```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import euclidean

# Define the problem: coordinates of cities
cities = np.array([
    [0, 0], [2, 3], [5, 2], [6, 6], [8, 3],
    [1, 5], [4, 7], [7, 8], [9, 5], [3, 1]
])
num_cities = len(cities)

# Parameters
num_ants = 10
num_iterations = 100
alpha = 1  # Importance of pheromone
beta = 2  # Importance of heuristic (1/distance)
rho = 0.5  # Pheromone evaporation rate
initial_pheromone = 1.0

# Distance matrix
distances = np.array([[euclidean(cities[i], cities[j]) for j in
range(num_cities)] for i in range(num_cities)])

# Heuristic information (1 / distance), avoiding division by zero
heuristics = np.zeros_like(distances)  # Initialize as zero
for i in range(num_cities):
    for j in range(num_cities):
        if i != j:
            heuristics[i, j] = 1 / distances[i, j]  # Only calculate for non-
diagonal elements
        else:
            heuristics[i, j] = 0  # Set diagonal to zero (no heuristic for the
same city)

# Pheromone matrix
pheromones = np.full((num_cities, num_cities), initial_pheromone)

# Ant class
class Ant:
    def __init__(self):
        self.visited = []
        self.total_distance = 0

    def choose_next_city(self, current_city):
        probabilities = []
        for city in range(num_cities):
            if city not in self.visited:
                pheromone = pheromones[current_city][city] ** alpha
                heuristic = heuristics[current_city][city] ** beta
                probabilities.append(pheromone * heuristic)
            else:
                probabilities.append(0)

        # Convert to numpy array for easier manipulation
        probabilities = np.array(probabilities)

        # Check if all probabilities are zero, avoid NaN
        if probabilities.sum() == 0:
            unvisited_cities = [city for city in range(num_cities) if city not
in self.visited]
            return np.random.choice(unvisited_cities)

        # Normalize probabilities to make sure they sum to 1
        probabilities /= probabilities.sum()
        return np.random.choice(range(num_cities), p=probabilities)

    def complete_tour(self):
        # Complete the tour by returning to the start city
        self.total_distance += distances[self.visited[-1]][self.visited[0]]
        self.visited.append(self.visited[0])

# ACO implementation
def ant_colony_optimization():
    global pheromones
    best_solution = None
    best_distance = float('inf')
    best_history = []

    for iteration in range(num_iterations):
        all_ants = []

        # Each ant constructs a solution
        for _ in range(num_ants):
            ant = Ant()
            current_city = np.random.randint(0, num_cities)  # Start at a random
city
            ant.visited.append(current_city)

            while len(ant.visited) < num_cities:
                next_city = ant.choose_next_city(current_city)
                ant.total_distance += distances[current_city][next_city]
                ant.visited.append(next_city)
                current_city = next_city

            # Complete the tour
            ant.complete_tour()
            all_ants.append(ant)

            # Update global best
            if ant.total_distance < best_distance:
                best_distance = ant.total_distance
                best_solution = ant.visited

        # Update pheromone trails
        pheromones *= (1 - rho)  # Evaporation
        for ant in all_ants:
            for i in range(num_cities):
                from_city = ant.visited[i]
                to_city = ant.visited[i + 1] if i + 1 < len(ant.visited) else
ant.visited[0]
                pheromones[from_city][to_city] += 1 / ant.total_distance

        # Track the best distance in history
        best_history.append(best_distance)
        print(f"Iteration {iteration + 1}, Best Distance: {best_distance}")

    return best_solution, best_distance, best_history

# Run the ACO algorithm
best_solution, best_distance, best_history = ant_colony_optimization()

# Print the results
print("\nBest route found:", best_solution)
print("Shortest distance:", best_distance)

# Plot the best route
route_cities = cities[best_solution]
plt.figure(figsize=(8, 6))
plt.plot(route_cities[:, 0], route_cities[:, 1], 'o-', label='Best Route')
plt.title("Best Route Found by ACO")
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.legend()
plt.grid()
plt.show()

# Plot the convergence
plt.figure()
plt.plot(best_history)
plt.title("ACO Convergence")
plt.xlabel("Iteration")
plt.ylabel("Shortest Distance")
plt.grid()
plt.show()
```