INDEX

Sr. No	Practical	Date	Sign
1.	A] Design a simple linear neural network model. B] Calculate the output of neural net using both binary and bipolar sigmoidal function		
2.	A] Generate AND/NOT function using McCulloch-Pitts neural net. B] Generate XOR function using McCulloch-Pitts neural net.		
3.	A] Write a program to implement Hebb's rule. B] Implement the Delta Rules		
4.	A] Write a program for Back Propagation Algorithm. B] Write a program for error Back Propagation algorithm		
5.	A] Write a program for Hopfield Network. B] Write a program for Radial Basis function		
6.	A] Implementation of Kohonen Self Organising Map B] Implementation Adaptive Resonance Theory		
7.	A] Write a program for Linear separation. B] Write a program for Hopfield network model for associative memory.		
8.	A] Membership and Identity Operators | in, not in B] Membership and Identity Operators is True or False		
9.	A] Find ratios using fuzzy logic B] Solve Tipping problem using fuzzy logic		
10.	A] Implementation of Simple genetic algorithm. B] Create two classes: City and Fitness using Genetic algorithm		

Practical No: 1

A] Design a simple linear neural network model.

CODE :

```
x=float(input("Enter value of x:"))
w=float(input("Enter value of weight w:"))
b=float(input("Enter value of bias b:"))
net = int(w*x+b)
if(net<0):
	out=0
elif((net>=0)&(net<0)):
	out =net
else:
	out=1
print("net=",net)
print("output=",out)
```

OUTPUT :

```
In [4]: runfile('C:/Users/DELL/Desktop/practicals/sct/Practical 1A.py', wdir='C:/Users/DELL/Desktop/practicals/sct')
Enter value of x:7
Enter value of weight w:4
Enter value of bias b:8
net= 36
output= 1
```

B] Calculate the output of neural net using both binary and bipolar sigmoidal function

CODE :

```
import math
inputs=int(input("Enter the no. of input layer neurons="))
print("Enter the input neurons values:")
inputsn=[]
for i in range (0,inputs):
    elements=float(input())
    inputsn.append(elements)
print(inputsn)
print("Enter the weight for input layer neurons:")
weight=[]
for i in range (0,inputs):
    weele=float(input())
    weight.append(weele)
print(weight)
print("Calculating the net input the output nueron")
Yinn=[]
for i in range (0,inputs):
    Yinn.append(inputsn[i]*weight[i])
    Yin=(round(sum(Yinn),3))
print(Yin)
print("The output from the neuron in case of a binary Sigmoidal Acyivation Function")
Y=1/(1+math.exp(-Yin))
print(Y)
print("The output from the neuron in case of a Bipolar Sigmoidal Activation Function")
Y=2/(1+math.exp(-Yin))
print(Y)
```

OUTPUT :

```
In [7]: runfile('C:/Users/DELL/Desktop/practicals/sct/Practical 1B.py', wdir='C:/Users/
DELL/Desktop/practicals/sct')
Enter the no. of input layer neurons=3
Enter the input neurons values:
0.3
0.5
0.6
[0.3, 0.5, 0.6]
Enter the weight for input layer neurons:
0.2
0.1
-0.3
[0.2, 0.1, -0.3]
Calculating yhe net inputn the output nueron
-0.07
The output from the neuron in case of a binary Sigmoidal Acyivation Function
0.48250714233361025
The output from the neuron in case of a Bipolar Sigmoidal Activation Function
0.9650142846672205
```

Practical No: 2

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

CODE :

```
num_ip = int(input("Enter the number of inputs: "))
w1 = 1
w2 = -1
print("For the", num_ip, "inputs, calculate the net input using net input formula")
x1 = []
x2 = []
for j in range(num_ip):
    element1 = int(input("X1 = "))
    element2 = int(input("X2 = "))
    x1.append(element1)
    x2.append(element2)
print("X1 =", x1)
print("X2 =", x2)
n = [element * w1 for element in x1]
m = [element * w2 for element in x2]
Yin_sum = [n[i] + m[i] for i in range(num_ip)]  # Sum of weighted inputs
Yin_diff = [n[i] - m[i] for i in range(num_ip)]  # Difference of weighted inputs
print("Yin (Sum) =", Yin_sum)
print("After assuming one weight as excitatory and the other as inhibitory, Yin
(Difference) =", Yin_diff)
Y = []
for i in range(num_ip):
    if Yin_sum[i] >= 1:
        Y.append(1)
    else:
        Y.append(0)
print("Y =", Y)
```

OUTPUT :

```
In [24]: runfile('C:/Users/DELL/Desktop/practicals/sct/untitled0.py', wdir='C:/Users/
DELL/Desktop/practicals/sct')
Enter the number of inputs: 4
For the 4 inputs, calculate the net input using net input formula
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 (Sum) = [0, -1, 1, 0]
After assuming one weight as excitatory and the other as inhibitory, Yin (Difference) =
[0, 1, 1, 2]
Y = [0, 0, 1, 0]
```

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

CODE :

```
print("\nXOR function using McClloch-Pitts")
x1inputs = [1,1,0,0]
x2inputs = [1,0,1,0]
print("Calculating z1 = x1w11 + x2w12")
print("Considering one weight as exciatatory and other as inhibitory ")
w11 = [1,1,1,1]
w12 = [-1,-1,-1,-1]
print("x1","x2","z1")
z1 = []
for i in range(0,4):
    z1.append(x1inputs[i]*w11[i] + x2inputs[i]*w12[i])
    print(x1inputs[i]," ",x2inputs[i]," ",z1[i])
print("\nCalculating z1 = x1w21 + x2w22")
print("Considering one weight as exciatatory and other as inhibitory ")
w21 = [-1,-1,-1,-1]
w22 = [1,1,1,1]
print("x1","x2","z2")
z2 = []
for i in range(0,4):
    z2.append(x1inputs[i]*w21[i] + x2inputs[i]*w22[i])
    print(x1inputs[i]," ",x2inputs[i]," ",z2[i])
print("\nApplying Threshold = 1 for z1 and z2")
for i in range(0,4):
    if(z1[i]>=1):
        z1[i] = 1
    else:
        z1[i] = 0
    if(z2[i]>=1):
        z2[i]=1
    else:
        z2[i]=0
print("z1","z2")
for i in range(0,4):
    print(z1[i]," ",z2[i]," ")
print("x1" , "x2" , "yin")
yin = []
v1 = 1
v2 = 1
for i in range(0,4):
    yin.append(z1[i]*v1 + z2[i]*v2)
    print(x1inputs[i]," ",x2inputs[i]," ",yin[i])
y=[]
for i in range(0,4):
    if(yin[i]>=1):
```

```
        y.append(1)
    else:
        y.append(0)
print("x1","x2","y")
for i in range(0,4):
    print(x1inputs[i]," ",x2inputs[i]," ",y[i])
```

OUTPUT :

```
In [32]: runfile('C:/Users/DELL/Desktop/practicals/sct/Practical 2B.py', wdir='C:/Users/DELL/Desktop/practicals/
sct')

XOR function using McClloch-Pitts
Calculating z1 = x1w11 + x2w12
Considering one weight as exciatatory and other as inhibitory
x1 x2 z1
1   1   0
1   0   1
0   1   -1
0   0   0

Calculating z1 = x1w21 + x2w22
Considering one weight as exciatatory and other as inhibitory
x1 x2 z2
1   1   0
1   0   -1
0   1   1
0   0   0

Applying Threshold = 1 for z1 and z2
z1 z2
0   0
1   0
0   1
0   0
x1 x2 yin
1   1   0
1   0   1
0   1   1
0   0   0
x1 x2 y
1   1   0
1   0   1
0   1   1
0   0   0
```

Practical No: 3

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

CODE:

```
import numpy as np
x1 = np.array([1, 1, 1, -1, 1, -1, 1, 1, 1])
x2 = np.array([1, 1, 1, 1, -1, 1, 1, 1, 1])
b = 0
y = np.array([1, -1])
wtold = np.zeros(9)
wtnew = np.zeros(9)
wtnew = wtnew.astype(int)
wtold = wtold.astype(int)
eta = 1
print("First input with target=1")
for i in range(9):
    wtold[i] = wtold[i] + eta * x1[i] * y[0]
b = b + eta * y[0]
wtnew = wtold
print("New weights:", wtnew)
print("Bias value:", b)
print("Second input with target=-1")
for i in range(9):
    wtold[i] = wtold[i] + eta * x2[i] * y[1]
b = b + eta * y[1]
wtnew = wtold
print("New weights:", wtnew)
print("Bias value:", b)
```

OUTPUT :

```
In [33]: runfile('C:/Users/DELL/Desktop/practicals/sct/Practical 3A.py', wdir='C:/Users/DELL/
Desktop/practicals/sct')
First input with target=1
New weights: [ 1  1  1 -1  1 -1  1  1  1]
Bias value: 1
Second input with target=-1
New weights: [ 0  0  0 -2  2 -2  0  0  0]
Bias value: 0
```

B] Implement the Delta Rules.

