for i = 1:size(train_x,1)
54         % Forward propagation
55         for j = 1:H
56             z_train(j) = sigmoid(sum(w1(j,:).*train_x(i,:))+b1);
57         end
58
59         for j = 1:K
60             output(i,j) = sigmoid(sum(w2(j,:).*z_train)+b2);
61         end
62
63         % Back propagation
64         for j = 1:K
65             cost(k) = cost(k) + ((output(i,j) - train_y(i,j))^2);
66         end
67         cost(k) = 0.5*sqrt(cost(k))/K;
68         for j = 1:K
69             for h = 1:H
70                 delta_w2(j,h) = -alpha*(train_y(i,j)-output(i,j))*output(i,j)*(1-output(i,j))*z_train(h);
71             end
72             delta_b2 = -alpha*(train_y(i,j)-output(i,j))*output(i,j)*(1-output(i,j));
73         end
74
75         for h = 1:H
76             for j = 1:size(train_x,2)
77                 s = 0;
78                 for g = 1:K
79                     s = s + (train_y(i,g) - output(i,g))*w2(g,h);
80                 end

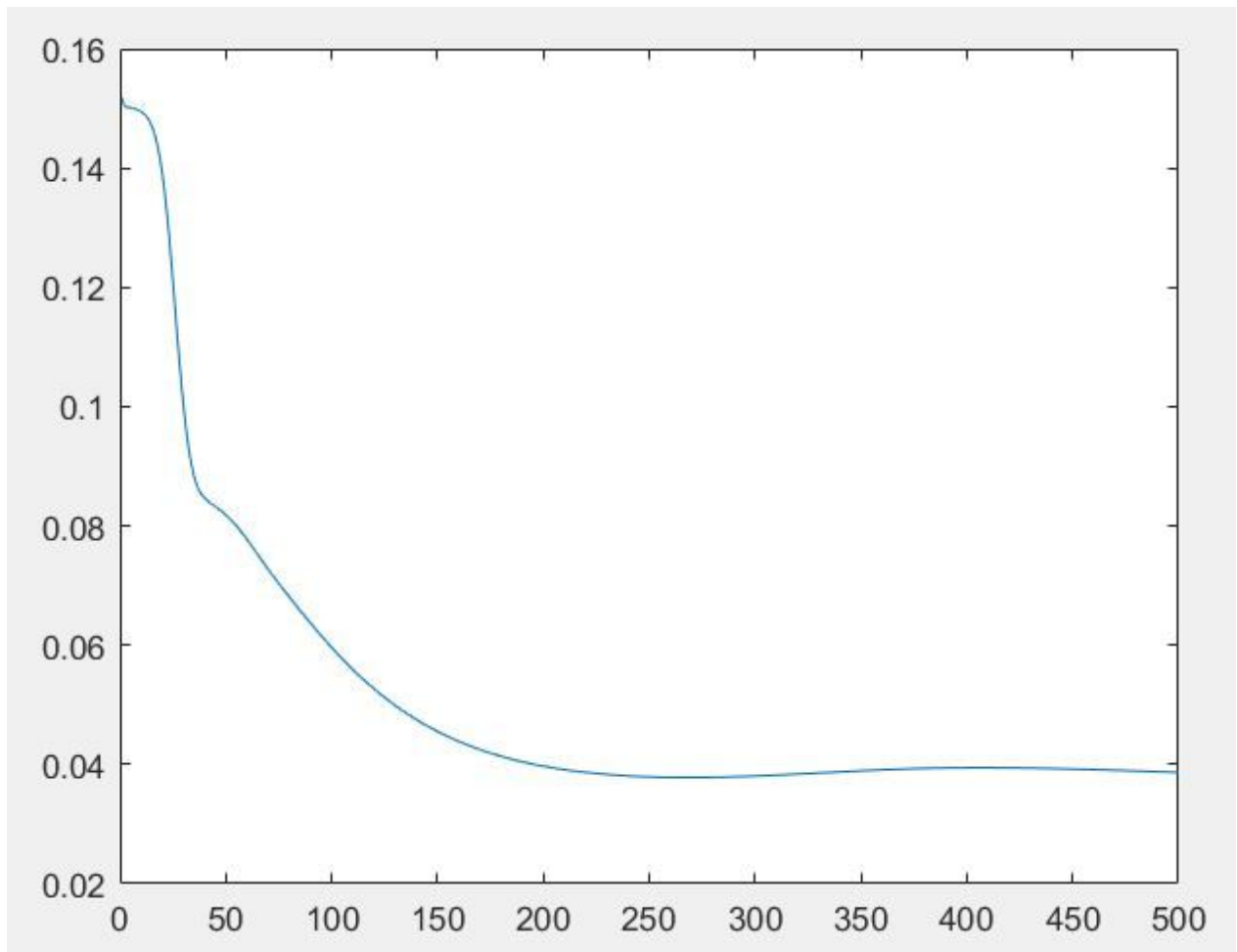
```

```

80         end
81         delta_w1(h,j) = -alpha*s*z_train(h)*(1-z_train(h))*train_x(i,j);
82     end
83     delta_b1 = -alpha*s*z_train(h)*(1-z_train(h));
