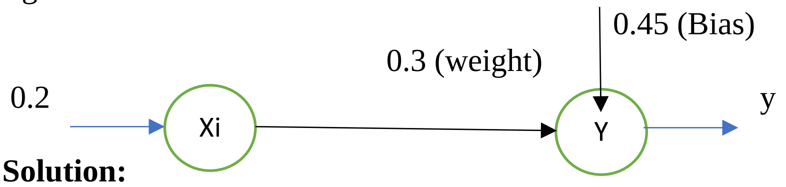
Practical No. 1

A. Design a simple linear neural network model.
B. Calculate the output of neural net using both binary and bipolar sigmoidal function.

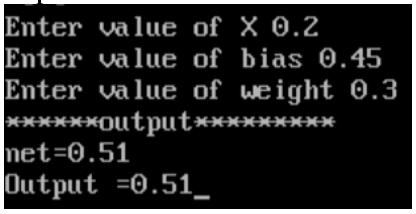
A. Design a simple linear neural network model.
Problem: Create C++ program to calculate net input to the output neuron for the network shown in figure below.



Solution:

```
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr();
float x,b,w,net;
float out;
cout<<"Enter value of X";
cin>>x;
cout<<"Enter value of bias";
cin>>b;
cout<<"Enter value of weight";
cin>>w;
net=(w*x+b);
cout<<"******output*********";
cout<<"\nnet="<<net<<endl;
if(net<0)
{out=0;}
else if((net>=0)&&(net<=1))
{out=net;}
else
out=1;
cout<<"Output ="<<out;
getch();
}
```

O/p



```
Enter value of X 0.2
Enter value of bias 0.45
Enter value of weight 0.3
******output*********
net=0.51
Output =0.51_
```

B. Calculate the output of neural net using both binary and bipolar sigmoidal function.
For the network shown in the figure, calculate the net input to output neuron.

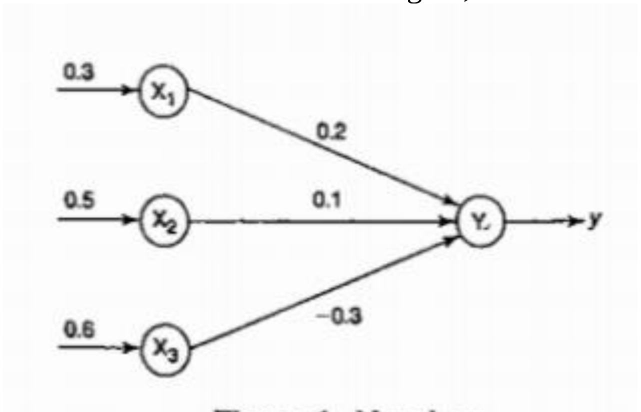


Figure 1 Neural net.

Solution :The given neural net consist of three input neurons and one output neuron. The inputs and weight are

[x1, x2, x3] = [0.8, 0.6 0.4]
[w1, w2, w3] = [0.1, 0.3, -0.2]

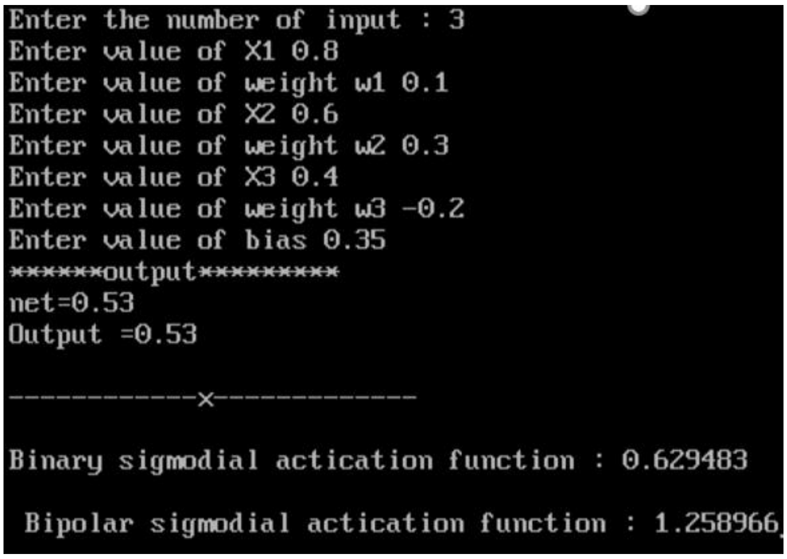
The net input can be calculated as

Yin = x1w1+x2w2+x3w3
= 0.8*0.1+0.6*0.3+0.4*(-0.2)
= 0.53

Code :

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main()
{
clrscr();
int i=0;
float x[10],b,w[10],net,n,sumxw=0,sigmo,e=2.71828;
float out;
cout<<"Enter the number of input : ";
cin>>n;
for (i=0;i<n;i++)
{
cout<<"Enter value of X"<<i+1;
cin>>x[i];
cout<<"Enter value of weight w"<<i+1;
cin>>w[i];
}
cout<<"Enter value of bias";
cin>>b;
for (i=0;i<n;i++)
{
        sumxw=sumxw+w[i]*x[i];
}
        net=(sumxw+b);
cout<<"******output*********";
cout<<"\nnet="<<net<<endl;
if(net<0)
{out=0;}
else if((net>=0)&&(net<=1))
{out=net;}
else
out=1;
cout<<"Output ="<<out;
cout<<"\n\n------------x-------------";
cout<<"\n\nBinary sigmodial actication function : "<<(1/(1+(pow(e,-net))));
cout<<"\n\nBipolar sigmodial actication function : "<<(2/(1+(pow(e,-net))));
getch();
}
```

Output :



```
Enter the number of input : 3
Enter value of X1 0.8
Enter value of weight w1 0.1
Enter value of X2 0.6
Enter value of weight w2 0.3
Enter value of X3 0.4
Enter value of weight w3 -0.2
Enter value of bias 0.35
******output*********
net=0.53
Output =0.53

------------x-------------

Binary sigmodial actication function : 0.629483

 Bipolar sigmodial actication function : 1.258966
```

Practical No. 2

A. Generate AND/NOT function using McCulloch-Pitts neural net.
B. Generate XOR function using McCulloch-Pitts neural net.

A. Generate AND/NOT function using McCulloch-Pitts neural net.

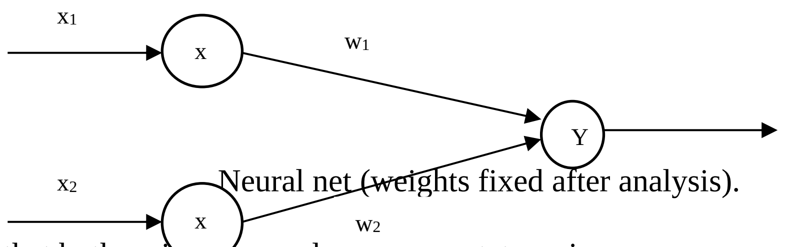
Solution:

In the case of ANDNOT function, the response is true if the first input is true and the second input is false. For all the other variations , the response is false. The truth table for ANDNOT function is given in Table below.

Truth Table:

x_1	x_2	y
0	0	0
0	1	0
1	0	1
1	1	0

The given function gives an output only when $x_1 = 1$ and $x_2 = 0$. The weights have to be decided only after the analysis. The net can be represent as shown in figure below:



Neural net (weights fixed after analysis).

Case 1: Assume that both weights w_1 and [illegible] tatory, i.e.,

$$w_1 = w_2 = 1$$

Then for the four inputs calculate the net input using

$$y_{ij} = x_1w_1 + x_2w_2$$

For inputs

(1, 1), $y_{ij} = 1 \text{ x } 1 + l \text{ x } 1 = 2$
(1, 0), $y_{ij} = 1 \text{ x } 1 + 0 \text{ x } 1 = 1$
(0, 1), $y_{ij} = 0 \text{ x } 1 + 1 \text{ x } 1 = 1$
(0, 0), $y_{ij} = 0 \text{ x } 1 + 0 \text{ x } 1 = 0$

From the calculated net inputs, it is not possible to fire the neuron form input (1, 0) only. Hence, J-. weights are not suitable.

Assume one weight as excitatory and the other as inhibitory, i.e.,

$$w_1 = 1, w_2 = -1$$

Now calculate the net input. For the inputs

(1,1), $y_{in} = 1 \text{ x } 1 + 1 \text{ x } -1 = 0$
(1,0), $y_{in} = 1 \text{ x } 1 + 0 \text{ x } -1 = 1$
(0,1), $y_{in} = 0 \text{ x } 1 + 1 \text{ x } -1 = -1$
(0, 0), $y_{in} = 0 \text{ x } 1 + 0 \text{ x } -1 = 0$

From the calculated net inputs, now it is possible to fire the neuron for input (1, 0) only by fixing a threshold of 1, i.e., $\theta \geq 1$ for Y unit. Thus,

$$w_1 = 1, w_2 = -1; \theta \geq 1$$

Note: The value is calculated using the following:

$\theta \geq nw - p$
$\theta \geq 2 \text{ x } 1 - 1$
$\theta \geq 1$

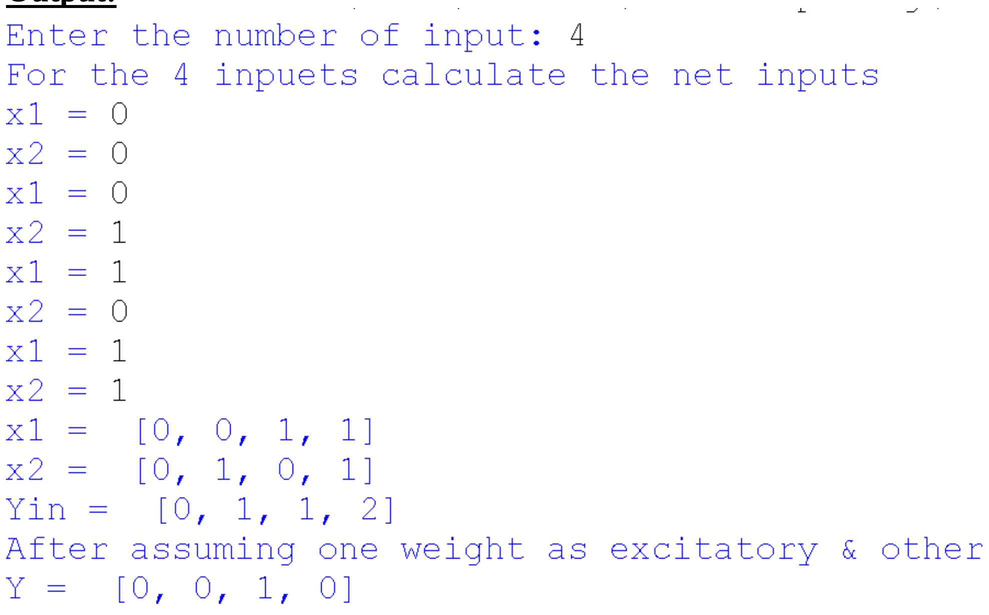
Thus, the output of neuron Y can be written as

$$y = f(y_{in}) = \begin{cases} 0 \text{ if } y_{in} \geq 1 \\ 1 \text{ if } y_{in} < 1 \end{cases}$$

Code:

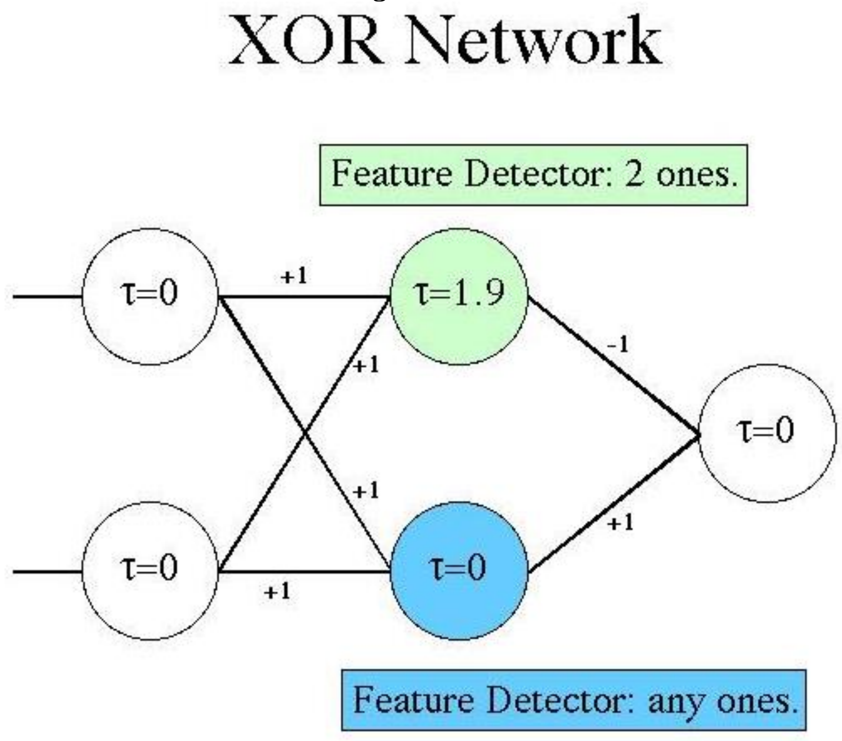
```
import numpy
num_ip=int(input("Enter the number of input: "))
w1 = 1
w2 = 1
print("For the",num_ip,"inpuets calculate the net inputs")
x1 = []
x2 = []
for j in range(0, num_ip):
    ele1 = int(input("x1 = "))
    ele2 = int(input("x2 = "))
    x1.append(ele1)
    x2.append(ele2)
print("x1 = ",x1)
print("x2 = ",x2)
n = x1 * w1
m = x2 * w2

Yin = []
for i in range(0, num_ip):
    Yin. append(n[i] + m[i])
print("Yin = ",Yin)
Yin = []
for i in range(0, num_ip):
    Yin. append(n[i] - m[i])
print("After assuming one weight as excitatory & other")
Y = []
for i in range(0, num_ip):
    if(Yin[i]>=1):
        ele=1
        Y.append(ele)
    if(Yin[i]<1):
        ele=0
        Y.append(ele)
print("Y = ",Y)
```

Output:



```
Enter the number of input: 4
For the 4 inpuets calculate the net inputs
x1 = 0
x2 = 0
x1 = 0
x2 = 1
x1 = 1
x2 = 0
x1 = 1
x2 = 1
x1 =  [0, 0, 1, 1]
x2 =  [0, 1, 0, 1]
Yin =  [0, 1, 1, 2]
After assuming one weight as excitatory & other
Y =  [0, 0, 1, 0]
```

B. Generate XOR function using McCulloch-Pitts neural net.



Code :

```
import math
import numpy
import random
# note that this only works for a single layer of depth
INPUT_NODES = 2
OUTPUT_NODES = 1
HIDDEN_NODES = 2
# 15000 iterations is a good point for playing with learning rate
MAX_ITERATIONS = 130000
# setting this too low makes everything change very slowly, but too high
# makes it jump at each and every example and oscillate. I found .5 to be good
LEARNING_RATE = .2
print ("Neural Network Program")
class network:
    def __init__(self, input_nodes, hidden_nodes, output_nodes, learning_rate):
        self.input_nodes = input_nodes
        self.hidden_nodes = hidden_nodes
        self.output_nodes = output_nodes
        self.total_nodes = input_nodes + hidden_nodes + output_nodes
        self.learning_rate = learning_rate
        # set up the arrays
        self.values = numpy.zeros(self.total_nodes)
        self.expectedValues = numpy.zeros(self.total_nodes)
        self.thresholds = numpy.zeros(self.total_nodes)
        # the weight matrix is always square
        self.weights = numpy.zeros((self.total_nodes, self.total_nodes))
        # set random seed! this is so we can experiment consistently
        random.seed(10000)
        # set initial random values for weights and thresholds
        # this is a strictly upper triangular matrix as there is no feedback
        # loop and there inputs do not affect other inputs
        for i in range(self.input_nodes, self.total_nodes):
            self.thresholds[i] = random.random() / random.random()
            for j in range(i + 1, self.total_nodes):
                self.weights[i][j] = random.random() * 2

    def process(self):
        # update the hidden nodes
        for i in range(self.input_nodes, self.input_nodes + self.hidden_nodes):
            # sum weighted input nodes for each hidden node, compare threshold, apply sigmoid
            W_i = 0.0
            for j in range(self.input_nodes):
                W_i += self.weights[j][i] * self.values[j]
            W_i -= self.thresholds[i]
            self.values[i] = 1 / (1 + math.exp(-W_i))
        # update the output nodes
        for i in range(self.input_nodes + self.hidden_nodes, self.total_nodes):
            # sum weighted hidden nodes for each output node, compare threshold, apply sigmoid
            W_i = 0.0
            for j in range(self.input_nodes, self.input_nodes + self.hidden_nodes):
                W_i += self.weights[j][i] * self.values[j]
            W_i -= self.thresholds[i]
            self.values[i] = 1 / (1 + math.exp(-W_i))
    def processErrors(self):
        sumOfSquaredErrors = 0.0
        # we only look at the output nodes for error calculation
        for i in range(self.input_nodes + self.hidden_nodes, self.total_nodes):
            error = self.expectedValues[i] - self.values[i]
            #print error
            sumOfSquaredErrors += math.pow(error, 2)
            outputErrorGradient = self.values[i] * (1 - self.values[i]) * error
            #print outputErrorGradient
            # now update the weights and thresholds
            for j in range(self.input_nodes, self.input_nodes + self.hidden_nodes):
                # first update for the hidden nodes to output nodes (1 layer)
                delta = self.learning_rate * self.values[j] * outputErrorGradient
                #print delta
                self.weights[j][i] += delta
                hiddenErrorGradient = self.values[j] * (1 - self.values[j]) * outputErrorGradient *
self.weights[j][i]
                # and then update for the input nodes to hidden nodes
                for k in range(self.input_nodes):
                    delta = self.learning_rate * self.values[k] * hiddenErrorGradient
                    self.weights[k][j] += delta
                # update the thresholds for the hidden nodes
                delta = self.learning_rate * -1 * hiddenErrorGradient
                #print delta
                self.thresholds[j] += delta
            # update the thresholds for the output node(s)
            delta = self.learning_rate * -1 * outputErrorGradient
```

```
                self.thresholds[i] += delta
            return sumOfSquaredErrors

class sampleMaker:
    def __init__(self, network):
        self.counter = 0
        self.network = network
    def setXor(self, x):
        if x == 0:
            self.network.values[0] = 1
            self.network.values[1] = 1
            self.network.expectedValues[4] = 0
        elif x == 1:
            self.network.values[0] = 0
            self.network.values[1] = 1
            self.network.expectedValues[4] = 1
        elif x == 2:
            self.network.values[0] = 1
            self.network.values[1] = 0
            self.network.expectedValues[4] = 1
        else:
            self.network.values[0] = 0
            self.network.values[1] = 0
            self.network.expectedValues[4] = 0
    def setNextTrainingData(self):
        self.setXor(self.counter % 4)
        self.counter += 1
# start of main program loop, initialize classes
net = network(INPUT_NODES, HIDDEN_NODES, OUTPUT_NODES, LEARNING_RATE)
samples = sampleMaker(net)

for i in range(MAX_ITERATIONS):
    samples.setNextTrainingData()
    net.process()
    error = net.processErrors()
    # prove that we got the right answers(ish)!
    if i > (MAX_ITERATIONS - 5):
        output = (net.values[0], net.values[1], net.values[4], net.expectedValues[4], error)
        print (output)
# display final parameters
print (net.weights)
print (net.thresholds)
```

Output :

