《数值计算与最优化技术》实验报告

年级、专业、班级	21 计卓 2 班	姓名	文红兵
实验题目	遗传算法的应用		
实验时间	2023 年 5 月 21 日	实验地点	DS3305
实验成绩		实验性质	□验证性 ■设计性 □综合性

教师评价：

□算法/实验过程正确；□源程序/实验内容提交 □程序结构/实验步骤合理；

□实验结果正确； □语法、语义正确； □报告规范；

其他：

评价教师签名：

一、实验目的

理解遗传算法的基本思想，并应用于求解实际问题。

二、实验项目内容

1.请利用遗传算法解决以下的旅行商问题：

一个旅行者需要到国内的 10 个城市旅行，各城市的坐标见 cities.csv 文档。请设计一个合理的线路使旅行者所行的路程（坐标间距离，非球面距离）之和最小。注意：每个城市只能访问一次，且最后要回到原来出来的城市。

要求：输出适应度函数的进化曲线，展示线路的动态变化过程及最终的线路图。

2. 用遗传算法求取以下目标函数的极大值。

$$X \cdot cos2\pi Y + Y \cdot \sin 2\pi \cdot X$$

要求：种群初始随机，范围分别为[-2, 2]、[-2, 2]，染色体长度为 20，交叉概率 0.7，变异概率为 0.01。

报告创建时间：

注意：实验语言不限，只提交本电子文档，注意本文件末尾的文件命名要求；源程序一节请先给出你的算法思路，并用代码备注的方式说明算法的关键点；实验结果一节需要提供调试过程中的中间结果及测试最终结果的截图并给出结果分析，若在调试过程中有错，请给出错误原因分析。

三、实验过程或算法（源程序）

问题一

1、TSP 问题是一个经典的组合优化问题，其目标是找到一条路径，使得旅行商可以依次访问一系列城市，且每个城市只能访问一次，路径的总长度最小。

2、该算法的主要流程如下：

- 初始化种群: 随机生成一些个体来构成初始种群。
- 评估: 用适应度函数计算每个个体的适应度值。
- 选择: 根据每个个体的适应度值，使用轮盘赌选择算法选择一些优秀的个体。
- 交叉: 对于选择出来的个体，以一定的概率进行交叉操作，生成新的个体。
- 变异: 对于新生成的个体，以一定的概率进行变异操作，生成更多的个体。
- 进化: 将新生成的个体加入到种群中，再次进行选择、交叉和变异操作。
- 终止: 当达到指定的迭代次数或者找到满足要求的最优解时，终止算法。

3、在实现中，使用了一个个体类(Individual)和一个种群类(Population)。个体类表示一个候选解，其中包括经过的城市顺序和对应的适应度值。种群类包括一组个体和相关的操作函数，如适应度函数、选择函数和进化函数等。

4、具体来说，适应度函数(fun)用于计算一个个体的适应度值，它基于每个城市之间的距离计算路径的总长度，最后用大常数 M 减去路径长度，将 TSP 问题转化为最大化适应度值的问题。

5、选择函数(roulette_wheel_selection 和 selection)使用轮盘赌选择算法，根据适应度值的大小选择优秀的个体。交叉函数(crossover)和变异函数(mutation)用于产生新的个体，并保持种群的多样性。进化函数(evolve)是整个算法的核心，它基于轮盘赌选择算法和