84 end
85 % Weight update:
86 for j = 1:K
87     for h = 1:H
88         w2(j,h) = w2(j,h) - delta_w2(j,h);
89     end
90     b2 = b2 - delta_b2;
91 end
92
93     for h = 1:H
94         for j = 1:size(train_x,2)
95             w1(h,j) = w1(h,j) - delta_w1(h,j);
96         end
97         b1 = b1 - delta_b1;
98     end
99 end
100 end
101
102 % Training :
103 z_test = zeros(1,H);
104 test_output = zeros(45,K);
105 for p = 1:size(test_x,1)
106     for h = 1:H
107         z_test(h) = sigmoid(sum(w1(h, :).*test_x(p, :)) + b1);
108     end
109     for i = 1:K
110         y_pred(p, i) = sigmoid(sum(w2(i, :).*z_test) + b2);
111     end
112 end
113
114 pl = zeros(1,size(y_pred,1));
115 pa = zeros(1,size(y_pred,1));
116 for i1 = 1:size(y_pred,1)
117     [~,pl(i1)] = max(y_pred(i1,:));
118     pa(i1) = test_y(i1,:);
119 end
120 [cm,~] = confusionmat(pa,pl);
121
122 diagonal = 0;
123 for i2 = 1:3
124     diagonal = diagonal + cm(i2,i2);
125 end
126 accuracy = diagonal/sum(sum(cm));
127 plot(cost)
128 end

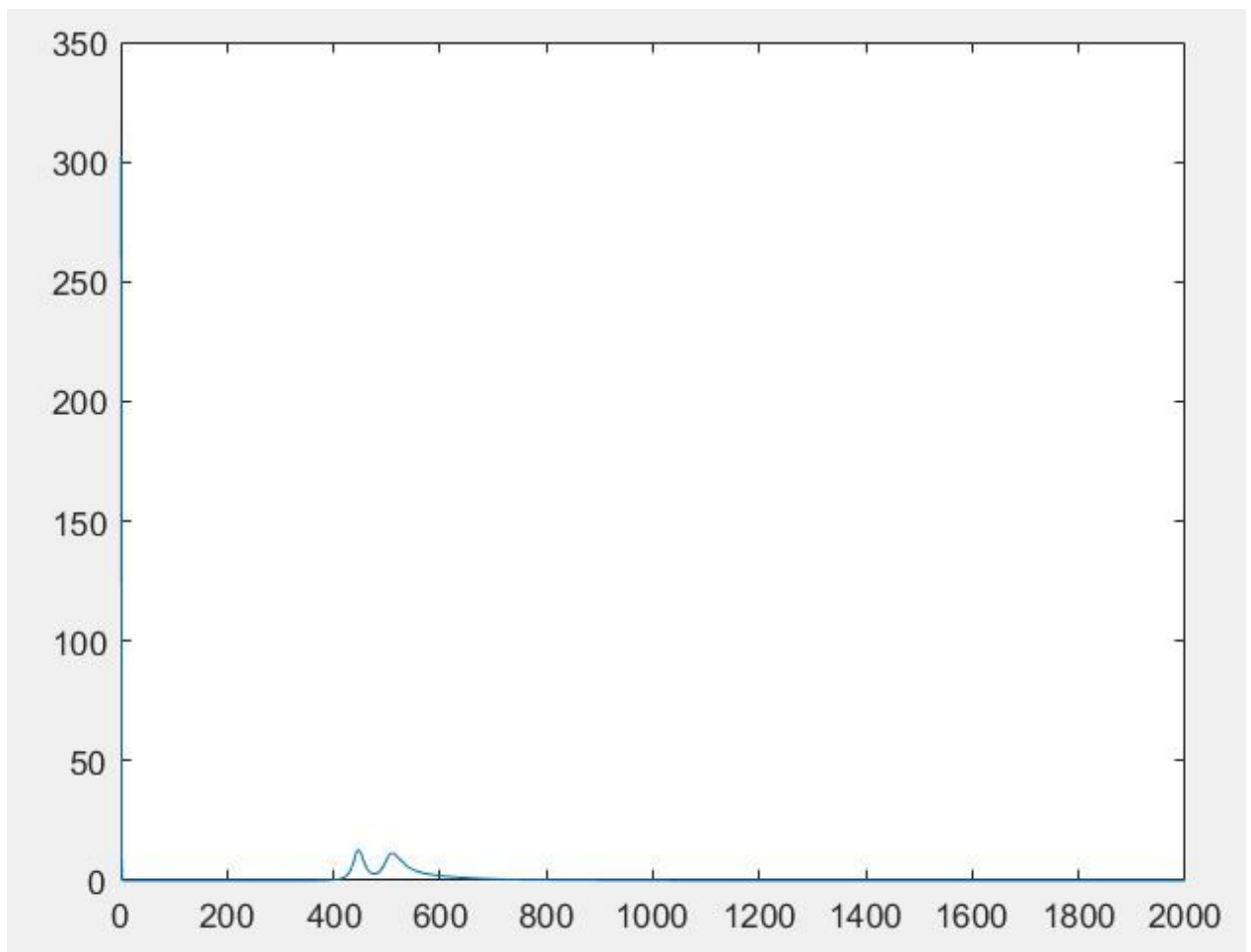
```

### Cost Function vs iterations



### Question 3:

**Cost**



**Code**