```
Neural Network Program
(1.0, 1.0, 0.014929208005738836, 0.0, 0.00022288125167860235)
(0.0, 1.0, 0.9857295047367691, 1.0, 0.00020364703505789487)
(1.0, 0.0, 0.9856250336871464, 1.0, 0.00020663965649567642)
(0.0, 0.0, 0.016607849913409613, 0.0, 0.0002758206787463397)
[[ 0.          0.          5.75231929 -6.31595212  0.        ]
 [ 0.          0.         -5.97540997  6.18899346  0.        ]
 [ 0.          0.          0.          1.93019719  9.6814855 ]
 [ 0.          0.          0.          0.          9.57128428]
 [ 0.          0.          0.          0.          0.        ]]
[0.         0.         3.1933078  3.44466182 4.75885176]
```

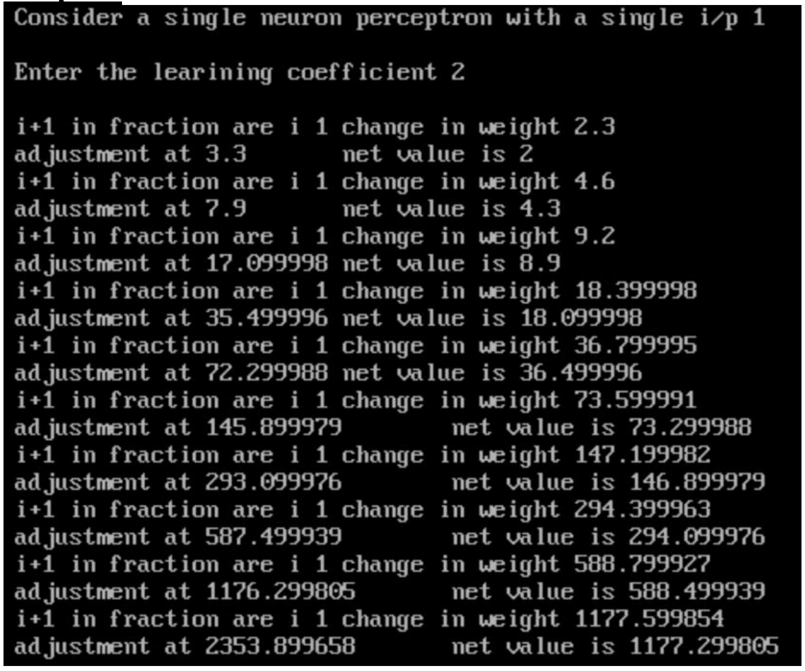
Practical No. 3

A. Write a program to implement Hebb's rule.

Solution :

```
#include<iostream.h>
#include<conio.h>
void main()
{
float n,w,x=1,net,d,div,a,at=0.3,dw;
clrscr();
cout<<"Consider a single neuron perceptron with a single i/p";
cin>>w;
cout<<"\nEnter the learining coefficient";
cin>>d;
for(int i=0; i<10;i++)
{
net=x+w;
if(w<0)
a=0;
else
a=1;
div=at+a+w;
w=w+div;
cout<<"\ni+1 in fraction are i "<<a<<"\tchange in weight "<<div<<"\nadjustment at "<<w<< "\tnet
value is "<<net;
}
getch();
}
```

Output :



```
Consider a single neuron perceptron with a single i/p 1

Enter the learining coefficient 2

i+1 in fraction are i 1 change in weight 2.3
adjustment at 3.3       net value is 2
i+1 in fraction are i 1 change in weight 4.6
adjustment at 7.9       net value is 4.3
i+1 in fraction are i 1 change in weight 9.2
adjustment at 17.099998 net value is 8.9
i+1 in fraction are i 1 change in weight 18.399998
adjustment at 35.499996 net value is 18.099998
i+1 in fraction are i 1 change in weight 36.799995
adjustment at 72.299988 net value is 36.499996
i+1 in fraction are i 1 change in weight 73.599991
adjustment at 145.899979        net value is 73.299988
i+1 in fraction are i 1 change in weight 147.199982
adjustment at 293.099976        net value is 146.899979
i+1 in fraction are i 1 change in weight 294.399963
adjustment at 587.499939        net value is 294.099976
i+1 in fraction are i 1 change in weight 588.799927
adjustment at 1176.299805       net value is 588.499939
i+1 in fraction are i 1 change in weight 1177.599854
adjustment at 2353.899658       net value is 1177.299805
```

Python Code:

```
#Learning Rules #
import math

def computeNet(input, weights):
	net = 0
	for i in range(len(input)):
		net = net + input[i]*weights[i]
	print ("NET:")
	print (net)
	return net
#print ("NET:")
	   #print net
	   #return net

def computeFNetBinary(net):
	f_net = 0
	if(net>0):
		f_net = 1
	if(net<0):
		f_net = -1
	return f_net

def computeFNetCont(net):
	f_net = 0
	f_net = (2/(1+math.exp(-net)))-1
	return f_net

def hebb(f_net):
	return f_net

def perceptron(desired, actual):
	   return (desired-actual)

def widrow(desired, actual):
	   return (desired-actual)

def adjustWeights(inputs, weights, last, binary, desired, rule):
	c = 1
	if(last):
		print ("COMPLETE")
		return
	current_input = inputs[0]
	inputs = inputs[1:]
	if desired :
		current_desired = desired[0]
		desired = desired[1:]
	if len(inputs) == 0:
		last = True
	net = computeNet(current_input, weights)
	if(binary):
		f_net = computeFNetBinary(net)
```

```
    else:
        f_net = computeFNetCont(net)
    if rule == "hebb":
      r = hebb(f_net)
    elif rule == "perceptron":
        r = perceptron(current_desired, f_net)
    elif rule == "widrow":
        r = widrow(current_desired, net)
    del_weights = []
    for i in range(len(current_input)):
        x = (c*r)*current_input[i]
        del_weights.append(x)
        weights[i] = x
    print("NEW WEIGHTS:")
    print(weights)
    adjustWeights(inputs, weights, last, binary, desired, rule)

if __name__=="__main__":
    #total_inputs = (int)raw_input("Enter Total Number of Inputs)
    #vector_length = (int)raw_input("Enter Length of vector)
    total_inputs = 3
    vector_length = 4
    #for i in range(vector_length):
    #weight.append(raw_input("Enter Initial Weight:")
    weights = [1,-1,0,0.5]
    inputs = [[1,-2,1.5,0],[1,-0.5,-2,-1.5],[0,1,-1,1.5]]
    desired = [1,2,1,-1]
    print("BINARY HEBB!")
    adjustWeights(inputs, [1,-1,0,0.5], False, True, None, "hebb")
    print("CONTINUOUS HEBB!")
    adjustWeights(inputs, [1,-1,0,0.5], False, False, None, "hebb")
    print("PERCEPTRON!")
    adjustWeights(inputs, [1,-1,0,0.5], False, True, desired, "perceptron")
    print("WIDROW HOFF!")
    adjustWeights(inputs, [1,-1,0,0.5], False, True, desired, "widrow")
```

<u>Output :</u>

```
============= RESTART: C:\Users\Yogesh Patil\Downloads\raw.py =============
BINARY HEBB!
NET:
3.0
NEW WEIGHTS:
[1, -2, 1.5, 0]
NET:
-1.0
NEW WEIGHTS:
[-1, 0.5, 2, 1.5]
NET:
0.75
NEW WEIGHTS:
[0, 1, -1, 1.5]
COMPLETE
```

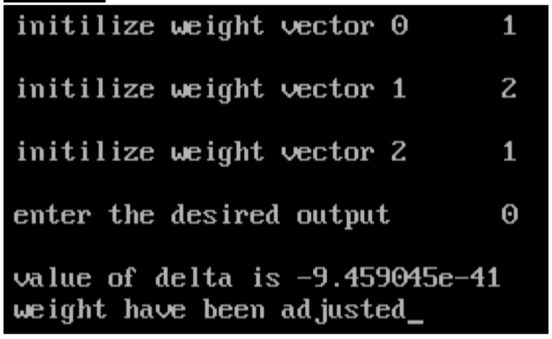
```
CONTINUOUS HEBB!
NET:
3.0
NEW WEIGHTS:
[0.9051482536448667, -1.8102965072897335, 1.3577223804673002, 0.0]
NET:
-0.905148253644867
NEW WEIGHTS:
[-0.42401264054072996, 0.21200632027036498, 0.8480252810814599, 0.6360189608110949]
NET:
0.31800948040554744
NEW WEIGHTS:
[0.0, 0.15767814164392502, -0.15767814164392502, 0.23651721246588753]
COMPLETE
PERCEPTRON!
NET:
3.0
NEW WEIGHTS:
[0, 0, 0.0, 0]
NET:
0.0
NEW WEIGHTS:
[2, -1.0, -4, -3.0]
NET:
-1.5
NEW WEIGHTS:
[0, 2, -2, 3.0]
COMPLETE
WIDROW HOFF!
NET:
3.0
NEW WEIGHTS:
[-2.0, 4.0, -3.0, -0.0]
NET:
2.0
NEW WEIGHTS:
[0.0, -0.0, -0.0, -0.0]
NET:
0.0
NEW WEIGHTS:
[0.0, 1.0, -1.0, 1.5]
COMPLETE
>>>
```

B. Write a program to implement of delta rule.

Solution :

```
#include<iostream.h>
#include<conio.h>
void main()
{
clrscr( );
float input[3],d,del,a,val[10],w[10],weight[3],delta;
for(int i=0;i < 3 ; i++)
{
cout<<"\n initilize weight vector "<<i<<"\t";
cin>>input[i];
}
cout<<"\n enter the desired output\t";
cin>>d;
do
{
del=d-a;
if(del<0)
for(i=0 ;i<3 ;i++)
w[i]=w[i]-input[i];
else if(del>0)
for(i=0;i<3;i++)
weight[i]=weight[i]+input[i];
for(i=0;i<3;i++)
{
val[i]=del*input[i];
weight[+1]=weight[i]+val[i];
}
cout<<"\n value of delta is "<<del;
cout<<"\n weight have been adjusted";
}while(del==0);
if(del==0)
cout<<"\n output is correct";
getch();
}
```

Output:



```
initilize weight vector 0      1

initilize weight vector 1      2

initilize weight vector 2      1

enter the desired output       0

value of delta is -9.459045e-41
weight have been adjusted_
```

Practical No. 3

A. Write a program for Back Propagation Algorithm.

Solution :

Python Code:

```
import math
import random
import sys

INPUT_NEURONS = 4
HIDDEN_NEURONS = 6
OUTPUT_NEURONS = 14

LEARN_RATE = 0.2 # Rho.
NOISE_FACTOR = 0.58
TRAINING_REPS = 10000
MAX_SAMPLES = 14

TRAINING_INPUTS = [[1, 1, 1, 0],
                [1, 1, 0, 0],
                [0, 1, 1, 0],
                [1, 0, 1, 0],
                [1, 0, 0, 0],
                [0, 1, 0, 0],
                [0, 0, 1, 0],
                [1, 1, 1, 1],
                [1, 1, 0, 1],
                [0, 1, 1, 1],
                [1, 0, 1, 1],
                [1, 0, 0, 1],
                [0, 1, 0, 1],
                [0, 0, 1, 1]]

TRAINING_OUTPUTS = [[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]]


class Example_4x6x16:
  def __init__(self, numInputs, numHidden, numOutput, learningRate, noise, epochs, numSamples,
inputArray, outputArray):
      self.mInputs = numInputs
```

```
        self.mHiddens = numHidden
        self.mOutputs = numOutput
        self.mLearningRate = learningRate
        self.mNoiseFactor = noise
        self.mEpochs = epochs
        self.mSamples = numSamples
        self.mInputArray = inputArray
        self.mOutputArray = outputArray

        self.wih = [] # Input to Hidden Weights
        self.who = [] # Hidden to Output Weights
        inputs = []
        hidden = []
        target = []
        actual = []
        erro = []
        errh = []
        return

    def initialize_arrays(self):
        for i in range(self.mInputs + 1): # The extra element represents bias node.
            self.wih.append([0.0] * self.mHiddens)
            for j in range(self.mHiddens):
                # Assign a random weight value between -0.5 and 0.5
                self.wih[i][j] = random.random() - 0.5

        for i in range(self.mHiddens + 1): # The extra element represents bias node.
            self.who.append([0.0] * self.mOutputs)
            for j in range(self.mOutputs):
                self.who[i][j] = random.random() - 0.5

        self.inputs = [0.0] * self.mInputs
        self.hidden = [0.0] * self.mHiddens
        self.target = [0.0] * self.mOutputs
        self.actual = [0.0] * self.mOutputs
        self.erro = [0.0] * self.mOutputs
        self.errh = [0.0] * self.mHiddens

        return

    def get_maximum(self, vector):
        # This function returns an array index of the maximum.
        index = 0
        maximum = vector[0]
        length = len(vector)

        for i in range(length):
            if vector[i] > maximum:
                maximum = vector[i]
                index = i

        return index
    def sigmoid(self, value):
        return 1.0 / (1.0 + math.exp(-value))

    def sigmoid_derivative(self, value):
        return value * (1.0 - value)

    def feed_forward(self):
        total = 0.0

        # Calculate input to hidden layer.
        for j in range(self.mHiddens):
            total = 0.0
            for i in range(self.mInputs):
                total += self.inputs[i] * self.wih[i][j]

            # Add in bias.
            total += self.wih[self.mInputs][j]
            self.hidden[j] = self.sigmoid(total)

        # Calculate the hidden to output layer.
        for j in range(self.mOutputs):
            total = 0.0
            for i in range(self.mHiddens):
                total += self.hidden[i] * self.who[i][j]

            # Add in bias.
            total += self.who[self.mHiddens][j]
            self.actual[j] = self.sigmoid(total)

        return

    def back_propagate(self):
        # Calculate the output layer error (step 3 for output cell).
        for j in range(self.mOutputs):
            self.erro[j] = (self.target[j] - self.actual[j]) * self.sigmoid_derivative(self.actual[j])

        # Calculate the hidden layer error (step 3 for hidden cell).
        for i in range(self.mHiddens):
            self.errh[i] = 0.0
            for j in range(self.mOutputs):
                self.errh[i] += self.erro[j] * self.who[i][j]

            self.errh[i] *= self.sigmoid_derivative(self.hidden[i])

        # Update the weights for the output layer (step 4).
        for j in range(self.mOutputs):
            for i in range(self.mHiddens):
                self.who[i][j] += (self.mLearningRate * self.erro[j] * self.hidden[i])

            # Update the bias.
            self.who[self.mHiddens][j] += (self.mLearningRate * self.erro[j])
        # Update the weights for the hidden layer (step 4).
        for j in range(self.mHiddens):
            for i in range(self.mInputs):
                self.wih[i][j] += (self.mLearningRate * self.errh[j] * self.inputs[i])

            # Update the bias.
            self.wih[self.mInputs][j] += (self.mLearningRate * self.errh[j])

        return

    def print_training_stats(self):
        sum = 0.0

        for i in range(self.mSamples):
            for j in range(self.mInputs):
                self.inputs[j] = self.mInputArray[i][j]

            for j in range(self.mOutputs):
                self.target[j] = self.mOutputArray[i][j]

            self.feed_forward()

            if self.get_maximum(self.actual) == self.get_maximum(self.target):
                sum += 1
            else:
                sys.stdout.write(str(self.inputs[0]) + "\t" + str(self.inputs[1]) + "\t" + str(self.inputs[2]) +
"\t" + str(self.inputs[3]) + "\n")
                sys.stdout.write(str(self.get_maximum(self.actual)) + "\t" +
str(self.get_maximum(self.target)) + "\n")

        sys.stdout.write("Network is " + str((float(sum) / float(MAX_SAMPLES)) * 100.0) + "%
correct.\n")

        return

    def train_network(self):
        sample = 0

        for i in range(self.mEpochs):
            sample += 1
            if sample == self.mSamples:
                sample = 0

            for j in range(self.mInputs):
                self.inputs[j] = self.mInputArray[sample][j]

            for j in range(self.mOutputs):
                self.target[j] = self.mOutputArray[sample][j]

            self.feed_forward()
            self.back_propagate()

        return

    def test_network(self):
        for i in range(self.mSamples):
            for j in range(self.mInputs):
                self.inputs[j] = self.mInputArray[i][j]

            self.feed_forward()

            for j in range(self.mInputs):
                sys.stdout.write(str(self.inputs[j]) + "\t")

            sys.stdout.write("Output: " + str(self.get_maximum(self.actual)) + "\n")

        return

    def test_network_with_noise(self):
        # This function adds a random fractional value to all the training inputs greater than zero.
        for i in range(self.mSamples):
            for j in range(self.mInputs):
                self.inputs[j] = self.mInputArray[i][j] + (random.random() * NOISE_FACTOR)

            self.feed_forward()

            for j in range(self.mInputs):
                sys.stdout.write("{:03.3f}".format(((self.inputs[j] * 1000.0) / 1000.0)) + "\t")

            sys.stdout.write("Output: " + str(self.get_maximum(self.actual)) + "\n")

        return

if __name__ == '__main__':
    ex = Example_4x6x16(INPUT_NEURONS, HIDDEN_NEURONS, OUTPUT_NEURONS,
LEARN_RATE, NOISE_FACTOR, TRAINING_REPS, MAX_SAMPLES, TRAINING_INPUTS,
TRAINING_OUTPUTS)
    ex.initialize_arrays()
    ex.train_network()
    ex.print_training_stats()
    sys.stdout.write("\nTest network against original input:\n")
    ex.test_network()
    sys.stdout.write("\nTest network against noisy input:\n")
    ex.test_network_with_noise()
```

Output:

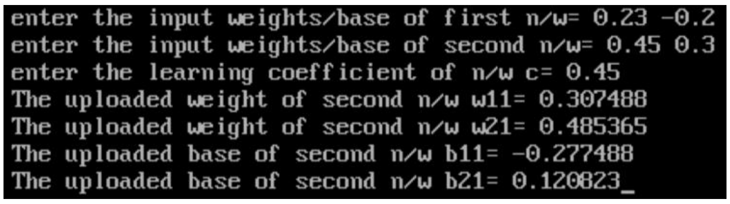
```
RESTART: C:\Users\Yogesh Patil\AppData\Local\Programs\Python\Python37-
32\backpropogationRule.py

Network is 100.0% correct.