交叉、变异操作，进行种群的进化，直到达到指定的迭代次数或者找到最优解为止。

6、最后，输出了找到的最优解和迭代过程中路径长度的变化情况，以便于结果的可视化展示

```
import random
import bisect
import itertools
import math
import matplotlib.pyplot as plt

M = 100000000
x_axis = []
y_axis = []

# 定义适应度函数,寻找最大值，必须是非负数
def fun(x):
    f = 0
    for i in range(1, len(x)):
        f += math.sqrt((x_axis[x[i-1]] - x_axis[x[i]]) ** 2 + (y_axis[x[i-1]] - y_axis[x[i]])**2)
    return -f + M


# 定义个体类
class Individual:
    def __init__(self, x):
        """
        :param x: 经过的城市顺序
        """
        self.x = x
        self.fitness = fun(self.x)


# 定义种群类
class Population:
    def __init__(self, pop_size, pop_len, pc, pm):
        """
        :param pop_size: 种群大小
        :param pop_len: 个体的染色体长度
        :param pc: 交叉的概率
        :param pm: 变异的概率
        """
        self.pop_size = pop_size
        self.individuals = [Individual(random.sample([x for x in range(1, pop_len+1)], pop_len)) for _ in
range(pop_size)]
        self.individuals.sort(key=lambda x: x.fitness)
        self.pc = pc
        self.pm = pm
        self.pop_len = pop_len
        self.evolve_route = []
        self.route = []

    # 选择操作，轮盘赌选择
    def roulette_wheel_selection(self, probabilities):
        """
        :param probabilities: 个体概率的前缀和
        :return: 随机选择的个体下标
        """
        r = random.uniform(0, 1)
        return bisect.bisect_left(probabilities, r)

    # 选择一个群体
    def selection(self, population):
        """
        在新生成的群体中选取pop_size个个体
        :param population: 新生成的群体
        :return: None
        """
        population.sort(key=lambda x: x.fitness)
        fitness_sum = sum([x.fitness for x in population])
        fitness = [x.fitness / fitness_sum for x in population]
        probabilities = list(itertools.accumulate(fitness))
        for i in range(self.pop_size):
            self.individuals[i] = population[self.roulette_wheel_selection(probabilities)]
```

```python
    # 交叉
    def crossover(self, parent1, parent2):
        """
        交叉
        :param parent1: 父母1的染色体
        :param parent2: 父母2的染色体
        :return: 父母1，  父母2
        """
        # 随机交叉点
        p1 = random.randint(1, len(parent1) - 1)
        p2 = random.randint(1, len(parent1) - 1)
        p1, p2 = min(p1, p2), max(p1, p2)

        # 进行交叉操作
        child1 = parent2[p1:p2]
        child2 = parent1[p1:p2]
        for city in parent1:
            if city not in child1:
                child1.append(city)
        for city in parent2:
            if city not in child2:
                child2.append(city)

        return child1, child2

    # 变异
    def mutation(self, child):
        p1 = random.randint(1, len(child) - 1)
        p2 = random.randint(1, len(child) - 1)
        child[p1], child[p2] = child[p2], child[p1]
        return child

    # 繁殖
    def reproduction(self, parent1, parent2):
        point = self.pop_len // 2
        if random.random() < self.pc:
            parent1, parent2 = self.crossover(parent1, parent2)
        pa1_set = set(parent1[:point])
        child = parent1[:point] + [x for x in parent2 if x not in pa1_set]
        if random.random() < self.pm:
            child = self.mutation(child)
        return child

    # 进化操作
    def evolve(self):
        population = self.individuals
        fitness_sum = sum([x.fitness for x in population])
        fitness = [x.fitness / fitness_sum for x in population]
        probabilities = list(itertools.accumulate(fitness))
        for j in range(self.pop_size):
            parent1 = self.individuals[self.roulette_wheel_selection(probabilities)].x
            parent2 = self.individuals[self.roulette_wheel_selection(probabilities)].x
            child = self.reproduction(parent1, parent2)
            population.append(Individual(child))
        self.selection(population)
```

```
        print(self.get_ans())
        a, b = self.get_ans()
        self.route = a
        self.evolve_route.append(b)
        print(a, b)

    # 获取最好的个体
    def get_ans(self):
        return self.individuals[-1].x, -(self.individuals[-1].fitness - M)

if __name__ == '__main__':
    pop_size = 5  # 种群大小
    pop_len = 10  # 个体的染色体长度
    pc = 0.8  # 交叉的概率
    pm = 0.01  # 变异的概率
    iter = 4000  # 迭代的次数
    y_axis = [0, 96.1, 94.44, 92.54, 93.37, 97.24, 96.29, 97.38, 98.12, 97.38,
95.59]
    x_axis = [0, 16.47, 16.47, 20.09, 22.39, 25.23, 17.2, 16.3, 14.05, 16.53, 21.52]
    # 初始化种群
    pop = Population(pop_size, pop_len, pc, pm)

    # 迭代 100 次
    for i in range(iter):
        pop.evolve()

    plt.figure()
    plt.scatter([x for x in range(1, iter+1)],pop.evolve_route)
    plt.xlabel('Iter')
    plt.ylabel('Distance')
    plt.title('Iter-Distance')
    plt.show()

    plt.figure()
    plt.scatter(x_axis[1:], y_axis[1:])
    plt.plot([x_axis[x] for x in pop.route], [y_axis[x] for x in pop.route])
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title('Route')
    plt.show()
```

