Assingment 05

EE-527 Machine Learning Laboratory

Group Members

RAJENDRA KUJUR (214161008)
ROHIT RAJ SINGH CHAUHAN (214161009)

In [1]:

```
# Importing all necessary libraries
import numpy as np
import pandas as pd
from math import e
from sympy import symbols, diff
import sympy as sy
from sympy.functions import *
from sympy.utilities.lambdify import lambdify
from matplotlib import cm
import matplotlib.pyplot as plt
import random
```

Consider the objective function f (x, y) given by

In [2]:

```
# Given function
def f(x, y):
    return 1.7 * e ** (-( ((x-3)**2/10) + ((y-3)**2/10) )) + e ** (-( ((x+5)**2/8)
```

Problem 1

In [3]:

```
# To plot the given function
x = np.linspace(-10, 10, 100)
y = np.linspace(-10, 10, 100)
xx, yy = np.meshgrid(x, y)
z = f(xx, yy)
```

In [4]:

```
# calculates the partial derivative at x0, y0 and returns
def getDerivative(x0, y0):
    x, y = symbols('x y', real = True)
    f = 1.7 * e ** (-( ((x-3)**2/10) + ((y-3)**2/10) )) + e ** (-( ((x+5)**2/8) + (
    dx = diff(f, x)
    dy = diff(f, y)
    dx = lambdify((x, y), dx)
    dy = lambdify((x, y), dy)
    return dx(x0, y0), dy(x0, y0)
```

In [5]:

```
# Visualizes the Gradient ascent
def VisualizeGA(x0, y0, learning_rate, iterations):
    """Visualizes the step by step gradient ascent"""

    # to store the x and y values at each step
    x_traj = np.array([x0])
    y_traj = np.array([y0])

    for _ in range(iterations):
        # change in x and y
        nx, ny = getDerivative(x0, y0)

        # update x0 and y0 for each iteration
        x0 = round(x0 + learning_rate * nx, 5)
        y0 = round(y0 + learning_rate * ny, 5)

        # append the new x0, y0 in the trajectory values
        x_traj = np.append(x_traj, x0)
        y_traj = np.append(y_traj, y0)

    # plot the trajectory
    plt.plot(x_traj, y_traj, color = 'red')
    plt.contour(x, y, z, cmap = cm.twilight_shifted);
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Contour Plot in 2D')
    plt.show()
    return x0, y0
```

First Test

In [6]:

```
# taking x0 and y0 value such that all the corners and center of the plot will be c
x0 = np.array([-6, -6, 0, 6, 6])
y0 = np.array([-6, 6, 0, -6, 6])


final_x0 = np.empty(0)
final_y0 = np.empty(0)

result = np.empty(0)
```

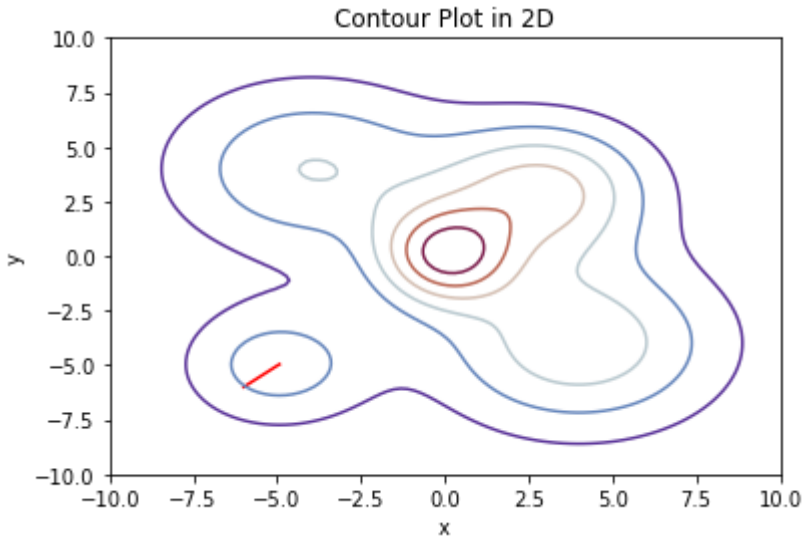
In [7]:

```
# adding 0.0001 so that learning rate could not become zero even in case of round o
learning_rate = round(random.random(), 4) + 0.0001
iterations = 1000


final_x, final_y = VisualizeGA(x0[0], y0[0], learning_rate, iterations)

# update the final_x0, final_y0 array for further presentation
final_x0 = np.append(final_x0, final_x)
final_y0 = np.append(final_y0, final_y)
# store the z value for corresponding returned value of x and y to find the best of
result = np.append(result, f(final_x, final_y))


print(f'Starting Value of x: {x0[0]} and y: {y0[0]}')
print(f'Final value of x: {final_x} and y: {final_y}')
```




```
Starting Value of x: -6 and y: -6
Final value of x: -4.92962 and y: -4.95774
```

