Name: **Kunal Sinha** Batch: **Swarm R1- G2** Roll no: **2K17/CO/164**

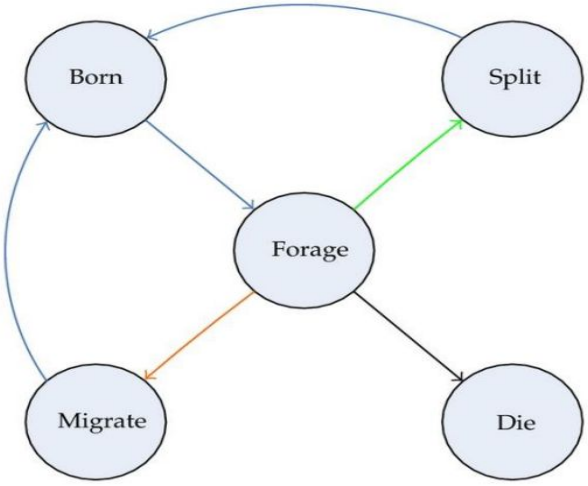
Experiment 6

Aim: Write a program to implement the Bacterial Foraging algorithm.

Theory:

The Bacterial Foraging Optimization, is inspired by the social foraging behavior of E.coli. During foraging of the real bacteria, locomotion is achieved by a set of tensile flagella. Tumble or swim, are two basic operations performed by a bacterium at the time of foraging. When they rotate the flagella in the clockwise direction, each flagellum pulls on the cell.In the above-mentioned algorithm the bacteria undergoes chemotaxis, where they like to move towards a nutrient gradient and avoid a noxious environment. Generally the bacteria move for a longer distance in a friendly environment.

When they get food in sufficient quantities, they are increased in length and in presence of suitable temperature they break in the middle to from an exact replica of itself. Due to the occurrence of sudden environmental changes or attack, the chemotactic progress may be destroyed and a group of bacteria may move to some other places or some other may be introduced in the swarm of concern. This constitutes the event of elimination-dispersal in the real bacterial population, where all the bacteria in a region are killed or a group is dispersed into a new part of the environment.



Working of Bacterial Foraging algorithm

Bacterial Foraging Optimization has four main steps:

- Chemotaxis
- Reproduction
- Elimination
- Dispersal

Algorithm:

1. Initialize randomly the bacteria foraging optimization population
2. Calculate the fitness of each agent
3. Set global best agent to best agent
4. FOR number of iterations
    a. FOR number of chemotactic steps
        i. FOR each search agent
            1. Move agent to the random direction
            2. Calculate the fitness of the moved agent
            3. FOR swimming length
                a. IF current fitness is better than previous
                    i. Move agent to the same direction
                b. ELSE
                    i. Move agent to the random direction
        ii. Calculate the fitness of each agent
    b. END FOR
    c. Compute and sort sum of fitness function of all chemotactic loops (health of agent)
    d. Let live and split only half of the population according to their health
    e. IF not the last iteration
        i. FOR each search agent
            1. With some probability replace agent with new random generated
    f. END IF
    g. Update the best search agent
5. Calculate the fitness of each agent

Source Code:

bacteria.py

```
import numpy as np
from random import random
from . import intelligence

class bfo(intelligence.sw):
    def __init__(self, n, function, lb, ub, dimension, iteration, Nc=2, Ns=12, C=0.2, Ped=1.15):
        """
n: number of agents, function: test function , lb&ub: lower and  upper limits for plot axes
dimension: space dimension , iteration: the number of iterations
Nc: number of chemotactic steps , Ns: swimming length
C: the size of step taken in the random direction specified by the tumble
Ped: elimination-dispersal probability """
```

```
super(bfo, self).__init__()
self.__agents = np.random.uniform(lb, ub, (n, dimension))
self._points(self.__agents)

n_is_even = True
if n & 1:
    n_is_even = False

J = np.array([function(x) for x in self.__agents])
Pbest = self.__agents[J.argmin()]
Gbest = Pbest

C_list = [C - C * 0.9 * i / iteration for i in range(iteration)]
Ped_list = [Ped - Ped * 0.5 * i / iteration for i in range(iteration)]
J_last = J[::1]

for t in range(iteration):
    J_chem = [J[::1]]
    for j in range(Nc):
        for i in range(n):
            dell = np.random.uniform(-1, 1, dimension)
            self.__agents[i] += C_list[t] * np.linalg.norm(dell) * dell

            for m in range(Ns):
                if function(self.__agents[i]) < J_last[i]:
                    J_last[i] = J[i]
                    self.__agents[i] += C_list[t] * np.linalg.norm(dell) \ * dell
                else:
                    dell = np.random.uniform(-1, 1, dimension)
                    self.__agents[i] += C_list[t] * np.linalg.norm(dell) \* dell

        J = np.array([function(x) for x in self.__agents])
        J_chem += [J]

    J_chem = np.array(J_chem)
    J_health = [(sum(J_chem[:, i]), i) for i in range(n)]
    J_health.sort()
    alived_agents = []
    for i in J_health:
        alived_agents += [list(self.__agents[i[1]])]
```