问题二

整体算法思路如下：

1、定义个体类（Individual）来表示一个解。每个个体包含多元函数的变量（二进制编码）、实际变量值（十进制）、适应度值（函数值）。

2、定义种群类（Population）来表示一组解。种群包含多个个体、交叉概率、变异概率、自变量边界等参数。

4、初始化种群。随机生成包含一定数量个体的种群。

5、对种群进行多次迭代，每次迭代包含以下步骤：

- 选择：使用轮盘赌选择策略从种群中选择两个父代个体。
- 交叉：以一定概率对父代个体进行交叉操作，生成两个子代个体。
- 变异：以一定概率对子代个体进行变异操作。
- 更新种群：将新生成的子代个体加入种群，然后使用轮盘赌选择策略从种群中选取一定数量的个体，作为新一代种群。
- 记录当前种群中最优解。

6、绘制迭代过程中最优解的变化曲线。

```python
import random
import bisect
import itertools
import math
import matplotlib.pyplot as plt
M = 100000000
# 定义适应度函数,寻找最大值，必须是非负数
def fun(x):
    return x[0] * math.cos(2 * math.pi * x[1]) + x[1] * math.sin(2 * math.pi * x[0]) + M
# 定义个体类
class Individual:
    def __init__(self, x, lb, ub, x_size):
        """
        :param x:多元函数的x，y,z的二进制列表，相当于是个二维数组
        :param lb:多元变量的下界
        :param ub:多元变量的上界
        :param s_size:多元函数的变量个数
        """
        self.x = x
        pop_len = len(x[0])
        # 十进制数值
        self.x_dec = []
        for i in range(x_size):
            self.x_dec.append(lb[i] + (ub[i] - lb[i]) / (((1 << pop_len) - 1)) * ((int(''.join([str(_) for _ in
x[i]]), 2))))
        self.fitness = fun(self.x_dec)
# 定义种群类
class Population:
    def __init__(self, pop_size, pop_len, pc, pm, lb, ub, x_size):
        """
        :param pop_size: 种群大小
        :param pop_len: 个体的染色体长度
        :param pc: 交叉的概率
        :param pm: 变异的概率
        :param lb: 个体取值的最小值，列表
        :param ub: 个体取值的最大值，列表
        :param x_size: 多元函数变量的个数
        """
        self.pop_size = pop_size
        self.individuals = [Individual([[random.choice([0, 1]) for _ in range(pop_size)] for __ in range(x_size)], lb,
ub, x_size) for _ in range(pop_size)]
        self.individuals.sort(key=lambda x: x.fitness)
        self.pc = pc
        self.pm = pm
        self.lb = lb
        self.ub = ub
        self.pop_len = pop_len
        self.x_size = x_size
        self.evolve_route = []
```