CODE :

```
import numpy as np
x=np.zeros ((3,))
weights=np.zeros((3,))
desired=np.zeros((3,))
actual=np.zeros((3,))
for i in range(0,3):
        x[i]=float(input("Intial inputs:"))
for i in range(0,3):
        weights[i]=float(input("Intial weights:"))
for i in range(0,3):
    desired[i]=float(input("Intial desired:"))
a=float(input("Enter learning rate:"))
actual=x*weights
print("Actual initial",actual)
print("Actual desired",desired)
while True:
        if np.array_equal(desired,actual):
            break
        else:
            for i in range(0,3):
                weights[i]=weights[i]+a*(desired[i]-actual[i])
            actual=x*weights
print("*" * 30)
print("Final output using delta rule")
print("Corrected weights",weights)
print("actual",actual)
print("desired",desired)
```

OUTPUT :

```
In [49]: runfile('C:/Users/DELL/Desktop/practicals/sct/Practical 3B.py', wdir='C:/Users/DELL/Desktop/practicals/sct')
Intial inputs:1
Intial inputs:1
Intial inputs:1
Intial weights:1
Intial weights:1
Intial weights:1
Intial desired:2
Intial desired:3
Intial desired:4
Enter learning rate:1
Actual initial [1. 1. 1.]
Actual desired [2. 3. 4.]
******************************
Final output using delta rule
Corrected weights [2. 3. 4.]
actual [2. 3. 4.]
desired [2. 3. 4.]
```

Practical No: 4

A] Write a program for Back Propagation Algorithm.

CODE :

```
import numpy as np
import decimal
import math
np.set_printoptions(precision=2)
v1=np.array([0,6,0,3])
v2=np.array([-0.1,0.4])
w=np.array([-0.2,0.4,0.1])
b1=0.3
b2=0.5
x1=0
x2=1
alpha=0.25
print("calculate net input to z1 layer")
zin1=round(b1+x1*v1[0]+x2*v2[0],4)
print("z1=", round(zin1,3))
print("calculate net input to z2 layer")
zin2=round(b2+x1*v1[1]+x2*v2[1],4)
print("z2=", round(zin2,4))
print("Apply activation function to calculate output")
z1=1/(1+math.exp(-zin1))
z1=round(z1,4)
z2=1/(1+math.exp(-zin2))
z2=round(z2,4)
print("z1=",z1)
print("z2=",z2)
print("calculate net input to output layer")
yin=w[0]+z1*w[1]+z2*w[2]
print("Yin=",yin)
print("calculate net output")
y=1/(1+math.exp(-yin))
print("y=",y)
fyin=y*(1-y)
dk=(1-y)*fyin
print("dk=",dk)
dw1= alpha * dk * z1
dw2= alpha * dk * z2
dw0= alpha * dk
print("compute error portion in delta")
din1=dk* w[1]
din2=dk* w[2]
print("din1=",din1)
print("din2=",din2)
```

```
print("error in delta")
fzin1= z1 *(1-z1)
print("fzin1",fzin1)
d1=din1*fzin1
fzin2=z2*(1-z2)
print("fzin2",fzin2)
d2=din2 * fzin2
print("d1=",d1)
print("d2=",d2)
print("changes in weights between input and hidden layer")
dv11=alpha * d1 * x1
print("dv11=",dv11)
dv21=alpha * d1 * x2
print("dv21=",dv21)
dv01=alpha * d1
print("dv01=",dv01)
dv12=alpha * d2 * x1
print("dv12=",dv12)
dv22=alpha * d2 * x2
print("dv22=",dv22)
dv02=alpha * d2
print("dv02=",dv02)
print("Final weights of network")
v1[0]=v1[0]+dv11
v1[1]=v1[1]+dv12
print("v=",v1)
v2[0]=v2[0]+dv21
v2[1]=v2[1]+dv22
print("v2=",v2)
w[1]=w[1]+dw1
w[2]=w[2]+dw2
b1=b1+dv01
b2=b2+dv02
w[0]=w[0]+dw0
print("w=",w)
print("bias b1=",b1, "b2=",b2)
```

OUTPUT :

```
In [50]: runfile('C:/Users/DELL/Desktop/practicals/sct/Practical 4A.py', wdir='C:/Users/DELL/
Desktop/practicals/sct')
calculate net input to z1 layer
z1= 0.2
calculate net input to z2 layer
z2= 0.9
Apply activation function to calculate output
z1= 0.5498
z2= 0.7109
calculate net input to output layer
Yin= 0.09101
calculate net output
y= 0.5227368084248941
dk= 0.11906907074145694
compute error portion in delta
din1= 0.04762762829658278
din2= 0.011906907074145694
error in delta
fzin1 0.24751996
fzin2 0.20552119000000002
d1= 0.011788788650865037
d2= 0.0024471217110978417
changes in weights between input and hidden layer
dv11= 0.0
dv21= 0.0029471971627162592
dv01= 0.0029471971627162592
dv12= 0.0
dv22= 0.0006117804277744604
dv02= 0.0006117804277744604
Final weights of network
v= [0 6 0 3]
v2= [-0.1  0.4]
w= [-0.17  0.42  0.12]
bias b1= 0.30294719716271623 b2= 0.5006117804277744
```

B] Write a program for error Back Propagation algorithm.

CODE :

```
import math
a0=-1
t=-1
w10=float(input("Enter weight first network:"))
b10=float(input("Enter base first network:"))
w20=float(input("Enter weight second network:"))
b20=float(input("Enter base second network:"))
c=float(input("Enter learning coefficient:"))
n1=float(w10*c+b10)
a1=math.tanh(n1)
n2=float(w20*a1+b20)
a2=math.tanh(float(n2))
e=t-a2
s2=-2*(1-a2*a2)*e
s1=(1-a1*a1)*w20*s2
w21=w20-(c*s2*a1)
w11=w10-(c*s1*a0)
b21=b20-(c*s2)
b11=b10-(c*s1)
print("The updated weight of first n/w w11=",w11)
print("The uploaded weight of second n/w w21=",w21)
print("The updated base of first n/w b10=",b10)
print("The uploaded base of second n/w b20=",b20)
```

OUTPUT :

```
In [52]: runfile('C:/Users/DELL/Desktop/practicals/sct/Practical 4B.py', wdir='C:/Users/DELL/
Desktop/practicals/sct')
Enter weight first network:12
Enter base first network:35
Enter weight second network:23
Enter base second network:45
Enter learning coefficient:11
The updated weight of first n/w w11= 12.0
The uploaded weight of second n/w w21= 23.0
The updated base of first n/w b10= 35.0
The uploaded base of second n/w b20= 45.0
```

Practical No: 5

A] Write a program for Hopfield Network.