Second Test

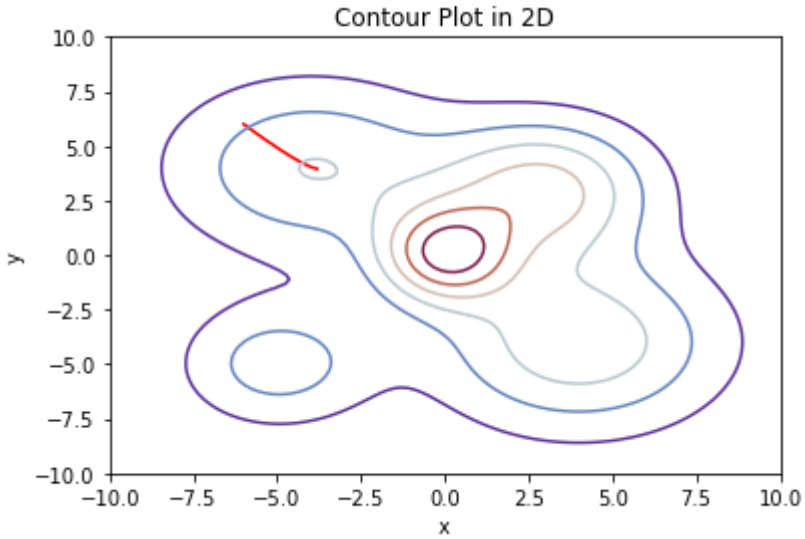
In [8]:

```
# adding 0.0001 so that learning rate could not become zero even in case of round o
learning_rate = round(random.random(), 4) + 0.0001
iterations = 2000

final_x, final_y = VisualizeGA(x0[1], y0[1], learning_rate, iterations)

# update the final_x0, final_y0 array for further presentation
final_x0 = np.append(final_x0, final_x)
final_y0 = np.append(final_y0, final_y)
# store the z value for corresponding returned value of x and y to find the best of
result = np.append(result, f(final_x, final_y))

print(f'Starting Value of x: {x0[1]} and y: {y0[1]}')
print(f'Final value of x: {final_x} and y: {final_y}')
```




```
Starting Value of x: -6 and y: 6
Final value of x: -3.80493 and y: 3.94878
```

Third Test

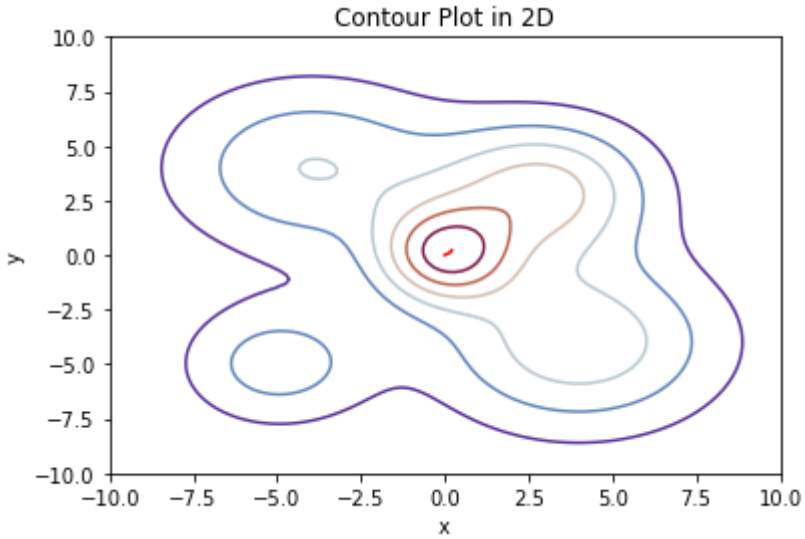
In [9]:

```
# adding 0.0001 so that learning rate could not become zero even in case of round o
learning_rate = round(random.random(), 4) + 0.0001
iterations = 2500

final_x, final_y = VisualizeGA(x0[2], y0[2], learning_rate, iterations)

# update the final_x0, final_y0 array for further presentation
final_x0 = np.append(final_x0, final_x)
final_y0 = np.append(final_y0, final_y)
# store the z value for corresponding returned value of x and y to find the best of
result = np.append(result, f(final_x, final_y))

print(f'Starting Value of x: {x0[2]} and y: {y0[2]}')
print(f'Final value of x: {final_x} and y: {final_y}')
```




```
Starting Value of x: 0 and y: 0
Final value of x: 0.21147 and y: 0.21445
```

