实验1：遗传算法求最值问题

实验目标

遗传算法是一种模拟自然选择和遗传机制的全局优化算法，适用于复杂函数的优化问题。它通过选择、交叉和变异操作，不断迭代逼近最优解。 在本实验中，我们需要求解目标函数：

$$y = 10 \times \sin(5x) + 7|x - 5| + 10$$

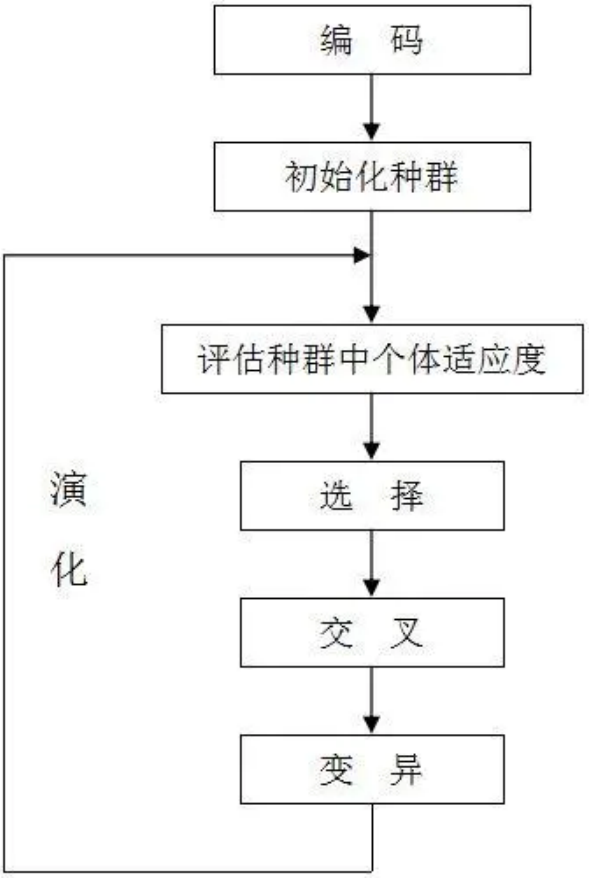
的最小值，定义域为x∈[0,10]。由于目标函数具有多个局部极值点，选择遗传算法以探索全局最优解。

实验原理

遗传算法是一种基于自然选择和遗传机制的优化算法。它通过模拟生物进化过程（选择、交叉、变异）来搜索问题的最优解。其核心思想是通过迭代生成种群，逐步优化个体的适应度，最终找到最优解。

1. **编码设计**：将问题的解表示为染色体。本实验采用实数编码，直接使用 x 的值作为染色体。
2. **适应度函数**：用于评价个体的优劣。本实验的目标是最小化函数值，因此适应度函数取目标函数值的相反数。
3. **选择**：根据适应度选择优秀个体。本实验采用轮盘赌选择。
4. **交叉**：通过组合两个父代个体的基因生成子代。本实验采用单点交叉。

5. **变异**：对个体的基因进行随机扰动，增加种群的多样性。本实验采用均匀变异。



https://blog.csdn.net/zyx_bx

实验思路

1. **初始化种群**：在 x 的范围内随机生成初始种群。
2. **适应度计算**：计算每个个体的适应度值。
3. **选择**：根据适应度值选择个体进入下一代。
4. **交叉**：对选中的个体进行交叉操作，生成子代。
5. **变异**：对子代进行变异操作，增加多样性。
6. **迭代**：重复上述步骤，直到达到最大迭代次数。
7. **结果输出**：输出最优解及其对应的函数值。

实验过程

1. **定义目标函数**：

```
def objective_function(x):
    return 10 * np.sin(5 * x) + 7 * np.abs(x - 5) + 10
```

2. **初始化种群**:

```
population = np.random.uniform(x_bounds[0], x_bounds[1], pop_size)
```

3. **适应度计算**:

```
def fitness(pop):
    return -objective_function(pop)
```

4. **选择**:

```
fitness_values = fitness(population)
probabilities = (fitness_values - fitness_values.min()) /
(fitness_values.max() - fitness_values.min())
probabilities /= probabilities.sum()
selected = np.random.choice(population, size=pop_size, p=probabilities)
```

5. **交叉**:

```
offspring = []
for _ in range(pop_size // 2):
    if np.random.rand() < cross_prob:
        p1, p2 = np.random.choice(selected, size=2, replace=False)
        cross_point = np.random.rand()
        child1 = cross_point * p1 + (1 - cross_point) * p2
        child2 = cross_point * p2 + (1 - cross_point) * p1
        offspring.extend([child1, child2])
    else:
        offspring.extend(np.random.choice(selected, size=2))
population = np.array(offspring)
```

6. **变异**:

```
for i in range(pop_size):
    if np.random.rand() < mut_prob:
        population[i] += np.random.uniform(-0.5, 0.5)
        population[i] = np.clip(population[i], x_bounds[0],
x_bounds[1])
```

7. **结果展示**:

