Lab 3

Ant colony optimization

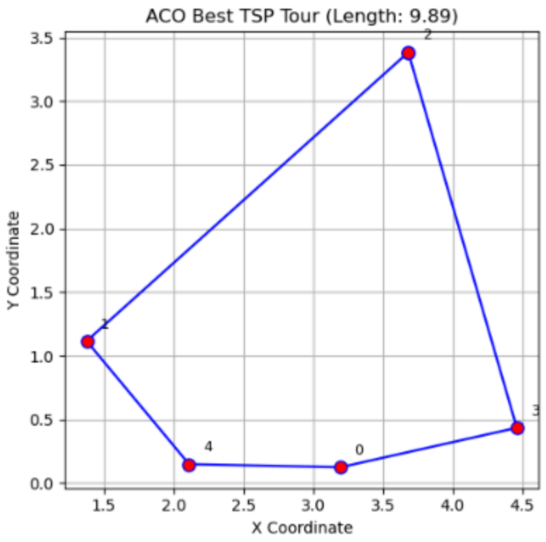
```python
import numpy as np
import random
import matplotlib.pyplot as plt
def create_distance_matrix(cities):
    """Calculates the Euclidean distance matrix between all pairs of cities."""
    num_cities = len(cities)
    dist_matrix = np.zeros((num_cities, num_cities))
    for i in range(num_cities):
        for j in range(i + 1, num_cities):
            distance = np.sqrt((cities[i][0] - cities[j][0])**2 + (cities[i][1] - cities[j][1])**2)
            dist_matrix[i, j] = dist_matrix[j, i] = distance
    return dist_matrix

def calculate_tour_length(tour, dist_matrix):
    """Calculates the total length of a given tour (sequence of city indices)."""
    length = 0
    num_cities = len(tour)
    for i in range(num_cities):
        city_a = tour[i]
        city_b = tour[(i + 1) % num_cities]
        length += dist_matrix[city_a, city_b]
    return length

def aco_tsp_solver(cities, num_ants=10, max_iterations=100, alpha=1.0, beta=5.0, rho=0.5, initial_pheromone=1.0):
    """
    Ant Colony Optimization (ACO) algorithm for the Traveling Salesman Problem (TSP).

    Args:
        cities (list of tuples): List of (x, y) coordinates for each city.
        num_ants (int): Number of artificial ants (M).
        max_iterations (int): Maximum number of generations to run.
        alpha (float): Influence of the pheromone trail (tau).
        beta (float): Influence of the heuristic information (eta, 1/distance).
        rho (float): Pheromone evaporation rate.
        initial_pheromone (float): Initial pheromone value (tau_0).
    """
    num_cities = len(cities)
    dist_matrix = create_distance_matrix(cities)
    eta_matrix = 1.0 / (dist_matrix + np.finfo(float).eps)
    np.fill_diagonal(eta_matrix, 0)

    pheromone_matrix = np.full((num_cities, num_cities), initial_pheromone)
    best_tour = None
    best_length = float('inf')
```

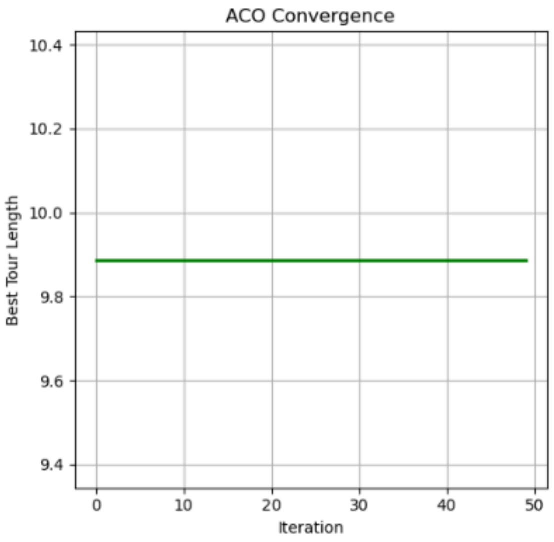
```python
pheromone_matrix = np.full((num_cities, num_cities), initial_pheromone)
best_tour = None
best_length = float('inf')
history = []

print(f"Starting ACO for {num_cities} cities with {num_ants} ants...")

