BIS LAB 12/09/2025

Shreya Sathyanarayana (1BM23CS318)

CODE:

```
import random

# Objective (fitness) function: De Jong function
def fitness_function(position):
    x, y = position
    return x**2 + y**2  # Minimize this function (simple sphere function)

# PSO parameters
num_particles = 10
num_iterations = 50
W = 0.3  # Inertia weight (from PDF)
C1 = 2   # Cognitive coefficient
C2 = 2   # Social coefficient

# Initialize particles and velocities
particles = [[random.uniform(-10, 10), random.uniform(-10, 10)] for _ in
range(num_particles)]
velocities = [[0.0, 0.0] for _ in range(num_particles)]

# Initialize personal bests
pbest_positions = [p[:] for p in particles]
pbest_values = [fitness_function(p) for p in particles]

# Initialize global best
gbest_index = pbest_values.index(min(pbest_values))
gbest_position = pbest_positions[gbest_index][:]
```

```
gbest_value = pbest_values[gbest_index]

# PSO main loop
for iteration in range(num_iterations):
    for i in range(num_particles):
        r1, r2 = random.random(), random.random()

        # Update velocity
        velocities[i][0] = (W * velocities[i][0] +
                            C1 * r1 * (pbest_positions[i][0] - particles[i][0]) +
                            C2 * r2 * (gbest_position[0] - particles[i][0]))

        velocities[i][1] = (W * velocities[i][1] +
                            C1 * r1 * (pbest_positions[i][1] - particles[i][1]) +
                            C2 * r2 * (gbest_position[1] - particles[i][1]))

        # Update position
        particles[i][0] += velocities[i][0]
        particles[i][1] += velocities[i][1]

        # Evaluate fitness
        current_value = fitness_function(particles[i])

        # Update personal best
        if current_value < pbest_values[i]:
            pbest_positions[i] = particles[i][:]
            pbest_values[i] = current_value

        # Update global best
```

```
        if current_value < gbest_value:
            gbest_value = current_value
            gbest_position = particles[i][:]

    print(f"Iteration {iteration + 1}/{num_iterations} | Best Value: {gbest_value:.6f} at {gbest_position}")

# Final result
print("\n Optimal Solution Found:")
print(f"Best Position: {gbest_position}")
print(f"Minimum Value: {gbest_value}")
```

Output:

```
Iteration 1/50 | Best Value: 8.350949 at [-0.13285633641609695, -2.8867452859744938]
Iteration 2/50 | Best Value: 3.444807 at [1.2689795097374952, -1.3544363028467488]
Iteration 3/50 | Best Value: 2.633939 at [1.263361022333822, -1.0187529560002615]
Iteration 4/50 | Best Value: 2.597594 at [1.5498117781139409, -0.4423548096752685]
Iteration 5/50 | Best Value: 2.205992 at [1.4679197494061158, -0.2262818171368084]
Iteration 6/50 | Best Value: 1.363363 at [0.823417538513798, -0.8278563712856422]
Iteration 7/50 | Best Value: 0.460178 at [0.5185051431831358, -0.4374135786354102]
Iteration 8/50 | Best Value: 0.098650 at [0.27056083682766224, -0.15952035972173184]
Iteration 9/50 | Best Value: 0.068873 at [0.2303028966684491, 0.12582964448442996]
Iteration 10/50 | Best Value: 0.050659 at [0.1760701209850321, 0.14020642063831784]
Iteration 11/50 | Best Value: 0.017810 at [0.12615603800958625, -0.0435293117389462]
Iteration 12/50 | Best Value: 0.008377 at [0.05819951337372323, -0.07063777838689785]
Iteration 13/50 | Best Value: 0.002704 at [-0.02092857974858304, -0.047601265736911716]
Iteration 14/50 | Best Value: 0.001833 at [-0.03708673731031219, -0.021380281149983915]
Iteration 15/50 | Best Value: 0.001833 at [-0.03708673731031219, -0.021380281149983915]
Iteration 16/50 | Best Value: 0.000520 at [-0.013749637046011162, -0.01819400916811135]
Iteration 17/50 | Best Value: 0.000107 at [-9.033398915178972e-05, -0.010349359374246222]
Iteration 18/50 | Best Value: 0.000080 at [0.0040074569279060215, -0.007995964436086684]
```

