SIES (NERUL) COLLEGE OF ARTS, SCIENCE AND COMMERCE

NAAC ACCREDITED 'A' GRADE COLLEGE

(ISO 9001:2015 CERTIFIED INSTITUTION)

NERUL, NAVI MUMBAI - 400706

Certificate

Seat No: 3713538

Certified that VARMA VISHAL VIJAY

Of Class MSC.IT PART-1 has duly completed the practical

course in the subject of SOFT COMPUTING TECHNIQUES

during the academic year 2021-22 as per the syllabus

prescribed by the University of Mumbai.

Subject Teacher **External Examiner**

Head of Department **Principal**

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 C]Calculate the net input for the network with	27/09/2021	
2.	A] Generate AND/NOT function using McCulloch-Pitts neural net. B] Generate XOR function using McCulloch-Pitts neural net.	01/10/2021	
3.	A] Write a program to implement Hebb's rule. B] Implement the Delta Rules	06/10/2021	
4.	A] Write a program for Back Propagation Algorithm. B] Write a program for error Back Propagation algorithm	26/10/2021	
5.	A] Write a program for Hopfield Network. B] Write a program for Radial Basis function	29/10/2021	
6.	A] Implementation of Kohonen Self Organising Map B] Implementation Adaptive Resonance Theory	15/11/2021	
7.	A] Write a program for Linear separation. B] Write a program for Hopfield network model for associative memory.	22/11/2021	
8.	A] Membership and Identity Operators | in, not in B] Membership and Identity Operators is True or False	29/11/2021	
9.	A] Find ratios using fuzzy logic B] Solve Tipping problem using fuzzy logic	03/12/2021	
10.	A] Implementation of Simple genetic algorithm. B] Create two classes: City and Fitness using Genetic algorithm	07/12/2021	

Practical No: 1

Aim: Implement the following:

A] Design a simple linear neural network model.

Code:

```
x = int(input("Enter the value of x: "))
b = int(input("Enter the value of bias: "))
w = int(input("Enter the value of weight: "))

ynet = (w*x) + b
print("Net input for y neuron = ",ynet)
print("Apply Activation function over net input, Ramp function")

if ynet<0:
    y = 0
elif ynet >= 0 and ynet <= 1:
    y = ynet
else:
    y = 1
print("Y = ",y)
```

OUTPUT:

```
Enter the value of x: -2
Enter the value of bias: 1
Enter the value of weight: 1
Net input for y neuron =  -1
Apply Activation function over net input, Ramp function
Y =  0
```

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 yhe net inputn 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:

```
= RESTART: C:\Users\Vishal Kunal\OneDrive\Desktop\MSCIT\MSC PRATICAL\SCT PRATICAL\PRATICAL 1B\PRATICAL 1B.py
Enter the no. of input layer neurons3
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
```

C]Calculate the net input for the network with Bias

CODE:

```
import math
print("Enter the bias for the network")
bias=float(input())
print("Enter the threshold for neuron")
theta=float(input())
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 yhe net inputn the output nueron")
Yinn=[]
for i in range (0,inputs):
    Yinn.append(inputsn[i]*weight[i])
Yin=round(sum(Yinn),3)+bias
print(Yin)
if Yin>=theta:
    print("The neural network fires and the ouput is 1")
else:
    print("The neural networkdoes not fires and the ouput is 0")
```

OUTPUT:

```
= RESTART: C:\Users\Vishal Kunal\OneDrive\Desktop\MSCIT\MSC PRATICAL\SCT PRATICAL\PRATICAL 1C\PRATICAL 1C.py
Enter the bias for the network
3
Enter the threshold for neuron
2
Enter the no. of input layer neurons3
Enter the input neurons values
1
1
2
[1.0, 1.0, 2.0]
Enter the weight for input layer neurons
2
1
2
[2.0, 1.0, 2.0]
Calculating yhe net inputn the output nueron
10.0
The neural network fires and the ouput is 1
```

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=2
print("For the ",num_ip," input calcilate the net input using net input formula")
x1=[]
x2=[]
for j in range (0,num_ip):
      element1=int(input("X1="))
      element2=int(input("X2="))
      x1.append(element1)
      x2.append(element2)
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 and the other as inhibitory Yin=",Yin)
Y=[]
for i in range(0,num_ip):
      if Yin[i]>=1:
          element=1
          Y.append(element)
      if Yin[i]<1:
          element=0
          Y.append(element)
print("Y=:",Y)
```

OUTPUT:

```
Python 3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:\Users\Vishal Kunal\OneDrive\Desktop\MSCIT\MSC PRATICAL\SCT PRATICAL\PRATICAL 2A\PRATICAL 2A.py
Enter the number of inputs4
For the  4  input calcilate 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= [0, 1, 1, 2]
After assuming one weight as excitatory and the other as inhibitory Yin= [0, -1, 1, 0]
Y=: [0, 0, 1, 0]
```

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