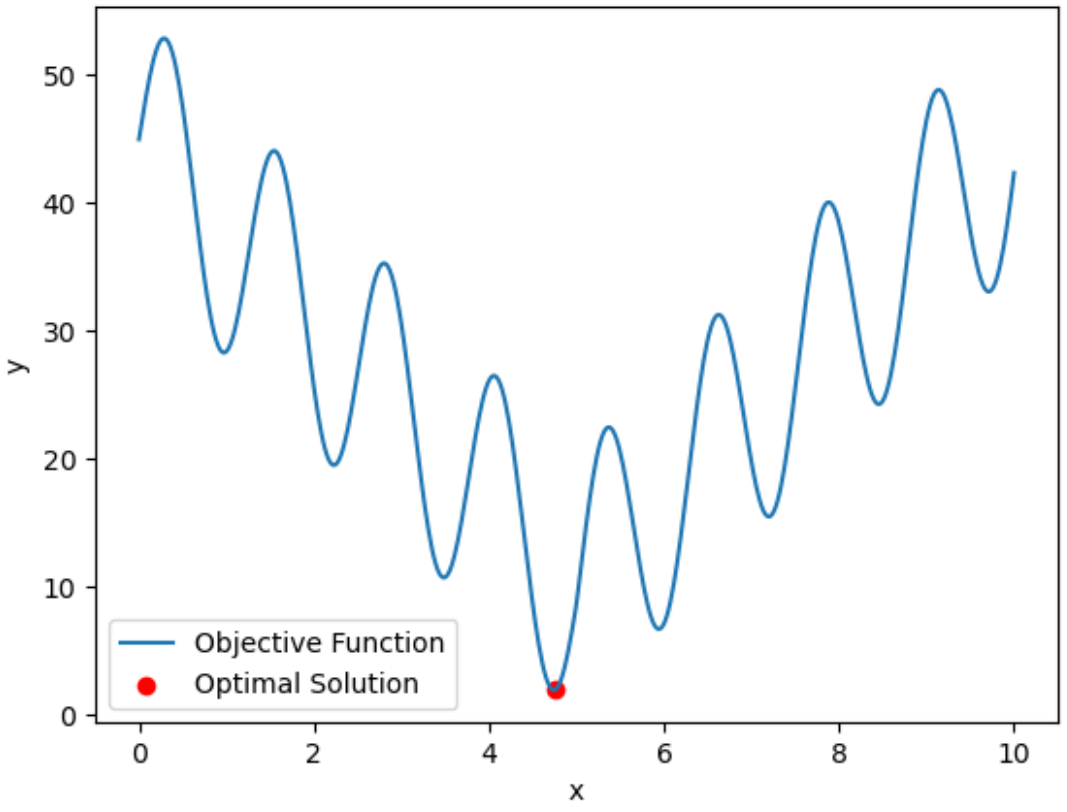
```
best_x = population[np.argmin(objective_function(population))]
best_y = objective_function(best_x)
print(f"最优解：x = {best_x}, y = {best_y}")
```

8. **可视化**：

```
x = np.linspace(x_bounds[0], x_bounds[1], 500)
y = objective_function(x)
plt.plot(x, y, label="Objective Function")
plt.scatter(best_x, best_y, color='red', label="Optimal Solution")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```

实验结果

- 最优解：(x = 4.740791908801333), (y = 1.9151280640255077)
- 可视化结果：



遇到的问题及解决方案

1. **适应度函数设计**：
   - 问题：目标是最小化函数值，但遗传算法通常最大化适应度。
   - 解决：将适应度函数定义为目标函数值的相反数。
2. **种群多样性不足**：
   - 问题：种群过早收敛，导致无法找到全局最优解。
   - 解决：调整变异概率，增加变异操作的扰动范围。
3. **参数选择**：
   - 问题：交叉概率和变异概率的选择对结果影响较大。
   - 解决：通过多次实验调整参数，找到合适的值。最终选择cross_prob = 0.8 ,mut_prob = 0.1

实验总结

通过本次实验，我成功使用遗传算法求解了函数的最小值。实验过程中，通过调整参数和优化算法设计，解决了种群多样性不足和适应度函数设计等问题。遗传算法在求解复杂非线性优化问题时表现出较强的鲁棒性和全局搜索能力。

附录

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

# 定义目标函数
def objective_function(x):
    return 10 * np.sin(5 * x) + 7 * np.abs(x - 5) + 10

# 遗传算法参数
pop_size = 50  # 种群大小
generations = 100  # 迭代次数
cross_prob = 0.8  # 交叉概率
mut_prob = 0.1  # 变异概率
x_bounds = [0, 10]  # x的范围

# 初始化种群
population = np.random.uniform(x_bounds[0], x_bounds[1], pop_size)

# 适应度计算
def fitness(pop):
    return -objective_function(pop)  # 目标是最小值，适应度取反
```

```python
# 遗传算法主循环
for generation in range(generations):
    # 选择
    fitness_values = fitness(population)
    probabilities = (fitness_values - fitness_values.min()) /
(fitness_values.max() - fitness_values.min())
    probabilities /= probabilities.sum()
    selected = np.random.choice(population, size=pop_size, p=probabilities)

    # 交叉
    offspring = []
    for _ in range(pop_size // 2):
        if np.random.rand() < cross_prob:
            p1, p2 = np.random.choice(selected, size=2, replace=False)
            cross_point = np.random.rand()
            child1 = cross_point * p1 + (1 - cross_point) * p2
            child2 = cross_point * p2 + (1 - cross_point) * p1
            offspring.extend([child1, child2])
        else:
            offspring.extend(np.random.choice(selected, size=2))
    population = np.array(offspring)

    # 变异
    for i in range(pop_size):
        if np.random.rand() < mut_prob:
            population[i] += np.random.uniform(-0.5, 0.5)
            population[i] = np.clip(population[i], x_bounds[0], x_bounds[1])

# 结果展示
best_x = population[np.argmin(objective_function(population))]
best_y = objective_function(best_x)
print(f"最优解：x = {best_x}, y = {best_y}")

# 可视化
x = np.linspace(x_bounds[0], x_bounds[1], 500)
y = objective_function(x)
plt.plot(x, y, label="Objective Function")
plt.scatter(best_x, best_y, color='red', label="Optimal Solution")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```