```python
import numpy as np
import random
import matplotlib.pylab as pylab
import math

#Declaration of Variables
X=[]
Xnew=[]
Class1=[]
Class2=[]
Cen1=[]
Cen1rest=[]
Cen2=[]
Cen2rest=[]
Centers=[]

desired = [[0 for x in range(1)] for y in range(100)]
g=        [[0 for x in range(1)] for y in range(100)]
act=      [[0 for x in range(1)] for y in range(100)]

desired=    np.matrix(desired)
g=          np.matrix(g)
act=        np.matrix(act)

variance=0
res1=1000
theta=random.uniform(-1,1)
N=100
n=10 #Number of Center in each Class : 10,2
rem=10

def weights(m):
    w=[]
    for i in range(2*m):
        w.append(random.uniform(-1,1))
    return w

def gen_inputs(N):#Generating Inputs
    inputs=[]
    for i in range(N):
        x1=[]
        for j in range(2):
            temp=random.uniform(0,1)
            x1.append(temp)
        inputs.append(x1)
    return inputs

def plot(N,X): #Generating Plot for generated Values

    pylab.title("Plot")
    for i in range(N):
        pylab.plot(X[i][0],X[i][1],'o')
    pylab.show()

def desired_outputs(N,X): #Getting plot of Sun and Moon
    for i in range(N):
        if((X[i][1]) < (((1/5)*(np.sin((10)*(X[i][0])))) + 0.3 )  or ((np.power((X[i][1]-0.8),2) +
            desired[i]=1
            Class1.append(X[i])
            pylab.plot(X[i][0],X[i][1],'gx')
        else:
            desired[i]=-1
            Class2.append(X[i])
            pylab.plot(X[i][0],X[i][1],'rx')

def class_seperation(n,rem):
    sc1=len(Class1)
    print("Length of Class C1 = ",sc1)
    sc2=len(Class2)
    print("Length of Class C2 = ",sc2)

    #Class1 Computation
    for i in range(n):
        t1 = Class1[i]
        Cen1.append(t1)
        pylab.plot(Cen1[i][0],Cen1[i][1],'bo')#Blue CENTERS plotted here

    #Values that are in Class1 but are not Cluster Centers
    lc1=sc1-rem
    print ("length of Points not included in Cluster centers",lc1)
    for j in range(lc1):
        t2 = Class1[j+rem]
        Cen1rest.append(t2)

    #Class-1 Computation
    for i in range(n):
        t3 = Class2[i]
        Cen2.append(t3)
        pylab.plot(Cen2[i][0],Cen2[i][1],'ko')

    #Values that are in Class-1 but are not Cluster Centers
    lc2=sc2-rem
    print ("length of Points not included in Cluster centers",lc2)
    for j in range(lc2):
        t4 = Class2[j+rem]
        Cen2rest.append(t4)

def kmeans(n):#Running kmeans for Cluster Centers update
    len1=len(Cen1rest)
    len2=len(Cen1)
    len3=len(Cen2rest)
    len4=len(Cen2)

    diff1=1
    diff2=1
    flag=1
    count=0
    average1=0
    average2=0

    while(flag!=0):
        cluster1={}
        cluster2={}
        #For Cen1
        for i in range(n):
            cluster1.setdefault(i,[])   #appending a new list!

        for i in range(len1):
            tlist=[]
            for j in range(len2):
                tlist.append(np.linalg.norm(Cen1rest[i]-Cen1[j]))
            cluster1[np.argmin(tlist)].append(Cen1rest[i])

        out1=0.0
        for i in range(n):
            sum=0
            if(len(cluster1[i])!=0):
                for j in cluster1[i]:
                    sum=sum+j
                average1=sum/(len(cluster1[i]))
                diff1=np.linalg.norm(average1-Cen1[i])
                Cen1[i]=average1
                out1=diff1+out1

        #For Cen2
        for i in range(n):
            cluster2.setdefault(i,[])   #appending a new list!

        for i in range(len3):
            tlist=[]
            for j in range(len4):
                tlist.append(np.linalg.norm(Cen2rest[i]-Cen2[j]))
            cluster2[np.argmin(tlist)].append(Cen2rest[i])

        out2=0.0
        for i in range(n):
            sum=0
            if(len(cluster2[i])!=0):
                for j in cluster2[i]:
                    sum=sum+j
                average2=sum/(len(cluster2[i]))
                diff2=np.linalg.norm(average2-Cen2[i])
                Cen2[i]=average2
                out2=diff2+out2

        if out1==0.0 and out2==0.0:
            flag=0

        count+=1
        print("Count",count)

    for i in range(n):
            pylab.plot(Cen1[i,0],Cen1[i,1],'go')#Checking whether centers are in correct region fo
            pylab.plot(Cen2[i,0],Cen2[i,1],'ro')#Checking whether centers are in correct region fo

def combining_points():
    global Centers
    xnew=[]#Converting all matrix to list and then returning the value as list
    temp1=Cen1.tolist()
    temp2=Cen1rest.tolist()
    temp3=Cen2.tolist()
    temp4=Cen2rest.tolist()
    Centers=temp1 + temp3
    xnew=temp1 + temp2 + temp3 + temp4
    return xnew

def afunc(x):#Signum Activation Function
    if (x >= 0):
        return 1
    else:
        return -1

def rbf(rbfvar):
    global variance
    temp1=(-1.0 * math.pow(rbfvar , 2)) / (2 * math.pow(variance , 2))
    ans=math.exp(temp1)
    return ans

def ptaupdate(N,Clist):
    global g
    global eta
    global weights
    global desired
    epoch=0

    #Goal is to find out g(x) first and then do weight update
    eta=1
    while(epoch!=300):
        error=0
        for i in range(N):
            g = 0
            for j in range(Clist):
                t=np.linalg.norm(Xnew[i] - Centers[j])
                g = (Weights[j] * rbf(t)) + g
            g = g + theta
            act[i]=afunc(g)

        for i in range(N):
            difference=desired[i]-act[i]
            for j in range(Clist):
                r=np.linalg.norm(Xnew[i] - Centers[j])
                Weights[j] = Weights[j] + (eta * rbf(r) * difference)
            if (desired[i]!=act[i]):
                error+=1
        epoch+=1
        print("Epoch  = ",epoch,"  Error = ",error)

def plot_final(m):#Function to print G(X)=0
    global res1
    global Centers
    h=0
    for x1 in range(res1):
        for x2 in range(res1):
            g=0
            x1t=x1/res1
            x2t=x2/res1
            arr=np.array([x1t,x2t])
            for j in range(2*m):
                t=arr-Centers[j]
                g= (Weights[j]*rbf(t))+g
            g=g+theta
            if(g < 0.05 and g > -0.05):
                pylab.plot(x1t,x2t,'b.',markersize=3,label='Decision Boundary' if h==0 else "")
                h=1
        print("Resolution",x1)
    pylab.legend(loc='upper right')
    pylab.show()

#Control Flow of Execution of Program
X=gen_inputs(N)
Xin=np.matrix(X)
plot(N,X)
desired_outputs(N,X)

class_seperation(n,rem)
Cen1=np.matrix(Cen1)
Cen1rest=np.matrix(Cen1rest)
Cen2=np.matrix(Cen2)
Cen2rest=np.matrix(Cen2rest)

kmeans(n)
pylab.show()

Xnew=combining_points()
Xnew=np.matrix(Xnew)

variance=np.var(Xnew)
Centers=np.matrix(Centers)
Weights=weights(n)

ptaupdate(N,20)
plot_final(n)
```