Iteration 19/50 | Best Value: 0.000076 at [0.004214936047380493, -0.00764228669591201]

Iteration 20/50 | Best Value: 0.000067 at [0.0006753902763684777, -0.008139977309183035]

Iteration 21/50 | Best Value: 0.000052 at [-0.0028816077907871706, -0.006627025201237949]

Iteration 22/50 | Best Value: 0.000013 at [0.002317629225177361, -0.002733058389431016]

Iteration 23/50 | Best Value: 0.000013 at [0.002317629225177361, -0.002733058389431016]

Iteration 24/50 | Best Value: 0.000012 at [0.002615979981203413, -0.0022270032798487774]

Iteration 25/50 | Best Value: 0.000006 at [-0.0012207718892239634, -0.0021565393119762424]

Iteration 26/50 | Best Value: 0.000006 at [-0.0007074770183736296, -0.0023299525348497293]

Iteration 27/50 | Best Value: 0.000000 at [4.6121856184661025e-05, -0.0006307362091496761]

Iteration 28/50 | Best Value: 0.000000 at [4.6121856184661025e-05, -0.0006307362091496761]

Iteration 29/50 | Best Value: 0.000000 at [0.00023395245975185556, -0.0005291936419950416]

Iteration 30/50 | Best Value: 0.000000 at [0.00023395245975185556, -0.0005291936419950416]

Iteration 31/50 | Best Value: 0.000000 at [0.0004840154124449775, -0.0001303484258812932]

Iteration 32/50 | Best Value: 0.000000 at [8.202319230135786e-05, 0.00019046408790314372]

Iteration 33/50 | Best Value: 0.000000 at [8.202319230135786e-05, 0.00019046408790314372]

Iteration 34/50 | Best Value: 0.000000 at [8.202319230135786e-05, 0.00019046408790314372]

Iteration 35/50 | Best Value: 0.000000 at [8.202319230135786e-05, 0.00019046408790314372]

Iteration 36/50 | Best Value: 0.000000 at [6.581138869102385e-05, 0.00019511343386258387]

Iteration 37/50 | Best Value: 0.000000 at [-9.379793541379456e-05, 9.502473972091346e-05]

Iteration 38/50 | Best Value: 0.000000 at [-7.190831547577639e-05, -5.5718477581929906e-05]

Iteration 39/50 | Best Value: 0.000000 at [-7.190831547577639e-05, -5.5718477581929906e-05]

Iteration 40/50 | Best Value: 0.000000 at [-7.190831547577639e-05, -5.5718477581929906e-05]

Iteration 41/50 | Best Value: 0.000000 at [-3.1734913564110745e-05, -5.7220547491965823e-05]

Iteration 42/50 | Best Value: 0.000000 at [3.632935297631629e-06, -5.689422970721357e-05]

Iteration 43/50 | Best Value: 0.000000 at [1.4867243368358823e-05, 2.455415596777931e-05]

Iteration 44/50 | Best Value: 0.000000 at [1.4867243368358823e-05, 2.455415596777931e-05]

Iteration 45/50 | Best Value: 0.000000 at [8.00132977819793e-06, 4.377199099630766e-06]

Iteration 46/50 | Best Value: 0.000000 at [8.00132977819793e-06, 4.377199099630766e-06]

Iteration 47/50 | Best Value: 0.000000 at [8.00132977819793e-06, 4.377199099630766e-06]

Iteration 48/50 | Best Value: 0.000000 at [8.00132977819793e-06, 4.377199099630766e-06]

Iteration 49/50 | Best Value: 0.000000 at [8.00132977819793e-06, 4.377199099630766e-06]

Iteration 50/50 | Best Value: 0.000000 at [8.00132977819793e-06, 4.377199099630766e-06]

Optimal Solution Found:

Best Position: [8.00132977819793e-06, 4.377199099630766e-06]

Minimum Value: 8.318115017728534e-11