for iteration in range(max_iterations):
    all_tours = []
    all_lengths = []
    for ant in range(num_ants):
        start_city = random.randint(0, num_cities - 1)
        tour = [start_city]
        visited = {start_city}

        for _ in range(num_cities - 1):
            current_city = tour[-1]
            unvisited_cities = [c for c in range(num_cities) if c not in visited]

            if not unvisited_cities:
                break
            probabilities = []
            denominator = 0.0

            for next_city in unvisited_cities:
                tau = pheromone_matrix[current_city, next_city] ** alpha
                eta = eta_matrix[current_city, next_city] ** beta
                numerator = tau * eta
                probabilities.append((next_city, numerator))
                denominator += numerator

            if denominator == 0:
                next_city = random.choice(unvisited_cities)
            else:
                prob_values = [p[1] / denominator for p in probabilities]
                next_city = random.choices(
                    [p[0] for p in probabilities],
                    weights=prob_values,
                    k=1
                )[0]

            tour.append(next_city)
            visited.add(next_city)

        tour_length = calculate_tour_length(tour, dist_matrix)
        all_tours.append(tour)
        all_lengths.append(tour_length)
```

```python
            tour_length = calculate_tour_length(tour, dist_matrix)
            all_tours.append(tour)
            all_lengths.append(tour_length)
            if tour_length < best_length:
                best_length = tour_length
                best_tour = tour

        history.append(best_length)
        pheromone_matrix = (1 - rho) * pheromone_matrix

        if best_tour is not None:

            delta_tau = 1.0 / best_length

            for i in range(num_cities):
                city_a = best_tour[i]
                city_b = best_tour[(i + 1) % num_cities]
                pheromone_matrix[city_a, city_b] += delta_tau
                pheromone_matrix[city_b, city_a] += delta_tau

        print(f"Iteration {iteration+1}/{max_iterations}: Best Length = {best_length:.2f}")

    return best_tour, best_length, history



def run_aco_example():

    random.seed(42)
    np.random.seed(42)
    cities = [(random.uniform(0, 5), random.uniform(0, 5)) for _ in range(5)]

    best_tour, best_length, history = aco_tsp_solver(
        cities,
        num_ants=10,
        max_iterations=50,
        alpha=1.0,
        beta=5.0,
        rho=0.1
    )

    print("\n--- Results ---")
    print(f"Cities: {cities}")
    print(f"Best Tour (City Indices): {best_tour}")
    print(f"Best Tour Length: {best_length:.4f}")
    )

    print("\n--- Results ---")
    print(f"Cities: {cities}")
    print(f"Best Tour (City Indices): {best_tour}")
    print(f"Best Tour Length: {best_length:.4f}")

    if best_tour:

        x_coords = [cities[i][0] for i in best_tour]
        y_coords = [cities[i][1] for i in best_tour]

        x_coords.append(x_coords[0])
        y_coords.append(y_coords[0])

        plt.figure(figsize=(10, 5))

        plt.subplot(1, 2, 1)
        plt.plot(x_coords, y_coords, 'o-', color='blue', markerfacecolor='red', markersize=8)
        for i, (x, y) in enumerate(cities):
            plt.text(x + 0.1, y + 0.1, str(i), fontsize=9)
        plt.title(f'ACO Best TSP Tour (Length: {best_length:.2f})')
        plt.xlabel('X Coordinate')
        plt.ylabel('Y Coordinate')
        plt.grid(True)

        plt.subplot(1, 2, 2)
        plt.plot(history, color='green', linewidth=2)
        plt.title('ACO Convergence')
        plt.xlabel('Iteration')
        plt.ylabel('Best Tour Length')
        plt.grid(True)
        plt.tight_layout()
        plt.show()

if __name__ == '__main__':
    run_aco_example()
```

```
Starting ACO for 5 cities with 10 ants...