CODE :

CPP Code:

```
#include "hop.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<<"\n activation is"<<nrn[i].activation;
output[i] = threshld(nrn[i].activation);
cout<<"\n output value is"<<output[i]<<"\n";
}
```

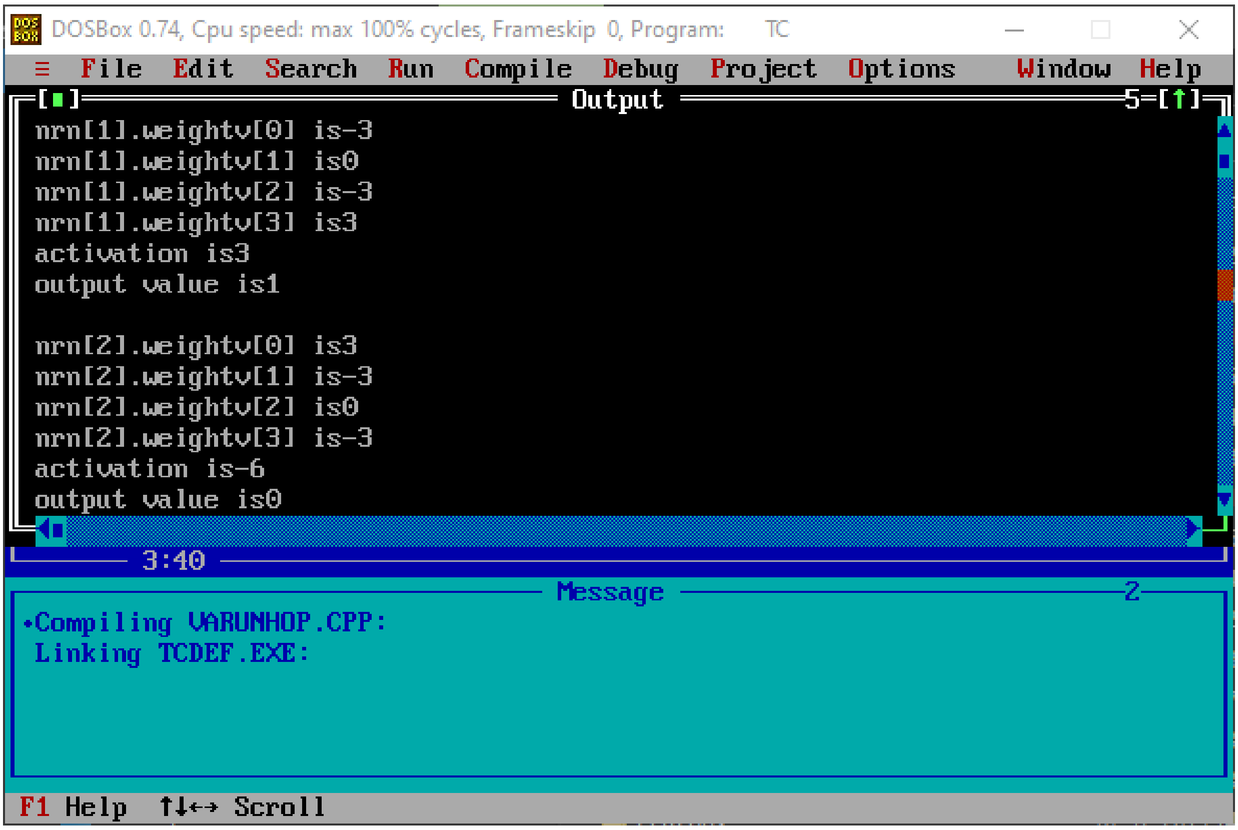
```
}
void main()
{
int patrn1[] = {1,0,1,0}, i;
int wt1[] = {0,-3,3,-3};
int wt2[] = {-3,0,-3,3};
int wt3[] = {3,-3,0,-3};
int wt4[] = {-3,3,-3,0};

cout<<"\n THIS PROGRAM IS FOR HOPFIELD NETWORK WITH A SINGLE LAYER OF";
cout<<"\n4 FULLY INTERCONNECTED NEURONS. THE NETWORK SHOULD RECALL THE";
cout<<"\n PATTERNS 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<<"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";
}
}
```

Header code:

```
#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*);
};
```

OUTPUT :



B] Write a program for Radial Basis function

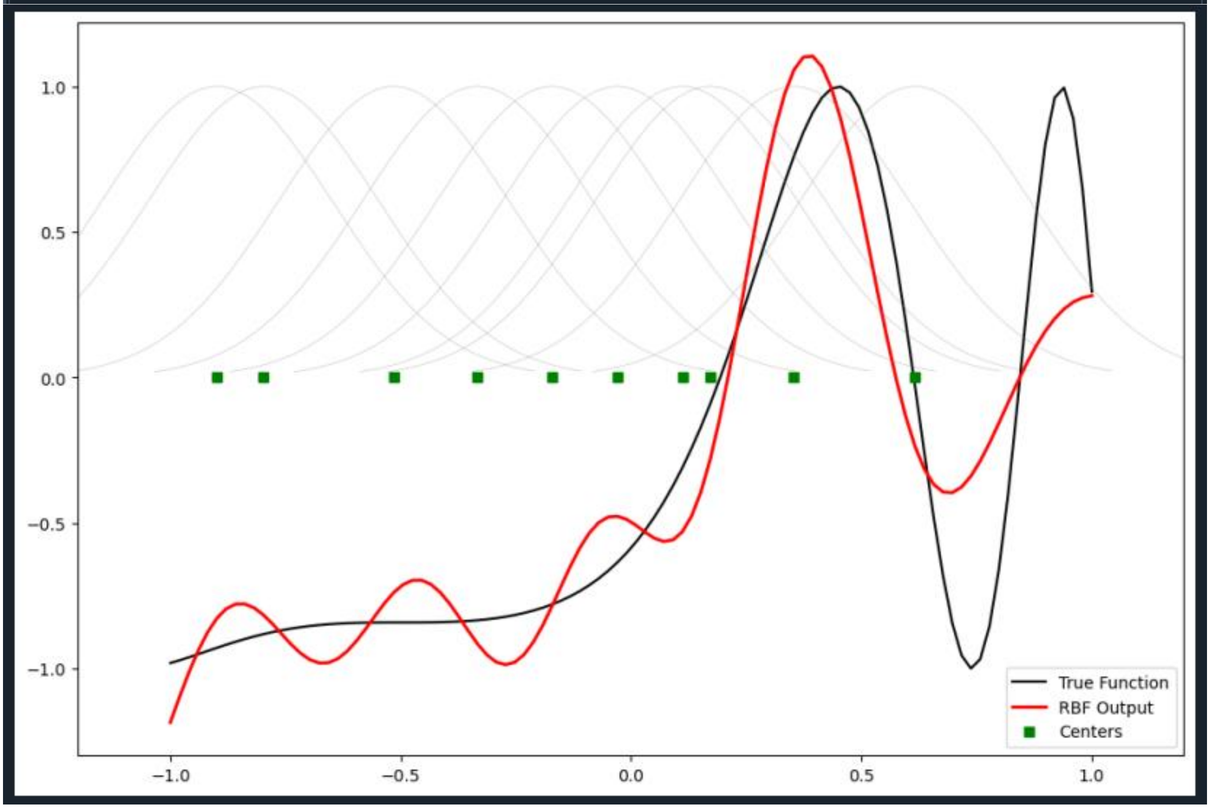
CODE :

```
from scipy.linalg import norm, pinv
from matplotlib import pyplot as plt
import numpy as np

class RBF:
    def __init__(self, indim, numCenters, outdim):
        self.indim = indim
        self.outdim = outdim
        self.numCenters = numCenters
        self.centers = [np.random.uniform(-1, 1, indim) for _ in range(numCenters)]
        self.beta = 8
        self.W = np.random.random((self.numCenters, self.outdim))
    def _basisfunc(self, c, d):
        """Gaussian Radial Basis Function"""
        assert len(d) == self.indim
        return np.exp(-self.beta * norm(c - d) ** 2)
    def _calcAct(self, X):
        """Calculate activation matrix G"""
        G = np.zeros((X.shape[0], self.numCenters), float)
        for ci, c in enumerate(self.centers):
            for xi, x in enumerate(X):
                G[xi, ci] = self._basisfunc(c, x)
        return G
    def train(self, X, Y):
        """Train the RBF network"""
        rnd_idx = np.random.permutation(X.shape[0])[:self.numCenters]
        self.centers = [X[i, :] for i in rnd_idx]
        print("Centers:", self.centers)
        G = self._calcAct(X)
        self.W = np.dot(pinv(G), Y)
    def test(self, X):
        """Test the RBF network"""
        G = self._calcAct(X)
        Y = np.dot(G, self.W)
        return Y
if __name__ == '__main__':
    n = 100
    x = np.mgrid[-1:1:complex(0, n)].reshape(n, 1)
    y = np.sin(3 * (x + 0.5) ** 3 - 1)
    rbf = RBF(1, 10, 1)
    rbf.train(x, y)
    z = rbf.test(x)
    plt.figure(figsize=(12, 8))
    plt.plot(x, y, 'k-', label='True Function')
```

```
plt.plot(x, z, 'r-', linewidth=2, label='RBF Output')
plt.plot(rbf.centers, np.zeros(rbf.numCenters), 'gs', label='Centers')
for c in rbf.centers:
    cx = np.arange(c - 0.7, c + 0.7, 0.01)
    cy = [rbf._basisfunc(np.array([cx_]), np.array([c])) for cx_ in cx]
    plt.plot(cx, cy, '-', color='gray', linewidth=0.2)
plt.xlim(-1.2, 1.2)
plt.legend()
plt.show()
```