```
import numpy as np
print('enter weights')
w11=int(input('Weight w11='))
w12=int(input('Weight w12='))
w21=int(input('Weight w21='))
w22=int(input('Weight w22='))
v1=int(input('Weight v1='))
v2=int(input('Weight v2='))

print('enter thresold value')
theta=int(input('theta'))
x1=np.array([0,0,1,1])
print(x1)
x2=np.array([0,1,0,1])
print(x2)
z=np.array([0,1,1,0])
print(z)
con=1
y1=np.zeros((4,))
print(y1)
y2=np.zeros((4,))
print(y2)
y=np.zeros((4,))
print(y)

while con==1:
   zin1=np.zeros((4,))
   zin2=np.zeros((4,))
   zin1=x1*w11+x2*w21
```

```
    zin2=x1*w12+x2*w22
    for i in range(0,4):

        if zin1[i]>=theta:
            y1[i]=1
        else :
            y1[i]=0
        if zin2[i]>=theta:
            y2[i]=1
        else:
            y2[i]=0

        yin=np.array([])
        yin=y1*v1+y2*v2
        for i in range (0,4):
            if yin[i]>=theta:
                y[i]=1
            else:
                y[i]=0
                print("yin",yin)
                print('output of net')
                y=y.astype(int)
                print("y",y)
                print("z",z)
                if np.array_equal(y,z):
                    con=0
                else:
                    print("net is not learing enter set of weight and threshold value")
                    w11=input("Weight w11=")
                    w12=input("Weight w12=")
                    w21=input("Weight w21=")
                    w22=input("Weight w22=")
                    v1=input("Weight v1=")
                    v2=input("Weight v2=")
                    theta=input("theta=")
print("McCulloch-Pits Net for XOR function")
print("Weight of Neuron Z1")
print(w11)
print(w21)
print("Weight of Neuron Z2")
print(w12)
print(w22)
print("Weight of Neuron Y")
print(v1)
print(v2)
print("Threshold Value")
print(theta)
```

OUTPUT:

```
= RESTART: C:\Users\Vishal Kunal\OneDrive\Desktop\MSCIT\MSC PRATICAL\SCT PRATICAL\PRATICAL 2B\2B.py
enter weights
Weight w11=1
Weight w12=-1
Weight w21=-1
Weight w22=-1
Weight v1=1
Weight v2=-1
enter thresold value
theta1
[0 0 1 1]
[0 1 0 1]
[0 1 1 0]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
yin [0. 0. 0. 0.]
output of net
y [0 0 0 0]
z [0 1 1 0]
net is not learing enter set of weight and threshold value
```

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))
print("first input with target=1")
for i in range (0,9):
   wtold[i]=wtold[i]*x1[i]*y[0]
wtnew=wtold
b=b+y[0]
print("new wt",wtnew)
print("bias value",b)
print("second input with target =-1")
for i in range (0,9):
     wtold[i]=wtold[i]*x2[i]*y[1]
wtnew=wtold
b=b+y[1]
print("new wt",wtnew)
print("bias value",b)
```

OUTPUT:

```
= RESTART: C:\Users\Vishal Kunal\OneDrive\Desktop\MSCIT\MSC PRATICAL\SCT PRATICAL\P3\P3A.py
first input with target=1
new wt [0 0 0 0 0 0 0 0 0]
bias value 1
second input with target =-1
new wt [0 0 0 0 0 0 0 0 0]
bias value 0
```

B] Implement the Delta Rules.

CODE:

```
import numpy as np
import time
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*weights
print("*"*30)
print("Final output using delta rule")
print("Corrected weights",weights)
print("actual",actual)
print("desired",desired)
```

OUTPUT:

```
>>>
= RESTART: C:/Users/Vishal Kunal/OneDrive/Desktop/MSCIT/MSC PRATICAL/SCT PRATICAL/3B/3B.py
Intial inputs:1
Intial inputs:1
Intial inputs:1
Intial weights:1
Intial weights:1
Intial weights:1
Intial desired:1
Intial desired:1
Intial desired:1
Enter learning rate:1
Actual initial [1. 1. 1.]
Actual desired [1. 1. 1.]
******************************
Final output using delta rule
Corrected weights [1. 1. 1.]
actual [1. 1. 1.]
desired [1. 1. 1.]
>>>
```

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 input layer")
yin=w[0]+z1*w[1]+z2*w[2]
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 [1]: runfile('C:/Users/Vishal Kunal/OneDrive/Desktop/MSCIT/MSC PRATICAL/SCT PRATICAL/P4/4a.py', wdir='C:/Users/Vishal
Kunal/OneDrive/Desktop/MSCIT/MSC PRATICAL/SCT PRATICAL/P4')
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 input layer
calculate net output
y= 0.5227368084248941
dk dk
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:

```
= RESTART: C:\Users\Vishal Kunal\OneDrive\Desktop\MSCIT\MSC PRATICAL\SCT PRATICAL\P4\practical_4b.py
Enter weight first network:1
Enter base first network:2
Enter weight second network:1
Enter base second network:3
Enter learning coefficient:1
The updated weight of first n/w w11= 1.0000534270164934
The uploaded weight of second n/w w21= 0.9946115339071285
The updated base of first n/w b10= 2.0
The uploaded base of second n/w b20= 3.0
```

Practical No. 5

A] Write a program for Hopfield Network.