Iteration 1/50: Best Length = 9.89
Iteration 2/50: Best Length = 9.89
Iteration 3/50: Best Length = 9.89
Iteration 4/50: Best Length = 9.89
Iteration 5/50: Best Length = 9.89
Iteration 6/50: Best Length = 9.89
Iteration 7/50: Best Length = 9.89
Iteration 8/50: Best Length = 9.89
Iteration 9/50: Best Length = 9.89
Iteration 10/50: Best Length = 9.89
Iteration 11/50: Best Length = 9.89
Iteration 12/50: Best Length = 9.89
Iteration 13/50: Best Length = 9.89
Iteration 14/50: Best Length = 9.89
Iteration 15/50: Best Length = 9.89
Iteration 16/50: Best Length = 9.89
Iteration 17/50: Best Length = 9.89
Iteration 18/50: Best Length = 9.89
Iteration 19/50: Best Length = 9.89
Iteration 20/50: Best Length = 9.89
Iteration 21/50: Best Length = 9.89
Iteration 22/50: Best Length = 9.89
Iteration 23/50: Best Length = 9.89
Iteration 24/50: Best Length = 9.89
Iteration 25/50: Best Length = 9.89
Iteration 26/50: Best Length = 9.89
Iteration 27/50: Best Length = 9.89
Iteration 28/50: Best Length = 9.89
Iteration 29/50: Best Length = 9.89
Iteration 30/50: Best Length = 9.89
Iteration 31/50: Best Length = 9.89
Iteration 32/50: Best Length = 9.89
Iteration 33/50: Best Length = 9.89
Iteration 34/50: Best Length = 9.89
Iteration 35/50: Best Length = 9.89
Iteration 36/50: Best Length = 9.89
Iteration 37/50: Best Length = 9.89
Iteration 38/50: Best Length = 9.89