OUTPUT :

```
In [53]: runfile('C:/Users/DELL/Desktop/practicals/sct/Practical 5B.py', wdir='C:/Users/DELL/Desktop/practicals/sct')
Centers: [array([-0.17]), array([0.62]), array([-0.03]), array([0.17]), array([-0.8]), array([-0.9]), array([0.35]), array([-0.52]), array([0.11]), array([-0.33])]
c:\users\dell\desktop\practicals\sct\practical 5b.py:48: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performing this operation. (Deprecated NumPy 1.25.)
  cx = np.arange(c - 0.7, c + 0.7, 0.01)
```



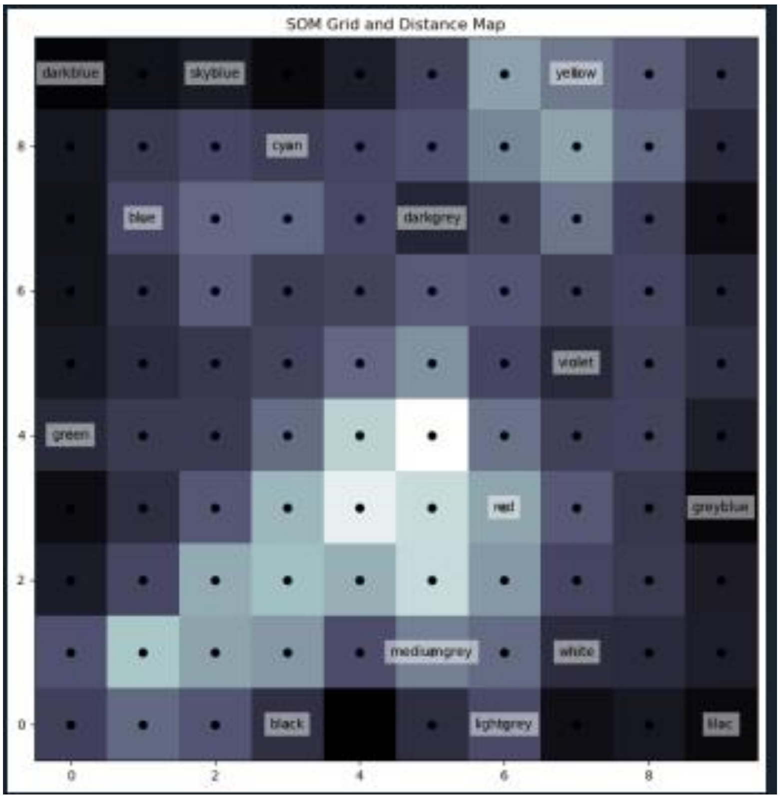
Practical No: 6

A] Implementation of Kohonen Self Organizing Map

CODE :

```
import numpy as np
from minisom import MiniSom
import matplotlib.pyplot as plt

colors = np.array([
    [0., 0., 0.],
    [0., 0., 1.],
    [0., 0., 0.5],
    [0.125, 0.529, 1.0],
    [0.33, 0.4, 0.67],
    [0.6, 0.5, 1.0],
    [0., 1., 0.],
    [1., 0., 0.],
    [0., 1., 1.],
    [1., 0., 1.],
    [1., 1., 0.],
    [1., 1., 1.],
    [0.33, 0.33, 0.33],
    [0.5, 0.5, 0.5],
    [0.66, 0.66, 0.66]
])

color_names = [
    'black', 'blue', 'darkblue', 'skyblue', 'greyblue', 'lilac',
    'green', 'red', 'cyan', 'violet', 'yellow', 'white',
    'darkgrey', 'mediumgrey', 'lightgrey'
]

som = MiniSom(x=10, y=10, input_len=3, sigma=1.0, learning_rate=0.5)
som.train_batch(colors, 1000)
plt.figure(figsize=(12, 10))
plt.imshow(som.distance_map().T, cmap='bone', origin='lower')
for i, color in enumerate(colors):
    w = som.winner(color)
    plt.text(w[1], w[0], color_names[i], ha='center', va='center',
            bbox=dict(facecolor='white', alpha=0.5, lw=0))
som_weights = som.get_weights()
n_rows, n_cols = som_weights.shape[0], som_weights.shape[1]
for i in range(n_rows):
    for j in range(n_cols):
        plt.plot(j, i, 'ko')

plt.title('SOM Grid and Distance Map')
plt.show()
```

OUTPUT :


SOM Grid and Distance Map
darkblue
skyblue
yellow
cyan
blue
darkgrey
violet
green
red
greyblue
mediumgrey
white
black
lightgrey
lilac
0
2
4
6
8

B] Adaptive Resonance Therory

CODE :

```
from __future__ import print_function
from __future__ import division
import numpy as np

class ART:
    def __init__(self, n=5, m=10, rho=.5):
        self.F1 = np.ones(n)          # Initial input field
        self.F2 = np.ones(m)          # Initial output field
        self.Wf = np.random.random((m, n))  # Forward weights
        self.Wb = np.random.random((n, m))  # Backward weights
        self.rho = rho                # Vigilance parameter
        self.active = 0               # Active neuron index

    def learn(self, X):
        self.F2[...] = np.dot(self.Wf, X)
        I = np.argsort(self.F2[:self.active].ravel())[::-1]
        for i in I:
            d = (self.Wb[:,i] * X).sum() / X.sum()  # Similarity measure
            if d >= self.rho:  # Match condition
                self.Wb[:, i] *= X   # Update weights
                self.Wf[i, :] = self.Wb[:, i] / (0.5 + self.Wb[:, i].sum())
                return self.Wb[:, i], i
        if self.active < self.F2.size:  # If no match, create a new active neuron
            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 no suitable neuron is found, return None


if __name__ == '__main__':
    np.random.seed(1)  # For reproducibility
    network = ART(5, 10, rho=0.5) # Create ART network instance
    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"
    ]    # Define the dataset (list of strings)

    max_length = max(len(s) for s in data)
    for i in range(len(data)):
        X = np.zeros(max_length)
        for j in range(len(data[i])):
            if data[i][j] == 'O':
                X[j] = 1
        Z, k = network.learn(X)
        if k is not None:
            print("|%s| -> class %d" % (data[i], k))
```

```
    else:
        print("|%s| -> no class assigned" % data[i])
```

OUTPUT :

```
In [10]: runfile('C:/Users/DELL/Desktop/practicals/sct/Practical 6B.py', wdir='C:/Users/DELL/Desktop/practicals/sct')
| 0 | -> class 0
| 0 0| -> class 1
| 0| -> class 2
| 0 0| -> class 1
| 0| -> class 2
| 0 0| -> class 1
| 0| -> class 2
| 00 0| -> class 3
| 00 | -> class 3
| 00 0| -> class 4
| 00 | -> class 3
|000 | -> class 5
|00 | -> class 6
|0 | -> class 6
|00 | -> class 7
|000 | -> class 8
|0000 | -> class 8
|00000| -> class 9
|0 | -> class 6
| 0 | -> class 2
| 0 | -> class 2
| 0 | -> class 2
| 0| -> class 2
| 0 0| -> class 1
| 00 0| -> class 8
| 00 | -> class 8
|000 | -> class 8
|00 | -> class 9
|0000 | -> no class assigned
|00000| -> no class assigned
```

