LAB - 03 **(13-11-24)**

Particle Swarm Optimization for Function Optimization:
Particle Swarm Optimization (PSO) is inspired by the social behavior of birds flocking or fish schooling. PSO is used to find optimal solutions by iteratively improving a candidate solution with regard to a given measure of quality. Implement the PSO algorithm using Python to optimize a mathematical function.

Implementation Steps:
1.Define the Problem: Create a mathematical function to optimize.
2. Initialize Parameters: Set the number of particles, inertia weight, cognitive and social coefficients.
3. Initialize Particles: Generate an initial population of particles with random positions and velocities.
4. Evaluate Fitness: Evaluate the fitness of each particle based on the optimization function.
5.Update Velocities and Positions: Update the velocity and position of each particle based on its own best position and the global best position.
6. Iterate: Repeat the evaluation, updating, and position adjustment for a fixed number of iterations or until convergence criteria are met.
7.Output the Best Solution: Track and output the best solution found during the iterations.

ALGORITHM/LOGIC -

Date 13/11/2024
Page 09

PARTICLE SWARM OPTIMIZATION (PSO)

- Particles → represents objects in problem
- Final solution → convergence
- Fitness function → used to verify answer
- pBest → personal best → individual value
  gBest → global best → overall value
- velocity update →

velocity → $v_i(t+1) = w \cdot v_i(t) + c_1 r_1 (pBest - x_i(t)) + c_2 r_2 (gBest - x_i(t))$

inertia weight (w), current pos. ($x_i(t)$)

- position update →
  $x_i(t+1) = x_i(t) + v_i(t+1)$
- $y = f(x) = -x^2 + 5x + 20$

ALGORITHM:

Initialization - numpy random
Fitness Evaluation
Update pBest, gBest
Update v_i & x_i
Repeat → $\dfrac{\text{convergence}}{\text{Optimal / Near optimal}}$

1. Create a 'population' of agents (particles) uniformly distributed over x

2. Evaluate each particle's position according to objective function
   ex: {$y = f(x) = -x^2 + 5x + 20$}

3. If a particle's current position is better than its previous best position update it.

4. Determine the best particle according to particle's previous best positions.

5. Update particle's velocities:

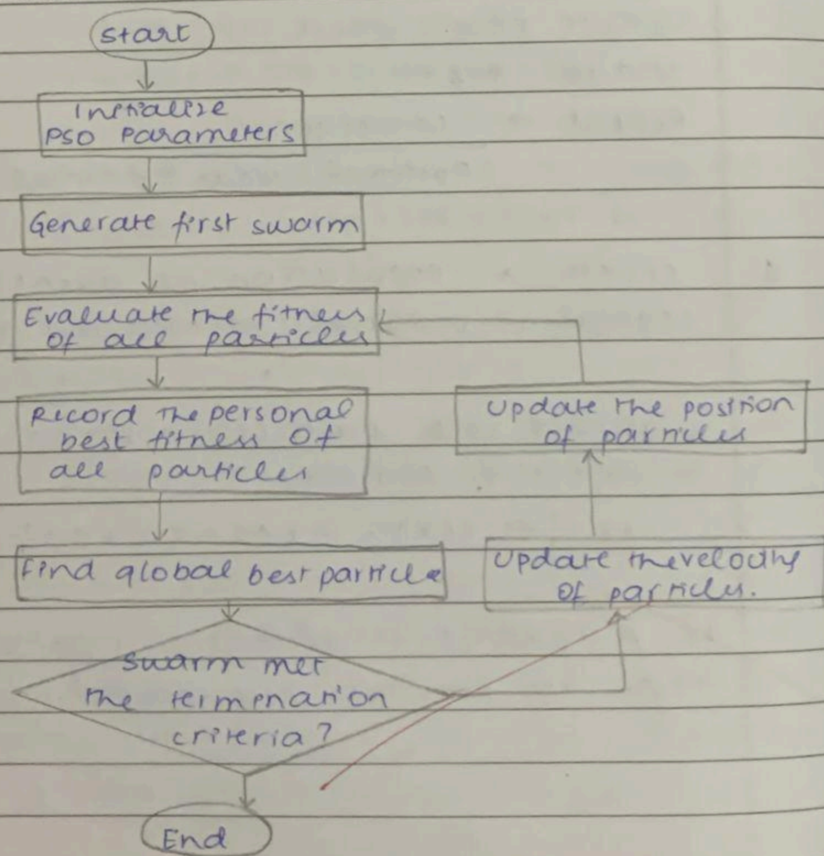
$$v_i^{t+1} = \underbrace{\underbrace{v_i^t}_{\text{inertia}}}_{\text{diversification}} + \underbrace{\underbrace{c_1 U_1^t (pb_i^t - p_i^t)}_{\text{personal influence}} + \underbrace{c_2 U_2^t (gb^t - p_i^t)}_{\text{social influence}}}_{\text{intensification}}$$

6. Move particles to their new position:

$$p_i^{t+1} = p_i^t + v_i^{t+1}$$

7. Go to step 2 until stopping criteria is satisfied.

FLOWCHART:



Date 13/11/2024
Page 11.

