Lab-2-Particle Swarm Optimization for Function Optimization

Code:

```
import numpy as np

# Define the 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])

class Particle:
    def __init__(self, dim):
        self.position = np.random.uniform(-5.12, 5.12, dim)
        self.velocity = np.random.uniform(-1, 1, dim)
        self.best_position = np.copy(self.position)
        self.best_value = rastrigin(self.position)

    def update_velocity(self, global_best_position, inertia_weight, cognitive_coef, social_coef):
        r1, r2 = np.random.rand(2)
        cognitive_velocity = cognitive_coef * r1 * (self.best_position - self.position)
        social_velocity = social_coef * r2 * (global_best_position - self.position)
        self.velocity = inertia_weight * self.velocity + cognitive_velocity + social_velocity

    def update_position(self):
        self.position += self.velocity
        # Keep the particle within the bounds
        self.position = np.clip(self.position, -5.12, 5.12)

        # Update the best position if necessary
        current_value = rastrigin(self.position)
        if current_value < self.best_value:
            self.best_value = current_value
            self.best_position = np.copy(self.position)

def pso(num_particles, dim, num_iterations):
    inertia_weight = 0.7
    cognitive_coef = 1.5
    social_coef = 1.5
```

```
    # Initialize particles
    particles = [Particle(dim) for _ in range(num_particles)]
    global_best_position = particles[0].best_position
    global_best_value = particles[0].best_value

    # Main PSO loop
    for _ in range(num_iterations):
        for particle in particles:
            particle.update_velocity(global_best_position, inertia_weight, cognitive_coef,
social_coef)
            particle.update_position()

            # Update global best
            if particle.best_value < global_best_value:
                global_best_value = particle.best_value
                global_best_position = particle.best_position

    return global_best_position, global_best_value

# User input for parameters
num_particles = int(input("Enter the number of particles: "))
dim = int(input("Enter the number of dimensions: "))
num_iterations = int(input("Enter the number of iterations: "))

# Run PSO
best_position, best_value = pso(num_particles, dim, num_iterations)

print(f"Best Position: {best_position}")
print(f"Best Value: {best_value}")
```

Output:

```
Enter the number of particles: 600
Enter the number of dimensions: 3
Enter the number of iterations: 4
Best Position: [ 1.003255   -0.07832814 -0.01143473]
Best Value: 2.227487748175008
```