USN : 1BM22CS253

LAB-2 : Par cle Swarm Op miza on for Func on Op miza on:

CODE:

```
#lab-3: pso import
numpy as np import
random


# Define the op miza on problem (Rastrigin Func on)
def rastrigin(x):      A = 10      return A * len(x) + sum([(xi**2 - A *
np.cos(2 * np.pi * xi)) for xi in x])


# Par cle Swarm Op miza on (PSO) implementa on
class Par cle:
   def __init__(self, dimension, lower_bound, upper_bound):        # Ini alize the
par   cle   posi   on   and   velocity   randomly                       self.posi   on   =
np.random.uniform(lower_bound, upper_bound, dimension)        self.velocity =
np.random.uniform(-1, 1, dimension)        self.best_posi on = np.copy(self.posi
on)
       self.best_value = rastrigin(self.posi on)

   def update_velocity(self,
global_best_posi        # Update the velocity
of the par cle        r1 =
np.random.rand(len(self.posi on))        r2 =
np.random.rand(len(self.posi on))


       # Iner a term        iner a
= w * self.velocity
```

Cogni ve term (individual best)

on, w, c1, c2): cogni ve = c1 * r1 * (self.best_posi on - self.posi on)

Social term (global best) social = c2 * r2 *

(global_best_posi on - self.posi on)

Update velocity self.velocity =

iner a + cogni ve + social

def update_posi on(self, lower_bound, upper_bound):

Update the posi on of the par cle

self.posi on = self.posi on + self.velocity

Ensure the par cle stays within the bounds self.posi on =

np.clip(self.posi on, lower_bound, upper_bound)

def evaluate(self):

Evaluate the fitness of the par cle

fitness = rastrigin(self.posi on)

Update the par cle's best posi on if necessary

if fitness < self.best_value: self.best_value =

fitness

self.best_posi on = np.copy(self.posi on)

```
def par cle_swarm_op miza on(dim, lower_bound, upper_bound, num_par cles=30,
max_iter=100, w=0.5, c1=1.5, c2=1.5):
    # Ini alize par cles     par cles = [Par cle(dim, lower_bound, upper_bound) for _ in
range(num_par cles)]


    # Ini alize the global best posi on and value    global_best_posi
on = par cles[0].best_posi on          global_best_value = par
cles[0].best_value


    for i in range(max_iter):
# Update each par cle
for par cle in par cles:
            par cle.update_velocity(global_best_posi on, w, c1, c2)
par cle.update_posi on(lower_bound, upper_bound)
            par cle.evaluate()


            # Update global best posi on if needed                if par
cle.best_value < global_best_value:               global_best_value
= par cle.best_value               global_best_posi on = np.copy(par
cle.best_posi on)


        # Op onally print the progress
if (i+1 ) % 10 == 0:
            print(f"Itera on {i+1 }/{max_iter} - Best Fitness: {global_best_value}")

    return global_best_posi on, global_best_value
```

Set the parameters for the PSO algorithm dim = 2

Number of dimensions for the func on lower_bound = -

5.12 # Lower bound of the search space upper_bound =

5.12 # Upper bound of the search space num_par cles =

30 # Number of par cles in the swarm max_iter = 100

Number of itera ons

Run the PSO

best_posi on, best_value = par cle_swarm_op miza on(dim, lower_bound, upper_bound, num_par cles, max_iter)

Output the best solu on found

print("\nBest Solu on Found:")

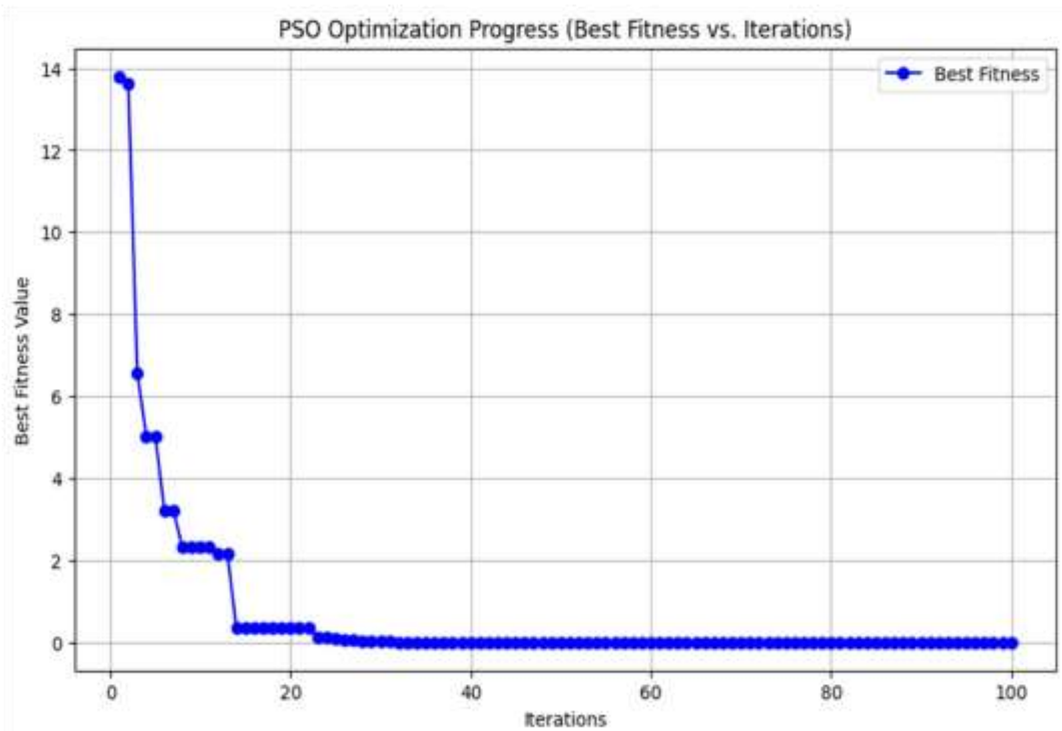
print("Posi on:", best_posi on)

