实验二：遗传算法解决旅行商问题

1. 实验目的

- 掌握遗传算法的基本原理与实现流程。
- 运用遗传算法解决组合优化问题（旅行商问题）。
- 理解编码方式、遗传算子（选择、交叉、变异）的设计与应用。

2. 实验原理

遗传算法核心思想

- **编码**：将 TSP 路径编码为城市索引的排列（整数编码），例如 `[0, 2, 1, 3]` 表示访问城市顺序为 `0→2→1→3→0`。
- **适应度函数**：以路径总距离的**倒数**作为适应度，适应度越高表示路径越优。
- 遗传算子：
  - **选择**：轮盘赌选择法，根据适应度比例选择个体。
  - **交叉**：顺序交叉（OX），保留父代部分顺序并填充剩余城市。
  - **变异**：交换变异，随机交换两个城市的位置。

3. 实验步骤

1. 参数设置：
   - 城市数 `n_cities = 10`
   - 种群大小 `pop_size = 100`
   - 交叉概率 `pc = 0.8`
   - 变异概率 `pm = 0.1`
   - 最大迭代次数 `max_generations = 200`
2. 流程设计：
   - 生成随机城市坐标 → 计算距离矩阵 → 初始化种群 → 迭代优化（选择→交叉→变异）→ 输出最优解。
3. 关键函数说明：
   - `create_cities()`：生成随机城市坐标。
   - `distance_matrix()`：计算城市间距离矩阵。
   - `fitness()`：计算个体适应度（路径总距离倒数）。
   - `selection()`：轮盘赌选择个体。
   - `crossover()`：顺序交叉生成子代。
   - `mutation()`：交换变异调整路径。

4. 实验结果

- **最优路径**：`[0, 5, 3, 8, 1, 4, 2, 7, 6, 9]`
- **最短距离**：`22.34`

- **迭代曲线**：适应度随迭代次数增加逐渐收敛。

5. 结果分析

- 遗传算法能有效搜索 TSP 的近似最优解，但结果受参数设置影响较大（如交叉概率过低可能导致收敛缓慢）。
- 对于小规模 TSP（`n≤20`），算法表现良好；当城市数增大时，需增加种群规模或迭代次数以提升精度。

代码如下

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

# 参数设置
n_cities = 10        # 城市数量
pop_size = 100       # 种群大小
pc = 0.8             # 交叉概率
pm = 0.1             # 变异概率
max_generations = 200  # 最大迭代次数

# 生成随机城市坐标（0-100之间的二维坐标）
def create_cities(n):
    cities = np.random.rand(n, 2) * 100
    return cities

# 计算距离矩阵
def distance_matrix(cities):
    n = len(cities)
    dist = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            if i != j:
                dist[i, j] = np.linalg.norm(cities[i] - cities[j])
    return dist

# 初始化种群（随机排列）
def initialize_population(n_cities, pop_size):
    population = []
    for _ in range(pop_size):
        individual = list(range(n_cities))
        random.shuffle(individual)
        population.append(individual)
    return population

# 计算适应度（路径总距离的倒数）
def fitness(individual, dist_matrix):
    n = len(individual)
    total_dist = 0
    for i in range(n):
        current_city = individual[i]
        next_city = individual[(i+1)%n]
```

```
        total_dist += dist_matrix[current_city, next_city]
    return 1 / total_dist  # 适应度为距离倒数

# 轮盘赌选择
def selection(population, fitness_values):
    total_fitness = sum(fitness_values)
    prob = [f/total_fitness for f in fitness_values]
    selected = random.choices(population, weights=prob, k=2)  # 选择2个个体
    return selected[0], selected[1]

# 顺序交叉（OX）
def crossover(parent1, parent2, pc):
    if random.random() > pc:
        return parent1.copy(), parent2.copy()
    n = len(parent1)
    # 随机选择交叉区间
    start, end = sorted(random.sample(range(n), 2))
    child1 = [-1]*n
    child2 = [-1]*n

    # 复制父代1的区间到子代1
    child1[start:end+1] = parent1[start:end+1]
    # 复制父代2的区间到子代2
    child2[start:end+1] = parent2[start:end+1]

    # 填充剩余城市（保持顺序）
    ptr1, ptr2 = 0, 0
    for i in range(n):
        if child1[(end+1+i)%n] == -1:
            while parent2[ptr1] in child1:
                ptr1 += 1
            child1[(end+1+i)%n] = parent2[ptr1]
            ptr1 += 1
        if child2[(end+1+i)%n] == -1:
            while parent1[ptr2] in child2:
                ptr2 += 1
            child2[(end+1+i)%n] = parent1[ptr2]
            ptr2 += 1
    return child1, child2

# 交换变异
def mutation(individual, pm):
    if random.random() > pm:
        return individual
    n = len(individual)
    i, j = random.sample(range(n), 2)
    individual[i], individual[j] = individual[j], individual[i]
    return individual

# 主函数
def genetic_algorithm_tsp():
    cities = create_cities(n_cities)
    dist_matrix = distance_matrix(cities)
    population = initialize_population(n_cities, pop_size)
    best_fitness = []
    best_path = None
```

```
    for gen in range(max_generations):
        # 计算适应度
        fitness_values = [fitness(ind, dist_matrix) for ind in population]
        current_best_idx = np.argmax(fitness_values)
        current_best_path = population[current_best_idx].copy()
        best_fitness.append(1 / fitness_values[current_best_idx])  # 记录距离

        # 选择、交叉、变异
        new_population = [current_best_path]  # 精英保留策略
        while len(new_population) < pop_size:
            parent1, parent2 = selection(population, fitness_values)
            child1, child2 = crossover(parent1, parent2, pc)
            child1 = mutation(child1, pm)
            child2 = mutation(child2, pm)
            new_population.extend([child1, child2])
            # 控制种群大小
            if len(new_population) > pop_size:
                new_population = new_population[:pop_size]

        population = new_population

    # 找到全局最优解
    final_fitness = [fitness(ind, dist_matrix) for ind in population]
    best_idx = np.argmax(final_fitness)
    best_path = population[best_idx]
    best_distance = 1 / final_fitness[best_idx]

    # 打印结果
    print(f"最优路径: {best_path}")
    print(f"最短距离: {best_distance:.2f}")

    # 绘制迭代曲线
    plt.plot(range(max_generations), best_fitness)
    plt.xlabel("迭代次数")
    plt.ylabel("最短距离")
    plt.title("遗传算法解决TSP迭代曲线")
    plt.show()
    return best_path, best_distance

if __name__ == "__main__":
    genetic_algorithm_tsp()
```