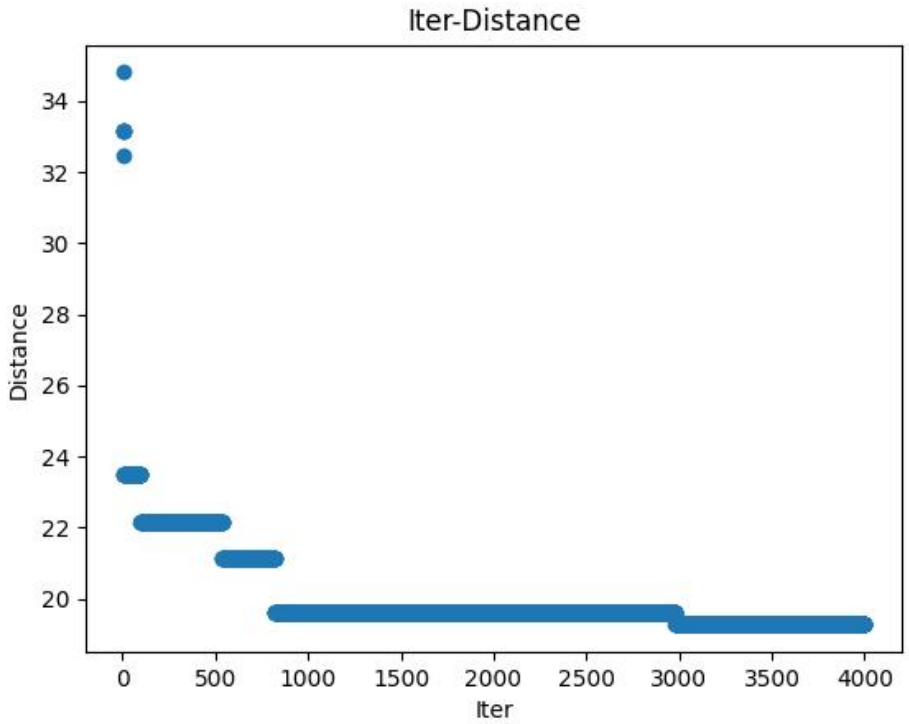
```python
    # 选择操作，轮盘赌选择
    def roulette_wheel_selection(self, probabilities):
        """
        :param probabilities: 个体概率的前缀和
        :return: 随机选择的个体下标
        """
        r = random.uniform(0, 1)
        return bisect.bisect_left(probabilities, r)

    # 选择一个群体
    def selection(self, population):
        """
        在新生成的群体中选取pop_size个个体
        :param population: 新生成的群体
        :return: None
        """
        population.sort(key=lambda x: x.fitness)
        fitness_sum = sum([x.fitness for x in population])
        fitness = [x.fitness / fitness_sum for x in population]
        probabilities = list(itertools.accumulate(fitness))
        for i in range(self.pop_size):
            self.individuals[i] = population[self.roulette_wheel_selection(probabilities)]

    # 交叉
    def crossover(self, parent1, parent2):
        """
        交叉
        :param parent1: 父母1的染色体
        :param parent2: 父母2的染色体
        :return: 父母1， 父母2
        """
        # 随机交叉点
        crossover_point = random.randint(1, len(parent1) - 1)
        # 生成新的子代染色体
        p1 = parent1[:crossover_point] + parent2[crossover_point:]
        p2 = parent2[:crossover_point] + parent1[crossover_point:]
        return p1, p2

    # 变异操作
    def mutation(self, children):
        # 随机变异点
        mutation_point = random.randint(1, len(children) - 1)
        children[mutation_point] = 0 if children[mutation_point] == 1 else 1
        return children

    # 繁殖
    def reproduction(self, parent1, parent2):
        point = self.pop_len // 2
        child = []
        for i in range(len(parent1)):
            if random.random() < self.pc:
                parent1[i], parent2[i] = self.crossover(parent1[i], parent2[i])
            child.append(parent1[i][:point] + parent2[i][point:])
        for i in range(len(parent1)):
            if random.random() < self.pm:
                child[i] = self.mutation(child[i])
        return child
```