Test network against original input:
1      1      1      0      Output: 0
1      1      0      0      Output: 1
0      1      1      0      Output: 2
1      0      1      0      Output: 3
1      0      0      0      Output: 4
0      1      0      0      Output: 5
0      0      1      0      Output: 6
1      1      1      1      Output: 7
1      1      0      1      Output: 8
0      1      1      1      Output: 9
1      0      1      1      Output: 10
1      0      0      1      Output: 11
0      1      0      1      Output: 12
0      0      1      1      Output: 13

Test network against noisy input:
1.129  1.530  1.184  0.132  Output: 0
1.487  1.044  0.468  0.464  Output: 1
0.168  1.555  1.184  0.032  Output: 2
1.316  0.013  1.108  0.453  Output: 10
1.063  0.174  0.049  0.095  Output: 4
0.109  1.064  0.079  0.264  Output: 5
0.477  0.528  1.560  0.083  Output: 0
1.386  1.438  1.554  1.109  Output: 7
1.074  1.234  0.171  1.313  Output: 8
0.236  1.134  1.497  1.336  Output: 9
1.375  0.037  1.374  1.384  Output: 10
1.017  0.463  0.448  1.389  Output: 11
0.252  1.202  0.343  1.447  Output: 7
0.533  0.388  1.252  1.342  Output: 13
>>>
```

B. Write a program for error Backpropagation algorithm.

Solution :

```
#include<conio.h>
#include<iostream.h>
#include<math.h>
void main()
{
clrscr();
float l,c,s1,n1,n2,w10,b10,w20,b20,w11,b11,w21,b21,p,t,a0=-1,a1,a2,e,s2;
cout<<"enter the input weights/base of second n/w= ";
cin>>w10>>b10;
cout<<"enter the input weights/base of second n/w= ";
cin>>w20>>b20;
cout<<"enter the learning coefficient of n/w c= ";
cin>>c;
/* Step1:Propagation of signal through n/w */
 n1=w10*p+b10;
a1=tanh(n1);
n2=w20*a1+b20;
a2=tanh(n2);
e=(t-a2); /* Back Propagation of Sensitivities */
s2=-2*(1-a2*a2)*e;
s1=(1-a1*a1)*w20*s2;
/* Updation of weights and bases */
w21=w20-(c*s2*a1);
w11=w10-(c*s1*a0);
b21=b20-(c*s2);
b11=b10-(c*s1);
cout<<"The uploaded weight of first n/w w11= "<<w11;
cout<<"\n"<<"The uploaded weight of second n/w w21= "<<w21;
cout<<"\n"<<"The uploaded base of second n/w b11= "<<b11;
cout<<"\n"<<"The uploaded base of second n/w b21= "<<b21;
getch();
}
```

Output:



```
enter the input weights/base of first n/w= 0.23 -0.2
enter the input weights/base of second n/w= 0.45 0.3
enter the learning coefficient of n/w c= 0.45
The uploaded weight of second n/w w11= 0.307488
The uploaded weight of second n/w w21= 0.485365
The uploaded base of second n/w b11= -0.277488
The uploaded base of second n/w b21= 0.120823_
```

Python Code :

```
import math
import random
import sys

NUM_INPUTS = 3 # Input nodes, plus the bias input.
NUM_PATTERNS = 4 # Input patterns for XOR experiment.
NUM_HIDDEN = 4
NUM_EPOCHS = 200
LR_IH = 0.7 # Learning rate, input to hidden weights.
LR_HO = 0.07 # Learning rate, hidden to output weights.

# The data here is the XOR data which has been rescaled to the range -1 to 1.
# An extra input value of 1 is also added to act as the bias.
# e.g: [Value 1][Value 2][Bias]
TRAINING_INPUT = [[1, -1, 1], [-1, 1, 1], [1, 1, 1], [-1, -1, 1]]

# The output must lie in the range -1 to 1.
TRAINING_OUTPUT = [1, 1, -1, -1]


class Backpropagation1:
    def __init__(self, numInputs, numPatterns, numHidden, numEpochs, i2hLearningRate,
h2oLearningRate, inputValues, outputValues):
        self.mNumInputs = numInputs
        self.mNumPatterns = numPatterns
        self.mNumHidden = numHidden
        self.mNumEpochs = numEpochs
        self.mI2HLearningRate = i2hLearningRate
        self.mH2OLearningRate = h2oLearningRate
        self.hiddenVal = [] # Hidden node outputs.
        self.weightsIH = [] # Input to Hidden weights.
        self.weightsHO = [] # Hidden to Output weights.
        self.trainInputs = inputValues
        self.trainOutput = outputValues # "Actual" output values.
        self.errThisPat = 0.0
        self.outPred = 0.0 # "Expected" output values.
        self.RMSerror = 0.0 # Root Mean Squared error.
        return

    def initialize_arrays(self):
        # Initialize weights to random values.
        for j in range(self.mNumInputs):
            newRow = []
            for i in range(self.mNumHidden):
                self.weightsHO.append((random.random() - 0.5) / 2.0)
                weightValue = (random.random() - 0.5) / 5.0
                newRow.append(weightValue)
                sys.stdout.write("Weight = " + str(weightValue) + "\n")
            self.weightsIH.append(newRow)
        self.hiddenVal = [0.0] * self.mNumHidden

        return

    def calc_net(self, patNum):
        # Calculates values for Hidden and Output nodes.
        for i in range(self.mNumHidden):
            self.hiddenVal[i] = 0.0
            for j in range(self.mNumInputs):
                self.hiddenVal[i] += (self.trainInputs[patNum][j] * self.weightsIH[j][i])

            self.hiddenVal[i] = math.tanh(self.hiddenVal[i])

        self.outPred = 0.0

        for i in range(self.mNumHidden):
            self.outPred += self.hiddenVal[i] * self.weightsHO[i]

        self.errThisPat = self.outPred - self.trainOutput[patNum] # Error = "Expected" - "Actual"
        return

    def adjust_hidden_to_output_weights(self):
        for i in range(self.mNumHidden):
            weightChange = self.mH2OLearningRate * self.errThisPat * self.hiddenVal[i]
            self.weightsHO[i] -= weightChange

            # Regularization of the output weights.
            if self.weightsHO[i] < -5.0:
                self.weightsHO[i] = -5.0
            elif self.weightsHO[i] > 5.0:
                self.weightsHO[i] = 5.0

        return

    def adjust_input_to_hidden_weights(self, patNum):
        for i in range(self.mNumHidden):
            for j in range(self.mNumInputs):
                x = 1 - math.pow(self.hiddenVal[i], 2)
                x = x * self.weightsHO[i] * self.errThisPat * self.mI2HLearningRate
                x = x * self.trainInputs[patNum][j]

                weightChange = x
                self.weightsIH[j][i] -= weightChange

        return

    def calculate_overall_error(self):
        errorValue = 0.0

        for i in range(self.mNumPatterns):
            self.calc_net(i)
            errorValue += math.pow(self.errThisPat, 2)
        errorValue /= self.mNumPatterns

        return math.sqrt(errorValue)

    def train_network(self):
        patNum = 0

        for j in range(self.mNumEpochs):
            for i in range(self.mNumPatterns):
                # Select a pattern at random.
                patNum = random.randrange(0, 4)

                # Calculate the output and error for this pattern.
                self.calc_net(patNum)

                # Adjust network weights.
                self.adjust_hidden_to_output_weights()
                self.adjust_input_to_hidden_weights(patNum)

            self.RMSerror = self.calculate_overall_error()

            sys.stdout.write("epoch = " + str(j) + " RMS Error = " + str(self.RMSerror) + "\n")

        return

    def display_results(self):
        for i in range(self.mNumPatterns):
            self.calc_net(i)
            sys.stdout.write("pat = " + str(i + 1) + " actual = " + str(self.trainOutput[i]) + " neural model
= " + str(self.outPred) + "\n")
        return

if __name__ == '__main__':
    bp1 = Backpropagation1(NUM_INPUTS, NUM_PATTERNS, NUM_HIDDEN, NUM_EPOCHS,
LR_IH, LR_HO, TRAINING_INPUT, TRAINING_OUTPUT)
    bp1.initialize_arrays()
    bp1.train_network()
    bp1.display_results()
```

Output :

```
RESTART: C:/Users/Yogesh Patil/AppData/Local/Programs/Python/Python37-
32/backpropogationRuleXOR.py
Weight = -0.076830377741923
Weight = 0.05545767965063293
Weight = 0.04681339243357252
Weight = -0.09587746729203184
Weight = 0.04052189546257452
Weight = -0.07220251631291778
Weight = -0.007749552037827767
Weight = 0.0019251805560653646
Weight = 0.07080044114386415
Weight = -0.09274060371150909
Weight = 0.06861531194281656
Weight = -0.0503784343248041 35
epoch = 0 RMS Error = 1.0009512196803645
epoch = 1 RMS Error = 1.0007742300312352
epoch = 2 RMS Error = 0.9999697046881315
epoch = 3 RMS Error = 1.000262809842168
epoch = 4 RMS Error = 1.0003385668587828
epoch = 5 RMS Error = 1.0003689607981894
epoch = 6 RMS Error = 1.0000417221070503
epoch = 7 RMS Error = 1.000677139732449
epoch = 8 RMS Error = 1.0007592165353387
epoch = 9 RMS Error = 1.0008703183629073
epoch = 10 RMS Error = 1.0008861430891467
epoch = 11 RMS Error = 1.0033119707170162
epoch = 12 RMS Error = 1.0002096196793477
epoch = 13 RMS Error = 1.0024201859681148
epoch = 14 RMS Error = 1.0246045183898609
epoch = 15 RMS Error = 1.0866139947291669
epoch = 16 RMS Error = 1.1180846338077137
epoch = 17 RMS Error = 1.0945116890437883
epoch = 18 RMS Error = 1.049671043392105
epoch = 19 RMS Error = 1.0152914017108694
epoch = 20 RMS Error = 1.0145557750670153
epoch = 21 RMS Error = 0.9932716005906015
epoch = 22 RMS Error = 0.9888288662456067
epoch = 23 RMS Error = 0.9653028002781683
epoch = 24 RMS Error = 0.9462033680943689
epoch = 25 RMS Error = 0.9635046766036908
epoch = 26 RMS Error = 0.9269561335697628
epoch = 27 RMS Error = 0.8899991117968605
epoch = 28 RMS Error = 0.8494727403855441
epoch = 29 RMS Error = 0.8501197677217179
epoch = 30 RMS Error = 0.8762543447244755
epoch = 31 RMS Error = 0.692734715088619
epoch = 32 RMS Error = 0.6544776768372573
epoch = 33 RMS Error = 0.6560015814555122
epoch = 34 RMS Error = 0.5847946010413321
epoch = 35 RMS Error = 0.5914082563666686
epoch = 36 RMS Error = 0.4790604826735862
epoch = 37 RMS Error = 0.4397556713159569
epoch = 38 RMS Error = 0.41377584615296625
epoch = 39 RMS Error = 0.4092770588919743
epoch = 40 RMS Error = 0.35296071912275606
epoch = 41 RMS Error = 0.3365550034266497
epoch = 42 RMS Error = 0.3280232258315713
epoch = 43 RMS Error = 0.3414255796938536
epoch = 44 RMS Error = 0.3533005009273525
epoch = 45 RMS Error = 0.28281089343673405
epoch = 46 RMS Error = 0.2904741808739498
epoch = 47 RMS Error = 0.2454063473896513
epoch = 48 RMS Error = 0.25409458767312154
epoch = 49 RMS Error = 0.20305645683840176
epoch = 50 RMS Error = 0.18393541468120506
epoch = 51 RMS Error = 0.17846776577735193
epoch = 52 RMS Error = 0.18334390019732144
epoch = 53 RMS Error = 0.15508220138915574
epoch = 54 RMS Error = 0.139229488350614
epoch = 55 RMS Error = 0.11974475852816462
epoch = 56 RMS Error = 0.11421376611765871
epoch = 57 RMS Error = 0.10268100281075816
epoch = 58 RMS Error = 0.10482234090600366
epoch = 59 RMS Error = 0.0999742036362297
epoch = 60 RMS Error = 0.09458874373044308
epoch = 61 RMS Error = 0.08254877094465272
epoch = 62 RMS Error = 0.08088721000198916
epoch = 63 RMS Error = 0.08307283801412396
epoch = 64 RMS Error = 0.06969713131699112
epoch = 65 RMS Error = 0.06463747572427869
epoch = 66 RMS Error = 0.05673540946521857
epoch = 67 RMS Error = 0.05222188742710054
epoch = 68 RMS Error = 0.04788054379879637
epoch = 69 RMS Error = 0.04230121876786113
epoch = 70 RMS Error = 0.03763746732085769
epoch = 71 RMS Error = 0.03345238653179893
epoch = 72 RMS Error = 0.03368828484699634
epoch = 73 RMS Error = 0.029034612333551403
epoch = 74 RMS Error = 0.025997629514787002
epoch = 75 RMS Error = 0.023283968034267345
epoch = 76 RMS Error = 0.02275553127586748
epoch = 77 RMS Error = 0.02266094188303696
epoch = 78 RMS Error = 0.02376199011376938
epoch = 79 RMS Error = 0.024441464034659403
epoch = 80 RMS Error = 0.025075352824191304
epoch = 81 RMS Error = 0.02617665075137898
epoch = 82 RMS Error = 0.01992863074760936
epoch = 83 RMS Error = 0.014545322962480053
epoch = 84 RMS Error = 0.013045591218816883
epoch = 85 RMS Error = 0.011229028517607825
epoch = 86 RMS Error = 0.01025903711675883
epoch = 87 RMS Error = 0.01002411657073447
epoch = 88 RMS Error = 0.010305972985037241
epoch = 89 RMS Error = 0.008650306018804928
epoch = 90 RMS Error = 0.009152497894035112
epoch = 91 RMS Error = 0.009301414031326888
epoch = 92 RMS Error = 0.007604408287698745
epoch = 93 RMS Error = 0.007785189814859717
epoch = 94 RMS Error = 0.008036562022304714
epoch = 95 RMS Error = 0.0065135928740881616
epoch = 96 RMS Error = 0.007015538276941596
epoch = 97 RMS Error = 0.007387497614085007
epoch = 98 RMS Error = 0.0050775653839133265
epoch = 99 RMS Error = 0.005027281384600543
epoch = 100 RMS Error = 0.003859324377077451
epoch = 101 RMS Error = 0.0035112042833212735
epoch = 102 RMS Error = 0.0034997996973881213
epoch = 103 RMS Error = 0.0028952254399767694
epoch = 104 RMS Error = 0.002568423552607127
epoch = 105 RMS Error = 0.002361451160813095
epoch = 106 RMS Error = 0.0021552800611884114
epoch = 107 RMS Error = 0.0019218421461379923
epoch = 108 RMS Error = 0.001731515387392159
epoch = 109 RMS Error = 0.0015986322737241336
epoch = 110 RMS Error = 0.0015128292602829211
epoch = 111 RMS Error = 0.0015088366258953208
epoch = 112 RMS Error = 0.0012987112428903436
epoch = 113 RMS Error = 0.0011018629306539375
epoch = 114 RMS Error = 0.0011065577292172176
epoch = 115 RMS Error = 0.0011089771459426357
epoch = 116 RMS Error = 0.0010047112460113657
epoch = 117 RMS Error = 0.0010148585595403199
epoch = 118 RMS Error = 0.0009908442377691941
epoch = 119 RMS Error = 0.0008304460522332652
epoch = 120 RMS Error = 0.0007747089415223214
epoch = 121 RMS Error = 0.0007599820231754118
epoch = 122 RMS Error = 0.0006306923848544624
epoch = 123 RMS Error = 0.0005663541153403198
epoch = 124 RMS Error = 0.0005487201564723689
epoch = 125 RMS Error = 0.000522977360780205
epoch = 126 RMS Error = 0.0004657976207044722
epoch = 127 RMS Error = 0.0004080705106233442
epoch = 128 RMS Error = 0.00036998147989017216
epoch = 129 RMS Error = 0.0003550916524108227
epoch = 130 RMS Error = 0.00030223466175174473
epoch = 131 RMS Error = 0.0003357167368151245
epoch = 132 RMS Error = 0.0002568827392197539
epoch = 133 RMS Error = 0.0002304640331961993
epoch = 134 RMS Error = 0.00021628498153342633
epoch = 135 RMS Error = 0.00019493804135861293
epoch = 136 RMS Error = 0.00019803295528420247
epoch = 137 RMS Error = 0.00020907905596424757
epoch = 138 RMS Error = 0.00015335262787111439
epoch = 139 RMS Error = 0.00014607965852596715
epoch = 140 RMS Error = 0.00013087066606122427
epoch = 141 RMS Error = 0.00013107016697259374
epoch = 142 RMS Error = 0.00011507567991620269
epoch = 143 RMS Error = 0.00010284982050853022
epoch = 144 RMS Error = 9.292485378643208e-05
epoch = 145 RMS Error = 8.343447649285541e-05
epoch = 146 RMS Error = 8.793306207224718e-05
epoch = 147 RMS Error = 8.707046038560856e-05
epoch = 148 RMS Error = 7.38365399544037e-05
epoch = 149 RMS Error = 7.382182262986522e-05
epoch = 150 RMS Error = 7.761966487666859e-05
epoch = 151 RMS Error = 6.135295946788964e-05
epoch = 152 RMS Error = 5.360015054234105e-05
epoch = 153 RMS Error = 5.077960248256503e-05
epoch = 154 RMS Error = 4.4228858203910684e-05
epoch = 155 RMS Error = 4.458895304383517e-05
epoch = 156 RMS Error = 4.581503455710884e-05
epoch = 157 RMS Error = 3.5506424755293975e-05
epoch = 158 RMS Error = 3.496237540404982e-05
epoch = 159 RMS Error = 3.287644381675574e-05
epoch = 160 RMS Error = 2.8091049147891424e-05
epoch = 161 RMS Error = 2.5880939208598246e-05
epoch = 162 RMS Error = 2.2919817738225087e-05
epoch = 163 RMS Error = 2.1142834970792857e-05
epoch = 164 RMS Error = 1.9503917936346664e-05
epoch = 165 RMS Error = 1.8695389461958275e-05
epoch = 166 RMS Error = 1.6660212708568678e-05
epoch = 167 RMS Error = 1.582339600722714e-05
epoch = 168 RMS Error = 1.6218553296525844e-05
epoch = 169 RMS Error = 1.2446790077869108e-05
epoch = 170 RMS Error = 1.1364488841747833e-05
epoch = 171 RMS Error = 1.1492799291221317e-05
epoch = 172 RMS Error = 1.1935239845489047e-05
epoch = 173 RMS Error = 9.8945275937417e-06
epoch = 174 RMS Error = 1.034526737443712e-05
epoch = 175 RMS Error = 1.0941352084405505e-05
epoch = 176 RMS Error = 8.64019424588387e-06
epoch = 177 RMS Error = 6.948345561870279e-06
epoch = 178 RMS Error = 6.841000724835082e-06
epoch = 179 RMS Error = 7.498689427415963e-06
epoch = 180 RMS Error = 7.537188714500655e-06
epoch = 181 RMS Error = 5.24014087532078e-06
epoch = 182 RMS Error = 4.5932620447212635e-06
epoch = 183 RMS Error = 4.247745851030866e-06
epoch = 184 RMS Error = 4.1552714340205e-06
epoch = 185 RMS Error = 4.222795805050674e-06
epoch = 186 RMS Error = 4.3907358630242534e-06
epoch = 187 RMS Error = 3.3014402308858223e-06
epoch = 188 RMS Error = 3.307976669902875e-06
epoch = 189 RMS Error = 3.487729575969274e-06
epoch = 190 RMS Error = 3.521219273796291e-06
epoch = 191 RMS Error = 3.6528551948410966e-06
epoch = 192 RMS Error = 3.765133447115511e-06
epoch = 193 RMS Error = 3.8074467357187595e-06
epoch = 194 RMS Error = 2.908648600092539e-06
epoch = 195 RMS Error = 2.4799267073853265e-06
epoch = 196 RMS Error = 2.5555389700947974e-06
epoch = 197 RMS Error = 2.3252854109401005e-06
epoch = 198 RMS Error = 2.2771751916395466e-06
epoch = 199 RMS Error = 2.105442499550845e-06
pat = 1 actual = 1 neural model = 0.9999970001299875
pat = 2 actual = 1 neural model = 0.9999996793968098
pat = 3 actual = -1 neural model = -0.9999996101382688
pat = 4 actual = -1 neural model = -0.9999970883760874
>>>
```

Practical No. 5

A. Write a program for Hopfield Network.

Given Pattern

[1,0,1,0] AND [0,1,0,1]

Given weighted vector

wt1 {0,-3,3,-3}

wt2{-3,0,-3,3}

wt3 {3,-3,0,-3}

wt4 {-3,3,-3,0}

Solution :

Save HOP.H file in INCLUDE folder in C:\TurboC3\Include

HOP.H