Fourth Test

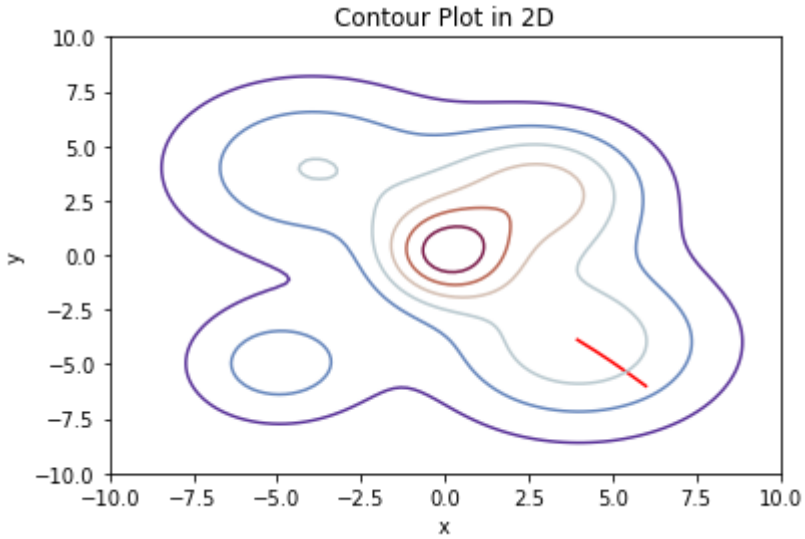
In [10]:

```
# adding 0.0001 so that learning rate could not become zero even in case of round o
learning_rate = round(random.random(), 4) + 0.0001
iterations = 4000

final_x, final_y = VisualizeGA(x0[3], y0[3], learning_rate, iterations)

# update the final_x0, final_y0 array for further presentation
final_x0 = np.append(final_x0, final_x)
final_y0 = np.append(final_y0, final_y)
# store the z value for corresponding returned value of x and y to find the best of
result = np.append(result, f(final_x, final_y))

print(f'Starting Value of x: {x0[3]} and y: {y0[3]}')
print(f'Final value of x: {final_x} and y: {final_y}')
```




```
Starting Value of x: 6 and y: -6
Final value of x: 3.95654 and y: -3.87962
```

Fifth Test

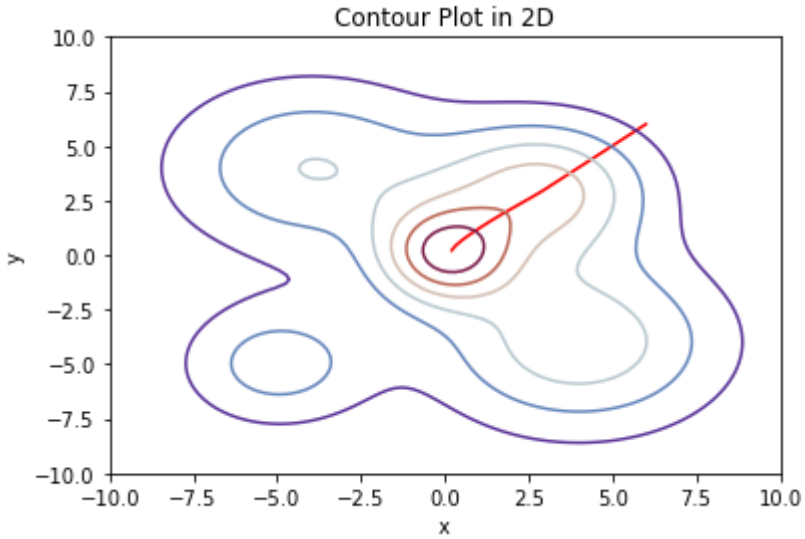
In [11]:

```
# adding 0.0001 so that learning rate could not become zero even in case of round o
learning_rate = round(random.random(), 4) + 0.0001
iterations = 5000

final_x, final_y = VisualizeGA(x0[4], y0[4], learning_rate, iterations)

# update the final_x0, final_y0 array for further presentation
final_x0 = np.append(final_x0, final_x)
final_y0 = np.append(final_y0, final_y)
# store the z value for corresponding returned value of x and y to find the best of
result = np.append(result, f(final_x, final_y))

print(f'Starting Value of x: {x0[4]} and y: {y0[4]}')
print(f'Final value of x: {final_x} and y: {final_y}')
```




```
Starting Value of x: 6 and y: 6
Final value of x: 0.21147 and y: 0.21445
```

