

**进化计算课程报告**

徽标

描述已自动生成

|  |  |
| --- | --- |
| 作业题目 | 遗传算法在求解函数最小值中的应用 |
| 学院 | 计算机学院 |
| 专业 | 计算机科学与技术专业 |
| 姓名 | 孙谦昊 |
| 学号 | 2023141460321 |
| 课程老师 | 孙亚楠 |
| 助教 | 谢婧荣 |

2025年03月 24日

目录

[1. 实验步骤 1](#_Toc1367916680)

[1.1 初始化 1](#_Toc1763199625)

[1.2 适应度评估 1](#_Toc969252422)

[1.3 环境选择 1](#_Toc1561994059)

[1.4 交叉 1](#_Toc1594048685)

[1.5 变异 1](#_Toc1317752470)

[2. 实验组件 1](#_Toc466911779)

[2.1 交叉变异类型 1](#_Toc481023515)

[2.2 交叉变异概率 1](#_Toc1433769297)

[2.3 适应度值评估 1](#_Toc446571692)

[2.4 群体大小 1](#_Toc75081179)

[2.5 精英主义的比例 1](#_Toc1316474664)

[3. 重复运行结果 1](#_Toc465662807)

[3.1 结果展示 1](#_Toc964387581)

[3.2 平均值 1](#_Toc1402989958)

[3.3 标准值 1](#_Toc681780046)

[4. 进化过程 1](#_Toc1851976377)

[4.1 进化过程图 1](#_Toc538988621)

[4.2 结论 2](#_Toc695730101)

[5. 同一代的平均值、最大值、最小值 2](#_Toc87349592)

[5.1 每一代的平均值、最大值和最小值图 2](#_Toc1353261843)

[5.2 结论 2](#_Toc272489924)

**遗传算法在求解函数最小值中的应用**

**摘要**：本文基于遗传算法（Genetic Algorithm, GA）实现了对二维Sphere函数等函数的最小化优化。算法通过模拟生物进化过程，包含以下关键步骤：1）初始化种群：在[-5.12, 5.12]范围内随机生成500个二维个体；2) 适应度评估：以函数值的负数作为适应度，引导种群向最小值方向进化；3) 选择操作：采用锦标赛选择与精英保留策略；4) 交叉与变异：单点交叉生成子代，并以一定概率进行随机变异。通过Matplotlib动态展示了种群在函数热力图上的分布演化，并记录了各代最佳、最差及平均适应度。实验结果表明，算法能有效收敛至近最优解，适应度曲线显示种群整体性能逐步提升，标准差减小，验证了遗传算法在连续优化问题中的有效性。

# 实验步骤

## 初始化

群规模：设定为500个个体，在搜索空间[-5.12,5.12]×[-5.12,5.12]内均匀随机分布，通过较大规模种群保证初始解的多样性。

基因编码：每个决策变量（x₁/x₂）采用100位二进制基因表示（总基因长度200位），通过decimal\_to\_binary函数将浮点数值映射为二进制串，实现连续空间离散化编码。

进化代数：限定50代迭代，平衡收敛速度与计算效率。

锦标赛选择：采用规模为5的竞争选择机制，保留优质基因的同时维持选择压力。

精英保留：每代保留2%最优个体（约10个），防止优秀基因丢失。

变异概率：设置20%的高变异率，增强算法跳出局部最优的能力。

初始化方法：通过np.random.uniform在定义域内生成均匀分布的初始种群（500×2矩阵），既避免初始解聚集，又为二进制编码提供实数空间锚点。

# 初始化种群

population\_size = 500  # 种群个体数

genes = 100            # 基因数

generation = 50        # 种群代数

mutation\_prob = 0.2   # 变异概率

elite\_prob = 0.02     # 精英概率

tournament\_size = 5    # 锦标赛大小

population = np.random.uniform(LOW, HIGHT, (population\_size, 2))

## 适应度评估

environment(x1, x2)函数为环境，借助**-** environment(x1, x2)函数为当代中的每个个体进行适应度评估。其中**-** environment(x1, x2)的值越大表明当代中的该个体对环境的适应度越高。

## 环境选择

采用经营保留的策略的将当代中对环境适应最好的个体放入elite\_individuals中以直接进入下一代。此外再通过锦标赛选择法选取出出精英个体外对环境适应度较好的个体。