```
#include <stdio.h>
#include <iostream.h>
#include <math.h>
class neuron
{
protected:
    int activation;
    friend class network;
public:
    int weightv[4];
    neuron() {};
    neuron(int *j) ;
    int act(int, int*);
};
class network
{
public:
    neuron   nrn[4];
    int output[4];
    int threshld(int) ;
    void activation(int j[4]);
    network(int*,int*,int*,int*);
};
```

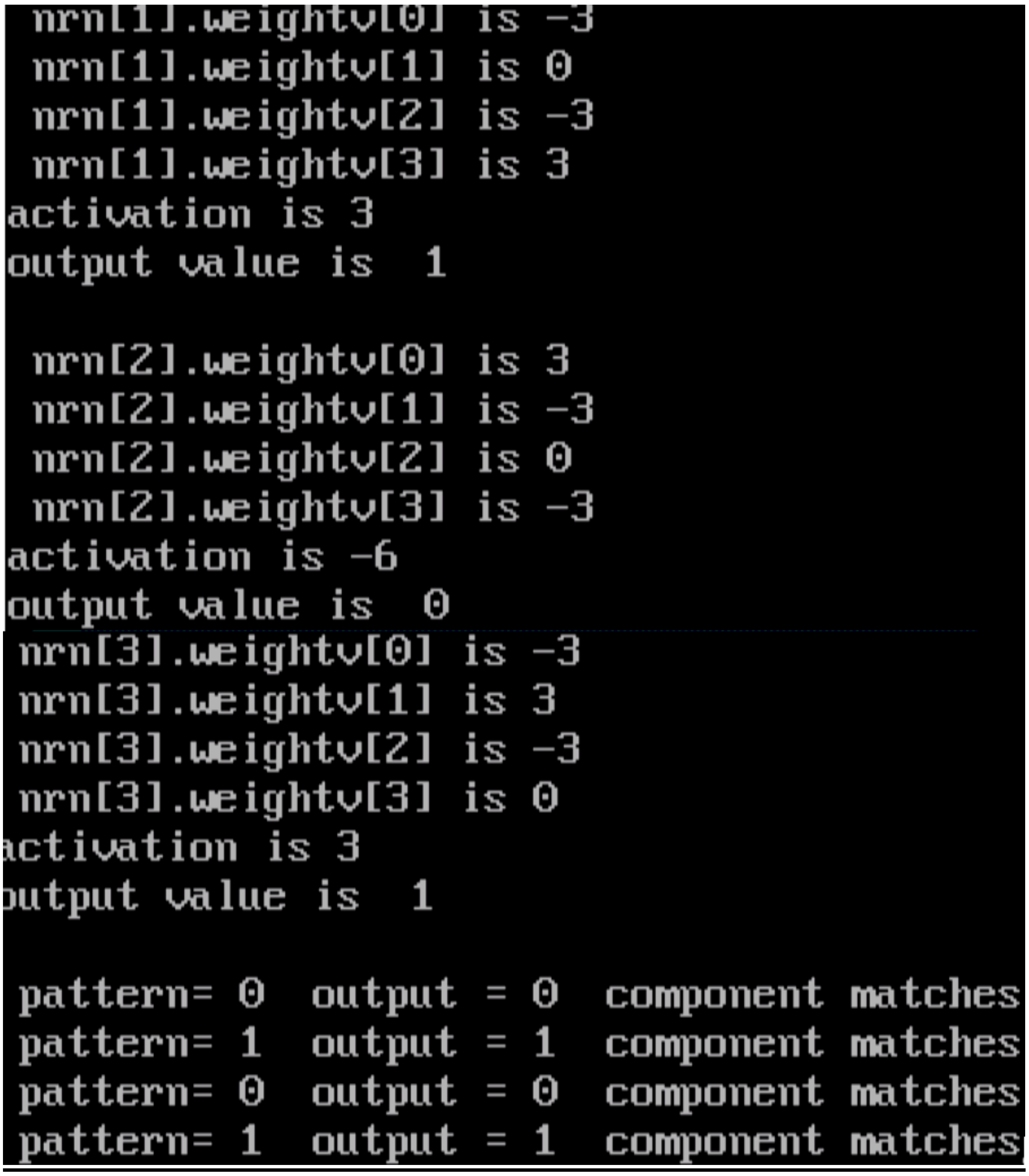
________________________header file HOP.H ends here________________

Main program (hopnet.cpp)

```
#include "hop.h"
#include<conio.h>
neuron::neuron(int *j)
{
int i;
for(i=0;i<4;i++)
    {
    weightv[i]= *(j+i);
    }
}
```

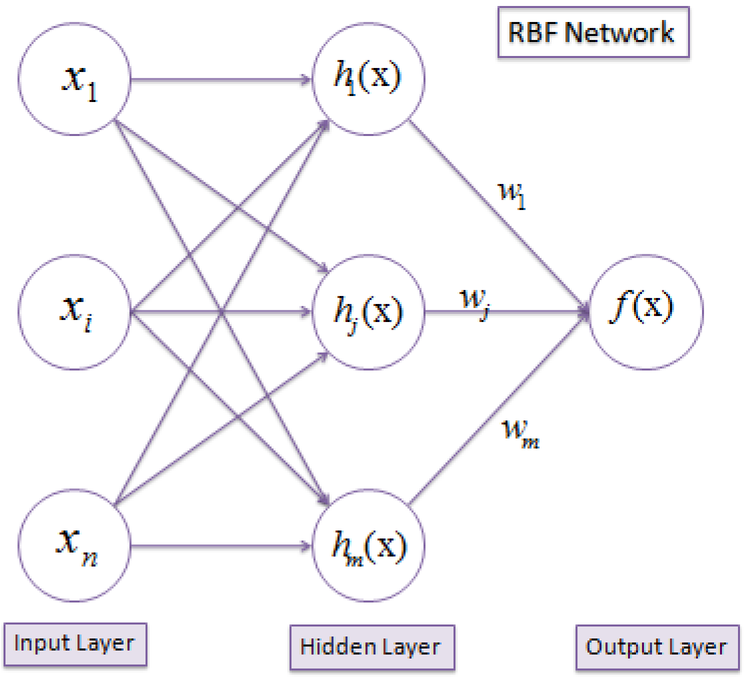
```
int neuron::act(int m, int *x)
{
int i;
int a=0;
for(i=0;i<m;i++)
    {
    a += x[i]*weightv[i];
    }
return a;
}
int network::threshld(int k)
{
if(k>=0)
    return (1);
else
    return (0);
}
network::network(int a[4],int b[4],int c[4],int d[4])
{
nrn[0] = neuron(a) ;
nrn[1] = neuron(b) ;
nrn[2] = neuron(c) ;
nrn[3] = neuron(d) ;
}

void network::activation(int *patrn)
{
int i,j;
for(i=0;i<4;i++)
    {
    for(j=0;j<4;j++)
        {
        cout<<"\n nrn["<<i<<"].weightv["<<j<<"] is "
            <<nrn[i].weightv[j];
        }
    nrn[i].activation = nrn[i].act(4,patrn);
    cout<<"\nactivation is "<<nrn[i].activation;
    output[i]=threshld(nrn[i].activation);
    cout<<"\noutput value is  "<<output[i]<<"\n";
    }
}
void main ()
{
int patrn1[]= {1,0,1,0},i;
int wt1[]= {0,-3,3,-3};
int wt2[]= {-3,0,-3,3};
int wt3[]= {3,-3,0,-3};
int wt4[]= {-3,3,-3,0};
cout<<"\nTHIS PROGRAM IS FOR A HOPFIELD NETWORK WITH A SINGLE LAYER
OF";
cout<<"\n4 FULLY INTERCONNECTED NEURONS. THE NETWORK SHOULD
RECALL
THE";
cout<<"\nPATTERNS 1010 AND 0101 CORRECTLY.\n";
//create the network by calling its constructor.
// the constructor calls neuron constructor as many times as the number of
// neurons in the network.
network h1(wt1,wt2,wt3,wt4);
//present a pattern to the network and get the activations of the neurons
h1.activation(patrn1);
//check if the pattern given is correctly recalled and give message
for(i=0;i<4;i++)
    {
    if (h1.output[i] == patrn1[i])
        cout<<"\n pattern= "<<patrn1[i]<<
        "  output = "<<h1.output[i]<<"  component matches";
    else
        cout<<"\n pattern= "<<patrn1[i]<<
        "  output = "<<h1.output[i]<<
        "  discrepancy occurred";
    }
cout<<"\n\n";
int patrn2[]= {0,1,0,1};
h1.activation(patrn2);
for(i=0;i<4;i++)
    {
    if (h1.output[i] == patrn2[i])
        cout<<"\n pattern= "<<patrn2[i]<<
        "  output = "<<h1.output[i]<<"  component matches";
    else
        cout<<"\n pattern= "<<patrn2[i]<<
        "  output = "<<h1.output[i]<<
        "  discrepancy occurred";
     }
getch();
}
```

Output:



```
 nrn[1].weightv[0] is -3
 nrn[1].weightv[1] is 0
 nrn[1].weightv[2] is -3
 nrn[1].weightv[3] is 3
activation is 3
output value is  1

 nrn[2].weightv[0] is 3
 nrn[2].weightv[1] is -3
 nrn[2].weightv[2] is 0
 nrn[2].weightv[3] is -3
activation is -6
output value is  0
 nrn[3].weightv[0] is -3
 nrn[3].weightv[1] is 3
 nrn[3].weightv[2] is -3
 nrn[3].weightv[3] is 0
activation is 3
output value is  1

 pattern= 0  output = 0  component matches
 pattern= 1  output = 1  component matches
 pattern= 0  output = 0  component matches
 pattern= 1  output = 1  component matches
```

B. Write a program for Radial Basis function

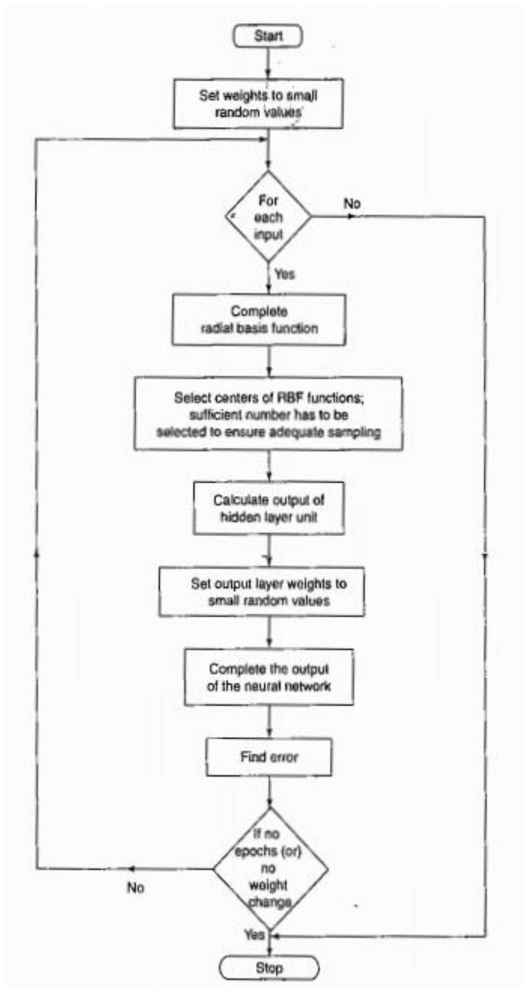


$$f(x) = \sum_{j=1}^{m} w_j h_j(x)$$

$$h(x) = \exp\left(-\frac{(x-c)^2}{r^2} \right)$$

h(x) is the Gaussian activation function with the parameters r (the radius or standard deviation) and c (the center or average taken from the input space) defined separately at each RBF unit. The learning process is based on adjusting the parameters of the network to reproduce a set of input-output patterns. There are three types of parameters; the weight w between the hidden nodes and the output nodes, the center c of each neuron of the hidden layer and the unit width r.

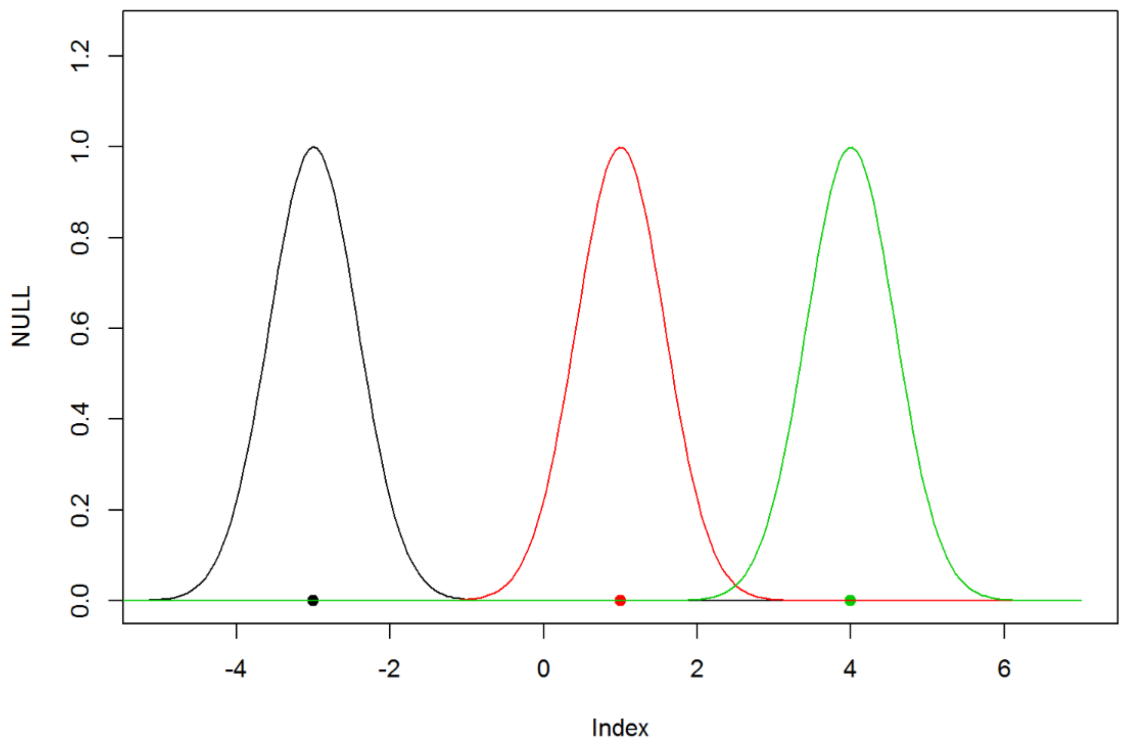
<u>Algorithm</u>


Start
Set weights to small random values
For each input
No
Yes
Complete radial basis function
Select centers of RBF functions; sufficient number has to be selected to ensure adequate sampling
Calculate output of hidden layer unit
Set output layer weights to small random values
Complete the output of the neural network
Find error
If no epochs (or) no weight change
No
Yes
Stop

One-dimensional dataset as an illustration of the gaussian influence:

```
rbf.gauss <- function(gamma=1.0) {

  function(x) {
    exp(-gamma * norm(as.matrix(x),"F")^2 )
  }
}

D <- matrix(c(-3,1,4), ncol=1) # 3 datapoints
N <- length(D[,1])

xlim  <- c(-5,7)
plot(NULL,xlim=xlim,ylim=c(0,1.25),type="n")
points(D,rep(0,length(D)),col=1:N,pch=19)
x.coord = seq(-7,7,length=250)
gamma <- 1.5
for (i in 1:N) {
  points(x.coord, lapply(x.coord - D[i,], rbf.gauss(gamma)), type="l", col=i)
}
```

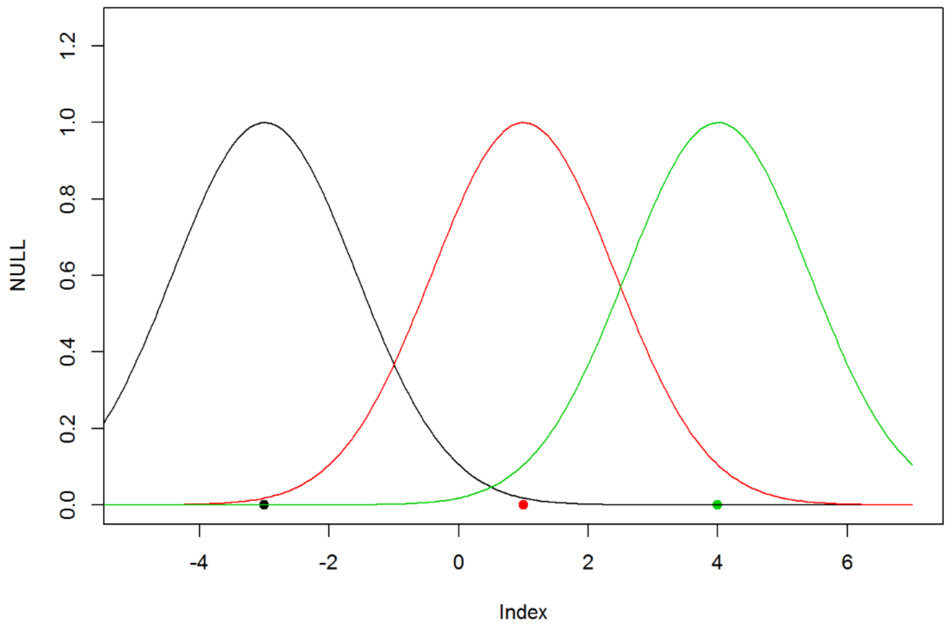
Output



The value of gamma controls how far or how little the influnce of each datapoint is felt:

```
plot(NULL,xlim=xlim,ylim=c(0,1.25),type="n")
points(D,rep(0,length(D)),col=1:N,pch=19)
x.coord = seq(-7,7,length=250)
gamma <- 0.25
for (i in 1:N) {
  points(x.coord, lapply(x.coord - D[i,], rbf.gauss(gamma)), type="l", col=i)
}
```

Output



Practical 6

A. Implementation of Kohonen Self Organising Map

A self-organizing map (SOM) is a bit hard to describe. Briefly, a SOM is a data structure that allows you to investigate the structure of a set of data. If you have data without class labels, a SOM can indicate how many classes there are in the data. If you have data with class labels, a SOM can be used for dimensionality reduction so the data can be graphed, where the resulting graph indicates how similar different classes are.

The example begins by creating a 30 x 30 SOM for the well-known Fisher's Iris dataset. The SOM is a data structure in memory. The demo uses the SOM to create what's called a U-Matrix, shown in Figure 1. In a U-Matrix, black cells indicate data items that are similar to each other and white cells indicate borders between groups of similar items. If you squint at the figure you can see that there appears to be three different areas of similar items, suggesting the data has three classes (which it does). An SOM can be used to reduce the number of dimensions in a dataset down to two, so the data can be graphed. The image in Figure 2 suggests that the items of class 0 (brown) are most different from items of class 2 (blue), and that items of class 1 (green) are somewhat similar to items of class 0 and class 2.

Execution of Program:

Step 1 : Save Data file and code file in same folder.
Step 2 : Execute code file **som_iris.py** in IDE python
Step 3 : Data will be start loading in to main memory
Step 4 : It will start building 30 X 30 U-matrix for given data set.
Step 5 : SOM can be used to reduce dimensionality so the data can be displayed as a two-dimensional graph in Figure 1.
Step 6 : Close Figure 1 then it will upload Figure 2.

Python Code som_iris.py :