Mathematical Example and Interpretation

1. De Jong Function

$$\min f(x,y) = x^2 + y^2$$

2. Rastrigin function

$$\sum_{i=1}^{D} (x_i - 10 \cdot \cos(2 \cdot \pi \cdot x_i))$$

3. Banana Function

$$f(x) = \sum_{i=1}^{d-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$$

CODE OUTPUT :

Iteration 1 : Best value = 1.7014947
Iteration 2 : Best value = 0.253033
Iteration 3: Best value = 0.2530337
:
:
:
Iteration 100: Best value = 4.069002

Global Best Position: [-7.4152439
1.87592380]
Global Best value : 4.06900299

frm
13/11/24

INPUT-

```
import numpy as np
# Objective function (simple sphere function)
def objective_function(x):
    return np.sum(x**2)

dim = 2
num_particles = 30
max_iter = 100
w = 0.5
c1 = 1.5
c2 = 1.5
# Initialize particles and velocities
position = np.random.uniform(-10, 10, (num_particles, dim))
velocity = np.random.uniform(-1, 1, (num_particles, dim))
best_position = np.copy(position)  # Best position of each particle
best_value = np.array([objective_function(p) for p in position])
global_best_position = position[np.argmin(best_value)]
global_best_value = np.min(best_value)
for iteration in range(max_iter):
    for i in range(num_particles):
        fitness = objective_function(position[i])
        if fitness < best_value[i]:
            best_value[i] = fitness
            best_position[i] = position[i]
        if fitness < global_best_value:
            global_best_value = fitness
            global_best_position = position[i]
    r1 = np.random.rand(num_particles, dim)
    r2 = np.random.rand(num_particles, dim)
    cognitive_velocity = c1 * r1 * (best_position - position)
    social_velocity = c2 * r2 * (global_best_position - position)
    velocity = w * velocity + cognitive_velocity + social_velocity
    position = position + velocity
    position = np.clip(position, -10, 10)
    print(f"Iteration {iteration + 1}: Best Value = {global_best_value}")

print(f"Global Best Position: {global_best_position}")
print(f"Global Best Value: {global_best_value}")
```

OUTPUT-

```
Iteration 1: Best Value = 1.7014947401095102
Iteration 2: Best Value = 0.25303378309282165
Iteration 3: Best Value = 0.25303378309282165
Iteration 4: Best Value = 0.07620297968731589
Iteration 5: Best Value = 0.07620297968731589
Iteration 6: Best Value = 0.017382130704648294
Iteration 7: Best Value = 0.017382130704648294
Iteration 8: Best Value = 0.006806027660032736
Iteration 9: Best Value = 0.006806027660032736
Iteration 10: Best Value = 0.0002629872625108001
Iteration 11: Best Value = 0.0002629872625108001
Iteration 12: Best Value = 0.0002629872625108001
Iteration 13: Best Value = 0.0002629872625108001
Iteration 14: Best Value = 0.0002629872625108001
Iteration 15: Best Value = 0.0002629872625108001
Iteration 16: Best Value = 4.0944344068559024e-05
Iteration 17: Best Value = 4.0944344068559024e-05
Iteration 18: Best Value = 8.464803493676039e-06
Iteration 19: Best Value = 8.464803493676039e-06
Iteration 20: Best Value = 8.464803493676039e-06
Iteration 21: Best Value = 8.464803493676039e-06
Iteration 22: Best Value = 8.464803493676039e-06
Iteration 23: Best Value = 8.464803493676039e-06
Iteration 24: Best Value = 4.912976553510539e-06
Iteration 25: Best Value = 7.679103335966e-07
Iteration 26: Best Value = 7.679103335966e-07
Iteration 27: Best Value = 3.8746558200222264e-07
Iteration 28: Best Value = 1.666962019114016e-07
Iteration 29: Best Value = 5.2065151822392155e-09
Iteration 30: Best Value = 5.2065151822392155e-09
Iteration 31: Best Value = 5.2065151822392155e-09
Iteration 32: Best Value = 5.2065151822392155e-09
Iteration 33: Best Value = 5.2065151822392155e-09
Iteration 34: Best Value = 1.5988754096957557e-09
Iteration 35: Best Value = 1.5988754096957557e-09
Iteration 36: Best Value = 2.319609044724985e-10
Iteration 37: Best Value = 2.319609044724985e-10
Iteration 38: Best Value = 2.319609044724985e-10
Iteration 39: Best Value = 2.319609044724985e-10
Iteration 40: Best Value = 2.319609044724985e-10
Iteration 41: Best Value = 2.319609044724985e-10
Iteration 42: Best Value = 4.8832938340287904e-11
Iteration 43: Best Value = 4.8832938340287904e-11
```

```
Iteration 86: Best Value = 3.864689774642681e-20
Iteration 87: Best Value = 3.864689774642681e-20
Iteration 88: Best Value = 2.9153190675205583e-20
Iteration 89: Best Value = 2.3654622639355475e-21
Iteration 90: Best Value = 5.4335713571518875e-22
Iteration 91: Best Value = 5.4335713571518875e-22
Iteration 92: Best Value = 5.4335713571518875e-22
Iteration 93: Best Value = 5.4335713571518875e-22
Iteration 94: Best Value = 5.34859527954185e-22
Iteration 95: Best Value = 5.34859527954185e-22
Iteration 96: Best Value = 1.8740078996512297e-22
Iteration 97: Best Value = 1.8740078996512297e-22
Iteration 98: Best Value = 4.6807574561835797e-23
Iteration 99: Best Value = 6.241452381268762e-24
Iteration 100: Best Value = 4.069002998359533e-24
Global Best Position: [-7.41562439e-13  1.87592328e-12]
Global Best Value: 4.069002998359533e-24
```