Particle Swarm Optimization for Function Optimization

```
import numpy as np


# Define the objective function to be optimized (e.g., f(x, y) = x^2 + y^2)
def objective_function(position):
    return position[0] ** 2 + position[1] ** 2


# Particle class to represent each particle in the swarm
class Particle:
    def __init__(self, bounds):
        # Initialize position and velocity randomly within bounds
        self.position = np.array([np.random.uniform(bound[0], bound[1]) for bound in bounds])
        self.velocity = np.array([np.random.uniform(-1, 1) for _ in bounds])
        self.best_position = self.position.copy()
        self.best_score = objective_function(self.position)


    def update_velocity(self, global_best_position, inertia_weight, cognitive_coef, social_coef):
        # Generate random factors for stochastic update
        r1, r2 = np.random.rand(2)


        # Update velocity based on inertia, cognitive, and social components
        cognitive_component = cognitive_coef * r1 * (self.best_position - self.position)
        social_component = social_coef * r2 * (global_best_position - self.position)


        self.velocity = inertia_weight * self.velocity + cognitive_component + social_component


    def update_position(self, bounds):
        # Update position based on velocity
        self.position += self.velocity
        # Enforce boundary conditions
        for i in range(len(bounds)):
```

```
            if self.position[i] < bounds[i][0]:
                self.position[i] = bounds[i][0]
            elif self.position[i] > bounds[i][1]:
                self.position[i] = bounds[i][1]

    def evaluate(self):
        # Evaluate the fitness of the particle
        score = objective_function(self.position)
        # Update personal best if the new position is better
        if score < self.best_score:
            self.best_score = score
            self.best_position = self.position.copy()

# Particle Swarm Optimization (PSO) algorithm
def particle_swarm_optimization(num_particles, bounds, inertia_weight, cognitive_coef, social_coef, max_iterations):
    # Initialize swarm
    swarm = [Particle(bounds) for _ in range(num_particles)]
    global_best_position = swarm[0].position.copy()
    global_best_score = objective_function(global_best_position)

    # Find initial global best
    for particle in swarm:
        if particle.best_score < global_best_score:
            global_best_score = particle.best_score
            global_best_position = particle.best_position.copy()

    # Optimization loop
    for iteration in range(max_iterations):
        for particle in swarm:
            # Update particle velocity and position
```

```
            particle.update_velocity(global_best_position, inertia_weight, cognitive_coef, social_coef)
            particle.update_position(bounds)
            particle.evaluate()

            # Update global best if the particle's best position is better
            if particle.best_score < global_best_score:
                global_best_score = particle.best_score
                global_best_position = particle.best_position.copy()

    # After all iterations, print the final global best score
    print(f"Final Iteration {max_iterations}, Global Best Score: {global_best_score}")
    return global_best_position, global_best_score

# Define the parameters for PSO
num_particles = 30                 # Number of particles in the swarm
bounds = [(5, 10), (5, 10)]      # Bounds for the search space (e.g., x and y can vary from -10 to 10)
inertia_weight = 0.5              # Inertia weight
cognitive_coef = 1.5              # Cognitive (personal best) coefficient
social_coef = 1.5                 # Social (global best) coefficient
max_iterations = 100              # Number of iterations to run the algorithm

# Run PSO to find the minimum of the objective function
best_position, best_score = particle_swarm_optimization(num_particles, bounds, inertia_weight,
cognitive_coef, social_coef, max_iterations)

print(f"Best Position: {best_position}")
print(f"Best Score: {best_score}")
```

```
Final Iteration 100, Global Best Score: 50.0
Best Position: [5. 5.]
Best Score: 50.0
```
s