```
import numpy as np
import matplotlib.pyplot as plt
# note: if this fails, try >pip uninstall matplotlib
# and then >pip install matplotlib

def closest_node(data, t, map, m_rows, m_cols):
  # (row,col) of map node closest to data[t]
  result = (0,0)
  small_dist = 1.0e20
  for i in range(m_rows):
    for j in range(m_cols):
      ed = euc_dist(map[i][j], data[t])
      if ed < small_dist:
        small_dist = ed
        result = (i, j)
  return result

def euc_dist(v1, v2):
  return np.linalg.norm(v1 - v2)
```

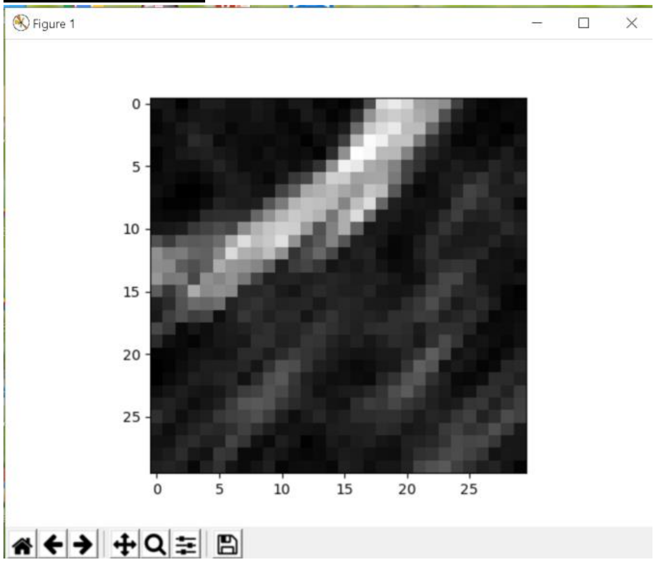
```
def manhattan_dist(r1, c1, r2, c2):
  return np.abs(r1-r2) + np.abs(c1-c2)

def most_common(lst, n):
  # lst is a list of values 0 . . n
  if len(lst) == 0: return -1
  counts = np.zeros(shape=n, dtype=np.int)
  for i in range(len(lst)):
    counts[lst[i]] += 1
  return np.argmax(counts)

# ==================================================================

def main():
  # 0. get started
  np.random.seed(1)
  Dim = 4
  Rows = 30; Cols = 30
  RangeMax = Rows + Cols
  LearnMax = 0.5
  StepsMax = 5000

  # 1. load data
  print("\nLoading Iris data into memory \n")
  data_file = ".\\iris_data_012.txt"
  data_x = np.loadtxt(data_file, delimiter=",", usecols=range(0,4),
    dtype=np.float64)
  data_y = np.loadtxt(data_file, delimiter=",", usecols=[4],
    dtype=np.int)
  # option: normalize data

  # 2. construct the SOM
  print("Constructing a 30x30 SOM from the iris data")
  map = np.random.random_sample(size=(Rows,Cols,Dim))
  for s in range(StepsMax):
    if s % (StepsMax/10) == 0: print("step = ", str(s))
    pct_left = 1.0 - ((s * 1.0) / StepsMax)
    curr_range = (int)(pct_left * RangeMax)
    curr_rate = pct_left * LearnMax

    t = np.random.randint(len(data_x))
    (bmu_row, bmu_col) = closest_node(data_x, t, map, Rows, Cols)
    for i in range(Rows):
      for j in range(Cols):
        if manhattan_dist(bmu_row, bmu_col, i, j) < curr_range:
          map[i][j] = map[i][j] + curr_rate * \
(data_x[t] - map[i][j])
  print("SOM construction complete \n")
  # 3. construct U-Matrix
  print("Constructing U-Matrix from SOM")
  u_matrix = np.zeros(shape=(Rows,Cols), dtype=np.float64)
  for i in range(Rows):
    for j in range(Cols):
      v = map[i][j]  # a vector
      sum_dists = 0.0; ct = 0

      if i-1 >= 0:    # above
        sum_dists += euc_dist(v, map[i-1][j]); ct += 1
      if i+1 <= Rows-1:   # below
        sum_dists += euc_dist(v, map[i+1][j]); ct += 1
      if j-1 >= 0:   # left
        sum_dists += euc_dist(v, map[i][j-1]); ct += 1
      if j+1 <= Cols-1:   # right
        sum_dists += euc_dist(v, map[i][j+1]); ct += 1

      u_matrix[i][j] = sum_dists / ct
  print("U-Matrix constructed \n")

  # display U-Matrix
  plt.imshow(u_matrix, cmap='gray')  # black = close = clusters
  plt.show()

  # 4. because the data has labels, another possible visualization:
  # associate each data label with a map node
  print("Associating each data label to one map node ")
  mapping = np.empty(shape=(Rows,Cols), dtype=object)
  for i in range(Rows):
    for j in range(Cols):
      mapping[i][j] = []

  for t in range(len(data_x)):
    (m_row, m_col) = closest_node(data_x, t, map, Rows, Cols)
    mapping[m_row][m_col].append(data_y[t])

  label_map = np.zeros(shape=(Rows,Cols), dtype=np.int)
  for i in range(Rows):
    for j in range(Cols):
      label_map[i][j] = most_common(mapping[i][j], 3)

  plt.imshow(label_map, cmap=plt.cm.get_cmap('terrain_r', 4))
  plt.colorbar()
  plt.show()

# ==============================================================

if __name__=="__main__":
  main()
```

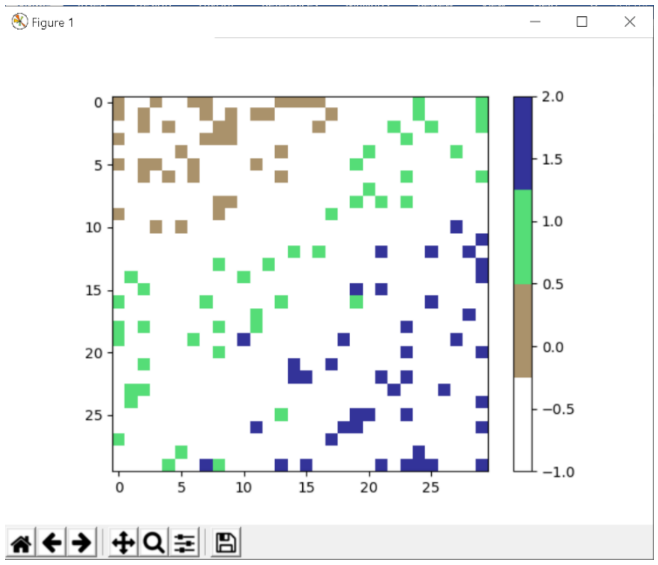
Output :

Loading Data set in to memory

```
Loading Iris data into memory

Constructing a 30x30 SOM from the iris data
step =  0
step =  500
step =  1000
step =  1500
step =  2000
step =  2500
step =  3000
step =  3500
step =  4000
step =  4500
SOM construction complete

Constructing U-Matrix from SOM
U-Matrix constructed

Associating each data label to one map node
```

Map Created :





B. Implementation of Adaptive resonance theory

```
from __future__ import print_function

from __future__ import division

import numpy as np



class ART:

    ''' ART class

    Usage example:

    --------------

    # Create a ART network with input of size 5 and 20 internal units

    >>> network = ART(5,10,0.5)

    '''

    def __init__(self, n=5, m=10, rho=.5):

        '''

        Create network with specified shape

        Parameters:

        -----------

        n : int

            Size of input

        m : int

            Maximum number of internal units

        rho : float

            Vigilance parameter

        '''

        # Comparison layer

        self.F1 = np.ones(n)

        # Recognition layer

        self.F2 = np.ones(m)

        # Feed-forward weights

        self.Wf = np.random.random((m,n))
        # Feed-back weights
        self.Wb = np.random.random((n,m))
        # Vigilance
        self.rho = rho
        # Number of active units in F2
        self.active = 0
    def learn(self, X):
        ''' Learn X '''
        # Compute F2 output and sort them (I)
        self.F2[...] = np.dot(self.Wf, X)
        I = np.argsort(self.F2[:self.active].ravel())[::-1]
        for i in I:
            # Check if nearest memory is above the vigilance level
            d = (self.Wb[:,i]*X).sum()/X.sum()
            if d >= self.rho:
                # Learn data
                self.Wb[:,i] *= X
                self.Wf[i,:] = self.Wb[:,i]/(0.5+self.Wb[:,i].sum())
                return self.Wb[:,i], i
        # No match found, increase the number of active units
        # and make the newly active unit to learn data
        if self.active < self.F2.size:
            i = self.active
            self.Wb[:,i] *= X
            self.Wf[i,:] = self.Wb[:,i]/(0.5+self.Wb[:,i].sum())
            self.active += 1
            return self.Wb[:,i], i
        return None,None
# -----------------------------------------------------------------------------
if __name__ == '__main__':
    np.random.seed(1)
    # Example 1 : very simple data
    # -------------------------------------------------------------------------
    network = ART( 5, 10, rho=0.5)
    data = ["   O ",
            "  O O",
            "    O",
            "  O O",
            "    O",
            "  O O",
            "    O",
            " OO O",
            " OO  ",
            " OO O",
            " OO  ",
            "OOO  ",
            "OO   ",
            "O    ",
            "OO   ",
            "OOO  ",
            "OOOO ",
            "OOOOO",
            "O    ",
            " O   ",
            "  O  ",
            "   O ",
            "    O",
            "  O O",
            " OO O",
            " OO  ",
            "OOO  ",
            "OO   ",
            "OOOO ",
            "OOOOO"]
```

```
X = np.zeros(len(data[0]))

for i in range(len(data)):

    for j in range(len(data[i])):

        X[j] = (data[i][j] == 'O')

    Z, k = network.learn(X)

    print("|%s|"%data[i],"-> class", k)

# Example 2 : Learning letters

# -------------------------------------------------------------------------

def letter_to_array(letter):

    ''' Convert a letter to a numpy array '''

    shape = len(letter), len(letter[0])

    Z = np.zeros(shape, dtype=int)

    for row in range(Z.shape[0]):

        for column in range(Z.shape[1]):

            if letter[row][column] == '#':

                Z[row][column] = 1

    return Z

def print_letter(Z):

    ''' Print an array as if it was a letter'''

    for row in range(Z.shape[0]):

        for col in range(Z.shape[1]):

            if Z[row,col]:

                print( '#', end="" )

            else:

                print( ' ', end="" )

        print( )


A = letter_to_array( [' #### ',

                      '#    #',

                      '#    #',

                      '######',

                      '#    #',
                '#    #',
                '#    #'] )
B = letter_to_array( ['##### ',
                '#    #',
                '#    #',
                '##### ',
                '#    #',
                '#    #',
                '##### '] )
C = letter_to_array( [' #### ',
                '#    #',
                '#     ',
                '#     ',
                '#     ',
                '#    #',
                ' #### '] )
D = letter_to_array( ['##### ',
                '#    #',
                '#    #',
                '#    #',
                '#    #',
                '#    #',
                '##### '] )
E = letter_to_array( ['######',
                '#     ',
                '#     ',
                '####  ',
                '#     ',
                '#     ',
                '######'] )
F = letter_to_array( ['######',
                '#     ',
            '#    ',

            '####  ',

            '#    ',

            '#    ',

            '#    '] )


samples = [A,B,C,D,E,F]

network = ART( 6*7, 10, rho=0.15 )


for i in range(len(samples)):

   Z, k = network.learn(samples[i].ravel())

   print("%c"%(ord('A')+i),"-> class",k)

   print_letter(Z.reshape(7,6))
```

Output

```
|   O | -> class 0
|  O O| -> class 1
|    O| -> class 1
|  O O| -> class 2
|    O| -> class 1
|  O O| -> class 3
|    O| -> class 1
| OO O| -> class 4
| OO  | -> class 5
| OO O| -> class 6
| OO  | -> class 6
|OOO  | -> class 6
|OO   | -> class 7
|O    | -> class 8
|OO   | -> class 9
|OOO  | -> class 6
|OOOO | -> class None
```

```
|OOOOO| -> class None

|O    | -> class 8

| O   | -> class 5

|  O  | -> class 6

|   O | -> class 0

|    O| -> class 1

|  O O| -> class 3

| OO O| -> class None

| OO  | -> class None

|OOO  | -> class None

|OO   | -> class 9

|OOOO | -> class None

|OOOOO| -> class None

A -> class 0

 ####

#    #

#    #

######

#    #

#    #

#    #

B -> class 0

 ####

#    #

#    #

#####

#    #

#    #

#

C -> class 0

 ####

#    #
```

```
#
#
#
#    #

D -> class 0
 ####
#    #
#
#
#
#    #

E -> class 0
 ####
#
#
#
#
#
F -> class 0
 ####
#
#
#
#
#
```

Example 2 : Testing ART by creating array dataset

```
import numpy as np
# compute sigmoid nonlinearity
def sigmoid(x):
    output = 1/(1+np.exp(-x))
    return output
# convert output of sigmoid function to its derivative
def sigmoid_output_to_derivative(output):
    return output*(1-output)
# input dataset
X = np.array([  [0,1],
                [0,1],
                [1,0],
                [1,0] ])
# output dataset
y = np.array([[0,0,1,1]]).T
# seed random numbers to make calculation
# deterministic (just a good practice)
np.random.seed(1)
# initialize weights randomly with mean 0
synapse_0 = 2*np.random.random((2,1)) - 1
for iter in range(10000):
    # forward propagation
    layer_0 = X
    layer_1 = sigmoid(np.dot(layer_0,synapse_0))
    # how much did we miss?
    layer_1_error = layer_1 - y
    # multiply how much we missed by the
    # slope of the sigmoid at the values in l1
    layer_1_delta = layer_1_error * sigmoid_output_to_derivative(layer_1)
    synapse_0_derivative = np.dot(layer_0.T,layer_1_delta)
    # update weights
    synapse_0 -= synapse_0_derivative

print ("Output After Training:")

print (layer_1)
```

Output

```
Output After Training:

[[0.00505119]

 [0.00505119]

 [0.99494905]

 [0.99494905]]
```

---X-------------------------------------

Example 3: By providing data pattern

```
import math

import sys

N = 4 # Number of components in an input vector.

M = 3 # Max number of clusters to be formed.

VIGILANCE = 0.4

PATTERNS = 7

TRAINING_PATTERNS = 4 # Use this many for training, the rest are for tests.

PATTERN_ARRAY = [[1, 1, 0, 0],

                [0, 0, 0, 1],

                [1, 0, 0, 0],

                [0, 0, 1, 1],

                [0, 1, 0, 0],

                [0, 0, 1, 0],

                [1, 0, 1, 0]]

class ART1_Example1:

    def __init__(self, inputSize, numClusters, vigilance, numPatterns, numTraining, patternArray):

        self.mInputSize = inputSize

        self.mNumClusters = numClusters

        self.mVigilance = vigilance

        self.mNumPatterns = numPatterns

        self.mNumTraining = numTraining
        self.mPatterns = patternArray
        self.bw = [] # Bottom-up weights.
        self.tw = [] # Top-down weights.
        self.f1a = [] # Input layer.
        self.f1b = [] # Interface layer.
        self.f2 = []
        return
    def initialize_arrays(self):
        # Initialize bottom-up weight matrix.
        sys.stdout.write("Weights initialized to:")
        for i in range(self.mNumClusters):
            self.bw.append([0.0] * self.mInputSize)
            for j in range(self.mInputSize):
                self.bw[i][j] = 1.0 / (1.0 + self.mInputSize)
                sys.stdout.write(str(self.bw[i][j]) + ", ")
            sys.stdout.write("\n")
        sys.stdout.write("\n")
        # Initialize top-down weight matrix.
        for i in range(self.mNumClusters):
            self.tw.append([0.0] * self.mInputSize)
            for j in range(self.mInputSize):
                self.tw[i][j] = 1.0
                sys.stdout.write(str(self.tw[i][j]) + ", ")
            sys.stdout.write("\n")
        sys.stdout.write("\n")
        self.f1a = [0.0] * self.mInputSize
        self.f1b = [0.0] * self.mInputSize
        self.f2 = [0.0] * self.mNumClusters
        return
    def get_vector_sum(self, nodeArray):
        total = 0
        length = len(nodeArray)
        for i in range(length):
            total += nodeArray[i]
        return total
    def get_maximum(self, nodeArray):
        maximum = 0;
        foundNewMaximum = False;
        length = len(nodeArray)
        done = False
        while not done:
            foundNewMaximum = False
            for i in range(length):
                if i != maximum:
                    if nodeArray[i] > nodeArray[maximum]:
                        maximum = i
                        foundNewMaximum = True
            if foundNewMaximum == False:
                done = True
        return maximum
    def test_for_reset(self, activationSum, inputSum, f2Max):
        doReset = False
        if(float(activationSum) / float(inputSum) >= self.mVigilance):
            doReset = False # Candidate is accepted.
        else:
            self.f2[f2Max] = -1.0 # Inhibit.
            doReset = True # Candidate is rejected.
        return doReset
    def update_weights(self, activationSum, f2Max):
        # Update bw(f2Max)
        for i in range(self.mInputSize):
            self.bw[f2Max][i] = (2.0 * float(self.f1b[i])) / (1.0 + float(activationSum))
        for i in range(self.mNumClusters):
            for j in range(self.mInputSize):
                sys.stdout.write(str(self.bw[i][j]) + ", ")
            sys.stdout.write("\n")
        sys.stdout.write("\n")
        # Update tw(f2Max)
        for i in range(self.mInputSize):
            self.tw[f2Max][i] = self.f1b[i]
        for i in range(self.mNumClusters):
            for j in range(self.mInputSize):
                sys.stdout.write(str(self.tw[i][j]) + ", ")
            sys.stdout.write("\n")
        sys.stdout.write("\n")
        return
    def ART1(self):
        inputSum = 0
        activationSum = 0
        f2Max = 0
        reset = True
        sys.stdout.write("Begin ART1:\n")
        for k in range(self.mNumPatterns):
            sys.stdout.write("Vector: " + str(k) + "\n\n")
            # Initialize f2 layer activations to 0.0
            for i in range(self.mNumClusters):
                self.f2[i] = 0.0
            # Input pattern() to f1 layer.
            for i in range(self.mInputSize):
                self.f1a[i] = self.mPatterns[k][i]
            # Compute sum of input pattern.
            inputSum = self.get_vector_sum(self.f1a)
            sys.stdout.write("InputSum (si) = " + str(inputSum) + "\n\n")
            # Compute activations for each node in the f1 layer.
            # Send input signal from f1a to the fF1b layer.
            for i in range(self.mInputSize):
                self.f1b[i] = self.f1a[i]

