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

# 1. Define the Problem: Create a set of cities with their coordinates
cities = np.array([
    [0, 0],  # City 0
    [1, 5],  # City 1
    [5, 1],  # City 2
    [6, 4],  # City 3
    [7, 8],  # City 4
])

# Calculate the distance matrix between each pair of cities
def calculate_distances(cities):
    num_cities = len(cities)
    distances = np.zeros((num_cities, num_cities))

    for i in range(num_cities):
        for j in range(num_cities):
            distances[i][j] = np.linalg.norm(cities[i] - cities[j])

    return distances

distances = calculate_distances(cities)

# 2. Initialize Parameters
num_ants = 10
num_cities = len(cities)
alpha = 1.0  # Influence of pheromone
beta = 5.0   # Influence of heuristic (inverse distance)
rho = 0.5    # Evaporation rate
num_iterations = 100
initial_pheromone = 1.0

# Pheromone matrix initialization
pheromone = np.ones((num_cities, num_cities)) * initial_pheromone

# 3. Heuristic information (Inverse of distance)
def heuristic(distances):
    with np.errstate(divide='ignore'):  # Ignore division by zero
        return 1 / distances

eta = heuristic(distances)

# 4. Choose next city probabilistically based on pheromone and heuristic info
def choose_next_city(pheromone, eta, visited):
    probs = []
    for j in range(num_cities):
        if j not in visited:
            pheromone_ij = pheromone[visited[-1], j] ** alpha
            heuristic_ij = eta[visited[-1], j] ** beta
            probs.append(pheromone_ij * heuristic_ij)
        else:
            probs.append(0)
    probs = np.array(probs)
    return np.random.choice(range(num_cities), p=probs / probs.sum())

# Construct solution for a single ant
def construct_solution(pheromone, eta):
    tour = [np.random.randint(0, num_cities)]
    while len(tour) < num_cities:
        next_city = choose_next_city(pheromone, eta, tour)
        tour.append(next_city)
    return tour

# 5. Update pheromones after all ants have constructed their tours
def update_pheromones(pheromone, all_tours, distances, best_tour):
```

```python
    pheromone *= (1 - rho)  # Evaporate pheromones

    # Add pheromones for each ant's tour
    for tour in all_tours:
        tour_length = sum([distances[tour[i], tour[i + 1]] for i in range(-1, num_cities - 1)])
        for i in range(-1, num_cities - 1):
            pheromone[tour[i], tour[i + 1]] += 1.0 / tour_length

    # Increase pheromones on the best tour
    best_length = sum([distances[best_tour[i], best_tour[i + 1]] for i in range(-1, num_cities - 1)])
    for i in range(-1, num_cities - 1):
        pheromone[best_tour[i], best_tour[i + 1]] += 1.0 / best_length

# 6. Main ACO Loop: Iterate over multiple iterations to find the best solution
def run_aco(distances, num_iterations):
    pheromone = np.ones((num_cities, num_cities)) * initial_pheromone
    best_tour = None
    best_length = float('inf')

    for iteration in range(num_iterations):
        all_tours = [construct_solution(pheromone, eta) for _ in range(num_ants)]
        all_lengths = [sum([distances[tour[i], tour[i + 1]] for i in range(-1, num_cities - 1)]) for tour in all_tours]

        current_best_length = min(all_lengths)
        current_best_tour = all_tours[all_lengths.index(current_best_length)]

        if current_best_length < best_length:
            best_length = current_best_length
            best_tour = current_best_tour

        update_pheromones(pheromone, all_tours, distances, best_tour)

        print(f"Iteration {iteration + 1}, Best Length: {best_length}")

    return best_tour, best_length

# Run the ACO algorithm
best_tour, best_length = run_aco(distances, num_iterations)

# 7. Output the Best Solution
print(f"Best Tour: {best_tour}")
print(f"Best Tour Length: {best_length}")

# 8. Plot the Best Route
def plot_route(cities, best_tour):
    plt.figure(figsize=(8, 6))
    for i in range(len(cities)):
        plt.scatter(cities[i][0], cities[i][1], color='red')
        plt.text(cities[i][0], cities[i][1], f"City {i}", fontsize=12)

    # Plot the tour as lines connecting the cities
    tour_cities = np.array([cities[i] for i in best_tour] + [cities[best_tour[0]]])  # Complete the loop by returning to the start
    plt.plot(tour_cities[:, 0], tour_cities[:, 1], linestyle='-', marker='o', color='blue')

    plt.title(f"Best Tour (Length: {best_length})")
    plt.xlabel("X Coordinate")
    plt.ylabel("Y Coordinate")
    plt.grid(True)
    plt.show()