Practical No: 7

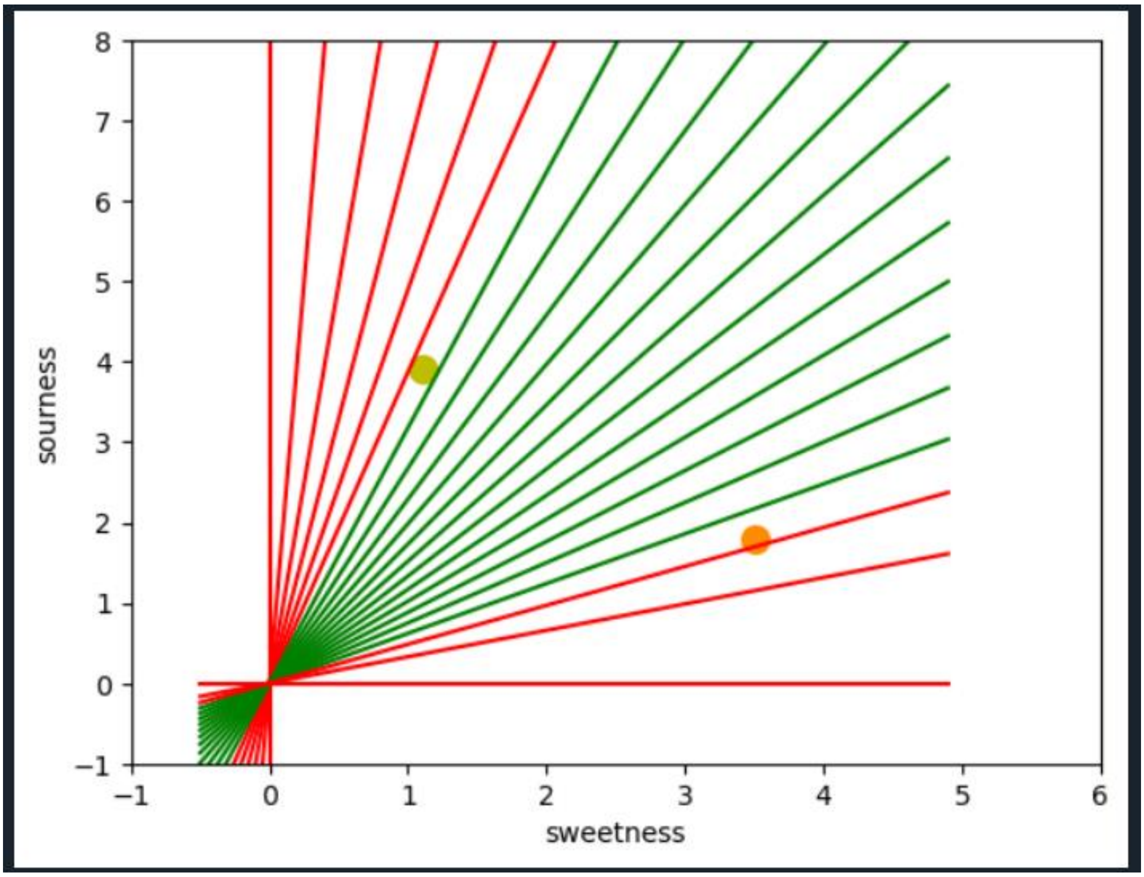
A] Write a program for Linear separation.

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)
    Y = slope * X
    results = [dist4line1(*point) for point in points]
```

```
    if results[0][1] != results[1][1]:
        ax.plot(X, Y, "g-")
    else:
        ax.plot(X, Y, "r-")

plt.show()
```

OUTPUT :

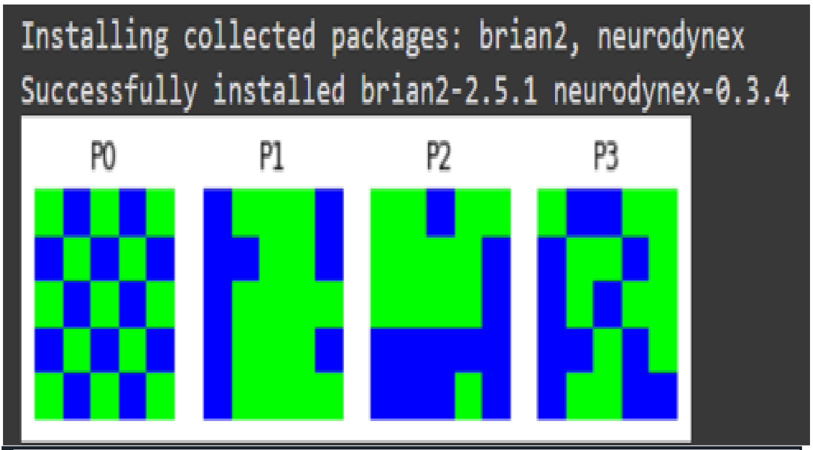


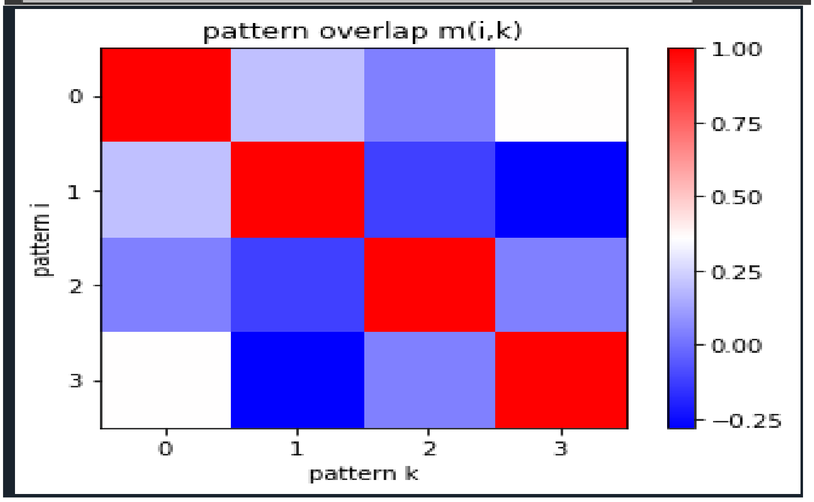
B] Write a program for Hopfield network model for associative memory

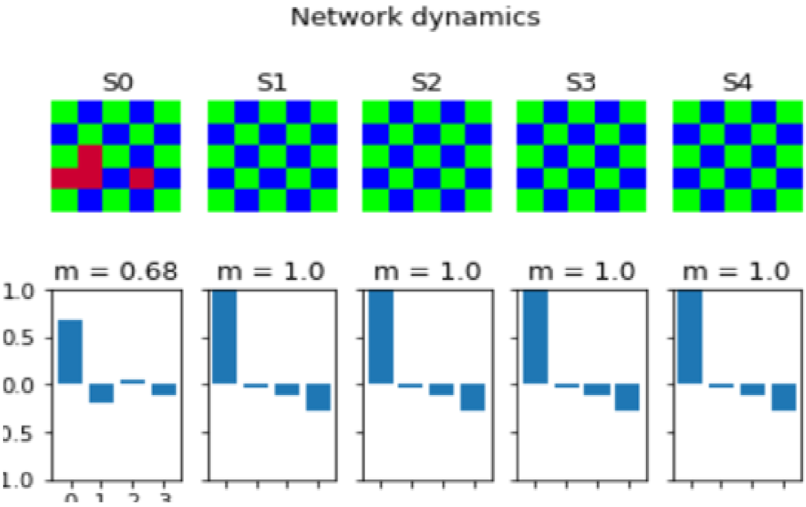
CODE:

```
#pip install neurodynex

from neurodynex.hopfield_network import network, pattern_tools, plot_tools
import matplotlib.pyplot as plt
import numpy as np

pattern_size = 5
hopfield_net = network.HopfieldNetwork(nr_neurons=pattern_size**2)

# Create pattern factory for generating patterns
factory = pattern_tools.PatternFactory(pattern_size, pattern_size)

# Generate a checkerboard pattern
checkerboard = factory.create_checkerboard()

# Create a list of patterns including random patterns
pattern_list = [checkerboard]
pattern_list.extend(factory.create_random_pattern_list(nr_patterns=3,
on_probability=0.5))

# Plot the patterns
plot_tools.plot_pattern_list(pattern_list)

# Compute and plot the overlap matrix
overlap_matrix = pattern_tools.compute_overlap_matrix(pattern_list)
plot_tools.plot_overlap_matrix(overlap_matrix)

# Flatten the patterns before storing them in the Hopfield network
pattern_list_flat = [pattern.flatten() for pattern in pattern_list]
hopfield_net.store_patterns(pattern_list_flat)

# Create a noisy initial state by flipping 4 bits in the checkerboard
noisy_init_state = pattern_tools.flip_n(checkerboard, nr_of_flips=4)