            # Compute net input for each node in the f2 layer.

            for i in range(self.mNumClusters):

                for j in range(self.mInputSize):

                    self.f2[i] += self.bw[i][j] * float(self.f1a[j])

                    sys.stdout.write(str(self.f2[i]) + ", ")

                sys.stdout.write("\n")

            sys.stdout.write("\n")

            reset = True

            while reset == True:

                # Determine the largest value of the f2 nodes.

                f2Max = self.get_maximum(self.f2)

                # Recompute the f1a to f1b activations (perform AND function)

                for i in range(self.mInputSize):

                    sys.stdout.write(str(self.f1b[i]) + " * " + str(self.tw[f2Max][i]) + " = " + str(self.f1b[i] *
self.tw[f2Max][i]) + "\n")

                    self.f1b[i] = self.f1a[i] * math.floor(self.tw[f2Max][i])

                # Compute sum of input pattern.

                activationSum = self.get_vector_sum(self.f1b)

                sys.stdout.write("ActivationSum (x(i)) = " + str(activationSum) + "\n\n")

                reset = self.test_for_reset(activationSum, inputSum, f2Max)

            # Only use number of TRAINING_PATTERNS for training, the rest are tests.

            if k < self.mNumTraining:

                self.update_weights(activationSum, f2Max)

            sys.stdout.write("Vector #" + str(k) + " belongs to cluster #" + str(f2Max) + "\n\n")

        return

    def print_results(self):

        sys.stdout.write("Final weight values:\n")

        for i in range(self.mNumClusters):

            for j in range(self.mInputSize):

                sys.stdout.write(str(self.bw[i][j]) + ", ")

            sys.stdout.write("\n")
        sys.stdout.write("\n")

        for i in range(self.mNumClusters):

            for j in range(self.mInputSize):

                sys.stdout.write(str(self.tw[i][j]) + ", ")

            sys.stdout.write("\n")

        sys.stdout.write("\n")

        return

if __name__ == '__main__':

    art1 = ART1_Example1(N, M, VIGILANCE, PATTERNS, TRAINING_PATTERNS,
PATTERN_ARRAY)

    art1.initialize_arrays()

    art1.ART1()

    art1.print_results()
```

Output

```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
== RESTART: E:/SPDT/MSCSem1/Soft Computing/MSC/MSC/Practical6/Pract6B_3.py ==
Weights initialized to:0.2, 0.2, 0.2, 0.2,
0.2, 0.2, 0.2, 0.2,
0.2, 0.2, 0.2, 0.2,
1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0,
Begin ART1:
Vector: 0
InputSum (si) = 2
0.2, 0.4, 0.4, 0.4,
0.2, 0.4, 0.4, 0.4,
0.2, 0.4, 0.4, 0.4,
1 * 1.0 = 1.0
1 * 1.0 = 1.0
0 * 1.0 = 0.0
0 * 1.0 = 0.0
ActivationSum (x(i)) = 2
0.6666666666666666, 0.6666666666666666, 0.0, 0.0,
0.2, 0.2, 0.2, 0.2,
0.2, 0.2, 0.2, 0.2,
1, 1, 0, 0,
1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0,
Vector #0 belongs to cluster #0
Vector: 1
InputSum (si) = 1
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.2,
0.0, 0.0, 0.0, 0.2,
0 * 1.0 = 0.0
0 * 1.0 = 0.0
0 * 1.0 = 0.0
1 * 1.0 = 1.0
ActivationSum (x(i)) = 1
0.6666666666666666, 0.6666666666666666, 0.0, 0.0,
0.0, 0.0, 0.0, 1.0,
0.2, 0.2, 0.2, 0.2,
1, 1, 0, 0,
0, 0, 0, 1,
1.0, 1.0, 1.0, 1.0,
Vector #1 belongs to cluster #1
Vector: 2
InputSum (si) = 1
0.6666666666666666, 0.6666666666666666, 0.6666666666666666, 0.6666666666666666,
0.0, 0.0, 0.0, 0.0,
0.2, 0.2, 0.2, 0.2,
1 * 1 = 1
0 * 1 = 0
0 * 0 = 0
0 * 0 = 0
ActivationSum (x(i)) = 1
1.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 1.0,
0.2, 0.2, 0.2, 0.2,
1, 0, 0, 0,
0, 0, 0, 1,
1.0, 1.0, 1.0, 1.0,
Vector #2 belongs to cluster #0
Vector: 3
InputSum (si) = 2
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 1.0,
0.0, 0.0, 0.2, 0.4,
0 * 0 = 0
0 * 0 = 0
1 * 0 = 0
1 * 1 = 1
ActivationSum (x(i)) = 1
1.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 1.0,
0.2, 0.2, 0.2, 0.2,
1, 0, 0, 0,
0, 0, 0, 1,
1.0, 1.0, 1.0, 1.0,
Vector #3 belongs to cluster #1
Vector: 4

InputSum (si) = 1
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
0.0, 0.2, 0.2, 0.2,
0 * 1.0 = 0.0
1 * 1.0 = 1.0
0 * 1.0 = 0.0
0 * 1.0 = 0.0
ActivationSum (x(i)) = 1
Vector #4 belongs to cluster #2
Vector: 5
InputSum (si) = 1
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.2, 0.2,
0 * 1.0 = 0.0
0 * 1.0 = 0.0
1 * 1.0 = 1.0
0 * 1.0 = 0.0
ActivationSum (x(i)) = 1
Vector #5 belongs to cluster #2
Vector: 6
InputSum (si) = 2
1.0, 1.0, 1.0, 1.0,
0.0, 0.0, 0.0, 0.0,
0.2, 0.2, 0.4, 0.4,
1 * 1 = 1
0 * 0 = 0
1 * 0 = 0
0 * 0 = 0
ActivationSum (x(i)) = 1

Vector #6 belongs to cluster #0
Final weight values:
1.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 1.0,
0.2, 0.2, 0.2, 0.2,
1, 0, 0, 0,
0, 0, 0, 1,
1.0, 1.0, 1.0, 1.0,
```

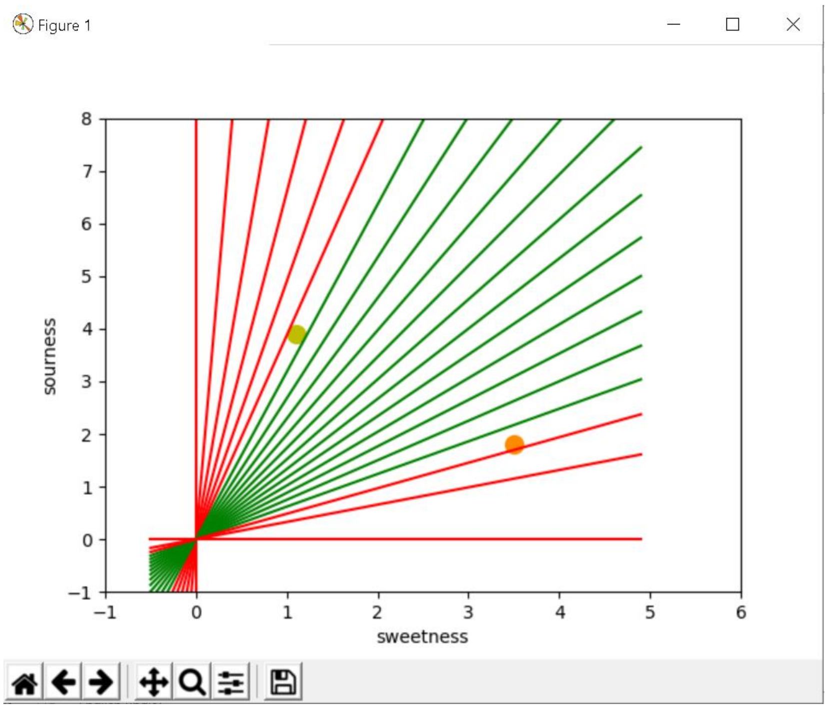
Practical 7

A. Write a program for Linear separation.

Python Code :

```
import numpy as np
import matplotlib.pyplot as plt
def create_distance_function(a, b, c):
    """ 0 = ax + by + c """
    def distance(x, y):
        """ returns tuple (d, pos)
            d is the distance
            If pos == -1 point is below the line,
            0 on the line and +1 if above the line
        """
        nom = a * x + b * y + c
        if nom == 0:
            pos = 0
        elif (nom<0 and b<0) or (nom>0 and b>0):
            pos = -1
        else:
            pos = 1
        return (np.absolute(nom) / np.sqrt( a ** 2 + b ** 2), pos)
    return distance
points = [ (3.5, 1.8), (1.1, 3.9) ]
fig, ax = plt.subplots()
ax.set_xlabel("sweetness")
ax.set_ylabel("sourness")
ax.set_xlim([-1, 6])
ax.set_ylim([-1, 8])
X = np.arange(-0.5, 5, 0.1)
colors = ["r", ""] # for the samples
size = 10
for (index, (x, y)) in enumerate(points):
    if index== 0:
        ax.plot(x, y, "o",
                color="darkorange",
                markersize=size)
    else:
        ax.plot(x, y, "oy",
                markersize=size)
step = 0.05
for x in np.arange(0, 1+step, step):
    slope = np.tan(np.arccos(x))
    dist4line1 = create_distance_function(slope, -1, 0)
    #print("x: ", x, "slope: ", slope)
    Y = slope * X
    results = []
    for point in points:
        results.append(dist4line1(*point))
    #print(slope, results)
    if (results[0][1] != results[1][1]):
        ax.plot(X, Y, "g-")
    else:
        ax.plot(X, Y, "r-")
plt.show()
```

Output :



B. Write a program for Hopfield network model for associative memory

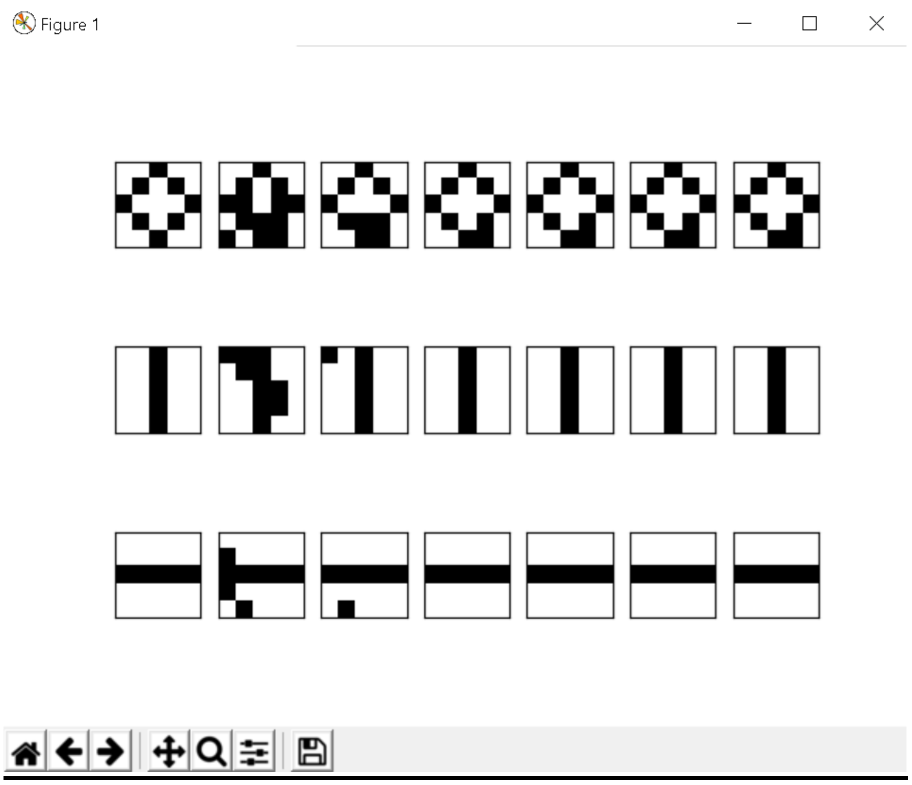
<u>Python Code :</u>

```
import numpy as np

import matplotlib.pyplot as plt

import tempfile

import argparse

############################################

#

# Some patterns that we use for testing

#

############################################

strings = []

strings.append("""

..X..

.X.X.

X...X

.X.X.

..X..""")


strings.append("""

..X..

..X..

..X..

..X..

..X..""")

strings.append("""

.....

.....

XXXXX

.....

.....""")
```

```
strings.append("""
X....
.X...
..X..
...X.
....X""")
strings.append("""
....X
...X.
..X..
.X...
X....""")
##############################################
#
# Some utility functions
#
##############################################
#
# Convert a string as above into a
# 5 x 5 matrix
#
def string_to_matrix(s):
    x = np.zeros(shape=(5,5), dtype=float)
    for i in range(len(s)):
        row, col = i // 5, i % 5
        x[row][col] = -1 if s[i] == 'X' else 1
    return x
#
# and back
#
def matrix_to_string(m):
    s = ""
    for i in range(5):

        for j in range(5):

            s = s + ('X' if m[i][j] < 0 else '')

        s = s + chr(10)

    return s

class HopfieldNetwork:

    #

    # Initialize a Hopfield network with N

    # neurons

    #

    def __init__(self, N):

        self.N = N

        self.W = np.zeros((N,N))

        self.s = np.zeros((N,1))

    #

    # Apply the Hebbian learning rule. The argument is a matrix S

    # which contains one sample state per row

    #

    def train(self, S):

        self.W = np.matmul(S.transpose(), S)

    #

    # Run one simulation step

    #

    def runStep(self):

        i = np.random.randint(0,self.N)

        a = np.matmul(self.W[i,:], self.s)

        if a < 0:

            self.s[i] = -1

        else:

            self.s[i] = 1

    #

    # Starting with a given state, execute the update rule
    # N times and return the resulting state
    #
    def run(self, state, steps):
        self.s = state
        for i in range(steps):
            self.runStep()
        return self.s
############################################
#
# Parse arguments
#
#############################################
def get_args():
    parser = argparse.ArgumentParser()
    parser.add_argument("--memories",
                    type=int,
                    default=3,
                    help="Number of patterns to learn")
    parser.add_argument("--epochs",
                    type=int,
                    default=6,
                    help="Number of epochs")
    parser.add_argument("--iterations",
                    type=int,
                    default=20,
                    help="Number of iterations per epoch")
    parser.add_argument("--errors",
                    type=int,
                    default=5,
                    help="Number of error that we add to each sample")
    parser.add_argument("--save",
                    type=int,
          default=0,

          help="Save output")

    return parser.parse_args()

############################################

#

# Main

#

#############################################

#

# Read parameters

#

args = get_args()

#

# Number of epochs. After each

# epoch, we capture one image

#

epochs = args.epochs

#

# Number of iterations

# per epoch

#

iterations = args.iterations

#

# Number of bits that we flip in each sample

#

errors = args.errors

#

# Number of patterns that we try to memorize

#

memories = args.memories

#

# Init network
```

```
#
HN = HopfieldNetwork(5*5)
#
# Prepare sample data and train network
#
M = []
for _ in range(memories):
    M.append(string_to_matrix(strings[_].replace(chr(10), '')).reshape(1,5*5))
S = np.concatenate(M)
HN.train(S)
#
# Run the network and display results
#
fig = plt.figure()
for pic in range(memories):
    state = (S[pic,:].reshape(25,1)).copy()
    #
    # Display original pattern
    #
    ax = fig.add_subplot(memories,epochs + 1, 1+pic*(epochs+1))
    ax.set_xticks([],[])
    ax.set_yticks([],[])
    ax.imshow(state.reshape(5,5), "binary_r")
    #
    # Flip a few bits
    #
    state = state.copy()
    for i in range(errors):
        index = np.random.randint(0,25)
        state[index][0] = state[index][0]*(-1)
    #
    # Run network and display the current state
    # at the beginning of each epoch
    #
    for i in range(epochs):
        ax = fig.add_subplot(memories,epochs + 1, i+2+pic*(epochs+1))
        ax.set_xticks([],[])
        ax.set_yticks([],[])
        ax.imshow(state.reshape(5,5), "binary_r")
        state = HN.run(state, iterations)


if 1 == args.save:
    outfile = tempfile.mktemp() + "_Hopfield.png"
    print("Using outfile ", outfile)
    plt.savefig(outfile)
plt.show()
```

Output:



Practical 8

A. Membership and Identity Operators | in, not in,

Python Code

```
# Python program to illustrate
# Finding common member in list
# using 'in' operator
list1=[1,2,3,4,5]
list2=[6,7,8,9]
for item in list1:
   if item in list2:
          print("overlapping")
else:
   print("not overlapping")
```

Output

```
   not overlapping
```

Python Code

```
# Python program to illustrate
# Finding common member in list
# without using 'in' operator

# Define a function() that takes two lists
def overlapping(list1,list2):

   c=0
   d=0
   for i in list1:
          c+=1
   for i in list2:
          d+=1
   for i in range(0,c):
          for j in range(0,d):
                 if(list1[i]==list2[j]):
                        return 1
   return 0
list1=[1,2,3,4,5]
list2=[6,7,8,9]
if(overlapping(list1,list2)):
   print("overlapping")
else:
   print("not overlapping")
```

Output

```
not overlapping
```

Python Code

```
# Python program to illustrate
# not 'in' operator
x = 24
y = 20
list = [10, 20, 30, 40, 50 ];

if ( x not in list ):
   print ("x is NOT present in given list")
else:
   print ("x is present in given list")

if ( y in list ):
   print ("y is present in given list")
else:
   print ("y is NOT present in given list")
```

Output

```
x is NOT present in given list
y is present in given list
```

B. Membership and Identity Operators is, is not

Python Code

```
# Python program to illustrate the use
# of 'is' identity operator
x = 5
if (type(x) is int):
            print ("true")
else:
            print ("false")
```

Output

```
true
```

Python Code

```
# Python program to illustrate the
# use of 'is not' identity operator
x = 5.2
if (type(x) is not int):
            print ("true")
else:
            print ("false")
```

Output

```
true
```

Practical 9

A. Find ratios using fuzzy logic

Python Code

```
# Python code showing all the ratios together,
# make sure you have installed fuzzywuzzy module

from fuzzywuzzy import fuzz
from fuzzywuzzy import process

s1 = "I love GeeksforGeeks"
s2 = "I am loving GeeksforGeeks"
print ("FuzzyWuzzy Ratio: ", fuzz.ratio(s1, s2))
print ("FuzzyWuzzy PartialRatio: ", fuzz.partial_ratio(s1, s2))
print ("FuzzyWuzzy TokenSortRatio: ", fuzz.token_sort_ratio(s1, s2))
print ("FuzzyWuzzy TokenSetRatio: ", fuzz.token_set_ratio(s1, s2))
print ("FuzzyWuzzy WRatio: ", fuzz.WRatio(s1, s2),'\n\n')

# for process library,
query = 'geeks for geeks'
choices = ['geek for geek', 'geek geek', 'g. for geeks']
print ("List of ratios: ")
print (process.extract(query, choices), '\n')
print ("Best among the above list: ",process.extractOne(query, choices))
```

Output