CODE:

P5.H SAVE IN INCLUDE

```
#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 threshold(int);
void activation(int j[4]);
network(int*,int*,int*,int*);
};
```

HOP.CPP SAVE IN BIN

```
#include "P5.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::threshold(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]=threshold(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
LAYEROF";
cout<<"\n4 FULLY INTERCONNECTED NEURONS. THE NETWORK SHOULD
RECALLTHE";
network h1(wt2,wt2,wt3,wt4);
h1.activation(patrn1);
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 occured";
}
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 occured";
}}
```

OUTPUT:

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


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

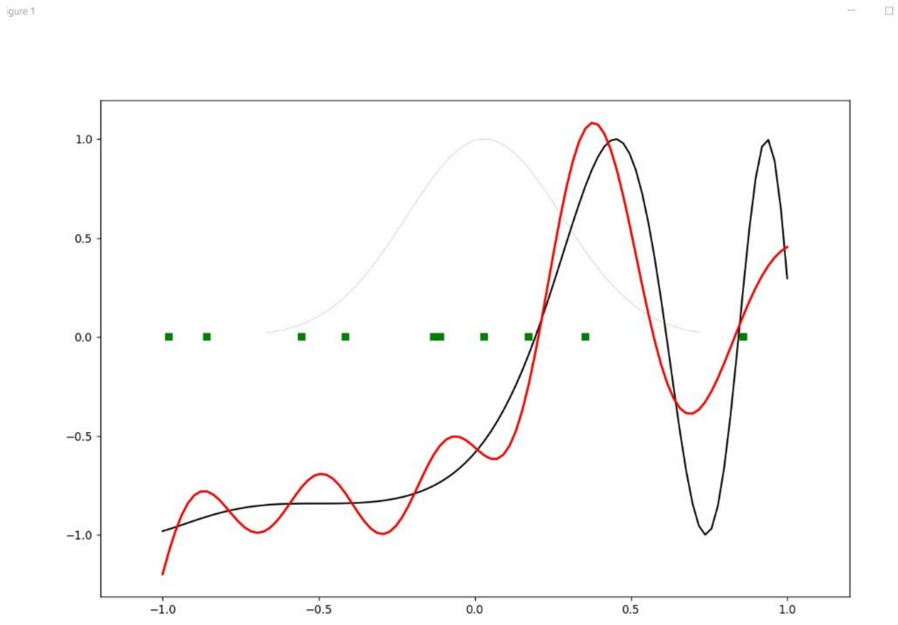
B] Write a program for Radial Basis function

CODE:

```
from scipy import *
from scipy.linalg import norm, pinv
from matplotlib import pyplot as plt
class RBF:
    def __init__(self, indim, numCenters, outdim):
        self.indim=indim
        self.outdim=outdim
        self.numCenters=numCenters
        self.centers=[random.uniform(-1,1,indim)
         for i in range (numCenters)]
        self.beta=8
        self.W =random.random((self.numCenters, self.outdim))
```

```
    def _basisfunc(self,c,d):
        assert len (d)==self.indim
        return exp(-self.beta *norm(c-d)**2)

    def _calcAct(self,X):
        G=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):
        """X: matrix of dimension n x indim
        y: column vetcor of dimension n x 1"""
        rnd_idx=random.permutation (X.shape[0])[:self.numCenters]
        self.centers =[X[i,:] for i in rnd_idx]
        print("center",self.centers)

        G=self._calcAct(X)
        print(G)
        self.W=dot(pinv(G),Y)

    def test(self,X):
        """ X: matrix of dimension n x indim """
        G=self._calcAct(X)
        Y=dot(G,self.W)
        return Y

if __name__=='__main__':
    n=100
    x=mgrid[-1:1:complex(0,n)].reshape(n,1)
    y=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-')
    plt.plot(x,z, 'r-',linewidth=2)
    plt.plot(rbf.centers, zeros(rbf.numCenters),'gs')
    for c in rbf.centers:
        cx=arange(c-0.7,c+0.7,0.01)
        cy=[rbf._basisfunc(array([cx_]),array([c])) for cx_ in cx]
        plt.plot(cx,cy,'-',color='gray' , linewidth=0.2)
        plt.xlim(-1.2,1.2)
        plt.show()
```

OUTPUT:



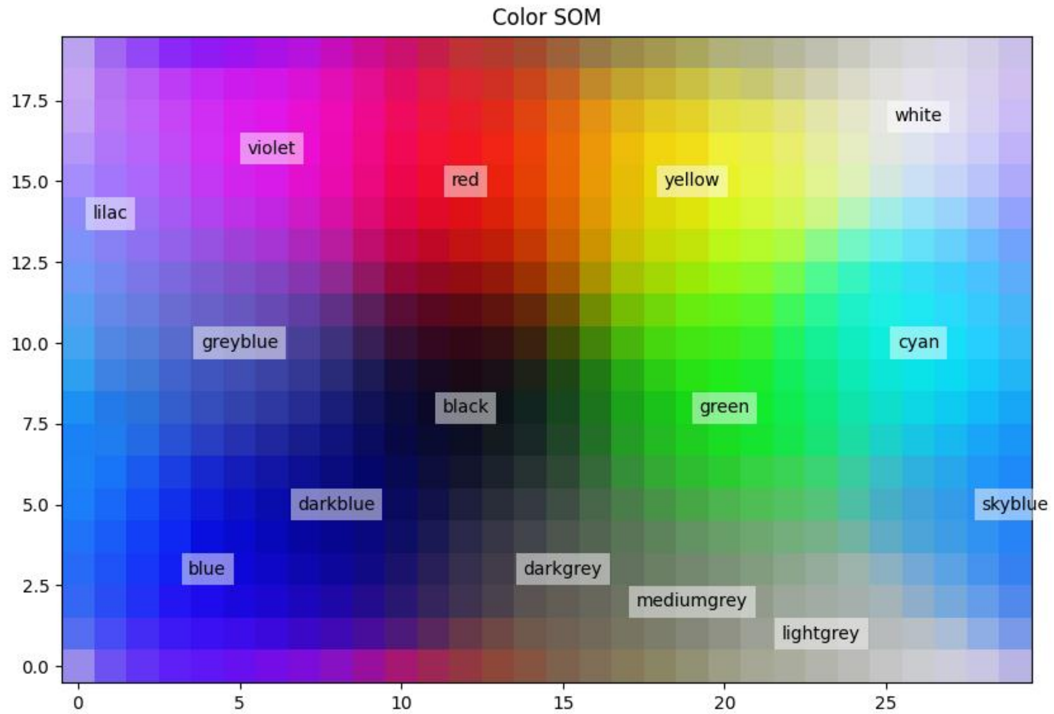
Practical 6

A] Implementation of Kohonen Self Organising Map

CODE:

```
from mvpa2 import *
from mvpa2.mappers.som import SimpleSOMMapper
#from suite import *
import matplotlib.pyplot as pl

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.],
                [.33,.33,.33],
                 [.5,.5,.5],
                [.66,.66,.66]])
color_names=\
['black','blue','darkblue','skyblue',
 'greyblue','lilac','green','red','cyan',
 'violet','yellow','white',
 'darkgrey','mediumgrey','lightgrey']
som=SimpleSOMMapper((2.,30),400,learning_rate=0.05)
som.train(colors)
pl.imshow(som.K,origin='lower')
mapped=som(colors)
pl.title('Colors SOM')
for i ,m in enumerate(mapped):
    pl.text(m[1],m[0],color_names[i],ha='center',va='center',
    bbox=dict(facecolor='white',alpha=0.5,lw=0))
    pl.plot()
    pl.show()
```

OUTPUT:



B] Adaptive Resonance Therory

CODE:

```
from __future__ import division
import numpy as np
from neupy.utils import format_data
from neupy.core.properties import (ProperFractionProperty, IntProperty)
from neupy.algorithms.base import BaseNetwork
import theano
__all__ = ('ART1',)

class ART1(BaseNetwork):
  rho = ProperFractionProperty(default=0.5)
  n_clusters = IntProperty(default=2, minval=2)

  def train(self, X):
    X = format_data(X)
    if X.ndim != 2:
      raise ValueError("Input value must be 2 dimensional, got ""{}".format(X.ndim))
    n_samples, n_features = X.shape
    n_clusters = self.n_clusters
    step = self.step
    rho = self.rho
```

```
        if np.any((X != 0) & (X != 1)):
            raise ValueError("ART1 Network works only with binary matrices")
        if not hasattr(self, 'weight_21'):
            self.weight_21 = np.ones((n_features, n_clusters))
        if not hasattr(self, 'weight_12'):
            scaler = step / (step + n_clusters-1)
            self.weight_12 = scaler * self.weight_21.T
            weight_21 = self.weight_21
            weight_12 = self.weight_12
        if n_features != weight_21.shape[0]:
            raise ValueError("Input data has invalid number of features. ""Got {} instead of
{}""".format(n_features, weight_21.shape[0]))
        classes = np.zeros(n_samples)

# Train network

        for i, p in enumerate(X):
            disabled_neurons = []
            reseted_values = []
            reset = True
            while reset:
                output1 = p
                input2 = np.dot(weight_12, output1.T)
                output2 = np.zeros(input2.size)
                input2[disabled_neurons] = -np.inf
                winner_index = input2.argmax()
                output2[winner_index] = 1
                expectation = np.dot(weight_21, output2)
                output1 = np.logical_and(p, expectation).astype(int)
                reset_value = np.dot(output1.T, output1) / np.dot(p.T, p)
                reset = reset_value < rho
                if reset:
                    disabled_neurons.append(winner_index)
                    reseted_values.append((reset_value, winner_index))
                if len(disabled_neurons) >= n_clusters:
                    reset = False
                    winner_index = None
                if not reset:
                    if winner_index is not None:
                        weight_12[winner_index, :] = (
                        step * output1) / (step + np.dot(output1.T, output1) - 1)
                        weight_21[:, winner_index] = output1
                    else:
                        winner_index = max(reseted_values)[1]
                        classes[i] = winner_index
            return classes
    def predict(self, X):
        return self.train(X)
```