Iteration 39/50: Best Length = 9.89
Iteration 40/50: Best Length = 9.89
Iteration 41/50: Best Length = 9.89
Iteration 42/50: Best Length = 9.89
Iteration 43/50: Best Length = 9.89
Iteration 44/50: Best Length = 9.89
Iteration 45/50: Best Length = 9.89
Iteration 46/50: Best Length = 9.89
Iteration 47/50: Best Length = 9.89
Iteration 48/50: Best Length = 9.89
Iteration 49/50: Best Length = 9.89
Iteration 50/50: Best Length = 9.89

--- Results ---
Cities: [(3.1971339922894186, 0.12505377611333468), (1.3751465918455963, 1.1160536907441139), (3.682356070820062, 3.3834974371145563), (4.46089783852422
7, 0.43469416314708076), (2.1096090984263522, 0.14898609719035172)]
Best Tour (City Indices): [2, 3, 0, 4, 1]
Best Tour Length: 9.8880
```

```
--- Results ---
Cities: [(3.1971339922894186, 0.12505377611333468), (1.3751465918455963, 1.1160536907441139), (3.682356070820062, 3.3834974371145563), (4.46089783852422
7, 0.43469416314708076), (2.1096090984263522, 0.14898609719035172)]
Best Tour (City Indices): [2, 3, 0, 4, 1]
Best Tour Length: 9.8880
```

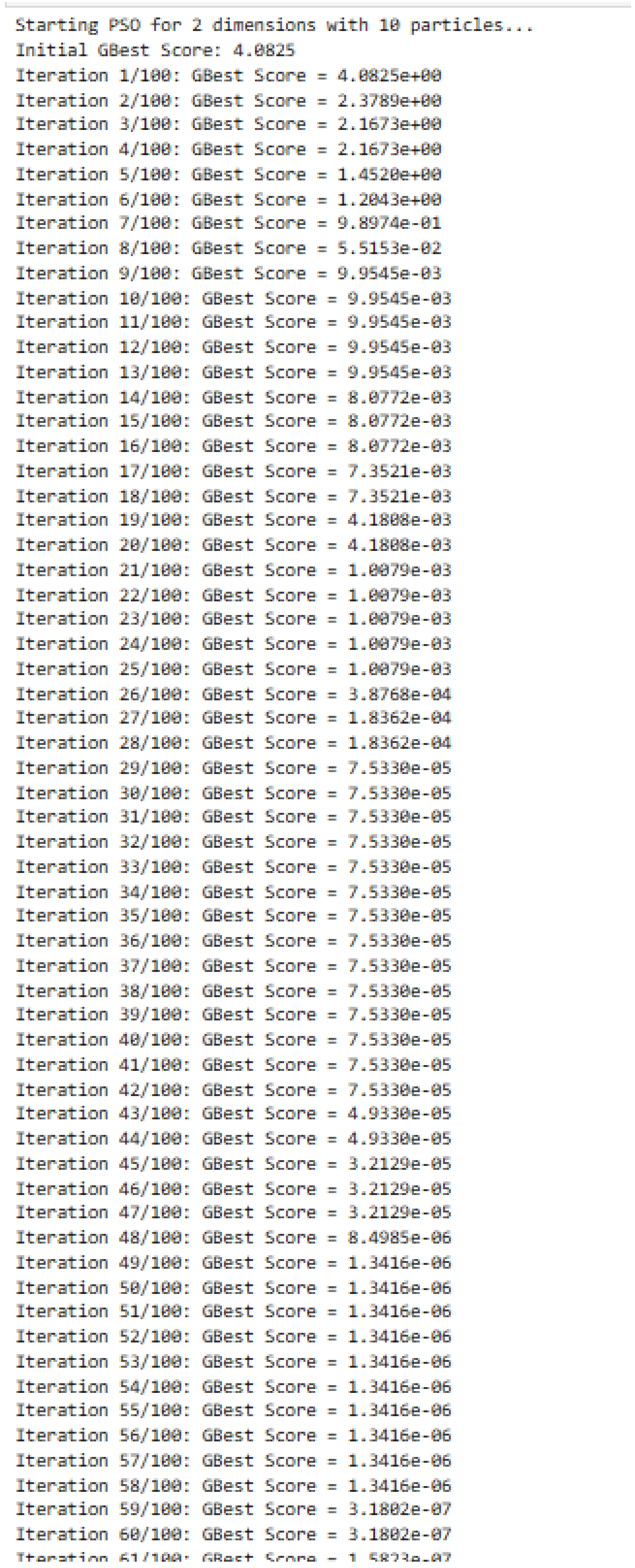




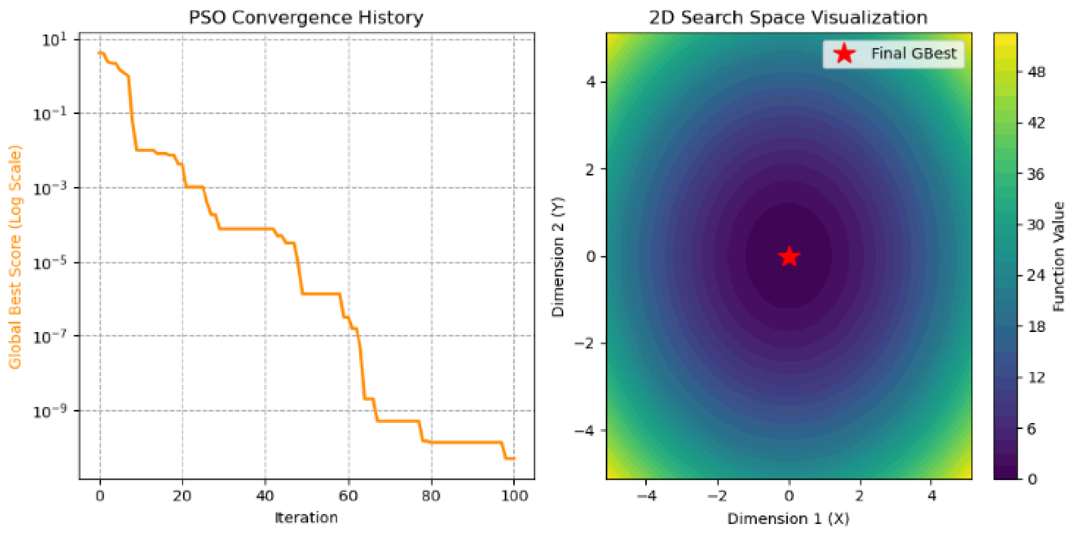
Particle Swarm optimization

```
import numpy as np
import random
import matplotlib.pyplot as plt
random.seed(42)
np.random.seed(42)
def sphere_function(position):
    """
    The classic Sphere function (f(x) = sum(x^2)), used for minimization.
