CODE:

```
import numpy as np

# Define the mathematical function to optimize (Rastrigin function as an example)
def rastrigin_function(position):
    """Rastrigin function: f(x) = 10n + sum(x_i^2 - 10 * cos(2 * pi * x_i))"""
    return 10 * len(position) + sum(x**2 - 10 * np.cos(2 * np.pi * x) for x in position)

# Particle Swarm Optimization implementation
class Particle:
    def __init__(self, dimensions, bounds):
        self.position = np.random.uniform(bounds[0], bounds[1], dimensions)  # Random position
        self.velocity = np.random.uniform(-1, 1, dimensions)  # Random velocity
        self.best_position = np.copy(self.position)  # Best position found by this particle
        self.best_score = float('inf')  # Best fitness score (minimization problem)

    def update_velocity(self, global_best_position, inertia, cognitive_coeff, social_coeff):
        r1 = np.random.uniform(0, 1, len(self.position))
        r2 = np.random.uniform(0, 1, len(self.position))
        cognitive_term = cognitive_coeff * r1 * (self.best_position - self.position)
        social_term = social_coeff * r2 * (global_best_position - self.position)
        self.velocity = inertia * self.velocity + cognitive_term + social_term

    def update_position(self, bounds):
        self.position += self.velocity
        # Apply boundary constraints
        self.position = np.clip(self.position, bounds[0], bounds[1])

def particle_swarm_optimization(
    func, dimensions, bounds, num_particles, num_iterations, inertia, cognitive_coeff,
social_coeff
):
    # Initialize particles
    particles = [Particle(dimensions, bounds) for _ in range(num_particles)]
    global_best_position = None
    global_best_score = float('inf')

    # Main optimization loop
    for iteration in range(num_iterations):
        for particle in particles:
```

```
            # Evaluate fitness
            fitness = func(particle.position)
            # Update particle's best known position and fitness
            if fitness < particle.best_score:
                particle.best_score = fitness
                particle.best_position = np.copy(particle.position)
            # Update global best if necessary
            if fitness < global_best_score:
                global_best_score = fitness
                global_best_position = np.copy(particle.position)

        # Update velocities and positions of particles
        for particle in particles:
            particle.update_velocity(global_best_position, inertia, cognitive_coeff, social_coeff)
            particle.update_position(bounds)

        # Print progress
        print(f"Iteration {iteration+1}/{num_iterations}, Best Score: {global_best_score}")

    return global_best_position, global_best_score

# Parameters for PSO
dimensions = 2  # Number of dimensions (e.g., 2 for visualization)
bounds = [-5.12, 5.12]  # Search space boundaries for Rastrigin function
num_particles = 30  # Number of particles
num_iterations = 100  # Number of iterations
inertia = 0.7  # Inertia weight
cognitive_coeff = 1.5  # Cognitive coefficient
social_coeff = 1.5  # Social coefficient

# Run PSO
best_position, best_score = particle_swarm_optimization(
    func=rastrigin_function,
    dimensions=dimensions,
    bounds=bounds,
    num_particles=num_particles,
    num_iterations=num_iterations,
    inertia=inertia,
    cognitive_coeff=cognitive_coeff,
    social_coeff=social_coeff
```

```
)

print("\nOptimal Solution Found:")
print(f"Best Position: {best_position}")
print(f"Best Score: {best_score}")
```

OUTPUT:

```
Iteration 1/5, Best Score: 13.3195103853709 54
Iteration 2/5, Best Score: 5.596161615907283
Iteration 3/5, Best Score: 4.804706909869424
Iteration 4/5, Best Score: 1.7254378781457085
Iteration 5/5, Best Score: 1.7254378781457085

Optimal Solution Found:
Best Position: [ 0.06056807 -0.0718261 ]
Best Score: 1.7254378781457085
```