```

1  clc;
2  clear;
3  close all;
4  data = xlsread('dataset.xlsx');
5  data = data(randperm(size(data,1)),:);
6  X = data(:,(1:7));
7  X = normalize(X);
8  Y = data(:,8);
9
10 % number of hidden neurons
11 H1 = 5;
12 H2 = 3;
13
14 alpha = 0.5; % learning rate
15 mf = 0.001; % Momentum factor
16 iter = 2000;
17 K = 3; % No. of output neurons = 3
18
19 % Sigmoid function definition
20 sigmoid = @(x) 1./(1 + exp(-x));
21
22 % Dividing data into test and training : 70-30 cross validation
23 train_x = X((1:105),:);
24 tr_y = Y((1:105),:);
25 train_y = zeros(105,3);
26 test_x = X((106:150),:);
27 test_y = Y((106:150),:);
28 [M, N] = size(train_x);
29 [P, Q] = size(test_x);
30
31 for i = 1:length(tr_y)
32     if (Y(i) == 1)
33         train_y(i,:) = [1,0,0];
34     elseif (Y(i) == 2)
35         train_y(i,:) = [0,1,0];
36     elseif (Y(i) == 3)
37         train_y(i,:) = [0,0,1];
38     end
39 end
40
41 % initializing random values of weight and bias between -0.01 and +0.01
42 rmin = -0.01;
43 rmax = 0.01;
44 w1 = rmin + rand(size(train_x,2)+1,H1)*(rmax-rmin);
45 w2 = rmin + (rmax-rmin)*rand(H1+1,H2);
46 w3 = rmin + (rmax-rmin)*rand(H2+1,K);
47 b = 1;
48
49

```

