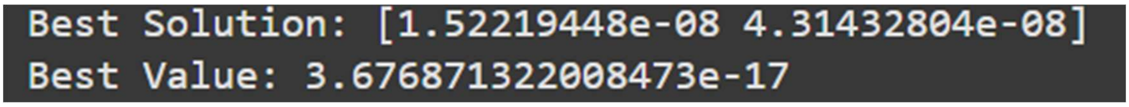
Lab 2 – Particle Swarm Optimization

```
import numpy as np

def objective_function(x):
    """Example objective function: Sphere function."""
    return sum(x_i**2 for x_i in x)

def pso(obj_func, bounds, num_particles=30, iterations=100, w=0.5, c1=1.5, c2=1.5):
    """
    Particle Swarm Optimization (PSO) implementation.

    Parameters:
        obj_func: The objective function to optimize.
        bounds: A 2D array of [min, max] for each dimension.
        num_particles: Number of particles in the swarm.
        iterations: Number of iterations.
        w: Inertia weight.
        c1: Cognitive coefficient.
        c2: Social coefficient.

    Returns:
        gbest_position: The best solution found.
        gbest_value: The fitness of the best solution.
    """
    num_dimensions = len(bounds)

    # Initialize particles' positions and velocities
    positions = np.random.uniform(bounds[:, 0], bounds[:, 1], (num_particles, num_dimensions))
    velocities = np.random.uniform(-1, 1, (num_particles, num_dimensions))
```

```python
# Initialize personal best positions and their fitness
personal_best_positions = np.copy(positions)
personal_best_values = np.array([obj_func(p) for p in positions])

# Initialize global best position and its fitness
gbest_index = np.argmin(personal_best_values)
gbest_position = personal_best_positions[gbest_index]
gbest_value = personal_best_values[gbest_index]

# Optimization loop
for _ in range(iterations):
    for i in range(num_particles):
        # Update velocity
        r1 = np.random.random(num_dimensions)
        r2 = np.random.random(num_dimensions)
        cognitive_component = c1 * r1 * (personal_best_positions[i] - positions[i])
        social_component = c2 * r2 * (gbest_position - positions[i])
        velocities[i] = w * velocities[i] + cognitive_component + social_component

        # Update position
        positions[i] += velocities[i]
        positions[i] = np.clip(positions[i], bounds[:, 0], bounds[:, 1])  # Boundary enforcement

        # Evaluate fitness
        fitness = obj_func(positions[i])

        # Update personal best
        if fitness < personal_best_values[i]:
            personal_best_positions[i] = positions[i]
            personal_best_values[i] = fitness
            # Update global best
            if fitness < gbest_value:
                gbest_position = positions[i]
                gbest_value = fitness

    return gbest_position, gbest_value

# Example usage
if __name__ == "__main__":
    # Define the problem
    bounds = np.array([[-10, 10], [-10, 10]])  # Search space bounds
    best_solution, best_value = pso(objective_function, bounds)
    print("Best Solution:", best_solution)
    print("Best Value:", best_value)
```

OUTPUT:



```
Best Solution: [1.52219448e-08 4.31432804e-08]
Best Value: 3.676871322008473e-17
```