# 精英保留

    elite\_size = int(population\_size \* elite\_prob)

    elite\_indices = np.argsort(fitness\_degree.flatten())[-elite\_size:]

    elite\_individuals = population[elite\_indices]

    # 选择

    selected\_indices = []

    for j in range(population\_size - elite\_size):

        candidates = np.random.choice(len(population), size=tournament\_size, replace=False)

        # 选择适应度最高的个体

        best\_candidate = candidates[np.argmax(fitness\_degree[candidates])]

        selected\_indices.append(best\_candidate)

    selected\_population = population[selected\_indices]

## 交叉

本文采用的是二进制交叉法。首先将个体的数值（表现形式）通过映射的方式映射为一个长度为50的字符串。随后在交叉融合基因的过程中借助随机数随机拼接父代和母代的基因。

## 变异

借助随机数将二进制基因编码的某一位的值进行翻转，完成变异。

# 变异

    for j in range(len(next\_population)):

        for k in range(2):

            if np.random.rand() < mutation\_prob:

                next\_population[j][k] = np.random.uniform(LOW, HIGHT)

# 实验组件

## 交叉变异类型

# 根据选择结果进行交配

    next\_population = np.zeros((population\_size - elite\_size, 2))

    for j in range(0, population\_size - elite\_size - 1, 2):

        # 选出父母

        parent1 = selected\_population[j]

        parent2 = selected\_population[j + 1]

        # 交叉融合基因

        crossover\_point = []

        for k in range(4):

            crossover\_point.append(np.random.randint(0, genes))

        child1\_bin = [decimal\_to\_binary(parent1[0], LOW, HIGHT, genes)[:crossover\_point[0]] + decimal\_to\_binary(parent2[0], LOW, HIGHT, genes)[crossover\_point[0]:],

                      decimal\_to\_binary(parent1[1], LOW, HIGHT, genes)[:crossover\_point[1]] + decimal\_to\_binary(parent2[1], LOW, HIGHT, genes)[crossover\_point[1]:]]

        child2\_bin = [decimal\_to\_binary(parent1[0], LOW, HIGHT, genes)[:crossover\_point[2]] + decimal\_to\_binary(parent2[0], LOW, HIGHT, genes)[crossover\_point[2]:],

                      decimal\_to\_binary(parent1[1], LOW, HIGHT, genes)[:crossover\_point[3]] + decimal\_to\_binary(parent2[1], LOW, HIGHT, genes)[crossover\_point[3]:]]

        child1 = [binary\_to\_decimal(gene, LOW, HIGHT, genes) for gene in child1\_bin]

        child2 = [binary\_to\_decimal(gene, LOW, HIGHT, genes) for gene in child2\_bin]

        next\_population[j] = child1

        next\_population[j + 1] = child2

## 交叉变异概率

mutation\_prob = 0.2   # 变异概率

## 适应度值评估

# 环境

def environment(x1, x2):

    return x1\*\*2 + x2\*\*2

# 计算适应度

    sum = 0

    for j in range(population\_size):

        a = environment(population[j][0], population[j][1])