    The global minimum is f(x)=0 at x=[0, 0, ..., 0].
    """
    return np.sum(position**2)
def pso_optimizer(
    objective_func,
    num_particles=30,
    dimensions=2,
    search_range=(-10, 10),
    max_iterations=100,
    w=0.729,
    c1=1.4944,
    c2=1.4944
):
    """
    Particle Swarm Optimization (PSO) algorithm for continuous optimization.

    Args:
        objective_func (callable): The function to minimize.
        num_particles (int): Number of particles (S).
        dimensions (int): Dimensionality of the search space.
        search_range (tuple): (min, max) bounds for particle positions.
        max_iterations (int): Maximum number of generations.
        w (float): Inertia weight.
        c1 (float): Cognitive constant.
        c2 (float): Social constant.
    """
    min_bound, max_bound = search_range
    positions = np.random.uniform(min_bound, max_bound, (num_particles, dimensions))
    velocities = np.random.uniform(-1, 1, (num_particles, dimensions))
    pbest_positions = positions.copy()
    pbest_scores = np.array([objective_func(p) for p in positions])
    gbest_index = np.argmin(pbest_scores)
    gbest_position = pbest_positions[gbest_index].copy()
    gbest_score = pbest_scores[gbest_index]
```

```python
    min_bound, max_bound = search_range
    positions = np.random.uniform(min_bound, max_bound, (num_particles, dimensions))
    velocities = np.random.uniform(-1, 1, (num_particles, dimensions))
    pbest_positions = positions.copy()
    pbest_scores = np.array([objective_func(p) for p in positions])
    gbest_index = np.argmin(pbest_scores)
    gbest_position = pbest_positions[gbest_index].copy()
    gbest_score = pbest_scores[gbest_index]
    history = [(gbest_score, gbest_position)]
    print(f"Starting PSO for {dimensions} dimensions with {num_particles} particles...")
    print(f"Initial GBest Score: {gbest_score:.4f}")
    for iteration in range(max_iterations):
        for i in range(num_particles):
            current_score = objective_func(positions[i])
            if current_score < pbest_scores[i]:
                pbest_scores[i] = current_score
                pbest_positions[i] = positions[i].copy()
        current_gbest_index = np.argmin(pbest_scores)
        current_gbest_score = pbest_scores[current_gbest_index]
        if current_gbest_score < gbest_score:
            gbest_score = current_gbest_score
            gbest_position = pbest_positions[current_gbest_index].copy()
        history.append((gbest_score, gbest_position.copy()))
        r1 = np.random.rand(num_particles, dimensions)
        r2 = np.random.rand(num_particles, dimensions)
        inertia_comp = w * velocities
        cognitive_comp = c1 * r1 * (pbest_positions - positions)
        social_comp = c2 * r2 * (gbest_position - positions)
        velocities = inertia_comp + cognitive_comp + social_comp
        positions = positions + velocities
        positions = np.clip(positions, min_bound, max_bound)
        print(f"Iteration {iteration+1}/{max_iterations}: GBest Score = {gbest_score:.4e}")
    return gbest_position, gbest_score, history
def run_pso_example():
    search_range = (-5.12, 5.12)