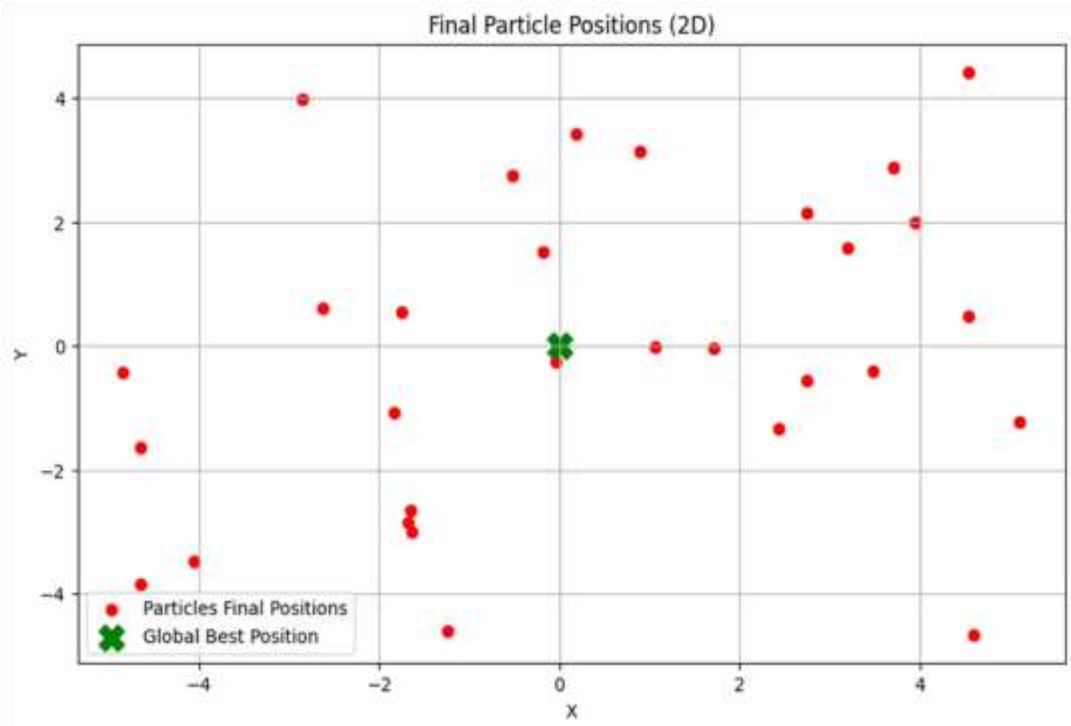
print("Fitness:", best_value)

OUTPUT:

```
Iteration 10/100 - Best Fitness: 2.3145203625443997
Iteration 20/100 - Best Fitness: 0.34026142761705813
Iteration 30/100 - Best Fitness: 0.0158886712260653
Iteration 40/100 - Best Fitness: 5.572809527620848e-06
Iteration 50/100 - Best Fitness: 3.493363465167931e-08
Iteration 60/100 - Best Fitness: 2.8475000135586015e-11
Iteration 70/100 - Best Fitness: 1.4210854715202004e-14
Iteration 80/100 - Best Fitness: 0.0
Iteration 90/100 - Best Fitness: 0.0
Iteration 100/100 - Best Fitness: 0.0

Best Solution Found:
Position: [ 1.64289135e-09 -1.88899730e-09]
Fitness: 0.0
```




Final Particle Positions (2D)
Particles Final Positions
Global Best Position
X
Y
−4
−2
0
2
4