```

50 % Training :
51 train_x = [b*ones(M, 1) train_x];
52 Dw1 = zeros(N+1, H1);
53 Dw2 = zeros(H1+1, H2);
54 Dw3 = zeros(H2+1, 3);
55 cost = zeros(iter, 1);
56
57 for k = 1:iter
58     % Forward Propagation :
59     z1 = [ones(M,1) sigmoid(train_x*w1 + b)];
60     z2 = [ones(M,1) sigmoid(z1*w2 + b)];
61     y = sigmoid(z2*w3);
62
63     % Back Propagation :
64     cost(k) = mean(sum(train_y - y).^2);
65     df = y.*(1-y);
66     d3 = df.*(train_y - y);
67     Dw3 = (alpha/N)*d3'*z2;
68     w3 = (1+mf)*w3 + Dw3';
69
70     df = z2.*(1-z2);
71     d2 = df.*(d3*w3');
72     d2 = d2(:, 2:end);
73     Dw2 = (alpha/N)*d2'*z1;
74     w2 = (1+mf)*w2 + Dw2';
75
76     df = z1.*(1-z1);
77     d1 = df.*(d2*w2');
78     d1 = d1(:, 2:end);
79     Dw1 = (alpha/N)*d1'*train_x;
80     w1 = (1+mf)*w1 + Dw1';
81 end
82
83 % Testing :
84 test_x = [ones(size(test_x,1),1) test_x];
85 z1_test = [ones(size(test_x,1),1) sigmoid(test_x*w1 + b)];
86 z2_test = [ones(size(test_x,1),1) sigmoid(z1_test*w2 + b)];
87 % z1_test = sigmoid(test_x*w1 + b);
88 % z2_test = sigmoid(z1_test*w2 + b);
89 y_output = sigmoid(z2_test*w3);
90
91 pl = zeros(1,size(y_output,1));
92 pa = zeros(1,size(y_output,1));
93 for i1 = 1:size(y_output,1)
94     [~,pl(i1)] = max(y_output(i1,:));
95     pa(i1) = test_y(i1,:);
96 end
97 [cm,~] = confusionmat(pa,pl);
98
99 diagonal = 0;
100 for i2 = 1:3
101     diagonal = diagonal + cm(i2,i2);

```



```

101     diagonal = diagonal + cm(i2,i2);
102 end
103 accuracy = diagonal/sum(sum(cm));
104 plot(cost)
105

```

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				

#### **Question 4:**

**Code**

```

1  % The input n is the number of neurons. To find n, Run 'grid_search.m'
2
3  function cm = rbfnn(n)
4  clc;
5  % clear;
6  close all;
7  data = xlsread('dataset.xlsx');
8  X = data(:,(1:7));
9  Y = data(:,8);
10 z = zeros(150,3);
11 % No. of output neurons = 3
12
13 for i = 1:length(Y)
14     if (Y(i) == 1)
15         z(i,:) = [1,0,0];
16     elseif (Y(i) == 2)
17         z(i,:) = [0,1,0];
18     elseif (Y(i) == 3)
19         z(i,:) = [0,0,1];
20     end
21 end
22
23 % Dividing data into test and training : 70-30 cross validation
24 traininput = [];
25 trainoutput = [];
26 testinput = [];
27 testoutput = [];
28
29 for j = 1:size(X,1)
30     if rand < 0.7
31         traininput = [traininput; X(j,:)];
32         trainoutput = [trainoutput; z(j,:)];
33     else
34         testinput = [testinput; X(j,:)];
35         testoutput = [testoutput; z(j,:)];
36     end
37 end
38
39 x = traininput;
40 y = trainoutput;
41 xt = testinput;
42 yt = testoutput;
43
44 [~, mu] = kmeans(x,n);
45
46 % Hidden layer eval
47 for i = 1:size(x,1)
48     for j = 1:size(mu,1)
49         h(i,j) = (norm( x(i,:) - mu(j,:)))^3;
50     end
51 end
52
53 % Weight eval