```
FuzzyWuzzy Ratio:  84
FuzzyWuzzy PartialRatio:  85
FuzzyWuzzy TokenSortRatio:  84
FuzzyWuzzy TokenSetRatio:  86
FuzzyWuzzy WRatio:  84
List of ratios:
[('g. for geeks', 95), ('geek for geek', 93), ('geek geek', 86)]

Best among the above list:  ('g. for geeks', 95)
```

B. Solve Tipping problem using fuzzy logic

Python Code

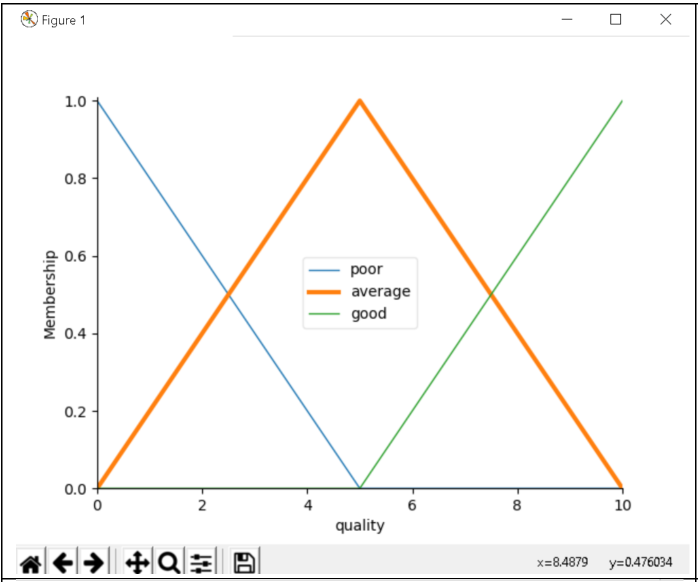
```
"""
==========================================
Fuzzy Control Systems: The Tipping Problem
==========================================
The 'tipping problem' is commonly used to illustrate the power of fuzzy logic principles to generate complex
behavior from a compact, intuitive set of expert rules.
If you're new to the world of fuzzy control systems, you might want to check out the `Fuzzy Control Primer
<../userguide/fuzzy_control_primer.html>`_ before reading through this worked example.
The Tipping Problem
-------------------
Let's create a fuzzy control system which models how you might choose to tip at a restaurant.  When
tipping, you consider the service and food quality, rated between 0 and 10.  You use this to leave a tip of
between 0 and 25%.
We would formulate this problem as:
* Antecednets (Inputs)
   - `service`
      * Universe (ie, crisp value range): How good was the service of the wait
        staff, on a scale of 0 to 10?
      * Fuzzy set (ie, fuzzy value range): poor, acceptable, amazing
   - `food quality`
      * Universe: How tasty was the food, on a scale of 0 to 10?
      * Fuzzy set: bad, decent, great
* Consequents (Outputs)
   - `tip`
      * Universe: How much should we tip, on a scale of 0% to 25%
      * Fuzzy set: low, medium, high
* Rules
   - IF the *service* was good  *or* the *food quality* was good,
     THEN the tip will be high.
   - IF the *service* was average, THEN the tip will be medium.
   - IF the *service* was poor *and* the *food quality* was poor
     THEN the tip will be low.
* Usage
   - If I tell this controller that I rated:
      * the service as 9.8, and
      * the quality as 6.5,
   - it would recommend I leave:
      * a 20.2% tip.
Creating the Tipping Controller Using the skfuzzy control API
-------------------------------------------------------------
We can use the `skfuzzy` control system API to model this.  First, let's
define fuzzy variables
"""
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
```

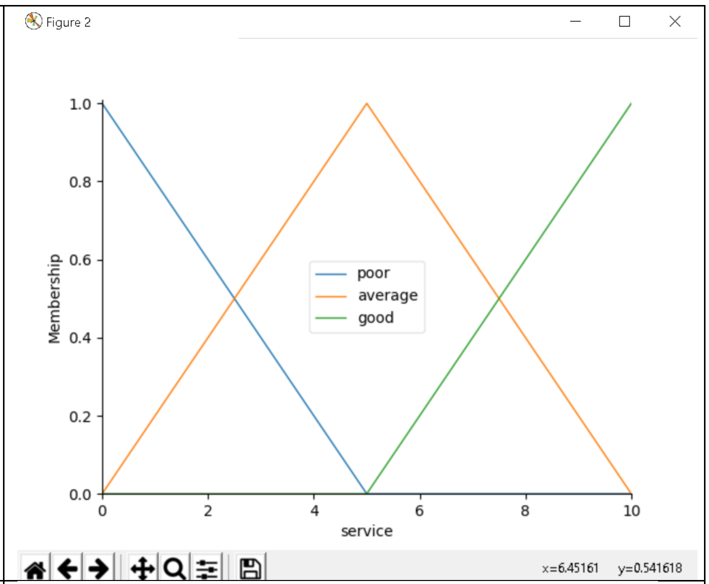
```
# New Antecedent/Consequent objects hold universe variables and membership
# functions
quality = ctrl.Antecedent(np.arange(0, 11, 1), 'quality')
service = ctrl.Antecedent(np.arange(0, 11, 1), 'service')
tip = ctrl.Consequent(np.arange(0, 26, 1), 'tip')

# Auto-membership function population is possible with .automf(3, 5, or 7)
quality.automf(3)
service.automf(3)

# Custom membership functions can be built interactively with a familiar,
# Pythonic API
tip['low'] = fuzz.trimf(tip.universe, [0, 0, 13])
tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])
tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])

"""
To help understand what the membership looks like, use the ``view`` methods.
"""

# You can see how these look with .view()
quality['average'].view()
"""
.. image:: PLOT2RST.current_figure
"""
service.view()
"""
.. image:: PLOT2RST.current_figure
"""
tip.view()
"""
.. image:: PLOT2RST.current_figure

Fuzzy rules
-----------
Now, to make these triangles useful, we define the *fuzzy relationship* between input and output variables.
For the purposes of our example, consider three simple rules:
1. If the food is poor OR the service is poor, then the tip will be low
2. If the service is average, then the tip will be medium
3. If the food is good OR the service is good, then the tip will be high.
Most people would agree on these rules, but the rules are fuzzy. Mapping the imprecise rules into a defined,
actionable tip is a challenge. This is the kind of task at which fuzzy logic excels.
"""

rule1 = ctrl.Rule(quality['poor'] | service['poor'], tip['low'])
rule2 = ctrl.Rule(service['average'], tip['medium'])
rule3 = ctrl.Rule(service['good'] | quality['good'], tip['high'])

rule1.view()
```

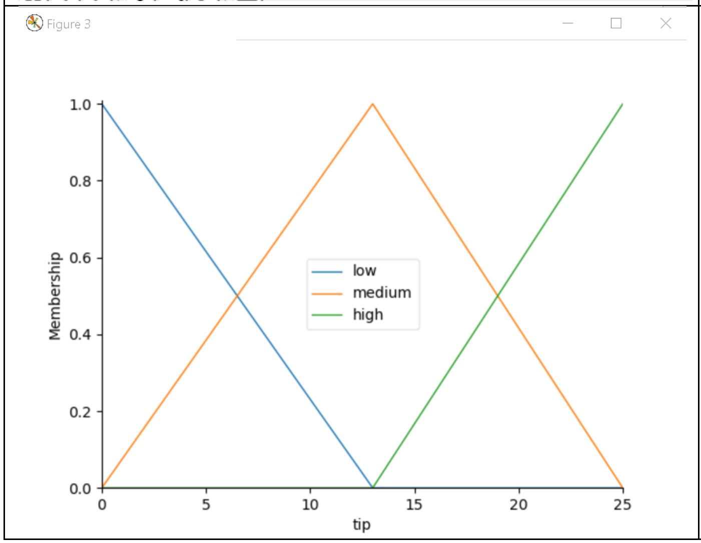
```
"""
.. image:: PLOT2RST.current_figure

Control System Creation and Simulation
---------------------------------------
Now that we have our rules defined, we can simply create a control system via:
"""
tipping_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])

"""
In order to simulate this control system, we will create a
``ControlSystemSimulation``.  Think of this object representing our controller applied to a specific set of
cirucmstances.  For tipping, this might be tipping Sharon at the local brew-pub.  We would create another
``ControlSystemSimulation`` when we're trying to apply our ``tipping_ctrl`` for Travis at the cafe because
the inputs would be different.
"""
tipping = ctrl.ControlSystemSimulation(tipping_ctrl)
"""
We can now simulate our control system by simply specifying the inputs and calling the ``compute``
method.  Suppose we rated the quality 6.5 out of 10 and the service 9.8 of 10.
"""
# Pass inputs to the ControlSystem using Antecedent labels with Pythonic API
# Note: if you like passing many inputs all at once, use .inputs(dict_of_data)
tipping.input['quality'] = 6.5
tipping.input['service'] = 9.8

# Crunch the numbers
tipping.compute()

"""
Once computed, we can view the result as well as visualize it.
"""
print (tipping.output['tip'])
tip.view(sim=tipping)

"""
.. image:: PLOT2RST.current_figure

The resulting suggested tip is **20.24%**.

Final thoughts
--------------
The power of fuzzy systems is allowing complicated, intuitive behavior based on a sparse system of rules
with minimal overhead. Note our membership function universes were coarse, only defined at the integers,
but ``fuzz.interp_membership`` allowed the effective resolution to increase on demand. This system can
respond to arbitrarily small changes in inputs, and the processing burden is minimal.
"""
```

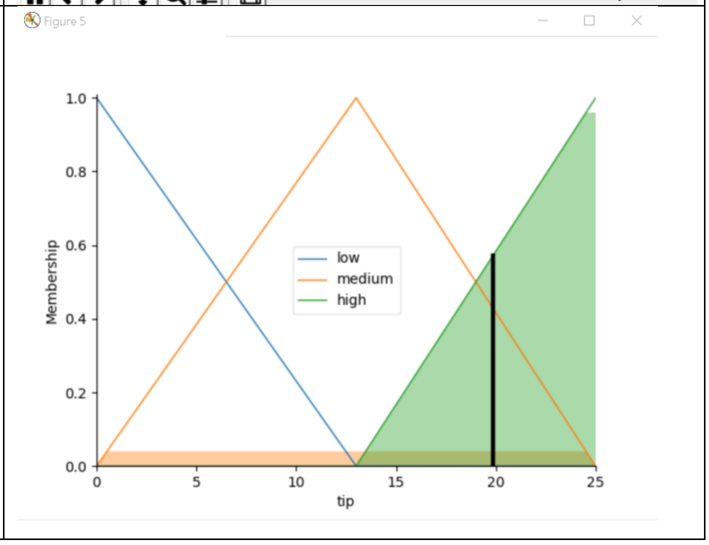
Output

```
19.847607361963192
```









Practical 10

A. Implementation of Simple genetic algorithm.

Python Code

```
import random

# Number of individuals in each generation
POPULATION_SIZE = 100

# Valid genes
GENES = '''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP
QRSTUVWXYZ 1234567890, .-;:_!"#%&/()=?@${[]}'''

# Target string to be generated
TARGET = "I love GeeksforGeeks"

class Individual(object):
    '''
    Class representing individual in population
    '''
    def __init__(self, chromosome):
        self.chromosome = chromosome
        self.fitness = self.cal_fitness()

    @classmethod
    def mutated_genes(self):
        '''
        create random genes for mutation
        '''
        global GENES
        gene = random.choice(GENES)
        return gene

    @classmethod
    def create_gnome(self):
        '''
        create chromosome or string of genes
        '''
        global TARGET
        gnome_len = len(TARGET)
        return [self.mutated_genes() for _ in range(gnome_len)]

    def mate(self, par2):
        '''
        Perform mating and produce new offspring
        '''

        # chromosome for offspring
        child_chromosome = []
        for gp1, gp2 in zip(self.chromosome, par2.chromosome):

            # random probability
            prob = random.random()

            # if prob is less than 0.45, insert gene
            # from parent 1
            if prob < 0.45:
                child_chromosome.append(gp1)

            # if prob is between 0.45 and 0.90, insert
            # gene from parent 2
            elif prob < 0.90:
                child_chromosome.append(gp2)

            # otherwise insert random gene(mutate),
            # for maintaining diversity
            else:
                child_chromosome.append(self.mutated_genes())

        # create new Individual(offspring) using
        # generated chromosome for offspring
        return Individual(child_chromosome)

    def cal_fitness(self):
        '''
        Calculate fittness score, it is the number of
        characters in string which differ from target
        string.
        '''
        global TARGET
        fitness = 0
        for gs, gt in zip(self.chromosome, TARGET):
            if gs != gt: fitness+= 1
        return fitness

# Driver code
def main():
    global POPULATION_SIZE

    #current generation
    generation = 1

    found = False
    population = []

    # create initial population
    for _ in range(POPULATION_SIZE):
                gnome = Individual.create_gnome()
            population.append(Individual(gnome))

    while not found:

        # sort the population in increasing order of fitness score
        population = sorted(population, key = lambda x:x.fitness)

        # if the individual having lowest fitness score ie.
        # 0 then we know that we have reached to the target
        # and break the loop
        if population[0].fitness <= 0:
            found = True
            break

        # Otherwise generate new offsprings for new generation
        new_generation = []

        # Perform Elitism, that mean 10% of fittest population
        # goes to the next generation
        s = int((10*POPULATION_SIZE)/100)
        new_generation.extend(population[:s])

        # From 50% of fittest population, Individuals
        # will mate to produce offspring
        s = int((90*POPULATION_SIZE)/100)
        for _ in range(s):
            parent1 = random.choice(population[:50])
            parent2 = random.choice(population[:50])
            child = parent1.mate(parent2)
            new_generation.append(child)

        population = new_generation

        print("Generation: {}\tString: {}\tFitness:
{}".format(generation,"".join(population[0].chromosome), population[0].fitness))

        generation += 1

    print("Generation: {}\tString: {}\tFitness: {}".format(generation,
"".join(population[0].chromosome), population[0].fitness))

if __name__ == '__main__':
    main()
```

Output

```
RESTART: C:\Users\Yogesh Patil\AppData\Local\Programs\Python\Python37-32\Pract10_A.py
Generation: 1  String: I%hQ8zp ]@x6,oZ!fL@}       Fitness: 18
Generation: 2  String: I%hQ8zp ]@x6,oZ!fL@}       Fitness: 18
Generation: 3  String: I-h1]z  75x=,o2=FL@
          Fitness: 17
Generation: 4  String: I#hinz /(FC6kor=fZ } Fitness: 16
Generation: 5  String: I#hinz /(FC6kor=fZ } Fitness: 16
Generation: 6  String: I-lQ8A @-]P=CorXft@
          Fitness: 15
Generation: 7  String: d-lt]J @e;asCor=F/gk Fitness: 14
Generation: 8  String: d-lt]J @e;asCor=F/gk Fitness: 14
Generation: 9  String: /-ldvP kme5G,orG-a@
          Fitness: 13
Generation: 10String: I[P}vN  xeC=,orGe"6t        Fitness: 12
Generation: 11String: I[l}vN  (e]=,orGe18Q Fitness: 11
Generation: 12String: I[l}vN  (e]=,orGe18Q Fitness: 11
Generation: 13String: d1lovz @ee5s4orGbM@       Fitness: 10
Generation: 14String: d1lovz @ee5s4orGbM@       Fitness: 10
Generation: 15String: d1lovz @ee5s4orGbM@       Fitness: 10
Generation: 16String: I-lnve HJe5s,orGe@@Q       Fitness: 9
Generation: 17String: I-lnve HJe5s,orGe@@Q       Fitness: 9
Generation: 18String: I loDN Fee5skorGeM8        Fitness: 8
Generation: 19String: I lov: Fee]s,orGe1g
          Fitness: 7
Generation: 20String: I lov: Fee]s,orGe1g
          Fitness: 7
Generation: 21String: I lov: Fee]s,orGe1g
          Fitness: 7
Generation: 22String: I lov  @eeYs,orGeen
          Fitness: 6
Generation: 23String: I lov  @eeYs,orGeen
          Fitness: 6
Generation: 24String: I lov  @eeYs,orGeen
          Fitness: 6
Generation: 25String: I lov  @eeYs,orGeen
          Fitness: 6
Generation: 26String: I lovR @eekskorGee;G        Fitness: 5
Generation: 27String: I lovR @eekskorGee;G        Fitness: 5
Generation: 28String: I lovR @eekskorGee;G        Fitness: 5
Generation: 29String: I lovR @eekskorGee;G        Fitness: 5
Generation: 30String: I lovR @eekskorGee;G        Fitness: 5
Generation: 31String: I lovR @eekskorGee;G        Fitness: 5
Generation: 32String: I lovR @eekskorGee;G        Fitness: 5
Generation: 33String: I lovR @eekskorGee;G        Fitness: 5
Generation: 34String: I lovR @eekskorGee;G        Fitness: 5
Generation: 35String: I lovR @eekskorGee;G        Fitness: 5
Generation: 36String: I lovR @eekskorGee;G        Fitness: 5
Generation: 37String: I lovR @eekskorGee;G        Fitness: 5
```

```
Generation: 38String: I lovR @eekskorGee;G          Fitness: 5
Generation: 39String: I lovR @eekskorGee;G          Fitness: 5
Generation: 40String: I lovR @eekskorGee;G          Fitness: 5
Generation: 41String: I lovR @eekskorGee;G          Fitness: 5
Generation: 42String: I lovR @eekskorGee;G          Fitness: 5
Generation: 43String: I lovR @eekskorGee;G          Fitness: 5
Generation: 44String: I lovR @eekskorGee;G          Fitness: 5
Generation: 45String: I lovR @eekskorGee;G          Fitness: 5
Generation: 46String: I lovR @eekskorGee;G          Fitness: 5
Generation: 47String: I lovR @eekskorGee;G          Fitness: 5
Generation: 48String: I love meekskorGee@G          Fitness: 4
Generation: 49String: I love meekskorGee@G          Fitness: 4
Generation: 50String: I love meekskorGee@G          Fitness: 4
Generation: 51String: I love meekskorGee@G          Fitness: 4
Generation: 52String: I love meekskorGee@G          Fitness: 4
Generation: 53String: I love meekskorGee@G          Fitness: 4
Generation: 54String: I love meekskorGee@G          Fitness: 4
Generation: 55String: I love meekskorGee@G          Fitness: 4
Generation: 56String: I love meekskorGee@G          Fitness: 4
Generation: 57String: I love meeksforGeenF Fitness: 3
Generation: 58String: I love meeksforGeenF Fitness: 3
Generation: 59String: I love meeksforGeenF Fitness: 3
Generation: 60String: I love meeksforGeenF Fitness: 3
Generation: 61String: I love meeksforGeenF Fitness: 3
Generation: 62String: I love GeeksforGeenF Fitness: 2
Generation: 63String: I love GeeksforGeenF Fitness: 2
Generation: 64String: I love GeeksforGeenF Fitness: 2
Generation: 65String: I love GeeksforGeenF Fitness: 2
Generation: 66String: I love GeeksforGeenF Fitness: 2
Generation: 67String: I love GeeksforGeenF Fitness: 2
Generation: 68String: I love GeeksforGeenF Fitness: 2
Generation: 69String: I love GeeksforGeenF Fitness: 2
Generation: 70String: I love GeeksforGeenF Fitness: 2
Generation: 71String: I love GeeksforGeenF Fitness: 2
Generation: 72String: I love GeeksforGeenF Fitness: 2
Generation: 73String: I love GeeksforGeenF Fitness: 2
Generation: 74String: I love GeeksforGeenF Fitness: 2
Generation: 75String: I love GeeksforGeenF Fitness: 2
Generation: 76String: I love GeeksforGeenF Fitness: 2
Generation: 77String: I love GeeksforGeenF Fitness: 2
Generation: 78String: I love GeeksforGeenF Fitness: 2
Generation: 79String: I love GeeksforGeenF Fitness: 2
Generation: 80String: I love GeeksforGeenF Fitness: 2
Generation: 81String: I love GeeksforGeenF Fitness: 2
Generation: 82String: I love GeeksforGeenF Fitness: 2
Generation: 83String: I love GeeksforGeenF Fitness: 2
Generation: 84String: I love GeeksforGeenF Fitness: 2
Generation: 85String: I love GeeksforGeenF Fitness: 2
Generation: 86String: I love GeeksforGeenF Fitness: 2
```