        sum += a

        #fitness\_degree[j] = (52.4288 - a) / 52.4288 \* 100

        fitness\_degree[j] = -a

## 群体大小

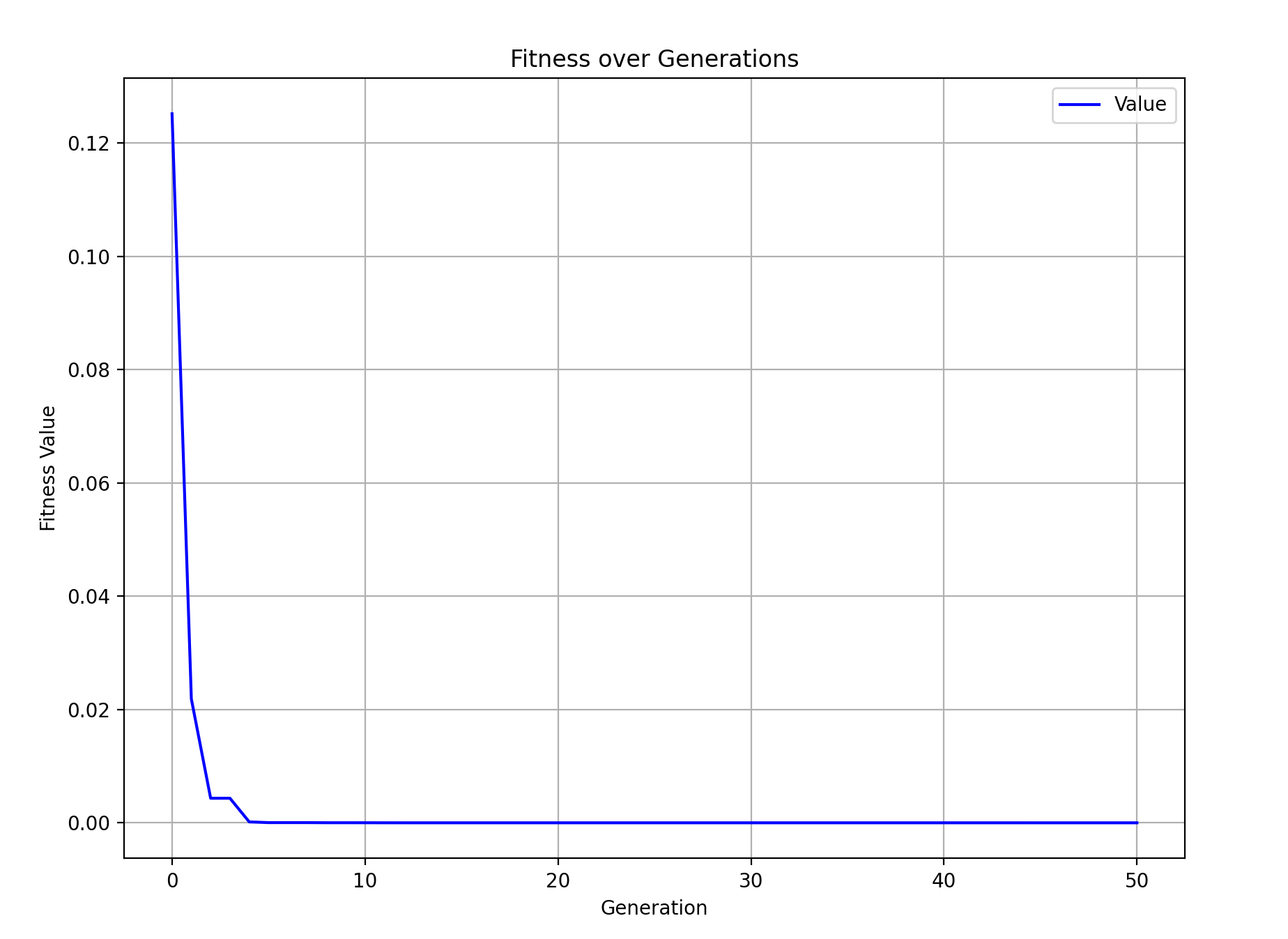
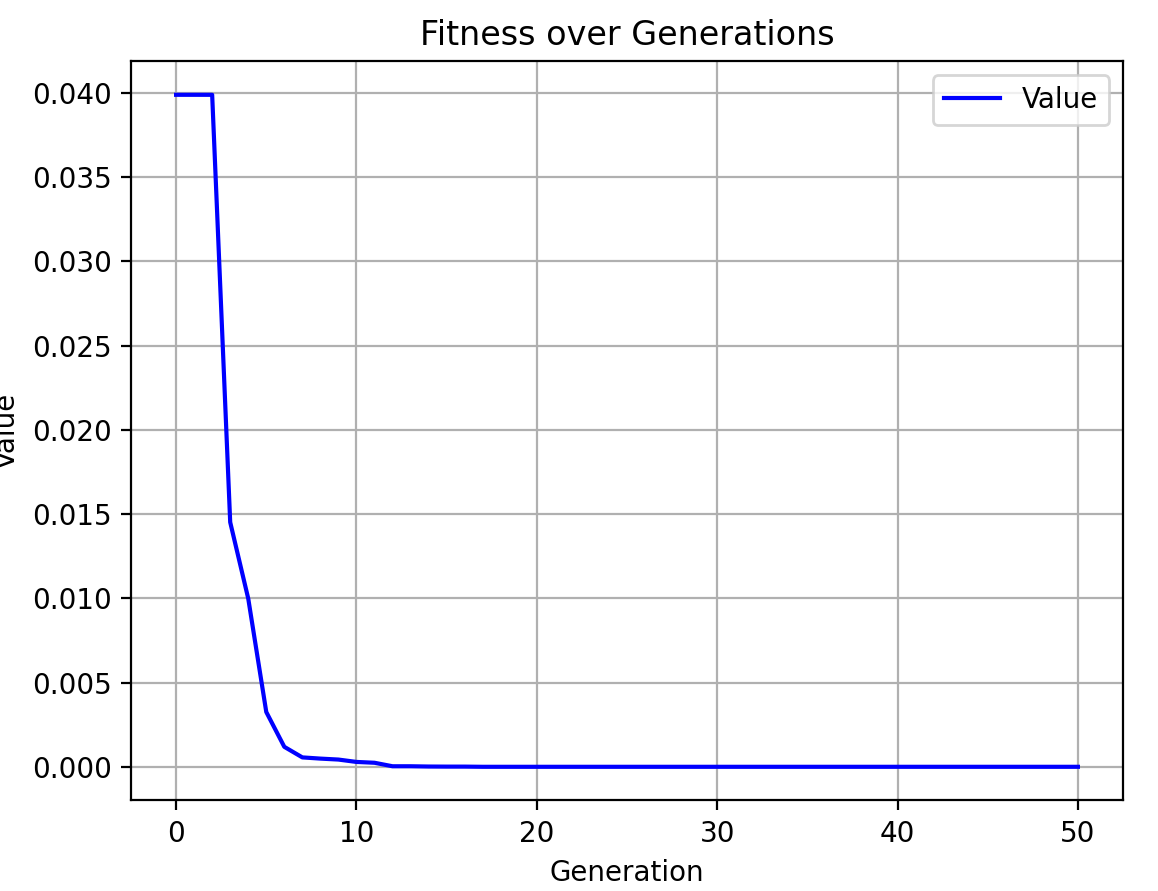
population\_size = 500  # 种群个体数