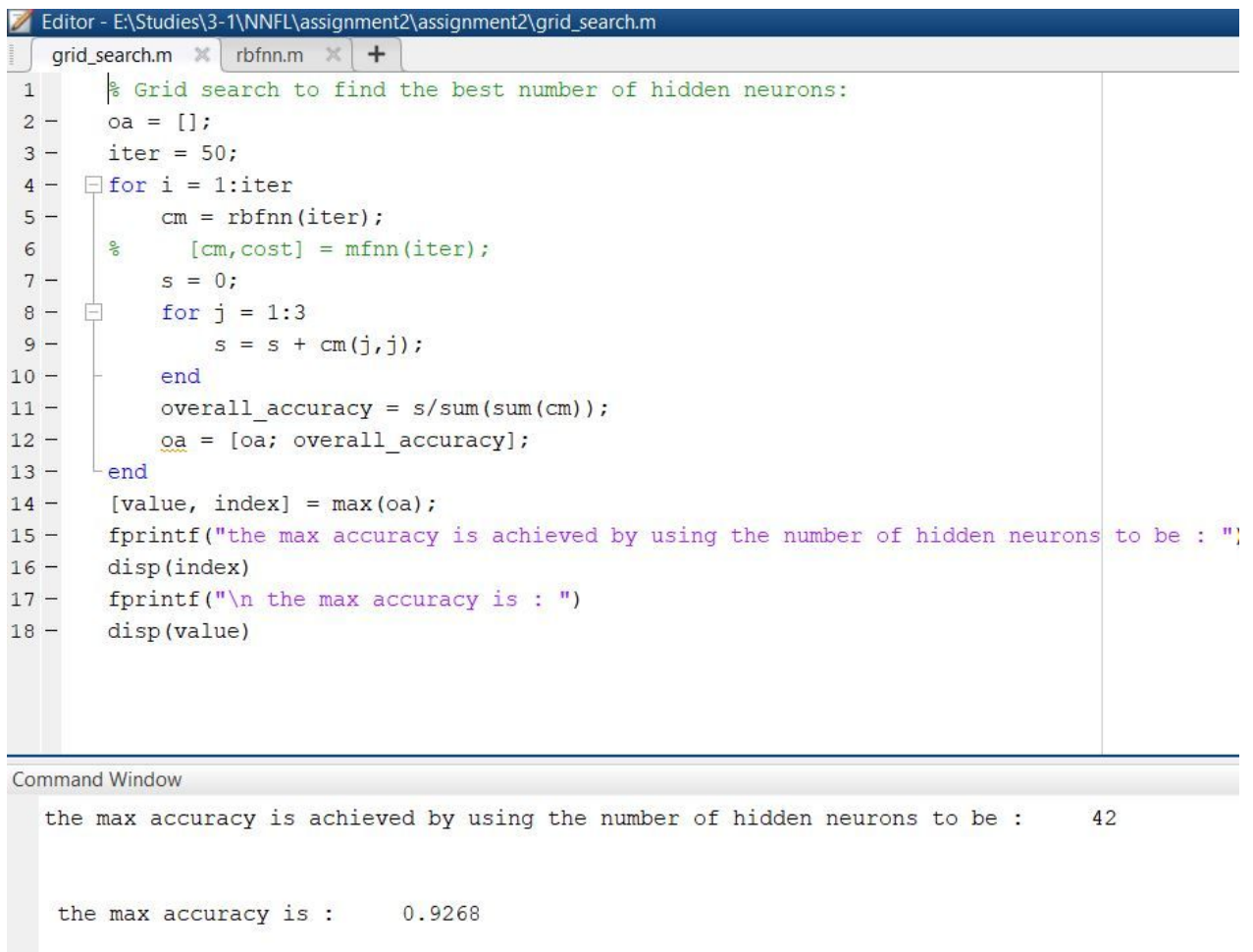
```

```

52
53 % Weight eval
54 W = pinv(h)*y;
55
56 % Test data eval
57
58 ▼ for i1 = 1:size(xt,1)
59     for j1 = 1:size(mu,1)
60         H(i1,j1) = (norm( xt(i1,:) - mu(j1,:)))^3;
61     end
62 end
63 output = H*W;
64 pl = zeros(1,40);
65 pa = zeros(1,40);
66
67 ▼ for i1 = 1:size(output,1)
68     [~,pl(i1)] = max(output(i1,:));
69     [~,pa(i1)] = max(yt(i1,:));
70 end
71
72 [cm,~] = confusionmat(pa,pl);
73
74 % fprintf("The confusion matrix is : \n");
75 % disp(cm)
76 end
77
78
79

```

Optimal number of hidden neurons :



The image shows a MATLAB Editor window with a script named `grid_search.m` and a Command Window below it. The script performs a grid search to find the best number of hidden neurons by comparing the accuracy of Radial Basis Function Neural Networks (rbfnn) and Multilayer Feedforward Neural Networks (mfnn) across different hidden neuron counts (1, 2, 3).