```
Generation: 87String: I love GeeksforGeenF Fitness: 2
Generation: 88String: I love GeeksforGeenF Fitness: 2
Generation: 89String: I love GeeksforGeenF Fitness: 2
Generation: 90String: I love GeeksforGeenF Fitness: 2
Generation: 91String: I love GeeksforGeenF Fitness: 2
Generation: 92String: I love GeeksforGeenF Fitness: 2
Generation: 93String: I love GeeksforGeenF Fitness: 2
Generation: 94String: I love GeeksforGeenF Fitness: 2
Generation: 95String: I love GeeksforGeenF Fitness: 2
Generation: 96String: I love GeeksforGeenF Fitness: 2
Generation: 97String: I love GeeksforGeenF Fitness: 2
Generation: 98String: I love GeeksforGeenF Fitness: 2
Generation: 99String: I love GeeksforGeenF Fitness: 2
Generation: 100         String: I love GeeksforGeenF Fitness: 2
Generation: 101         String: I love GeeksforGeenF Fitness: 2
Generation: 102         String: I love GeeksforGeenF Fitness: 2
Generation: 103         String: I love GeeksforGeenF Fitness: 2
Generation: 104         String: I love GeeksforGeenF Fitness: 2
Generation: 105         String: I love GeeksforGeenF Fitness: 2
Generation: 106         String: I love GeeksforGeenF Fitness: 2
Generation: 107         String: I love GeeksforGeenF Fitness: 2
Generation: 108         String: I love GeeksforGeenF Fitness: 2
Generation: 109         String: I love GeeksforGeenF Fitness: 2
Generation: 110         String: I love GeeksforGeenF Fitness: 2
Generation: 111         String: I love GeeksforGeenF Fitness: 2
Generation: 112         String: I love GeeksforGeenF Fitness: 2
Generation: 113         String: I love GeeksforGeenF Fitness: 2
Generation: 114         String: I love GeeksforGeenF Fitness: 2
Generation: 115         String: I love GeeksforGeenF Fitness: 2
Generation: 116         String: I love GeeksforGeenF Fitness: 2
Generation: 117         String: I love GeeksforGeenF Fitness: 2
Generation: 118         String: I love GeeksforGeenF Fitness: 2
Generation: 119         String: I love GeeksforGeenF Fitness: 2
Generation: 120         String: I love GeeksforGeenF Fitness: 2
Generation: 121         String: I love GeeksforGeenF Fitness: 2
Generation: 122         String: I love GeeksforGeenF Fitness: 2
Generation: 123         String: I love GeeksforGeenF Fitness: 2
Generation: 124         String: I love GeeksforGeenF Fitness: 2
Generation: 125         String: I love GeeksforGeenF Fitness: 2
Generation: 126         String: I love GeeksforGeenF Fitness: 2
Generation: 127         String: I love GeeksforGeenF Fitness: 2
Generation: 128         String: I love GeeksforGeenF Fitness: 2
Generation: 129         String: I love GeeksforGeenF Fitness: 2
Generation: 130         String: I love GeeksforGeenF Fitness: 2
Generation: 131         String: I love GeeksforGeenF Fitness: 2
Generation: 132         String: I love GeeksforGeenF Fitness: 2
Generation: 133         String: I love GeeksforGeenF Fitness: 2
Generation: 134         String: I love GeeksforGeenF Fitness: 2
Generation: 135         String: I love GeeksforGeenF Fitness: 2
```

```
Generation: 136	String: I love GeeksforGeenF	Fitness: 2
Generation: 137	String: I love GeeksforGeenF	Fitness: 2
Generation: 138	String: I love GeeksforGeenF	Fitness: 2
Generation: 139	String: I love GeeksforGeenF	Fitness: 2
Generation: 140	String: I love GeeksforGeenF	Fitness: 2
Generation: 141	String: I love GeeksforGeenF	Fitness: 2
Generation: 142	String: I love GeeksforGeenF	Fitness: 2
Generation: 143	String: I love GeeksforGeenF	Fitness: 2
Generation: 144	String: I love GeeksforGeenF	Fitness: 2
Generation: 145	String: I love GeeksforGeenF	Fitness: 2
Generation: 146	String: I love GeeksforGeek	Fitness: 1
Generation: 147	String: I love GeeksforGeek	Fitness: 1
Generation: 148	String: I love GeeksforGeek	Fitness: 1
Generation: 149	String: I love GeeksforGeek	Fitness: 1
Generation: 150	String: I love GeeksforGeek	Fitness: 1
Generation: 151	String: I love GeeksforGeek	Fitness: 1
Generation: 152	String: I love GeeksforGeek	Fitness: 1
Generation: 153	String: I love GeeksforGeek	Fitness: 1
Generation: 154	String: I love GeeksforGeek	Fitness: 1
Generation: 155	String: I love GeeksforGeek	Fitness: 1
Generation: 156	String: I love GeeksforGeek	Fitness: 1
Generation: 157	String: I love GeeksforGeek	Fitness: 1
Generation: 158	String: I love GeeksforGeek	Fitness: 1
Generation: 159	String: I love GeeksforGeek	Fitness: 1
Generation: 160	String: I love GeeksforGeek	Fitness: 1
Generation: 161	String: I love GeeksforGeek	Fitness: 1
Generation: 162	String: I love GeeksforGeek	Fitness: 1
Generation: 163	String: I love GeeksforGeek	Fitness: 1
Generation: 164	String: I love GeeksforGeek	Fitness: 1
Generation: 165	String: I love GeeksforGeek	Fitness: 1
Generation: 166	String: I love GeeksforGeek	Fitness: 1
Generation: 167	String: I love GeeksforGeek	Fitness: 1
Generation: 168	String: I love GeeksforGeek	Fitness: 1
Generation: 169	String: I love GeeksforGeek	Fitness: 1
Generation: 170	String: I love GeeksforGeek	Fitness: 1
Generation: 171	String: I love GeeksforGeek	Fitness: 1
Generation: 172	String: I love GeeksforGeek	Fitness: 1
Generation: 173	String: I love GeeksforGeek	Fitness: 1
Generation: 174	String: I love GeeksforGeek	Fitness: 1
Generation: 175	String: I love GeeksforGeek	Fitness: 1
Generation: 176	String: I love GeeksforGeek	Fitness: 1
Generation: 177	String: I love GeeksforGeek	Fitness: 1
Generation: 178	String: I love GeeksforGeek	Fitness: 1
Generation: 179	String: I love GeeksforGeek	Fitness: 1
Generation: 180	String: I love GeeksforGeek	Fitness: 1
Generation: 181	String: I love GeeksforGeek	Fitness: 1
Generation: 182	String: I love GeeksforGeek	Fitness: 1
Generation: 183	String: I love GeeksforGeek	Fitness: 1
Generation: 184	String: I love GeeksforGeek	Fitness: 1
```

```
Generation: 185     String: I love GeeksforGeek   Fitness: 1
Generation: 186     String: I love GeeksforGeek   Fitness: 1
Generation: 187     String: I love GeeksforGeek   Fitness: 1
Generation: 188     String: I love GeeksforGeek   Fitness: 1
Generation: 189     String: I love GeeksforGeek   Fitness: 1
Generation: 190     String: I love GeeksforGeek   Fitness: 1
Generation: 191     String: I love GeeksforGeek   Fitness: 1
Generation: 192     String: I love GeeksforGeek   Fitness: 1
Generation: 193     String: I love GeeksforGeek   Fitness: 1
Generation: 194     String: I love GeeksforGeek   Fitness: 1
Generation: 195     String: I love GeeksforGeek   Fitness: 1
Generation: 196     String: I love GeeksforGeek   Fitness: 1
Generation: 197     String: I love GeeksforGeek   Fitness: 1
Generation: 198     String: I love GeeksforGeek   Fitness: 1
Generation: 199     String: I love GeeksforGeek   Fitness: 1
Generation: 200     String: I love GeeksforGeek   Fitness: 1
Generation: 201     String: I love GeeksforGeek   Fitness: 1
Generation: 202     String: I love GeeksforGeek   Fitness: 1
Generation: 203     String: I love GeeksforGeek   Fitness: 1
Generation: 204     String: I love GeeksforGeek   Fitness: 1
Generation: 205     String: I love GeeksforGeek   Fitness: 1
Generation: 206     String: I love GeeksforGeek   Fitness: 1
Generation: 207     String: I love GeeksforGeek   Fitness: 1
Generation: 208     String: I love GeeksforGeek   Fitness: 1
Generation: 209     String: I love GeeksforGeek   Fitness: 1
Generation: 210     String: I love GeeksforGeek   Fitness: 1
Generation: 211     String: I love GeeksforGeek   Fitness: 1
Generation: 212     String: I love GeeksforGeek   Fitness: 1
Generation: 213     String: I love GeeksforGeek   Fitness: 1
Generation: 214     String: I love GeeksforGeek   Fitness: 1
Generation: 215     String: I love GeeksforGeek   Fitness: 1
Generation: 216     String: I love GeeksforGeek   Fitness: 1
Generation: 217     String: I love GeeksforGeek   Fitness: 1
Generation: 218     String: I love GeeksforGeek   Fitness: 1
Generation: 219     String: I love GeeksforGeek   Fitness: 1
Generation: 220     String: I love GeeksforGeek   Fitness: 1
Generation: 221     String: I love GeeksforGeek   Fitness: 1
Generation: 222     String: I love GeeksforGeek   Fitness: 1
Generation: 223     String: I love GeeksforGeek   Fitness: 1
Generation: 224     String: I love GeeksforGeek   Fitness: 1
Generation: 225     String: I love GeeksforGeek   Fitness: 1
Generation: 226     String: I love GeeksforGeek   Fitness: 1
Generation: 227     String: I love GeeksforGeek   Fitness: 1
Generation: 228     String: I love GeeksforGeek   Fitness: 1
Generation: 229     String: I love GeeksforGeek   Fitness: 1
Generation: 230     String: I love GeeksforGeek   Fitness: 1
Generation: 231     String: I love GeeksforGeek   Fitness: 1
Generation: 232     String: I love GeeksforGeek   Fitness: 1
Generation: 233     String: I love GeeksforGeek   Fitness: 1
```

```
Generation: 234         String: I love GeeksforGeek   Fitness: 1
Generation: 235         String: I love GeeksforGeek   Fitness: 1
Generation: 236         String: I love GeeksforGeek   Fitness: 1
Generation: 237         String: I love GeeksforGeek   Fitness: 1
Generation: 238         String: I love GeeksforGeeks Fitness: 0
>>>
```

B. Create two classes: City and Fitness using Genetic algorithm

Python Code

```
"""Applying a genetic algorithm to the travelling salesman problem"
"""
import math
import random
class City:
  def __init__(self, x=None, y=None):
    self.x = None
    self.y = None
    if x is not None:
      self.x = x
    else:
      self.x = int(random.random() * 200)
    if y is not None:
      self.y = y
    else:
      self.y = int(random.random() * 200)
  def getX(self):
    return self.x
  def getY(self):
    return self.y
  def distanceTo(self, city):
    xDistance = abs(self.getX() - city.getX())
    yDistance = abs(self.getY() - city.getY())
    distance = math.sqrt( (xDistance*xDistance) + (yDistance*yDistance) )
    return distance
  def __repr__(self):
    return str(self.getX()) + ", " + str(self.getY())
class TourManager:
  destinationCities = []
  def addCity(self, city):
    self.destinationCities.append(city)
  def getCity(self, index):
    return self.destinationCities[index]
  def numberOfCities(self):
    return len(self.destinationCities)
class Tour:
  def __init__(self, tourmanager, tour=None):
    self.tourmanager = tourmanager
    self.tour = []
    self.fitness = 0.0
    self.distance = 0
    if tour is not None:
      self.tour = tour
    else:
      for i in range(0, self.tourmanager.numberOfCities()):
        self.tour.append(None)
  def __len__(self):
    return len(self.tour)
  def __getitem__(self, index):
    return self.tour[index]
  def __setitem__(self, key, value):
    self.tour[key] = value
  def __repr__(self):
    geneString = "|"
    for i in range(0, self.tourSize()):
      geneString += str(self.getCity(i)) + "|"
    return geneString
  def generateIndividual(self):
    for cityIndex in range(0, self.tourmanager.numberOfCities()):
      self.setCity(cityIndex, self.tourmanager.getCity(cityIndex))
    random.shuffle(self.tour)
  def getCity(self, tourPosition):
    return self.tour[tourPosition]
  def setCity(self, tourPosition, city):
    self.tour[tourPosition] = city
    self.fitness = 0.0
    self.distance = 0
  def getFitness(self):
    if self.fitness == 0:
      self.fitness = 1/float(self.getDistance())
    return self.fitness
  def getDistance(self):
    if self.distance == 0:
      tourDistance = 0
      for cityIndex in range(0, self.tourSize()):
        fromCity = self.getCity(cityIndex)
        destinationCity = None
        if cityIndex+1 < self.tourSize():
          destinationCity = self.getCity(cityIndex+1)
        else:
          destinationCity = self.getCity(0)
        tourDistance += fromCity.distanceTo(destinationCity)
      self.distance = tourDistance
    return self.distance
  def tourSize(self):
    return len(self.tour)
  def containsCity(self, city):
    return city in self.tour
class Population:
  def __init__(self, tourmanager, populationSize, initialise):
    self.tours = []
    for i in range(0, populationSize):
      self.tours.append(None)
    if initialise:
      for i in range(0, populationSize):
        newTour = Tour(tourmanager)
```

```
            newTour.generateIndividual()
            self.saveTour(i, newTour)
    def __setitem__(self, key, value):
        self.tours[key] = value
    def __getitem__(self, index):
        return self.tours[index]
    def saveTour(self, index, tour):
        self.tours[index] = tour
    def getTour(self, index):
        return self.tours[index]
    def getFittest(self):
        fittest = self.tours[0]
        for i in range(0, self.populationSize()):
            if fittest.getFitness() <= self.getTour(i).getFitness():
                fittest = self.getTour(i)
        return fittest
    def populationSize(self):
        return len(self.tours)
class GA:
    def __init__(self, tourmanager):
        self.tourmanager = tourmanager
        self.mutationRate = 0.015
        self.tournamentSize = 5
        self.elitism = True
    def evolvePopulation(self, pop):
        newPopulation = Population(self.tourmanager, pop.populationSize(), False)
        elitismOffset = 0
        if self.elitism:
            newPopulation.saveTour(0, pop.getFittest())
            elitismOffset = 1
        for i in range(elitismOffset, newPopulation.populationSize()):
            parent1 = self.tournamentSelection(pop)
            parent2 = self.tournamentSelection(pop)
            child = self.crossover(parent1, parent2)
            newPopulation.saveTour(i, child)
        for i in range(elitismOffset, newPopulation.populationSize()):
            self.mutate(newPopulation.getTour(i))
        return newPopulation
    def crossover(self, parent1, parent2):
        child = Tour(self.tourmanager)
        startPos = int(random.random() * parent1.tourSize())
        endPos = int(random.random() * parent1.tourSize())
        for i in range(0, child.tourSize()):
            if startPos < endPos and i > startPos and i < endPos:
                child.setCity(i, parent1.getCity(i))
            elif startPos > endPos:
                if not (i < startPos and i > endPos):
                    child.setCity(i, parent1.getCity(i))
        for i in range(0, parent2.tourSize()):
            if not child.containsCity(parent2.getCity(i)):
               for ii in range(0, child.tourSize()):
                  if child.getCity(ii) == None:
                     child.setCity(ii, parent2.getCity(i))
                     break
         return child
      def mutate(self, tour):
         for tourPos1 in range(0, tour.tourSize()):
            if random.random() < self.mutationRate:
               tourPos2 = int(tour.tourSize() * random.random())
               city1 = tour.getCity(tourPos1)
               city2 = tour.getCity(tourPos2)
               tour.setCity(tourPos2, city1)
               tour.setCity(tourPos1, city2)
      def tournamentSelection(self, pop):
         tournament = Population(self.tourmanager, self.tournamentSize, False)
         for i in range(0, self.tournamentSize):
            randomId = int(random.random() * pop.populationSize())
            tournament.saveTour(i, pop.getTour(randomId))
         fittest = tournament.getFittest()
         return fittest
if __name__ == '__main__':
   tourmanager = TourManager()
   # Create and add our cities
   city = City(60, 200)
   tourmanager.addCity(city)
   city2 = City(180, 200)
   tourmanager.addCity(city2)
   city3 = City(80, 180)
   tourmanager.addCity(city3)
   city4 = City(140, 180)
   tourmanager.addCity(city4)
   city5 = City(20, 160)
   tourmanager.addCity(city5)
   city6 = City(100, 160)
   tourmanager.addCity(city6)
   city7 = City(200, 160)
   tourmanager.addCity(city7)
   city8 = City(140, 140)
   tourmanager.addCity(city8)
   city9 = City(40, 120)
   tourmanager.addCity(city9)
   city10 = City(100, 120)
   tourmanager.addCity(city10)
   city11 = City(180, 100)
   tourmanager.addCity(city11)
   city12 = City(60, 80)
   tourmanager.addCity(city12)
   city13 = City(120, 80)
```

```
    tourmanager.addCity(city13)
    city14 = City(180, 60)
    tourmanager.addCity(city14)
    city15 = City(20, 40)
    tourmanager.addCity(city15)
    city16 = City(100, 40)
    tourmanager.addCity(city16)
    city17 = City(200, 40)
    tourmanager.addCity(city17)
    city18 = City(20, 20)
    tourmanager.addCity(city18)
    city19 = City(60, 20)
    tourmanager.addCity(city19)
    city20 = City(160, 20)
    tourmanager.addCity(city20)
    # Initialize population
    pop = Population(tourmanager, 50, True);
    print ("Initial distance: " + str(pop.getFittest().getDistance()))
    # Evolve population for 50 generations
    ga = GA(tourmanager)
    pop = ga.evolvePopulation(pop)
    for i in range(0, 100):
       pop = ga.evolvePopulation(pop)
    # Print final results
    print ("Finished")
    print ("Final distance: " + str(pop.getFittest().getDistance()))
    print ("Solution:")
    print (pop.getFittest())
```

Output

```
Initial distance: 1902.801165633492
Finished
Final distance: 1072.7575103728066
Solution:
|40, 120|20, 160|60, 200|80, 180|180, 200|200, 160|180, 60|200, 40|160, 20|100, 40|60, 20|20, 20|20,
40|120, 80|180, 100|140, 140|140, 180|100, 160|100, 120|60, 80|
```

----------------------------------X------------------------------