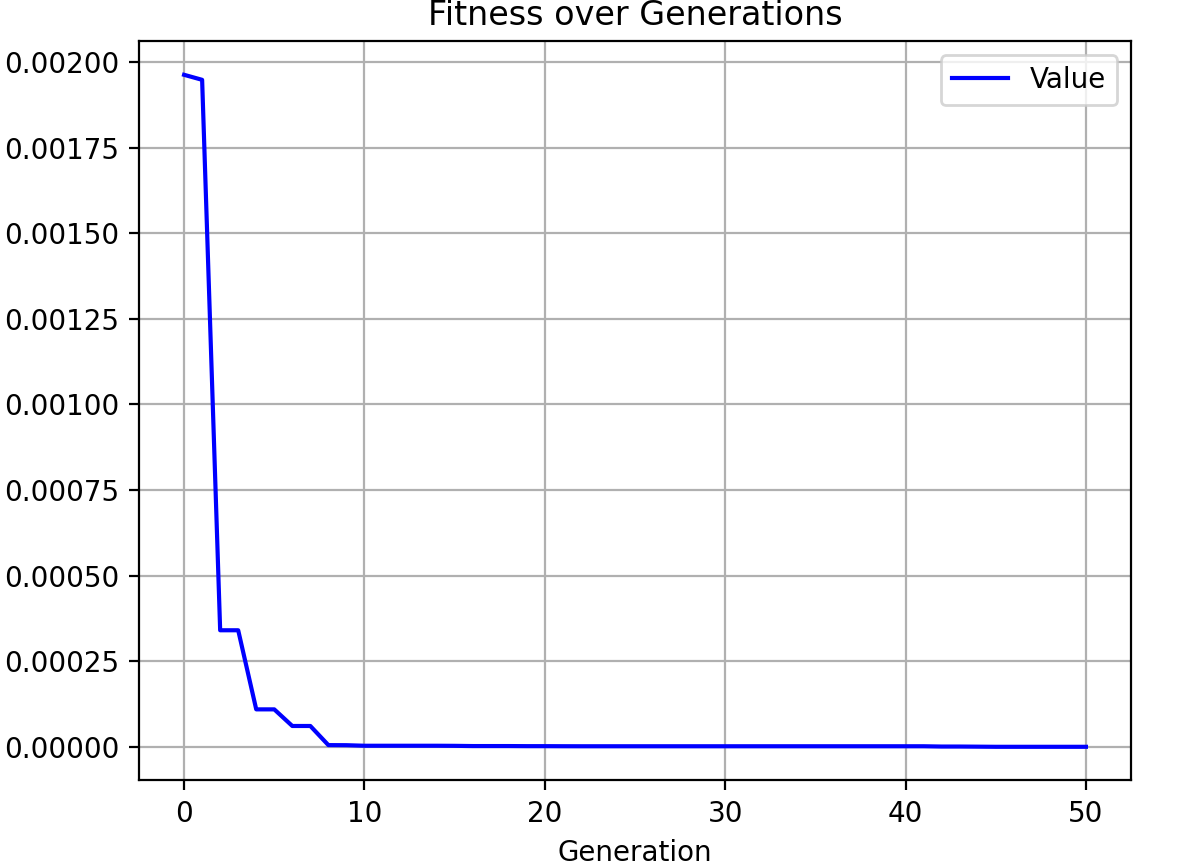
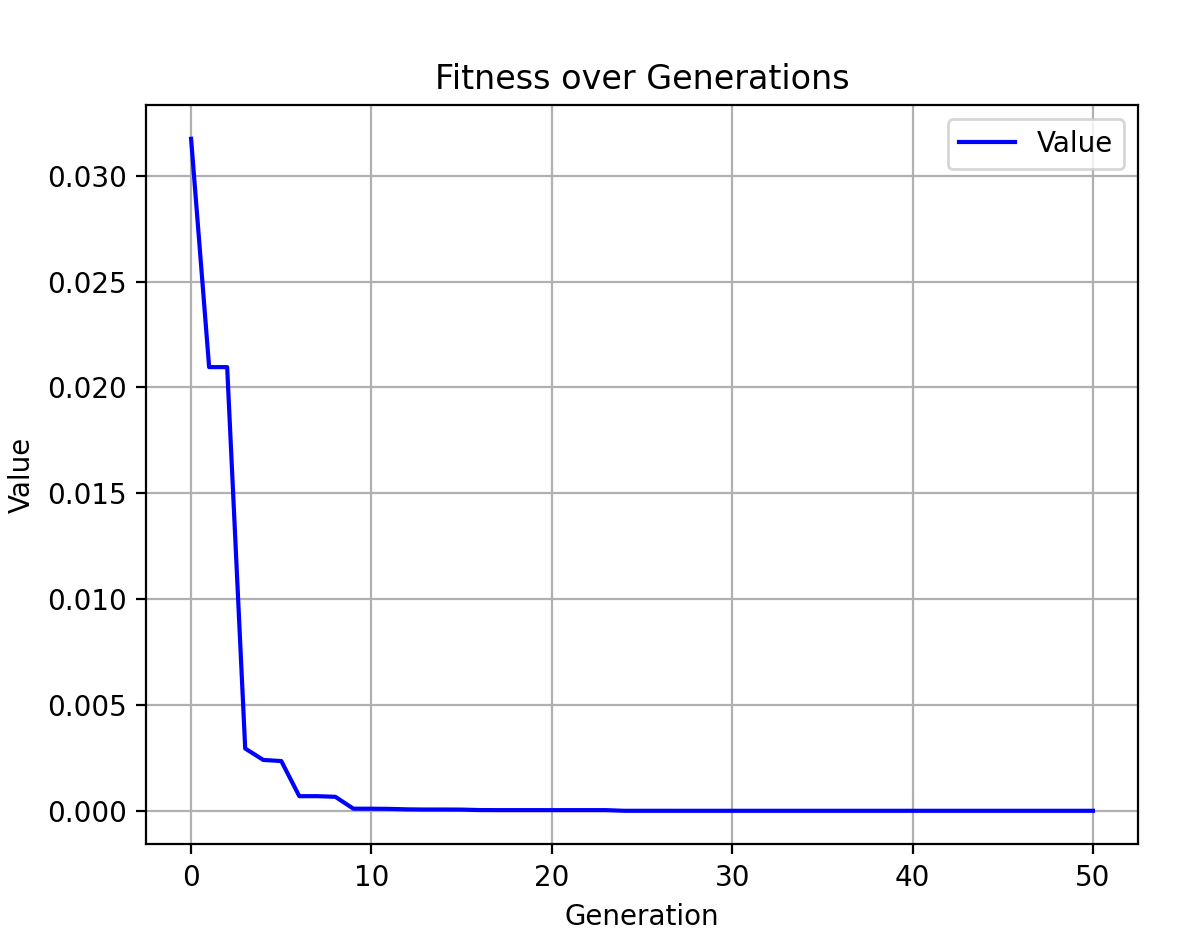
## 精英主义的比例

