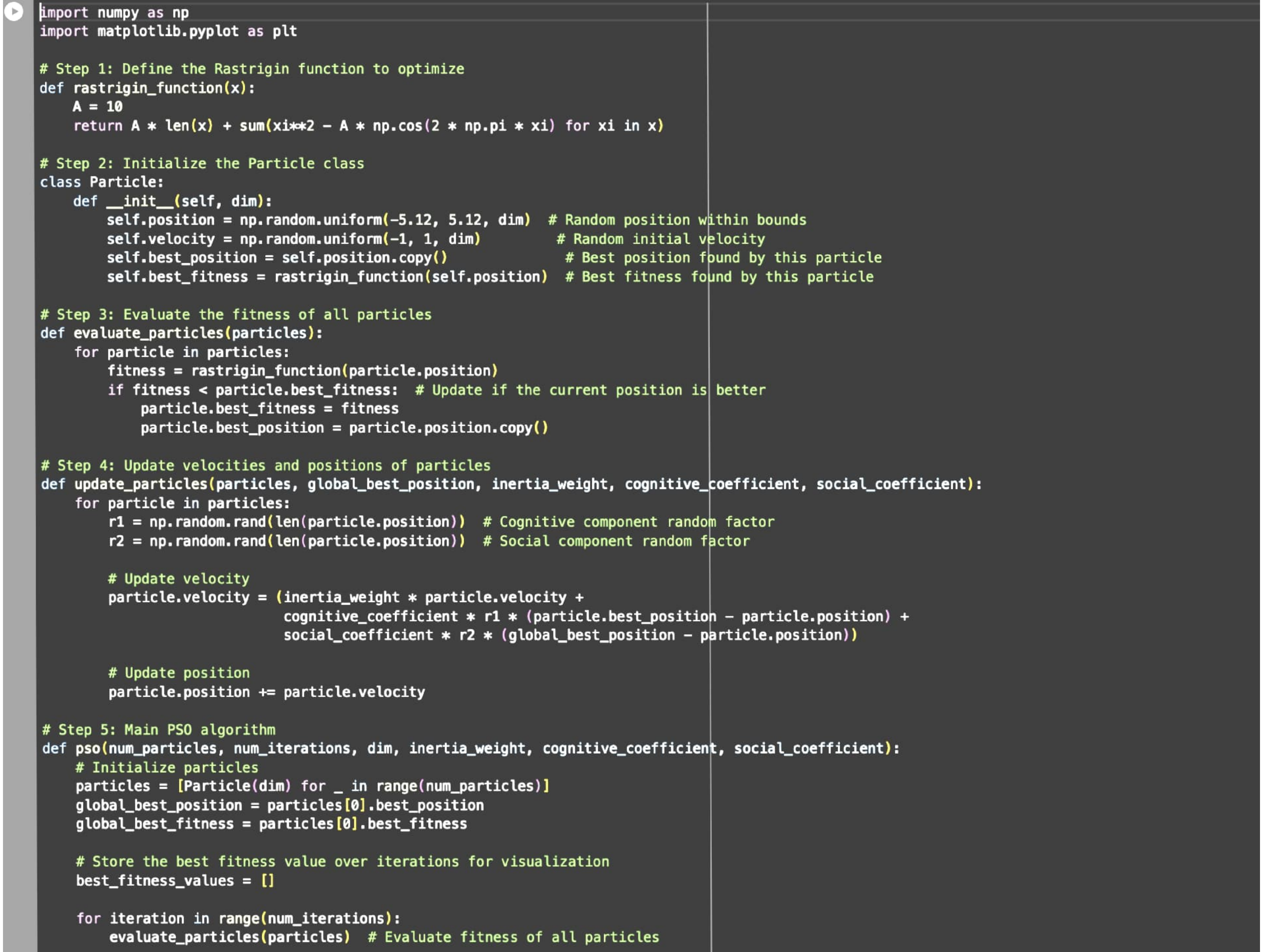
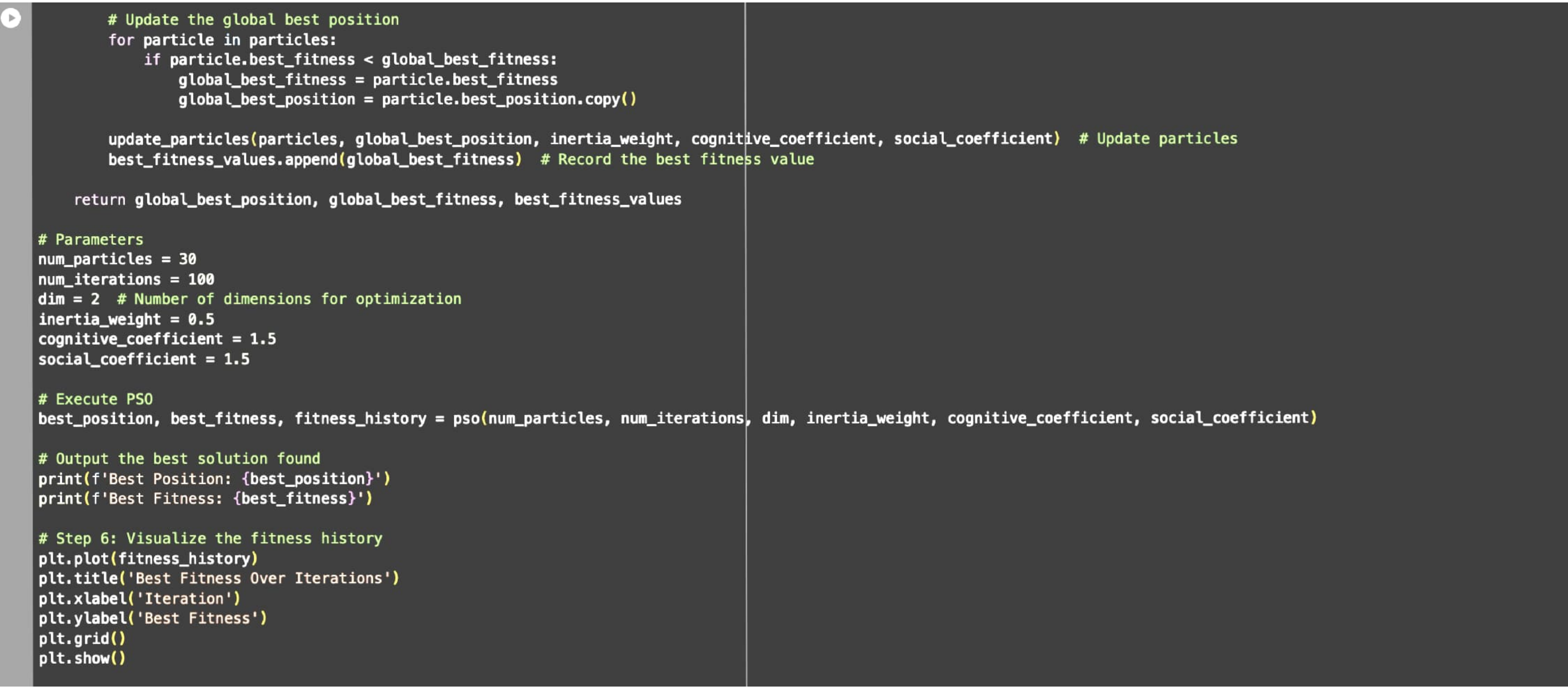
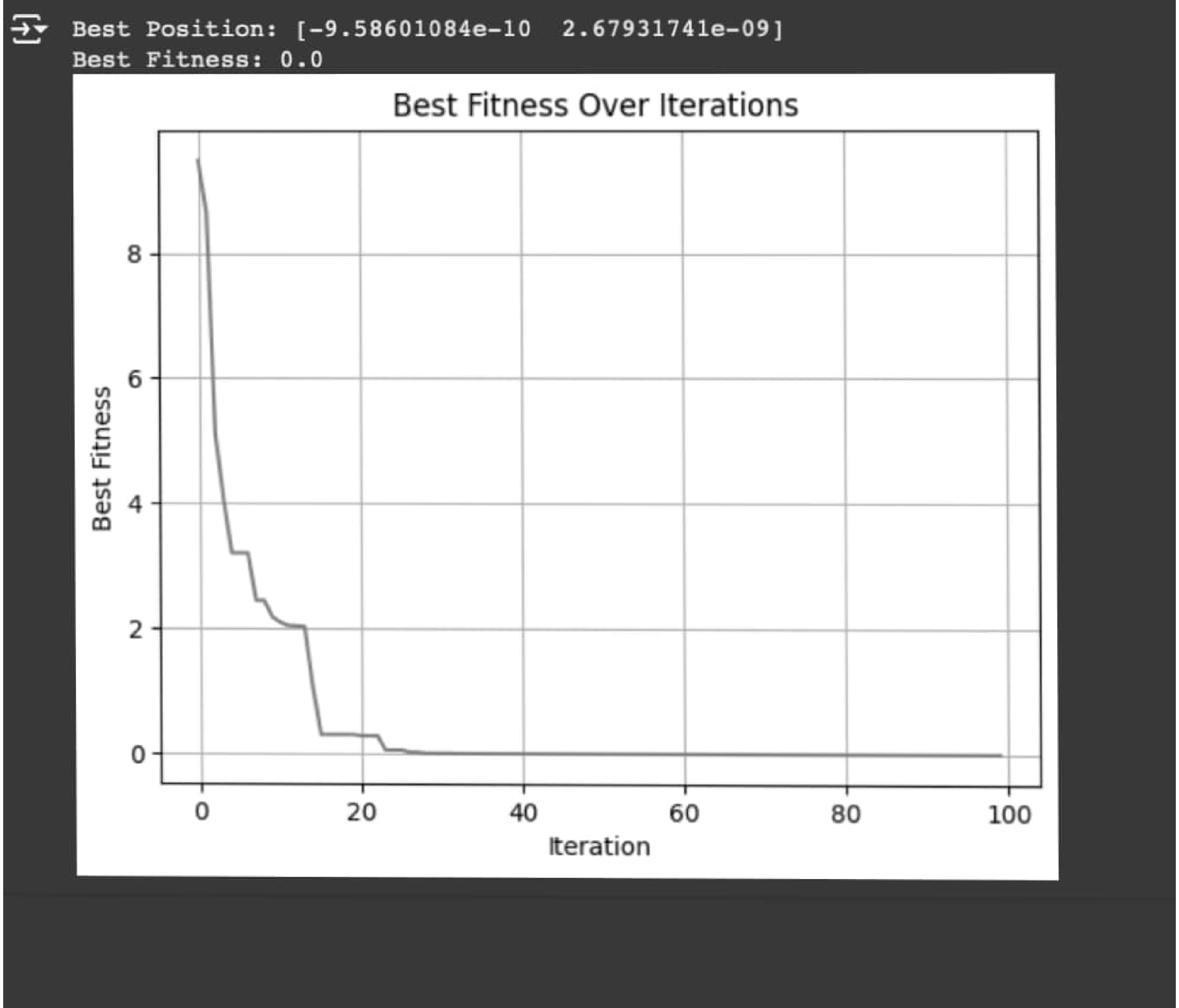


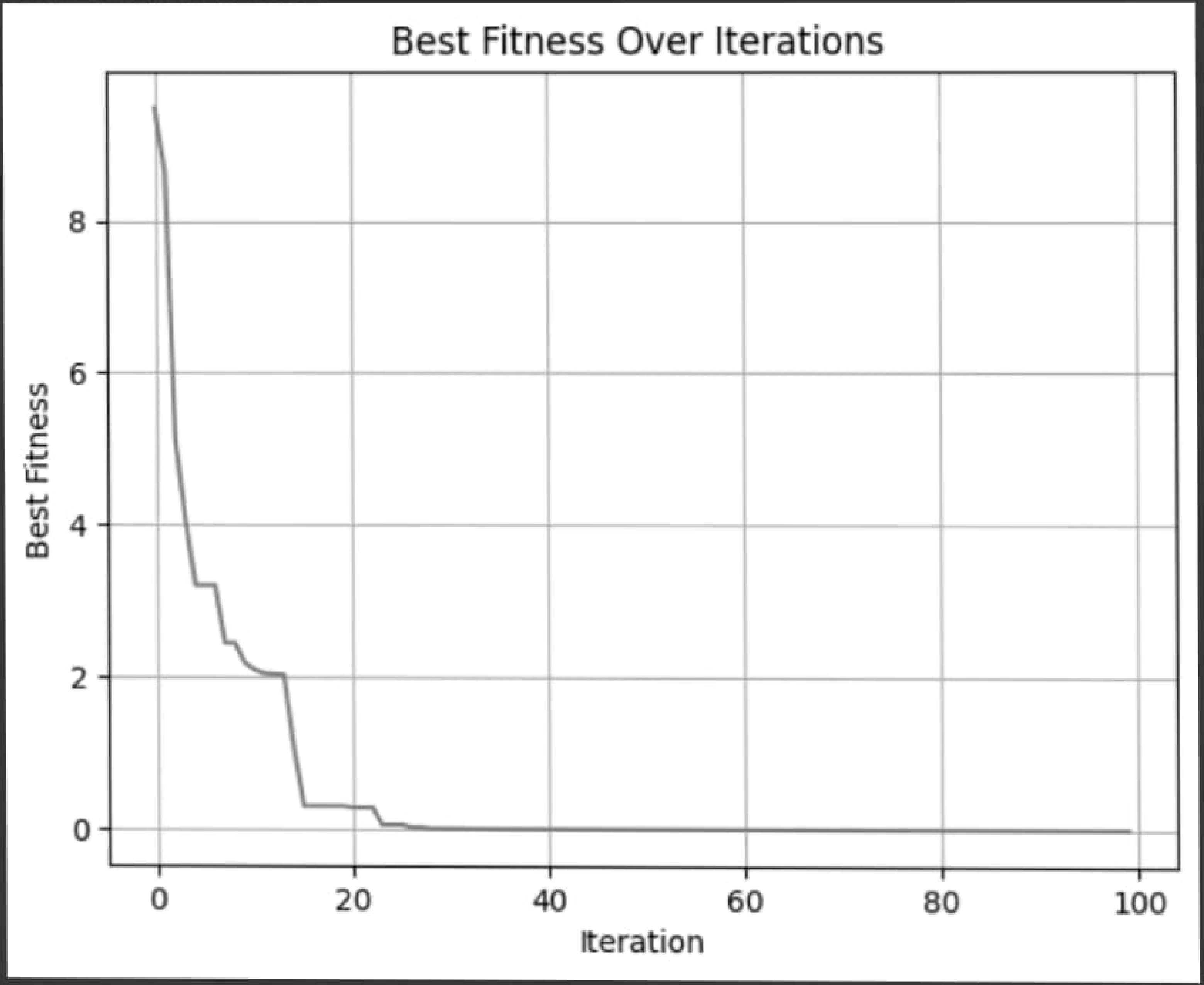
```python
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Define the Rastrigin function to optimize
def rastrigin_function(x):
    A = 10
    return A * len(x) + sum(xi**2 - A * np.cos(2 * np.pi * xi) for xi in x)

# Step 2: Initialize the Particle class
class Particle:
    def __init__(self, dim):
        self.position = np.random.uniform(-5.12, 5.12, dim)  # Random position within bounds
        self.velocity = np.random.uniform(-1, 1, dim)         # Random initial velocity
        self.best_position = self.position.copy()             # Best position found by this particle
        self.best_fitness = rastrigin_function(self.position)  # Best fitness found by this particle

# Step 3: Evaluate the fitness of all particles
def evaluate_particles(particles):
    for particle in particles:
        fitness = rastrigin_function(particle.position)
        if fitness < particle.best_fitness:  # Update if the current position is better
            particle.best_fitness = fitness
            particle.best_position = particle.position.copy()

# Step 4: Update velocities and positions of particles
def update_particles(particles, global_best_position, inertia_weight, cognitive_coefficient, social_coefficient):
    for particle in particles:
        r1 = np.random.rand(len(particle.position))  # Cognitive component random factor
        r2 = np.random.rand(len(particle.position))  # Social component random factor

        # Update velocity
        particle.velocity = (inertia_weight * particle.velocity +
                             cognitive_coefficient * r1 * (particle.best_position - particle.position) +
                             social_coefficient * r2 * (global_best_position - particle.position))

        # Update position
        particle.position += particle.velocity

# Step 5: Main PSO algorithm
def pso(num_particles, num_iterations, dim, inertia_weight, cognitive_coefficient, social_coefficient):
    # Initialize particles
    particles = [Particle(dim) for _ in range(num_particles)]
    global_best_position = particles[0].best_position
    global_best_fitness = particles[0].best_fitness

    # Store the best fitness value over iterations for visualization
    best_fitness_values = []

    for iteration in range(num_iterations):
        evaluate_particles(particles)  # Evaluate fitness of all particles
```



```
        # Update the global best position
        for particle in particles:
            if particle.best_fitness < global_best_fitness:
                global_best_fitness = particle.best_fitness
                global_best_position = particle.best_position.copy()

        update_particles(particles, global_best_position, inertia_weight, cognitive_coefficient, social_coefficient)  # Update particles
        best_fitness_values.append(global_best_fitness)  # Record the best fitness value

    return global_best_position, global_best_fitness, best_fitness_values

# Parameters
num_particles = 30
num_iterations = 100
dim = 2  # Number of dimensions for optimization
inertia_weight = 0.5
cognitive_coefficient = 1.5
social_coefficient = 1.5

# Execute PSO
best_position, best_fitness, fitness_history = pso(num_particles, num_iterations, dim, inertia_weight, cognitive_coefficient, social_coefficient)

# Output the best solution found
print(f'Best Position: {best_position}')
print(f'Best Fitness: {best_fitness}')

# Step 6: Visualize the fitness history
plt.plot(fitness_history)
plt.title('Best Fitness Over Iterations')
plt.xlabel('Iteration')
plt.ylabel('Best Fitness')
plt.grid()
plt.show()
```



```
Best Position: [-9.58601084e-10  2.67931741e-09]
Best Fitness: 0.0
```


Best Fitness Over Iterations
Best Fitness
Iteration