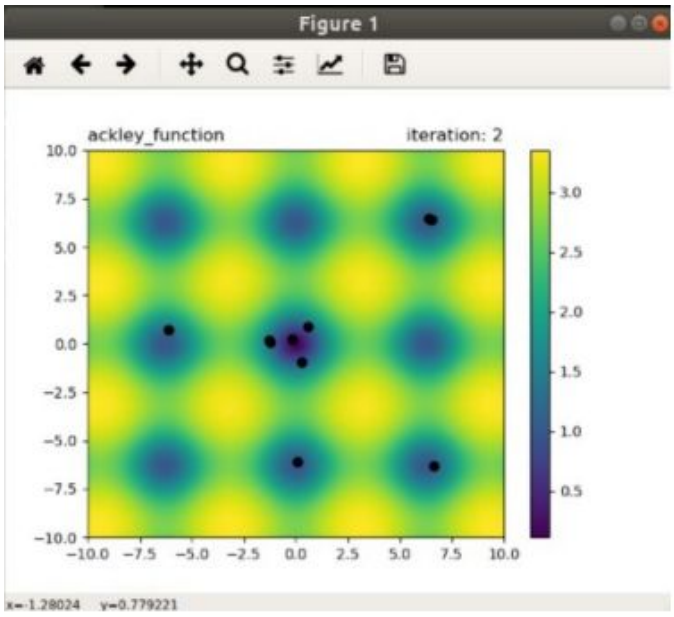
```
        if n_is_even:
            alived_agents = 2*alived_agents[:n//2]
            self.__agents = np.array(alived_agents)
        else:
            alived_agents = 2*alived_agents[:n//2] +\
                        [alived_agents[n//2]]
            self.__agents = np.array(alived_agents)

        if t < iteration - 2:
            for i in range(n):
                r = random()
                if r >= Ped_list[t]:
                    self.__agents[i] = np.random.uniform(lb, ub, dimension)

        J = np.array([function(x) for x in self.__agents])
        self._points(self.__agents)

        Pbest = self.__agents[J.argmin()]
        if function(Pbest) < function(Gbest):
            Gbest = Pbest
    self._set_Gbest(Gbest)
```

main.py

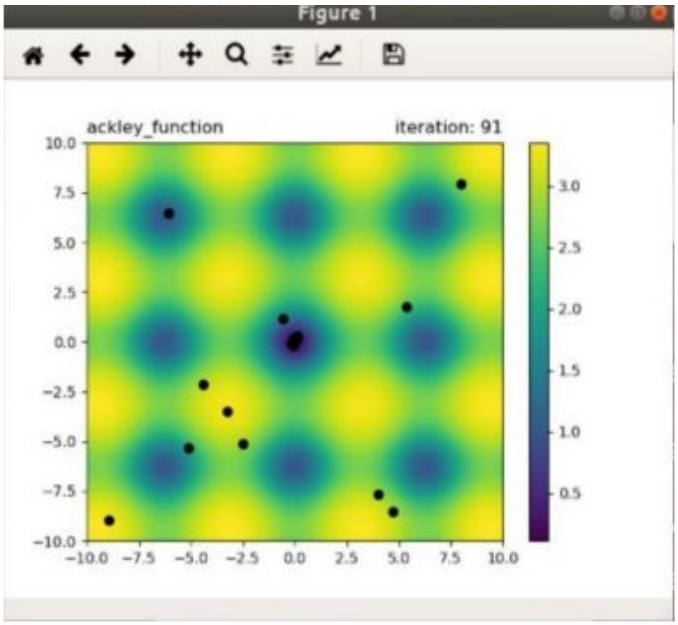
```
from math import *
import bacteria.py as bfo
import matplotlib.pyplot as plt

def ackley_function(x):
    return -exp(-sqrt(0.5*sum([i**2 for i in x]))) - \exp(0.5*sum([cos(i) for i in x])) + 1 + exp(1)

alh = bfo(50, ackely_function, -10,10, 2, 90,2, 12,0.2, 1.15)
plt(alh.get_agents(),easom_function, -10, 10)
```

Output:

For 90 iterations



Initial epoch



Final epoch

Finding and Learnings:

We have successfully implemented the Bacterial Foraging Algorithm (BFOA) in python. BFOA has been widely accepted as a global optimization algorithm of current interest for optimization and control. BFOA has already drawn the attention of researchers because of its efficiency in solving real-world optimization problems arising in several application domains.