Practical 7

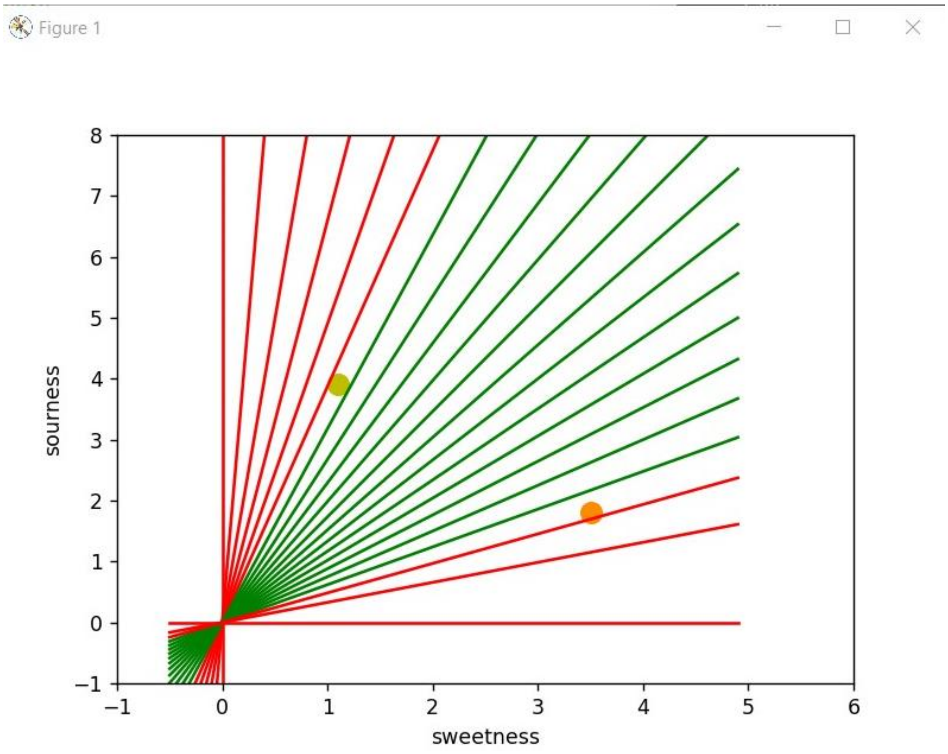
A] Write a program for Linear separation.

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)
                    #print("x: ", x, "slope: ", slope)
                Y = slope * X
                results = []
                for point in points:
                    results.append(dist4line1(*point))
                if (results[0][1] != results[1][1]):
                    ax.plot(X, Y, "g-")
```

```
        else:
                ax.plot(X, Y, "r-")
plt.show()
```

OUTPUT:



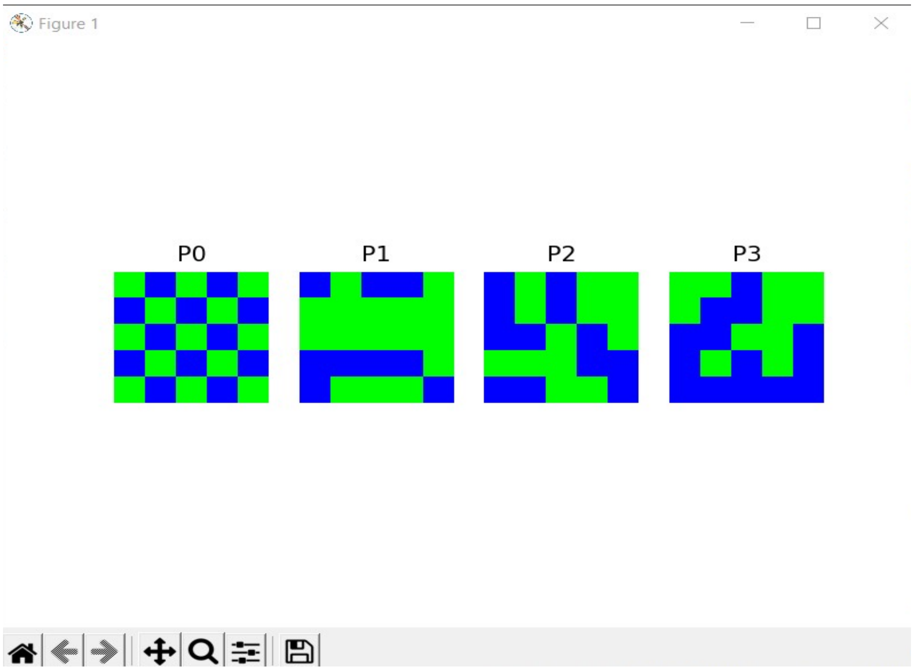
B] Write a program for Hopfield network model for associative memory.

CODE:

```
from typing import Pattern, overload
from neurodynex import hopfield_network
from neurodynex.hopfield_network import network ,pattern_tools,plot_tools
Pattern_size=5
hopfield_network=network.HopfieldNetwork(nr_neurons=Pattern_size**2)
factory=pattern_tools.PatternFactory(Pattern_size,Pattern_size)
checkerboard=factory.create_checkerboard()
Pattern_list=[checkerboard]
Pattern_list.extend(factory.create_random_pattern_list(nr_patterns=3,on_probability=0.5))
plot_tools.plot_pattern_list(Pattern_list)
overload_matrix=pattern_tools.compute_overlap_matrix(Pattern_list)
plot_tools.plot_overlap_matrix(overlap_matrix)
hopfield_not.store_patterns(Pattern_list)
noisy_init_state-pattern_tools.flip_n(checkerboard,nr_of_flips=4)
```

```
hopfield_net.set_state_from_pattern(noisy_init_state)
states=hopfield_net.run_with_monitoring(nr_steps=4)
states_as_patterns= factory.reshape_patterns(states)
plot_tools.plot_state_sequence_and_overlap(states_as_patterns, Pattern_list,
reference_idx=0, suptitle="network dynamics")
```

OUTPUT:



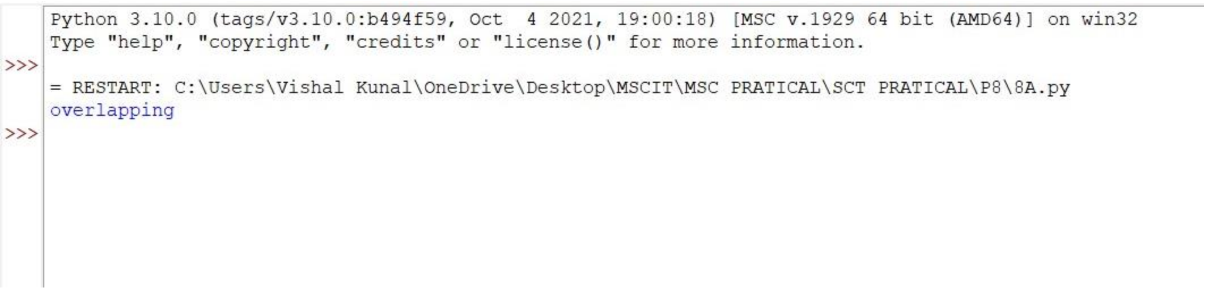
Practical 8

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:



```
Python 3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Vishal Kunal\OneDrive\Desktop\MSCIT\MSC PRATICAL\SCT PRATICAL\P8\8A.py
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:

```
Python 3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Vishal Kunal\OneDrive\Desktop\MSCIT\MSC PRATICAL\SCT PRATICAL\P8\8B.py
true
true
>>>
```

Practical 9

A. Find 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:

```
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)
```

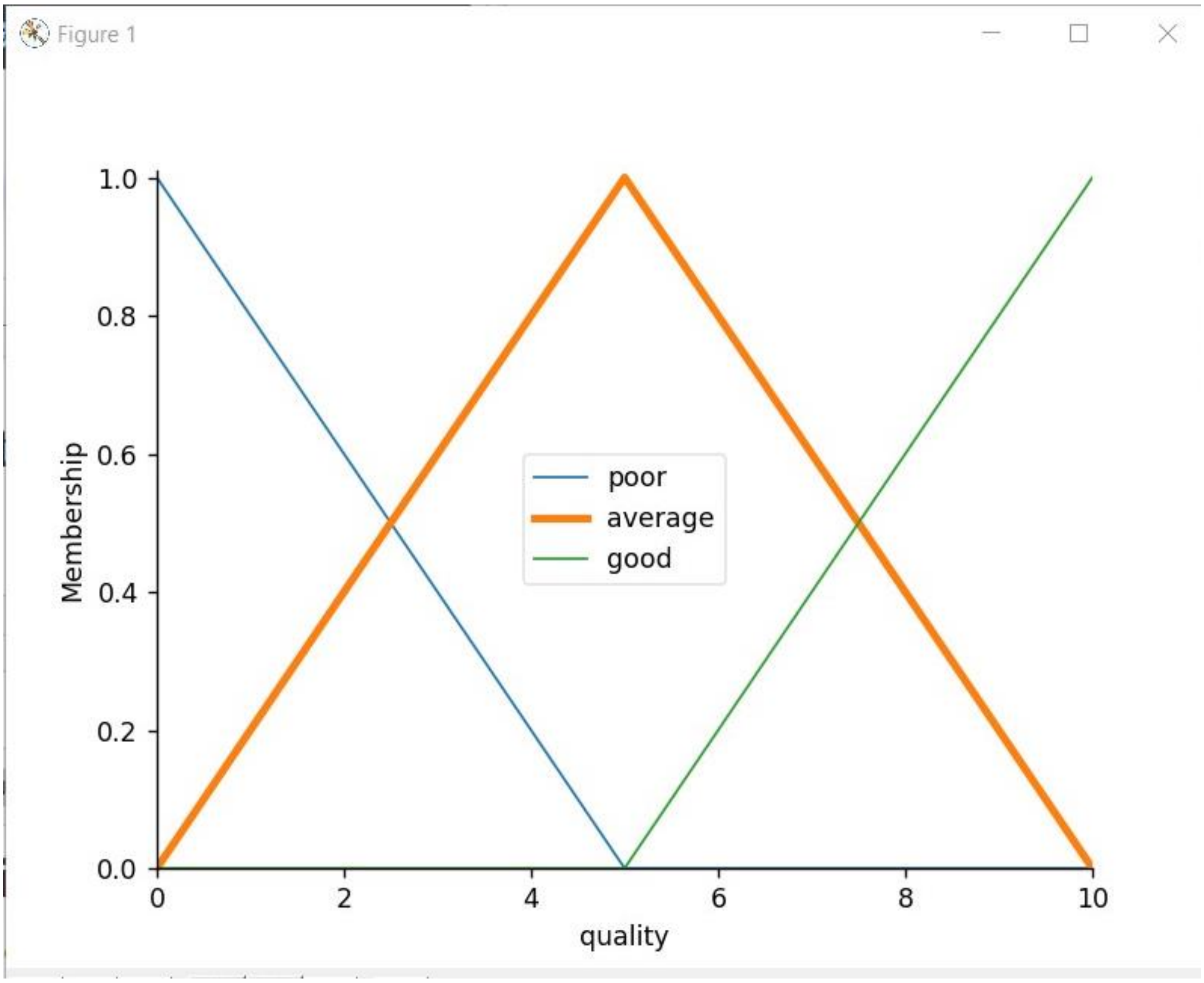
B] Solve Tipping problem using fuzzy logic