# Set the initial state in the Hopfield network
hopfield_net.set_state_from_pattern(noisy_init_state.flatten())  # Flatten initial state if
necessary

# Run the network with monitoring for 4 steps
states = hopfield_net.run_with_monitoring(nr_steps=4)

# Reshape the states to match the pattern size
states_as_patterns = factory.reshape_patterns(states)

# Plot the state sequence and overlap with the reference pattern
plot_tools.plot_state_sequence_and_overlap(states_as_patterns, pattern_list,
reference_idx=0, suptitle="Network dynamics")
```

OUTPUT:


Installing collected packages: brian2, neurodynex
Successfully installed brian2-2.5.1 neurodynex-0.3.4
P0
P1
P2
P3


pattern overlap m(i,k)
pattern i
pattern k
0
1
2
3
1.00
0.75
0.50
0.25
0.00
−0.25


Network dynamics
S0
S1
S2
S3
S4
m = 0.68
m = 1.0
m = 1.0
m = 1.0
m = 1.0
1.0
0.5
0.0

Practical No: 8

A] Membership and Identity Operators in, not in,

CODE :

```
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 = [5, 6, 7, 8, 9]

if (overlapping(list1, list2)):
    print("overlapping")
else:
    print("not overlapping")
```

OUTPUT :

```
In [70]: runfile('C:/Users/DELL/Desktop/practicals/sct/Practical 8A.py', wdir='C:/Users/DELL/
Desktop/practicals/sct')
overlapping
```

B] Membership and Identity Operators is True or False

CODE :

```
x=5
if (type(x) is int):
    print("true")
else:
    print("false")

x=5.2
if (type(x) is not int):
    print("true")
else:
    print("false")
```

OUTPUT :

```
In [73]: runfile('C:/Users/DELL/Desktop/practicals/sct/Practical 8B.py', wdir='C:/Users/DELL/
Desktop/practicals/sct')
true
true
```

Practical No: 9

A] Find the ratios using fuzzy logic

CODE :

```
from fuzzywuzzy import fuzz
from fuzzywuzzy import process
s1 = "I love fuzzywuzzys"
s2 = "I am loveing fuzzywuzzys"
print ("Fuzzywuzzy Ratio:", fuzz.ratio(s1,s2))
print ("FuzzywuzzyParialRatio:" ,fuzz.partial_ratio(s1,s2))
print ("FuzzywuzzyTokenSortRatio:" ,fuzz.token_sort_ratio(s1,s2))
print ("FuzzywuzzyTokenSortRatio:", fuzz.token_sort_ratio(s1,s2))
print ("FuzzywuzzyWRatio:" ,fuzz.WRatio(s1,s2))
query ='fuzzy for fuzzys'
choices=['fuzzy for fuzzy' , 'fuzzy fuzzy','g. for fuzzys']
print("list of ratio:")
print(process.extract(query,choices),'\n')
print("best among the above list:",process.extractOne(query,choices))
```

OUTPUT :

```
In [75]: runfile('C:/Users/DELL/Desktop/practicals/sct/Practical 9A.py', wdir='C:/Users/DELL/
Desktop/practicals/sct')
Fuzzywuzzy Ratio: 86
FuzzywuzzyParialRatio: 83
FuzzywuzzyTokenSortRatio: 86
FuzzywuzzyTokenSortRatio: 86
FuzzywuzzyWRatio: 86
list of ratio:
[('fuzzy for fuzzy', 97), ('fuzzy fuzzy', 95), ('g. for fuzzys', 86)]

best among the above list: ('fuzzy for fuzzy', 97)
```

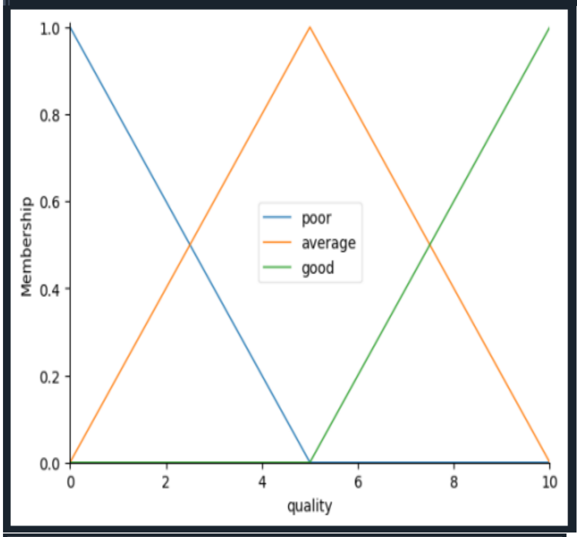
B] Solve Tipping problem using fuzzy logic

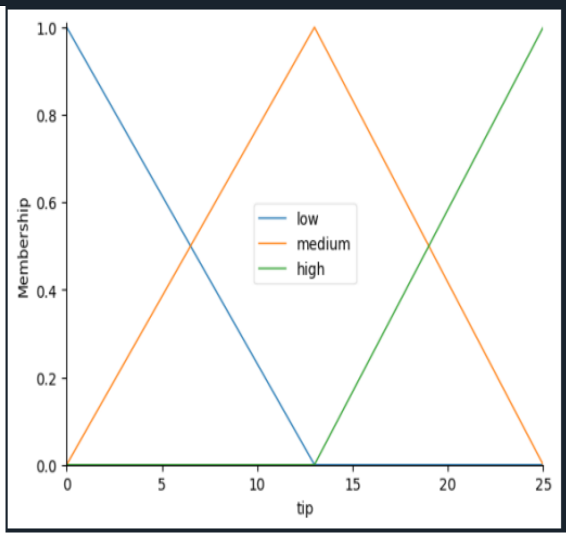
CODE :

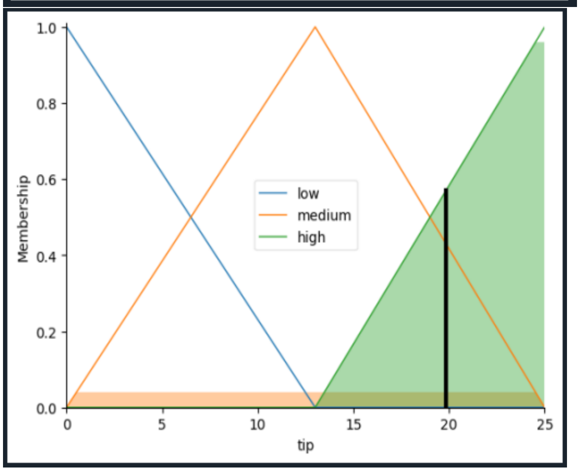
```
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import numpy as np
import matplotlib.pyplot as plt

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')
quality.automf(3)
service.automf(3)

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])

quality.view()
service.view()
tip.view()

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'])

tipping_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
tipping = ctrl.ControlSystemSimulation(tipping_ctrl)
tipping.input['quality'] = 6.5
tipping.input['service'] = 9.8
tipping.compute()
print(tipping.output['tip'])
tip.view(sim=tipping)
plt.show()

# if shows skfuzzy module error run following command "pip install -U scikit-fuzzy"
```

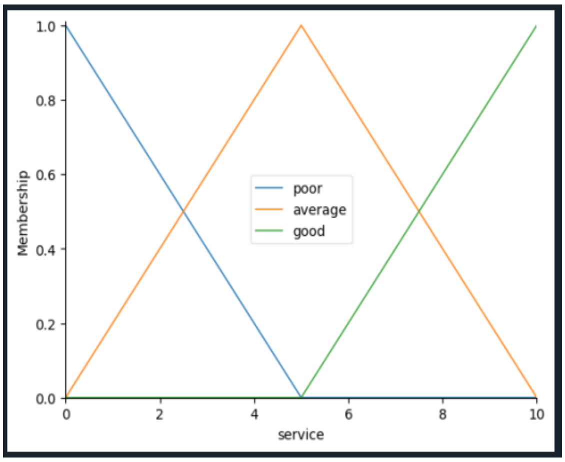
OUTPUT :