Final Conclusion of all the five tests

In [12]:

```
print('The best solution in above test cases is')
index = np.argmax(result)
print(f'x : {final_x0[index]}  y : {final_y0[index]} z : {result[index]}')
```

```
The best solution in above test cases is
x : 0.21147  y : 0.21445 z : 2.725935146467117
```

Problem 2

In [13]:

```
xmin = -10
xmax = 10
ymin = -10
ymax = 10
popSize = 100
nRRI = 100
nRLC = 100
maxItr = 10
```

In [14]:

```
def randomWalk(xyMatrix,Cj):
    lAmbda = np.random.uniform(0,1,size=(popSize))
    newMatrix2 = []
    for a in Cj:
        for b in range(0,int(a)):
            Vij = np.random.uniform(-1,1,size=(2))
            lengthofVij = np.linalg.norm(Vij)
            rij = lAmbda[int(a)]*(Vij/lengthofVij)
            newMatrix2.append(rij + xyMatrix[b])

    newMatrix2 = np.array(newMatrix2)
    return newMatrix2
```

In [15]:

```
def linearCombination(xyMatrix):
    newPoint3 = []
    for i in range(popSize):
        point1 = np.random.randint(0,popSize)
        point2 = np.random.randint(0,popSize)
        alpha1 = np.random.uniform(0,1)
        newPoint3.append((1-alpha1)*xyMatrix[point1] + (alpha1)*xyMatrix[point2])

    newPoint3 = np.array(newPoint3)
    return newPoint3
```

In [16]:

```
def randomReinitialization():
    newalpha = np.random.uniform(0,1,size = (popSize,2))
    xyMatrix1 = np.zeros(shape = (popSize, 2))
    xyMatrix1[:,0] = (1-newalpha[:,0])*xmin + newalpha[:,0]*xmax
    xyMatrix1[:,1]= (1-newalpha[:,1])*ymin + newalpha[:,1]*ymax
    return xyMatrix1
```

In [17]:

```
def calculateYt(x,y):
    Yt =  1.7 * e ** (-( ((x-3)**2/10) + ((y-3)**2/10) )) + e ** (-( ((x+5)**2/8) +
    return Yt
```

In [18]:

```
Smin = [xmin,xmax]
Smax = [ymin,ymax]
```

In [19]:

```
xj = np.linspace(-10,10,100)
yj = np.linspace(-10,10,100)
zj = np.zeros(shape = (100,100)) # z value will be for each combination of xj and y
for i in range(100):
    for j in range(100):
        # equation for calculating z value
        zj[i][j] = 1.7*e **(-((xj[i]-3)**2+(yj[j]-3)**2)/10)+e**(-((xj[i]+5)**2+(yj
```

In [20]:

```
def stochasticSearch(Smin,Smax, popSize,radius,nRRI,nRLC, maxItr):
    alpha = np.random.uniform(0,1,size = (popSize,2)) # alpha value uniformly distr
    xyMatrix = np.zeros(shape = (popSize, 2))
    xyMatrix[:,0] = (1-alpha[:,0])*xmin + alpha[:,0]*xmax
    xyMatrix[:,1]= (1-alpha[:,1])*ymin + alpha[:,1]*ymax
    Yt = 1.7*e**(-((xyMatrix[:,0]-3)**2+(xyMatrix[:,1]-3)**2)/10)+e**(-((xyMatrix[:

    for numofItr in range(0,maxItr):
        Ut = Yt-min(Yt) # Transforming to non-negative values
        Pt = Ut/sum(Ut) # Transforming to relative solution scores
        Cj = np.round(popSize*Pt) # children generation
        aMatrix = randomWalk(xyMatrix,Cj) # random walk function
        bMatrix = linearCombination(xyMatrix) # linear combination function output
        cMatrix = randomReinitialization() # random re-initialization function
        newSample = np.concatenate((aMatrix,bMatrix,cMatrix,xyMatrix)) # all the ou
        newYt = []
        for i in range(newSample.shape[0]):
            newYt.append(calculateYt(newSample[i,0],newSample[i,1])) # new function
        newYt = np.array(newYt)
        df = pd.DataFrame(newSample)
        df[2] = newYt
        df = df.sort_values(by = 2, ascending = False)
        xyMatrix = df[[0,1]].head(popSize).to_numpy() # top 100 values are found, f
        Yt = df[2].head(popSize).to_numpy() # corresponding to that function values
        fig,ax = plt.subplots()
        # Contour plot and scatter plot made
        ax.contour(xj,yj,zj,cmap = 'viridis',level = 30)
        ax.scatter(xyMatrix[:,0],xyMatrix[:,1])
```

In [21]:

```
stochasticSearch(Smin,Smax, popSize,1,nRRI,nRLC, maxItr)
```