CODE:

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
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, [0,0,25])
tip['high']=fuzz.trimf(tip.universe, [13,25,25])
""
To help understand what the membership looks like, use the "view" methods.
""
quality['average'].view()
""
..image::PLOT2RST.current_figure
""
service.view()
""
..image::PLOT2RST.current_figure
""
tip.view

OUTPUT:



Practical 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(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):
    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 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

def main():
    global POPULATION_SIZE
    #current generation
    generation =1
    found =False
    population =[]

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

    while not found:
        population =sorted(population, key =lambda x:x.fitness)
        if population[0].fitness <=0:
            found =True
            break
        new_generation =[]

        s =int((10*POPULATION_SIZE)/100)
        new_generation.extend(population[:s])

        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\Vishal Kunal\OneDrive\Desktop\MSCIT\MSC PRATICAL\SCT PRA
Generation: 1   String: EbA/E_P:]VHI):4p5P}&gq?4-rtiWPLs       Fitness:28
Generation: 1   String: EbA/E_P:]VHI):4p5P}&gq?4-rtiWPLs       Fitness:28
Generation: 1   String: EbA/E_P:]VHI):4p5P}&gq?4-rtiWPLs       Fitness:28
Generation: 1   String: ubl/E_y:]VH&8:4p5P}$N 74{rwiCdLs       Fitness:27
Generation: 1   String: ubl/E_y:]VH&8:4p5P}$N 74{rwiCdLs       Fitness:27
Generation: 1   String: v yKvkyIA@&$73,pV&NdQVT4Mrtif0Ls       Fitness:26
Generation: 1   String: v"AovKGwr0He6gUzJBiGg8GJ%?nifu4s       Fitness:24
Generation: 1   String: ITl-w#s: ]&M025pKP[hN TS{rniCuls       Fitness:23
Generation: 1   String: I6ZoEtyS&IkK8yIpzP}&g =x-WniCuys       Fitness:22
Generation: 1   String: I6ZoEtyS&IkK8yIpzP}&g =x-WniCuys       Fitness:22
Generation: 1   String: k6$ovtlS5I=YC)mr&{iLg:TxMyni.u&s       Fitness:20
Generation: 1   String: I Aov#g: l] l!VpHPY&g TTZWniIuys       Fitness:19
Generation: 1   String: I Aov#g: l] l!VpHPY&g TTZWniIuys       Fitness:19
Generation: 1   String: kbbovW5S&JsKC)mpkPicg Tx;lni.uys       Fitness:18
Generation: 1   String: kbbovW5S&JsKC)mpkPicg Tx;lni.uys       Fitness:18
Generation: 1   String: I Aov#g:9l] CSypu{Y)g TTMGni.uRs       Fitness:17
Generation: 1   String: I Aov#g:9l] CSypu{Y)g TTMGni.uRs       Fitness:17
Generation: 1   String: I Aov#g:9l] CSypu{Y)g TTMGni.uRs       Fitness:17
Generation: 1   String: I OovIMSF7= C!mp[Ui%g TTMNLi.uBs       Fitness:16
Generation: 1   String: I OovIMSF7= C!mp[Ui%g TTMNLi.uBs       Fitness:16
Generation: 1   String: I Aov#b:9f] CSmpuqi)g $T{Gni,uys       Fitness:15
Generation: 1   String: I Aov#b:9f] CSmpuqi)g $T{Gni,uys       Fitness:15
Generation: 1   String: I Aov#b:9f] CSmpuqi)g $T{Gni,uys       Fitness:15
Generation: 1   String: I lov5uS9f] C)mpNqimg TL{TniIuE;       Fitness:14
Generation: 1   String: I lov5uS9f] C)mpNqimg TL{TniIuE;       Fitness:14
Generation: 1   String: I lov5uS9f] C)mpNqimg TL{TniIuE;       Fitness:14
Generation: 1   String: I lov5uS9f] C)mpNqimg TL{TniIuE;       Fitness:14
Generation: 1   String: I lov5uS9f] C)mpNqimg TL{TniIuE;       Fitness:14
Generation: 1   String: I lov5uS9f] C)mpNqimg TL{TniIuE;       Fitness:14
Generation: 1   String: I lov5uS9f] C)mpNqimg TL{TniIuE;       Fitness:14
Generation: 1   String: I AovILSif] CCmpu3imi TA;GniIues       Fitness:13
Generation: 1   String: I AovILSif] CCmpu3imi TA;GniIues       Fitness:13
Generation: 1   String: I lov5uSxf] CCmpyqimg TL;TniIues       Fitness:12
Generation: 1   String: I lov5uSxf] CCmpyqimg TL;TniIues       Fitness:12
Generation: 1   String: I lov5uSxf] CCmpyqimg TL;TniIues       Fitness:12
Generation: 1   String: I lov5uSxf] CCmpyqimg TL;TniIues       Fitness:12
Generation: 1   String: I lov5uSxf] CCmpyqimg TL;TniIues       Fitness:12
Generation: 1   String: I lov5uSxf] CCmpyqimg TL;TniIues       Fitness:12
Generation: 1   String: I lov5uSxf] CCmpyqimg TL;TniIues       Fitness:12
Generation: 1   String: I lov5uSxf] CCmpyqimg TL;TniIues       Fitness:12
Generation: 1   String: I lov5uSxf] CCmpyqimg TL;TniIues       Fitness:12
Generation: 1   String: I lov5uSxf] CCmpyqimg TL;TniIues       Fitness:12
Generation: 1   String: I lovIeS9f] C)mpytimg TW;TniIues       Fitness:11
Generation: 1   String: I lov5LS@f. CWmputi6g TL;TniIues       Fitness:10
Generation: 1   String: I lov5LS@f. CWmputi6g TL;TniIues       Fitness:10
Generation: 1   String: I lov5LS@f. CWmputi6g TL;TniIues       Fitness:10
```