# Call the plot function
plot_route(cities, best_tour)
```

```
Iteration 1, Best Length: 24.191626245470978
Iteration 2, Best Length: 24.191626245470978
Iteration 3, Best Length: 24.191626245470978
Iteration 4, Best Length: 24.191626245470978
Iteration 5, Best Length: 24.191626245470978
Iteration 6, Best Length: 24.191626245470978
Iteration 7, Best Length: 24.191626245470978
Iteration 8, Best Length: 24.191626245470978
Iteration 9, Best Length: 24.191626245470978
Iteration 10, Best Length: 24.191626245470978
Iteration 11, Best Length: 24.191626245470978
Iteration 12, Best Length: 24.191626245470978
Iteration 13, Best Length: 24.191626245470978
Iteration 14, Best Length: 24.191626245470978
Iteration 15, Best Length: 24.191626245470978
Iteration 16, Best Length: 24.191626245470978
Iteration 17, Best Length: 24.191626245470978
Iteration 18, Best Length: 24.191626245470978
Iteration 19, Best Length: 24.191626245470978
Iteration 20, Best Length: 24.191626245470978
Iteration 21, Best Length: 24.191626245470978
Iteration 22, Best Length: 24.191626245470978
Iteration 23, Best Length: 24.191626245470978
Iteration 24, Best Length: 24.191626245470978
Iteration 25, Best Length: 24.191626245470978
Iteration 26, Best Length: 24.191626245470978
Iteration 27, Best Length: 24.191626245470978
Iteration 28, Best Length: 24.191626245470978
Iteration 29, Best Length: 24.191626245470978
Iteration 30, Best Length: 24.191626245470978
Iteration 31, Best Length: 24.191626245470978
Iteration 32, Best Length: 24.191626245470978
Iteration 33, Best Length: 24.191626245470978
Iteration 34, Best Length: 24.191626245470978
Iteration 35, Best Length: 24.191626245470978
Iteration 36, Best Length: 24.191626245470978
Iteration 37, Best Length: 24.191626245470978
Iteration 38, Best Length: 24.191626245470978
Iteration 39, Best Length: 24.191626245470978
Iteration 40, Best Length: 24.191626245470978
Iteration 41, Best Length: 24.191626245470978
Iteration 42, Best Length: 24.191626245470978
Iteration 43, Best Length: 24.191626245470978
Iteration 44, Best Length: 24.191626245470978
Iteration 45, Best Length: 24.191626245470978
Iteration 46, Best Length: 24.191626245470978
Iteration 47, Best Length: 24.191626245470978
Iteration 48, Best Length: 24.191626245470978
Iteration 49, Best Length: 24.191626245470978
Iteration 50, Best Length: 24.191626245470978
Iteration 51, Best Length: 24.191626245470978
Iteration 52, Best Length: 24.191626245470978
Iteration 53, Best Length: 24.191626245470978
Iteration 54, Best Length: 24.191626245470978
Iteration 55, Best Length: 24.191626245470978
Iteration 56, Best Length: 24.191626245470978
Iteration 57, Best Length: 24.191626245470978
Iteration 58, Best Length: 24.191626245470978
Iteration 59, Best Length: 24.191626245470978
Iteration 60, Best Length: 24.191626245470978
Iteration 61, Best Length: 24.191626245470978
Iteration 62, Best Length: 24.191626245470978
Iteration 63, Best Length: 24.191626245470978
Iteration 64, Best Length: 24.191626245470978
Iteration 65, Best Length: 24.191626245470978
Iteration 66, Best Length: 24.191626245470978
Iteration 67, Best Length: 24.191626245470978
Iteration 68, Best Length: 24.191626245470978
Iteration 69, Best Length: 24.191626245470978
Iteration 70, Best Length: 24.191626245470978
Iteration 71, Best Length: 24.191626245470978
Iteration 72, Best Length: 24.191626245470978
Iteration 73, Best Length: 24.191626245470978
Iteration 74, Best Length: 24.191626245470978
Iteration 75, Best Length: 24.191626245470978
Iteration 76, Best Length: 24.191626245470978
Iteration 77, Best Length: 24.191626245470978
Iteration 78, Best Length: 24.191626245470978
Iteration 79, Best Length: 24.191626245470978
Iteration 80, Best Length: 24.191626245470978
Iteration 81, Best Length: 24.191626245470978
Iteration 82, Best Length: 24.191626245470978
Iteration 83, Best Length: 24.191626245470978
Iteration 84, Best Length: 24.191626245470978
Iteration 85, Best Length: 24.191626245470978
Iteration 86, Best Length: 24.191626245470978
Iteration 87, Best Length: 24.191626245470978
Iteration 88, Best Length: 24.191626245470978
Iteration 89, Best Length: 24.191626245470978
Iteration 90, Best Length: 24.191626245470978
Iteration 91, Best Length: 24.191626245470978
Iteration 92, Best Length: 24.191626245470978
```

```
Iteration 93, Best Length: 24.191626245470978
Iteration 94, Best Length: 24.191626245470978
Iteration 95, Best Length: 24.191626245470978
Iteration 96, Best Length: 24.191626245470978
Iteration 97, Best Length: 24.191626245470978
Iteration 98, Best Length: 24.191626245470978
Iteration 99, Best Length: 24.191626245470978
Iteration 100, Best Length: 24.191626245470978
Best Tour: [1, 0, 2, 3, 4]
Best Tour Length: 24.191626245470978
```