```
1  % Grid search to find the best number of hidden neurons:
2  oa = [];
3  iter = 50;
4  for i = 1:iter
5      cm = rbfnn(iter);
6      % [cm, cost] = mfnn(iter);
7      s = 0;
8      for j = 1:3
9          s = s + cm(j,j);
10     end
11     overall_accuracy = s/sum(sum(cm));
12     oa = [oa; overall_accuracy];
13 end
14 [value, index] = max(oa);
15 fprintf("the max accuracy is achieved by using the number of hidden neurons to be : ");
16 disp(index)
17 fprintf("\n the max accuracy is : ")
18 disp(value)
```

The Command Window displays the output of the script:

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

the max accuracy is :    0.9268
```

### **Question 5:**

#### **Code**

```

1  clc;
2  clear;
3  close all;
4  data = xlsread('dataset.xlsx');
5  data = data(randperm(size(data,1)),:);
6  X = data(:,(1:7));
7  X = normalize(X);
8  Y = data(:,8);
9
10 alpha = 0.5; % learning rate
11 mf = 0.001; % Momentum factor
12 iter = 2000;
13 K = 3; % No. of output neurons = 3
14
15 % Sigmoid function definition
16 sigmoid = @(x) 1./(1 + exp(-x));
17
18 % Dividing data into test and training : 70-30 cross validation
19 train_x = X((1:105),:);
20 tr_y = Y((1:105),:);
21 train_y = zeros(105,3);
22 test_x = X((106:150),:);
23 test_y = Y((106:150),:);
24 [M, N] = size(train_x);
25 [P, Q] = size(test_x);
26
27 H1 = 10; % Number of hidden neurons in MFNN
28
29 for i = 1:length(tr_y)
30     if (Y(i) == 1)
31         train_y(i,:) = [1,0,0];
32     elseif (Y(i) == 2)
33         train_y(i,:) = [0,1,0];
34     elseif (Y(i) == 3)
35         train_y(i,:) = [0,0,1];
36     end
37 end
38
39 % initializing random values of weight and bias between -0.01 and +0.01
40 rmin = -0.01;
41 rmax = 0.01;
42 w1 = rmin + rand(size(train_x,2)+1,H1)*(rmax-rmin);
43 w2 = rmin + (rmax-rmin)*rand(H1+1,K);
44 b = 1;
45
46
47 % -----TRAINING-----
48 train_x = [b*ones(M, 1) train_x];
49 Dw1 = zeros(N+1, H1);
50 Dw2 = zeros(H1+1, K);
51 cost = zeros(iter, 1);
52
53

```

```

53
54 % MFNN Part
55 ▼ for k = 1:iter
56     % Forward Propagation
57     z = [ones(M,1) sigmoid(train_x*w1 + b)];
58     y = sigmoid(z*w2);
59
60     % Backward Propagation
61     cost(k) = mean(sum(train_y - y).^2);
62     df = y.*(1-y);
63     d2 = df.*(train_y - y);
64     Dw2 = (alpha/N)*d2'*z;
65     w2 = (1+mf)*w2 + Dw2';
66
67     df = z.*(1-z);
68     d1 = df.*(d2*w2');
69     d1 = d1(:, 2:end);
70     Dw1 = (alpha/N)*d1'*train_x;
71     w1 = (1+mf)*w1 + Dw1';
72 end
73
74 % RBFNN Part
75
76 n = 10; % Number of hidden neurons in RBFNN(no. of cluster centers)
77 [~, mu] = kmeans(y,n);
78
79 % Hidden layer eval
80 ▼ for i = 1:size(y,1)
81     for j = 1:size(mu,1)
82         h(i,j) = (norm( y(i,:) - mu(j,:)))^3;
83     end
84 end
85
86 % Weight eval
87 W = pinv(h)*train_y;
88
89 % -----TESTING-----
90
91 % MFNN Forward prop:
92 test_x = [ones(size(test_x,1),1) test_x];
93 z_test = [ones(size(test_x,1),1) sigmoid(test_x*w1 + b)];
94 mfnn_output = sigmoid(z_test*w2);
95
96 % RBFNN Forward prop
97 ▼ for i1 = 1:size(mfnn_output,1)
98     for j1 = 1:size(mu,1)
99         H(i1,j1) = (norm( mfnn_output(i1,:) - mu(j1,:)))^3;
100     end
101 end
102 final_output = H*W;
103