```
In [2]: runfile('C:/Users/DELL/Desktop/practicals/sct/Practical 9B.py', wdir='C:/Users/DELL/
Desktop/practicals/sct')
19.847607361963192
```









Practical No: 10

A] Implementation of Simple genetic algorithm

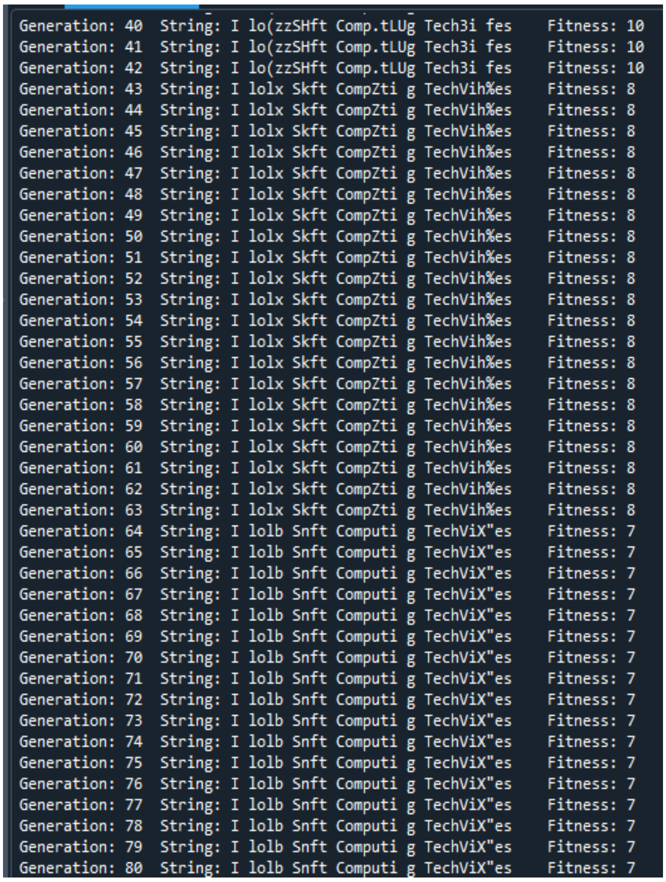
CODE :

```
import random

POPULATION_SIZE = 100
GENES = '''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890,
.-;:_!"#%&/()=?@${[]}'''
TARGET = "I love Soft Computing Techniques"

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

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

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

    def mate(self, par2):
        '''
        Perform mating and produce new offspring
        '''
        child_chromosome = []
        for gp1, gp2 in zip(self.chromosome, par2.chromosome):
            prob = random.random()
            if prob < 0.45:
                child_chromosome.append(gp1)
            elif prob < 0.90:
                child_chromosome.append(gp2)
            else:
                child_chromosome.append(self.mutated_genes())
        return Individual(child_chromosome)
```

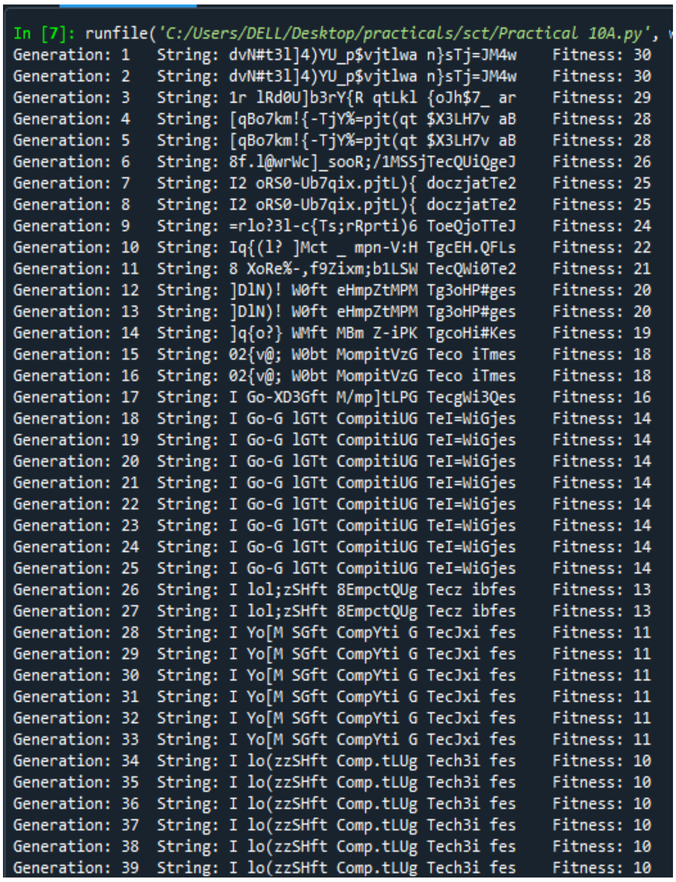
```
    def cal_fitness(self):
        '''
        Calculate fitness score, it is the number of characters in string which differ from
target string.
        '''
        global TARGET
        fitness = 0
        for gs, gt in zip(self.chromosome, TARGET):
            if gs != gt:
                fitness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

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 is 0, then we have reached the
target
        if population[0].fitness <= 0:
            found = True
            break

        # Otherwise generate new generation
        new_generation = []

        # Perform Elitism, that means 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 :



```
Generation: 40  String: I lo(zzSHft Comp.tLUg Tech3i fes    Fitness: 10
Generation: 41  String: I lo(zzSHft Comp.tLUg Tech3i fes    Fitness: 10
Generation: 42  String: I lo(zzSHft Comp.tLUg Tech3i fes    Fitness: 10
Generation: 43  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 44  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 45  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 46  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 47  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 48  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 49  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 50  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 51  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 52  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 53  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 54  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 55  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 56  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 57  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 58  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 59  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 60  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 61  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 62  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 63  String: I lolx Skft CompZti g TechVih%es    Fitness: 8
Generation: 64  String: I lolb Snft Computi g TechViX"es    Fitness: 7
Generation: 65  String: I lolb Snft Computi g TechViX"es    Fitness: 7
Generation: 66  String: I lolb Snft Computi g TechViX"es    Fitness: 7
Generation: 67  String: I lolb Snft Computi g TechViX"es    Fitness: 7
Generation: 68  String: I lolb Snft Computi g TechViX"es    Fitness: 7
Generation: 69  String: I lolb Snft Computi g TechViX"es    Fitness: 7
Generation: 70  String: I lolb Snft Computi g TechViX"es    Fitness: 7
Generation: 71  String: I lolb Snft Computi g TechViX"es    Fitness: 7
Generation: 72  String: I lolb Snft Computi g TechViX"es    Fitness: 7
Generation: 73  String: I lolb Snft Computi g TechViX"es    Fitness: 7
Generation: 74  String: I lolb Snft Computi g TechViX"es    Fitness: 7
Generation: 75  String: I lolb Snft Computi g TechViX"es    Fitness: 7
Generation: 76  String: I lolb Snft Computi g TechViX"es    Fitness: 7
Generation: 77  String: I lolb Snft Computi g TechViX"es    Fitness: 7
Generation: 78  String: I lolb Snft Computi g TechViX"es    Fitness: 7
Generation: 79  String: I lolb Snft Computi g TechViX"es    Fitness: 7
Generation: 80  String: I lolb Snft Computi g TechViX"es    Fitness: 7
```



```
In [7]: runfile('C:/Users/DELL/Desktop/practicals/sct/Practical 10A.py',
Generation: 1   String: dvN#t3l]4)YU_p$vjtlwa n}sTj=JM4w    Fitness: 30
Generation: 2   String: dvN#t3l]4)YU_p$vjtlwa n}sTj=JM4w    Fitness: 30
Generation: 3   String: 1r lRd0U]b3rY{R qtLkl {oJh$7_ ar    Fitness: 29
Generation: 4   String: [qBo7km!{-TjY%=pjt(qt $X3LH7v aB    Fitness: 28
Generation: 5   String: [qBo7km!{-TjY%=pjt(qt $X3LH7v aB    Fitness: 28
Generation: 6   String: 8f.l@wrWc]_sooR;/1MSSjTecQUiQgeJ    Fitness: 26
Generation: 7   String: I2 oRS0-Ub7qix.pjtL){ doczjatTe2    Fitness: 25
Generation: 8   String: I2 oRS0-Ub7qix.pjtL){ doczjatTe2    Fitness: 25
Generation: 9   String: =rlo?3l-c{Ts;rRprti)6 ToeQjoTTeJ    Fitness: 24
Generation: 10  String: Iq{(l? ]Mct _ mpn-V:H TgcEH.QFLs    Fitness: 22
Generation: 11  String: 8 XoRe%-,f9Zixm;b1LSW TecQWi0Te2    Fitness: 21
Generation: 12  String: ]DlN)! W0ft eHmpZtMPM Tg3oHP#ges    Fitness: 20
Generation: 13  String: ]DlN)! W0ft eHmpZtMPM Tg3oHP#ges    Fitness: 20
Generation: 14  String: ]q{o?} WMft MBm Z-iPK TgcoHi#Kes    Fitness: 19
Generation: 15  String: 02{v@; W0bt MompitVzG Teco iTmes    Fitness: 18
Generation: 16  String: 02{v@; W0bt MompitVzG Teco iTmes    Fitness: 18
Generation: 17  String: I Go-XD3Gft M/mp]tLPG TecgWi3Qes    Fitness: 16
Generation: 18  String: I Go-G lGTt CompitiUG TeI=WiGjes    Fitness: 14
Generation: 19  String: I Go-G lGTt CompitiUG TeI=WiGjes    Fitness: 14
Generation: 20  String: I Go-G lGTt CompitiUG TeI=WiGjes    Fitness: 14
Generation: 21  String: I Go-G lGTt CompitiUG TeI=WiGjes    Fitness: 14
Generation: 22  String: I Go-G lGTt CompitiUG TeI=WiGjes    Fitness: 14
Generation: 23  String: I Go-G lGTt CompitiUG TeI=WiGjes    Fitness: 14
Generation: 24  String: I Go-G lGTt CompitiUG TeI=WiGjes    Fitness: 14
Generation: 25  String: I Go-G lGTt CompitiUG TeI=WiGjes    Fitness: 14
Generation: 26  String: I lol;zSHft 8EmpctQUg Tecz ibfes    Fitness: 13
Generation: 27  String: I lol;zSHft 8EmpctQUg Tecz ibfes    Fitness: 13
Generation: 28  String: I Yo[M SGft CompYti G TecJxi fes    Fitness: 11
Generation: 29  String: I Yo[M SGft CompYti G TecJxi fes    Fitness: 11
Generation: 30  String: I Yo[M SGft CompYti G TecJxi fes    Fitness: 11
Generation: 31  String: I Yo[M SGft CompYti G TecJxi fes    Fitness: 11
Generation: 32  String: I Yo[M SGft CompYti G TecJxi fes    Fitness: 11
Generation: 33  String: I Yo[M SGft CompYti G TecJxi fes    Fitness: 11
Generation: 34  String: I lo(zzSHft Comp.tLUg Tech3i fes    Fitness: 10
Generation: 35  String: I lo(zzSHft Comp.tLUg Tech3i fes    Fitness: 10
Generation: 36  String: I lo(zzSHft Comp.tLUg Tech3i fes    Fitness: 10
Generation: 37  String: I lo(zzSHft Comp.tLUg Tech3i fes    Fitness: 10
Generation: 38  String: I lo(zzSHft Comp.tLUg Tech3i fes    Fitness: 10
Generation: 39  String: I lo(zzSHft Comp.tLUg Tech3i fes    Fitness: 10
```