    max_iter = 100
    best_position, best_score, history = pso_optimizer(
        objective_func=sphere_function,
        num_particles=10,
        dimensions=2,
        search_range=search_range,
        max_iterations=max_iter
    )
    print("\n--- Results ---")
    print(f"Objective Function: Sphere Function")
    print(f"Best Position Found: {best_position}")
    return gbest_position, gbest_score, history
def run_pso_example():
    search_range = (-5.12, 5.12)
    max_iter = 100
    best_position, best_score, history = pso_optimizer(
        objective_func=sphere_function,
        num_particles=10,
        dimensions=2,
        search_range=search_range,
        max_iterations=max_iter
    )
    print("\n--- Results ---")
    print(f"Objective Function: Sphere Function")
    print(f"Best Position Found: {best_position}")
    print(f"Minimum Score (Fitness): {best_score:.6e}")
    scores = [item[0] for item in history]
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.plot(range(len(scores)), scores, color='darkorange', linewidth=2)
    plt.title('PSO Convergence History')
    plt.xlabel('Iteration')
    plt.ylabel('Global Best Score (Log Scale)', color='darkorange')
    plt.yscale('log')
    plt.grid(True, which="both", ls="--")
    if history and len(history[0][1]) == 2:
        plt.subplot(1, 2, 2)
        x = np.linspace(search_range[0], search_range[1], 100)
        y = np.linspace(search_range[0], search_range[1], 100)
        X, Y = np.meshgrid(x, y)
        Z = np.array([[sphere_function(np.array([X[i, j], Y[i, j]])) for j in range(100)] for i in range(100)])
        plt.contourf(X, Y, Z, levels=50, cmap='viridis')
        plt.colorbar(label='Function Value')
        plt.plot(best_position[0], best_position[1], 'r*', markersize=15, label='Final GBest')
        plt.title('2D Search Space Visualization')
        plt.xlabel('Dimension 1 (X)')
        plt.ylabel('Dimension 2 (Y)')
        plt.legend()
    plt.tight_layout()
    plt.show()
if __name__ == '__main__':
    run_pso_example()
```



```
Starting PSO for 2 dimensions with 10 particles...
Initial GBest Score: 4.0825
Iteration 1/100: GBest Score = 4.0825e+00
Iteration 2/100: GBest Score = 2.3789e+00
Iteration 3/100: GBest Score = 2.1673e+00
Iteration 4/100: GBest Score = 2.1673e+00
Iteration 5/100: GBest Score = 1.4520e+00
Iteration 6/100: GBest Score = 1.2043e+00
Iteration 7/100: GBest Score = 9.8974e-01
Iteration 8/100: GBest Score = 5.5153e-02
Iteration 9/100: GBest Score = 9.9545e-03
Iteration 10/100: GBest Score = 9.9545e-03
Iteration 11/100: GBest Score = 9.9545e-03
Iteration 12/100: GBest Score = 9.9545e-03
Iteration 13/100: GBest Score = 9.9545e-03
Iteration 14/100: GBest Score = 8.0772e-03
Iteration 15/100: GBest Score = 8.0772e-03
Iteration 16/100: GBest Score = 8.0772e-03
Iteration 17/100: GBest Score = 7.3521e-03
Iteration 18/100: GBest Score = 7.3521e-03
Iteration 19/100: GBest Score = 4.1808e-03
Iteration 20/100: GBest Score = 4.1808e-03
Iteration 21/100: GBest Score = 1.0079e-03
Iteration 22/100: GBest Score = 1.0079e-03
Iteration 23/100: GBest Score = 1.0079e-03
Iteration 24/100: GBest Score = 1.0079e-03
Iteration 25/100: GBest Score = 1.0079e-03