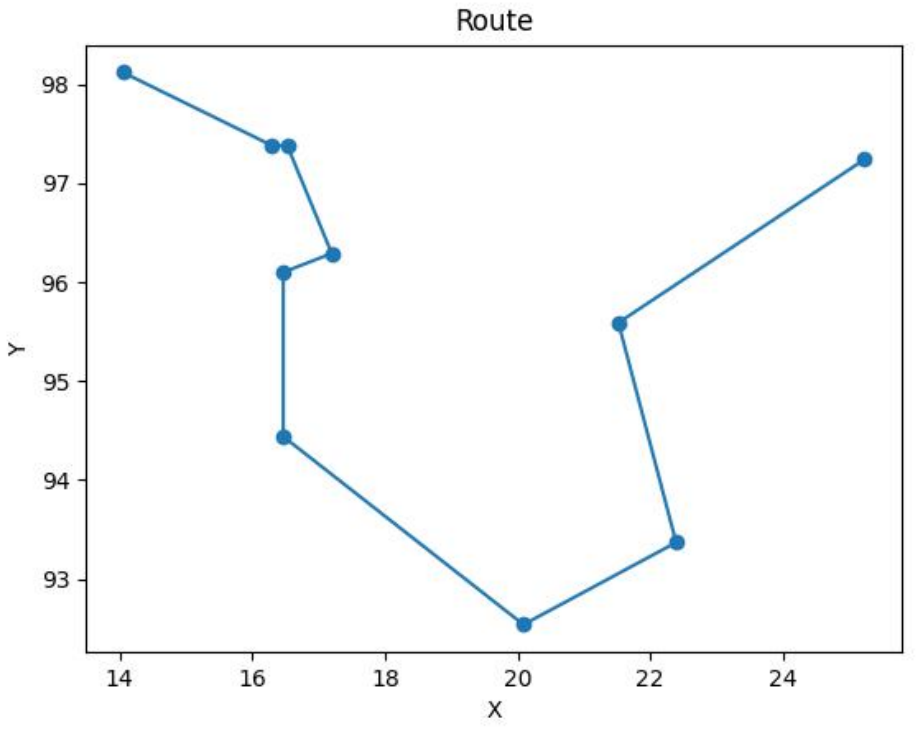
```
    # 进化操作
    def evolve(self):
        population = self.individuals
        fitness_sum = sum([x.fitness for x in population])
        fitness = [x.fitness / fitness_sum for x in population]
        probabilities = list(itertools.accumulate(fitness))
        for j in range(self.pop_size):
            parent1 = self.individuals[self.roulette_wheel_selection(probabilities)].x
            parent2 = self.individuals[self.roulette_wheel_selection(probabilities)].x
            child = self.reproduction(parent1, parent2)
            population.append(Individual(child, self.lb, self.ub, self.x_size))
        self.selection(population)
        self.evolve_route.append(self.get_ans()[1])
        print(self.get_ans())


    # 获取最好的个体
    def get_ans(self):
        return self.individuals[-1].x_dec, self.individuals[-1].fitness - M

if __name__ == '__main__':
    pop_size = 50  # 种群大小
    pop_len = 20  # 个体的染色体长度
    lb = [-2, -2]  # 数值的最小值
    ub = [2, 2]  # 数字的最大值
    pc = 0.7  # 交叉的概率
    pm = 0.01  # 变异的概率
    iter = 1000  # 迭代的次数

    # 初始化种群
    pop = Population(pop_size, pop_len, pc, pm, lb, ub, 2)

    # 迭代 100 次
    for i in range(iter):
        pop.evolve()

    plt.figure()
    plt.scatter([x for x in range(1, iter+1)],pop.evolve_route)
    plt.xlabel('Iter')
    plt.ylabel('Distance')
    plt.title('Iter-Distance')
    plt.show()
```

