

Lab:-2 Particle swarm optimization for function optimization:

code for pso

```
import numpy as np
import random

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

# Particle Swarm Optimization (PSO) implementation
class Particle:
    def __init__(self, dimension, lower_bound, upper_bound):
        # Initialize the particle position and velocity randomly
        self.position = np.random.uniform(lower_bound, upper_bound, dimension)
        self.velocity = np.random.uniform(-1, 1, dimension)
        self.best_position = np.copy(self.position)
        self.best_value = rastrigin(self.position)

    def update_velocity(self, global_best_position, w, c1, c2):
        # Update the velocity of the particle
        r1 = np.random.rand(len(self.position))
        r2 = np.random.rand(len(self.position))

        # Inertia term
        inertia = w * self.velocity

        # Cognitive term (individual best)
```

```

cognitive = c1 * r1 * (self.best_position - self.position)

# Social term (global best)
social = c2 * r2 * (global_best_position - self.position)

# Update velocity
self.velocity = inertia + cognitive + social

def update_position(self, lower_bound, upper_bound):
    # Update the position of the particle
    self.position = self.position + self.velocity

    # Ensure the particle stays within the bounds
    self.position = np.clip(self.position, lower_bound, upper_bound)

def evaluate(self):
    # Evaluate the fitness of the particle
    fitness = rastrigin(self.position)

    # Update the particle's best position if necessary
    if fitness < self.best_value:
        self.best_value = fitness
        self.best_position = np.copy(self.position)

def particle_swarm_optimization(dim, lower_bound, upper_bound, num_particles=30,
max_iter=100, w=0.5, c1=1.5, c2=1.5):
    # Initialize particles
    particles = [Particle(dim, lower_bound, upper_bound) for _ in range(num_particles)]

    # Initialize the global best position and value
    global_best_position = particles[0].best_position

```

```

global_best_value = particles[0].best_value

for i in range(max_iter):
    # Update each particle
    for particle in particles:
        particle.update_velocity(global_best_position, w, c1, c2)
        particle.update_position(lower_bound, upper_bound)
        particle.evaluate()

    # Update global best position if needed
    if particle.best_value < global_best_value:
        global_best_value = particle.best_value
        global_best_position = np.copy(particle.best_position)

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

return global_best_position, global_best_value

# Set the parameters for the PSO algorithm
dim = 2          # Number of dimensions for the function
lower_bound = -5.12 # Lower bound of the search space
upper_bound = 5.12  # Upper bound of the search space
num_particles = 30  # Number of particles in the swarm
max_iter = 100     # Number of iterations

# Run the PSO
best_position, best_value = particle_swarm_optimization(dim, lower_bound, upper_bound,
num_particles, max_iter)

```

```
# Output the best solution found

print("\nBest Solution Found:")

print("Position:", best_position)

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
```