Iteration 26/100: GBest Score = 3.8768e-04
Iteration 27/100: GBest Score = 1.8362e-04
Iteration 28/100: GBest Score = 1.8362e-04
Iteration 29/100: GBest Score = 7.5330e-05
Iteration 30/100: GBest Score = 7.5330e-05
Iteration 31/100: GBest Score = 7.5330e-05
Iteration 32/100: GBest Score = 7.5330e-05
Iteration 33/100: GBest Score = 7.5330e-05
Iteration 34/100: GBest Score = 7.5330e-05
Iteration 35/100: GBest Score = 7.5330e-05
Iteration 36/100: GBest Score = 7.5330e-05
Iteration 37/100: GBest Score = 7.5330e-05
Iteration 38/100: GBest Score = 7.5330e-05
Iteration 39/100: GBest Score = 7.5330e-05
Iteration 40/100: GBest Score = 7.5330e-05
Iteration 41/100: GBest Score = 7.5330e-05
Iteration 42/100: GBest Score = 7.5330e-05
Iteration 43/100: GBest Score = 4.9330e-05
Iteration 44/100: GBest Score = 4.9330e-05
Iteration 45/100: GBest Score = 3.2129e-05
Iteration 46/100: GBest Score = 3.2129e-05
Iteration 47/100: GBest Score = 3.2129e-05
Iteration 48/100: GBest Score = 8.4985e-06
Iteration 49/100: GBest Score = 1.3416e-06
Iteration 50/100: GBest Score = 1.3416e-06
Iteration 51/100: GBest Score = 1.3416e-06
Iteration 52/100: GBest Score = 1.3416e-06
Iteration 53/100: GBest Score = 1.3416e-06
Iteration 54/100: GBest Score = 1.3416e-06
Iteration 55/100: GBest Score = 1.3416e-06
Iteration 56/100: GBest Score = 1.3416e-06
Iteration 57/100: GBest Score = 1.3416e-06
Iteration 58/100: GBest Score = 1.3416e-06
Iteration 59/100: GBest Score = 3.1802e-07
Iteration 60/100: GBest Score = 3.1802e-07
Iteration 61/100: GBest Score = 1.5823e-07
Iteration 72/100: GBest Score = 5.0642e-10
Iteration 73/100: GBest Score = 5.0642e-10
Iteration 74/100: GBest Score = 5.0642e-10
Iteration 75/100: GBest Score = 5.0642e-10
Iteration 76/100: GBest Score = 5.0642e-10
Iteration 77/100: GBest Score = 5.0642e-10
Iteration 78/100: GBest Score = 1.4724e-10
Iteration 79/100: GBest Score = 1.4724e-10
Iteration 80/100: GBest Score = 1.3409e-10
Iteration 81/100: GBest Score = 1.3409e-10
Iteration 82/100: GBest Score = 1.3409e-10
Iteration 83/100: GBest Score = 1.3409e-10
Iteration 84/100: GBest Score = 1.3409e-10
Iteration 85/100: GBest Score = 1.3409e-10
Iteration 86/100: GBest Score = 1.3409e-10
Iteration 87/100: GBest Score = 1.3409e-10
Iteration 88/100: GBest Score = 1.3409e-10
Iteration 89/100: GBest Score = 1.3409e-10
Iteration 90/100: GBest Score = 1.3409e-10
Iteration 91/100: GBest Score = 1.3409e-10
Iteration 92/100: GBest Score = 1.3409e-10
Iteration 93/100: GBest Score = 1.3409e-10
Iteration 94/100: GBest Score = 1.3409e-10
Iteration 95/100: GBest Score = 1.3409e-10
Iteration 96/100: GBest Score = 1.3409e-10
Iteration 97/100: GBest Score = 1.3409e-10
Iteration 98/100: GBest Score = 4.9838e-11
Iteration 99/100: GBest Score = 4.9838e-11
Iteration 100/100: GBest Score = 4.9838e-11

--- Results ---
Objective Function: Sphere Function
Best Position Found: [ 6.78638486e-06 -1.94488851e-06]
Minimum Score (Fitness): 4.983761e-11
```


PSO Convergence History
Global Best Score (Log Scale)
10^{1}
10^{-1}
10^{-3}
10^{-5}
10^{-7}
10^{-9}
Iteration
2D Search Space Visualization
Final GBest
Dimension 2 (Y)
Dimension 1 (X)
Function Value