四、实验结果及分析和（或）源程序调试过程

问题一

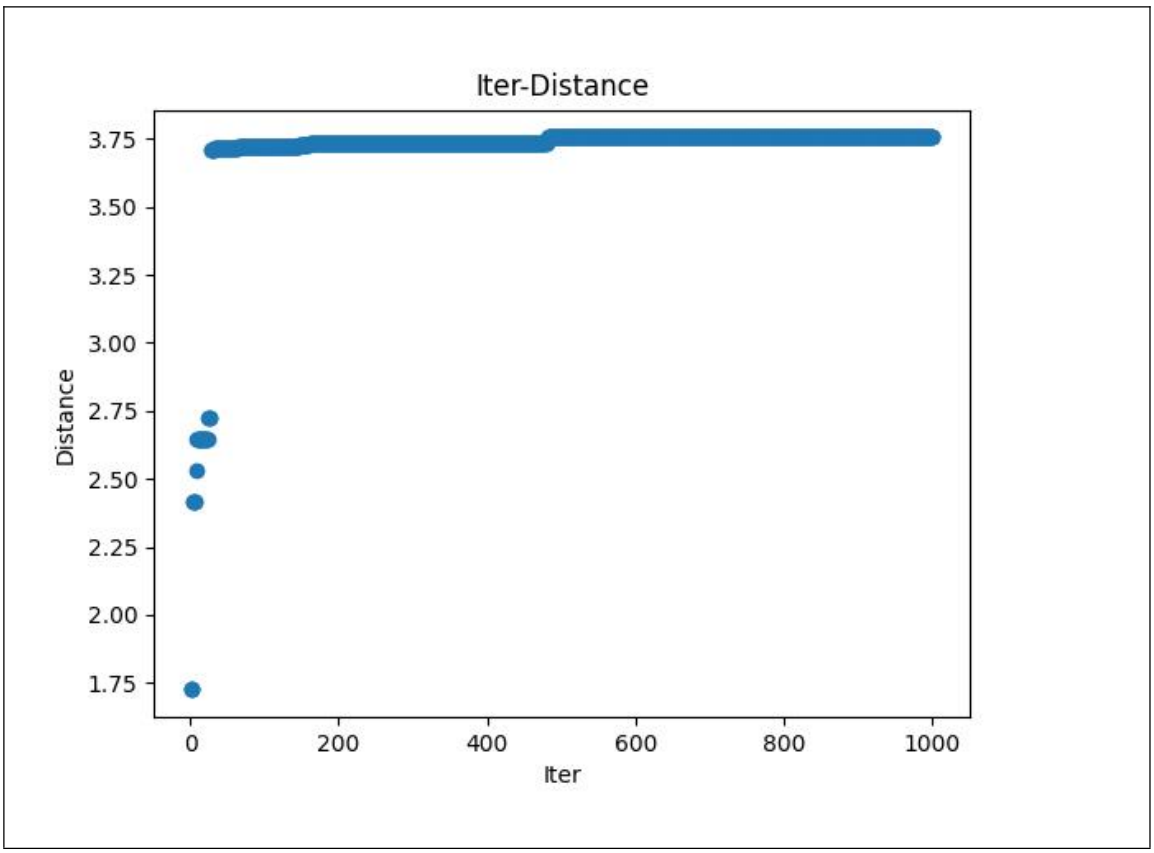
```
[8, 7, 9, 6, 1, 2, 3, 4, 10, 5] 19.27059878408909
([8, 7, 9, 6, 1, 2, 3, 4, 10, 5], 19.27059878408909)
[8, 7, 9, 6, 1, 2, 3, 4, 10, 5] 19.27059878408909
([8, 7, 9, 6, 1, 2, 3, 4, 10, 5], 19.27059878408909)
[8, 7, 9, 6, 1, 2, 3, 4, 10, 5] 19.27059878408909
```





问题二

```
([1.7631799380300315, -1.99998613280356], 3.75631220638752)
([1.7631799380300315, -1.99998613280356], 3.75631220638752)
([1.7631799380300315, -1.99998613280356], 3.75631220638752)
([1.7631799380300315, -1.99998613280356], 3.75631220638752)
([1.7631799380300315, -1.99998613280356], 3.75631220638752)
```


Iter-Distance
Distance
Iter
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
0
200
400
600
800
1000