```
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 1   String: I love SWft Computing Techniques       Fitness:1
Generation: 2   String: I love Soft Computing Techniques       Fitness: 0
```

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

CODE:

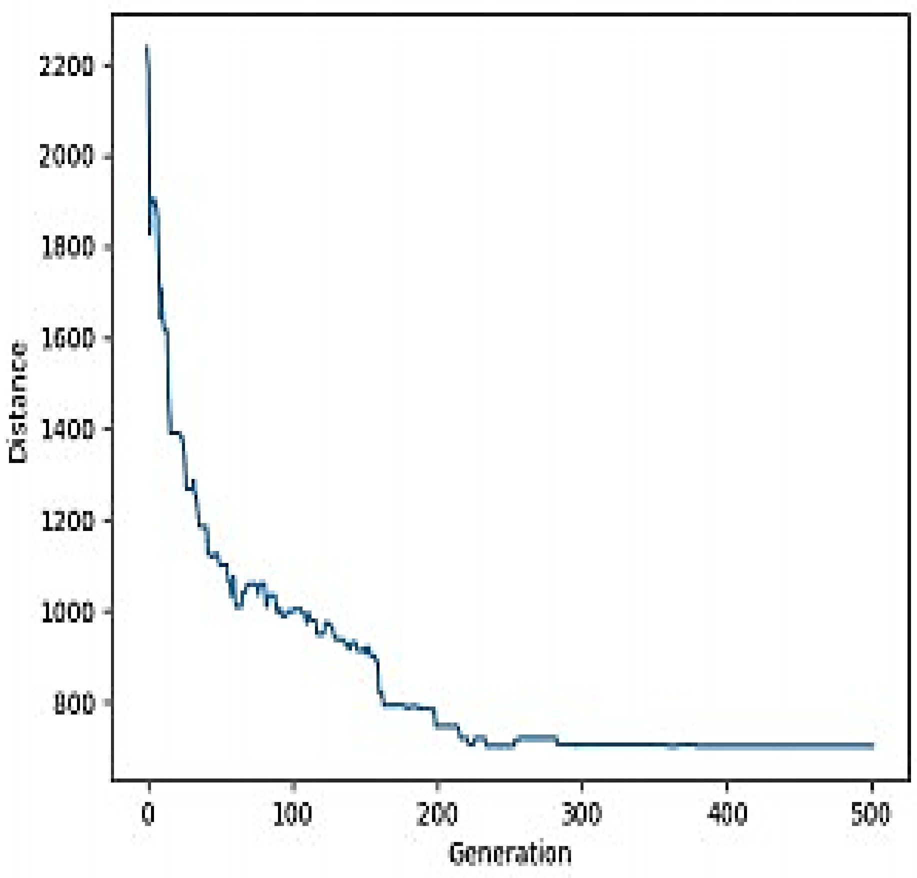
```
import numpy as np
import random
import operator
import pandas as pd
import matplotlib.pyplot as plt
from tkinter import Tk, Canvas, Frame, BOTH, Text
import math


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)

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

OUTPUT:



2200
2000
1800
1600
1400
1200
1000
800
Distance
0
100
200
300
400
500
Generation