```

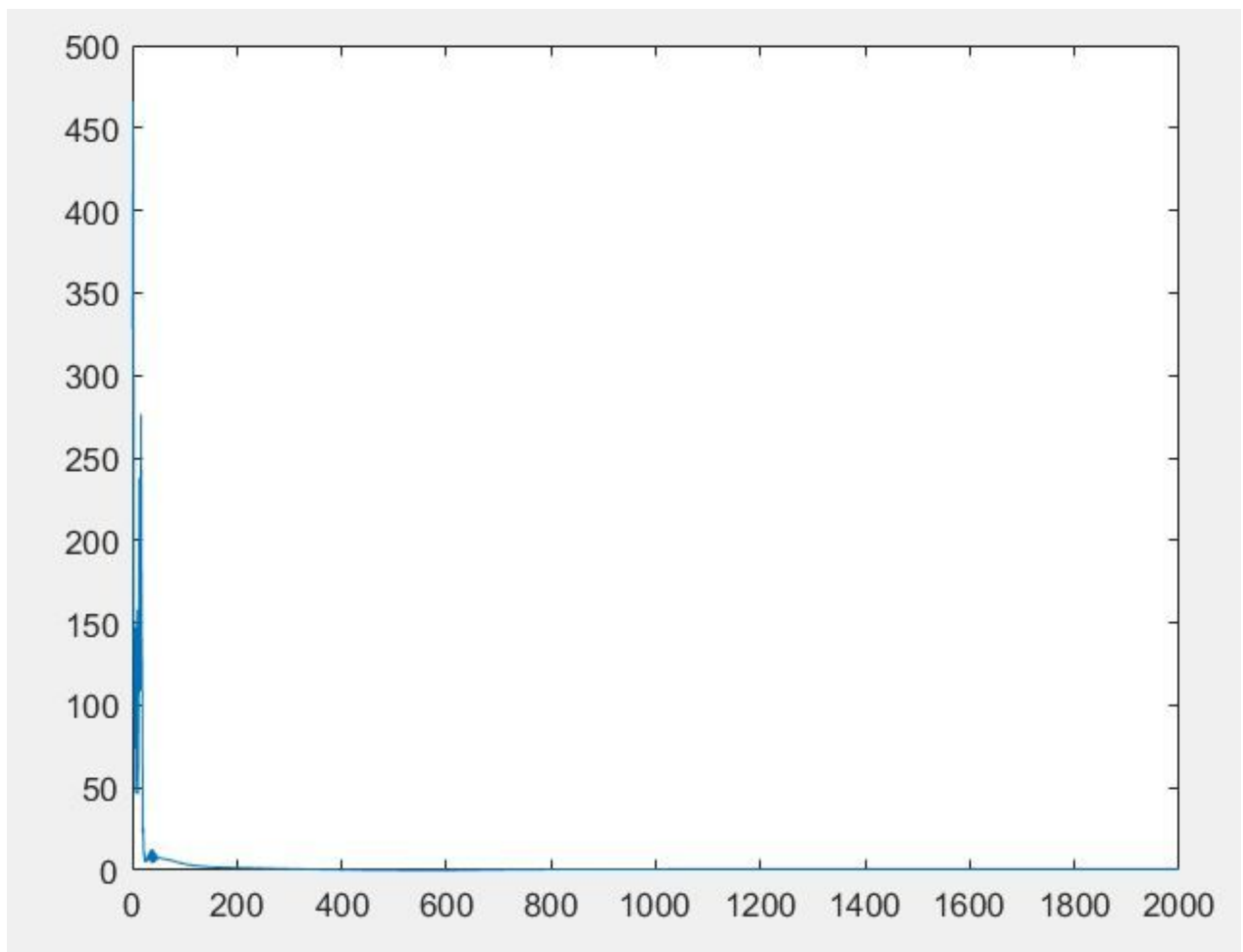


```

103
104 % Accuracy and confusion matrix eval
105 pl = zeros(1,size(final_output,1));
106 pa = zeros(1,size(final_output,1));
107 for i1 = 1:size(final_output,1)
108     [~,pl(i1)] = max(final_output(i1,:));
109     pa(i1) = test_y(i1,:);
110 end
111 [cm,~] = confusionmat(pa,pl);
112
113 diagonal = 0;
114 for i2 = 1:3
115     diagonal = diagonal + cm(i2,i2);
116 end
117 accuracy = diagonal/sum(sum(cm));
118 plot(cost)
119
120

```

**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				