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

CODE :

```
import numpy as np, random, operator, pandas as pd, matplotlib.pyplot as plt
class City:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def distance(self, city):
        xDis = abs(self.x - city.x)
        yDis = abs(self.y - city.y)
        distance = np.sqrt((xDis ** 2) + (yDis ** 2))
        return distance
    def __repr__(self):
        return "(" + str(self.x) + "," + str(self.y) + ")"
class Fitness:
    def __init__(self, route):
        self.route = route
        self.distance = 0
        self.fitness= 0.0
    def routeDistance(self):
        if self.distance ==0:
            pathDistance = 0
            for i in range(0, len(self.route)):
                fromCity = self.route[i]
                toCity = None
                if i + 1 < len(self.route):
                    toCity = self.route[i + 1]
                else:
                    toCity = self.route[0]
                pathDistance += fromCity.distance(toCity)
            self.distance = pathDistance
        return self.distance
    def routeFitness(self):
        if self.fitness == 0:
            self.fitness = 1 / float(self.routeDistance())
        return self.fitness
def createRoute(cityList):
    route = random.sample(cityList, len(cityList))
    return route
def initialPopulation(popSize, cityList):
    population = []
    for i in range(0, popSize):
        population.append(createRoute(cityList))
    return population
def rankRoutes(population):
    fitnessResults = {}
```

```
        for i in range(0,len(population)):
            fitnessResults[i] = Fitness(population[i]).routeFitness()
        return sorted(fitnessResults.items(), key = operator.itemgetter(1), reverse =
True)
    def selection(popRanked, eliteSize):
        selectionResults = []
        df = pd.DataFrame(np.array(popRanked), columns=["Index","Fitness"])
        df['cum_sum'] = df.Fitness.cumsum()
        df['cum_perc'] = 100*df.cum_sum/df.Fitness.sum()
        for i in range(0, eliteSize):
            selectionResults.append(popRanked[i][0])
        for i in range(0, len(popRanked) - eliteSize):
            pick = 100*random.random()
            for i in range(0, len(popRanked)):
                if pick <= df.iat[i,3]:
                    selectionResults.append(popRanked[i][0])
                    break
        return selectionResults
    def matingPool(population, selectionResults):
        matingpool = []
        for i in range(0, len(selectionResults)):
            index = selectionResults[i]
            matingpool.append(population[index])
        return matingpool
    def breed(parent1, parent2):
        child = []
        childP1 = []
        childP2 = []
        geneA = int(random.random() * len(parent1))
        geneB = int(random.random() * len(parent1))
        startGene = min(geneA, geneB)
        endGene = max(geneA, geneB)
        for i in range(startGene, endGene):
            childP1.append(parent1[i])
        childP2 = [item for item in parent2 if item not in childP1]
        child = childP1 + childP2
        return child
    def breedPopulation(matingpool, eliteSize):
        children = []
        length = len(matingpool) - eliteSize
        pool = random.sample(matingpool, len(matingpool))
        for i in range(0,eliteSize):
            children.append(matingpool[i])
        for i in range(0, length):
            child = breed(pool[i], pool[len(matingpool)-i-1])
            children.append(child)
        return children
```

```
def mutate(individual, mutationRate):
    for swapped in range(len(individual)):
        if(random.random() < mutationRate):
            swapWith = int(random.random() * len(individual))
            city1 = individual[swapped]
            city2 = individual[swapWith]
            individual[swapped] = city2
            individual[swapWith] = city1
    return individual
def mutatePopulation(population, mutationRate):
    mutatedPop = []
    for ind in range(0, len(population)):
        mutatedInd = mutate(population[ind], mutationRate)
        mutatedPop.append(mutatedInd)
    return mutatedPop
def nextGeneration(currentGen, eliteSize, mutationRate):
    popRanked = rankRoutes(currentGen)
    selectionResults = selection(popRanked, eliteSize)
    matingpool = matingPool(currentGen, selectionResults)
    children = breedPopulation(matingpool, eliteSize)
    nextGeneration = mutatePopulation(children, mutationRate)
    return nextGeneration
def geneticAlgorithm(population, popSize, eliteSize, mutationRate, generations):
    pop = initialPopulation(popSize, population)
    print("Initial distance: " + str(1 / rankRoutes(pop)[0][1]))
    for i in range(0, generations):
        pop = nextGeneration(pop, eliteSize, mutationRate)
    print("Final distance: " + str(1 / rankRoutes(pop)[0][1]))
    bestRouteIndex = rankRoutes(pop)[0][0]
    bestRoute = pop[bestRouteIndex]
    return bestRoute
def geneticAlgorithmPlot(population, popSize, eliteSize, mutationRate,
generations):
    pop = initialPopulation(popSize, population)
    progress = []
    progress.append(1 / rankRoutes(pop)[0][1])
    for i in range(0, generations):
        pop = nextGeneration(pop, eliteSize, mutationRate)
        progress.append(1 / rankRoutes(pop)[0][1])
    plt.plot(progress)
    plt.ylabel('Distance')
    plt.xlabel('Generation')
    plt.show()
def main():
    cityList = []
    for i in range(0,25):
        cityList.append(City(x=int(random.random() * 200), y=int(random.random() *
```

```
200)))
        geneticAlgorithmPlot(population=cityList, popSize=100, eliteSize=20,
mutationRate=0.01, generations=500)
    main()
    print('check the plotted graph')
```

OUTPUT :