elite\_prob = 0.02     # 精英概率

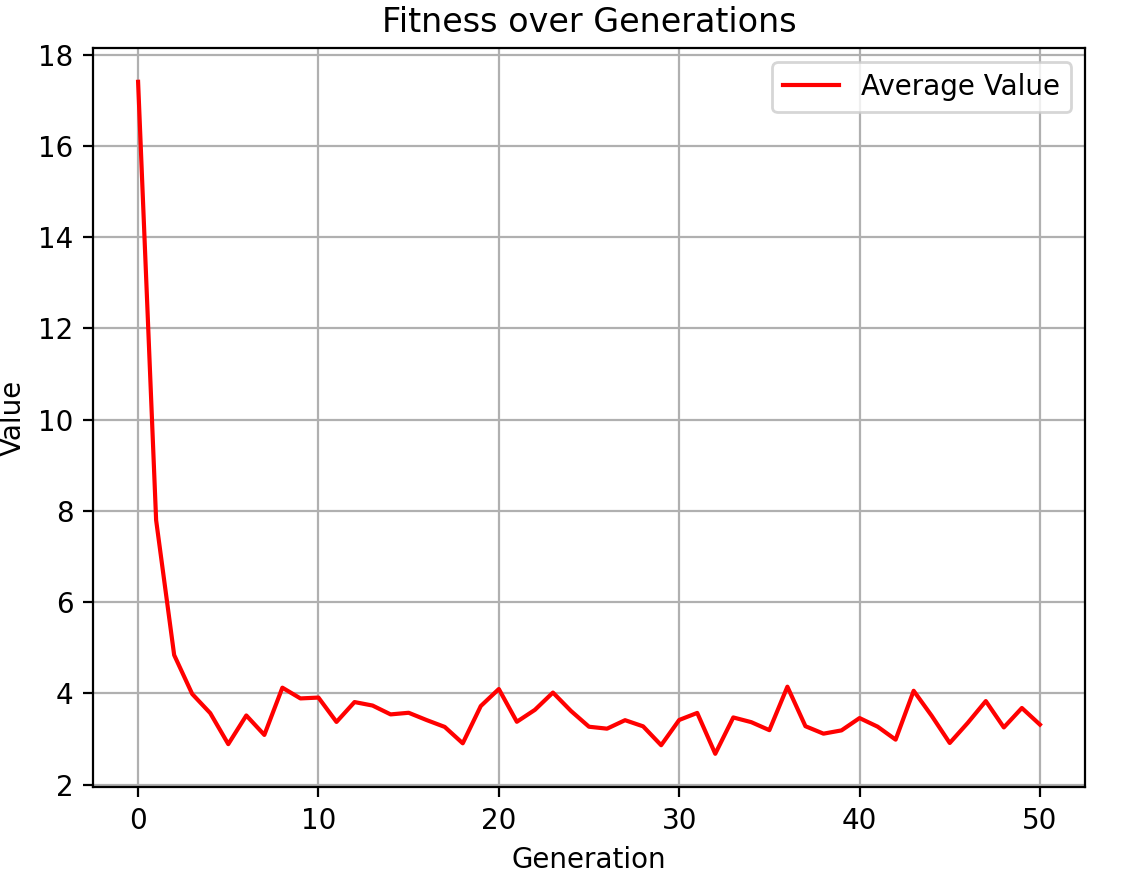
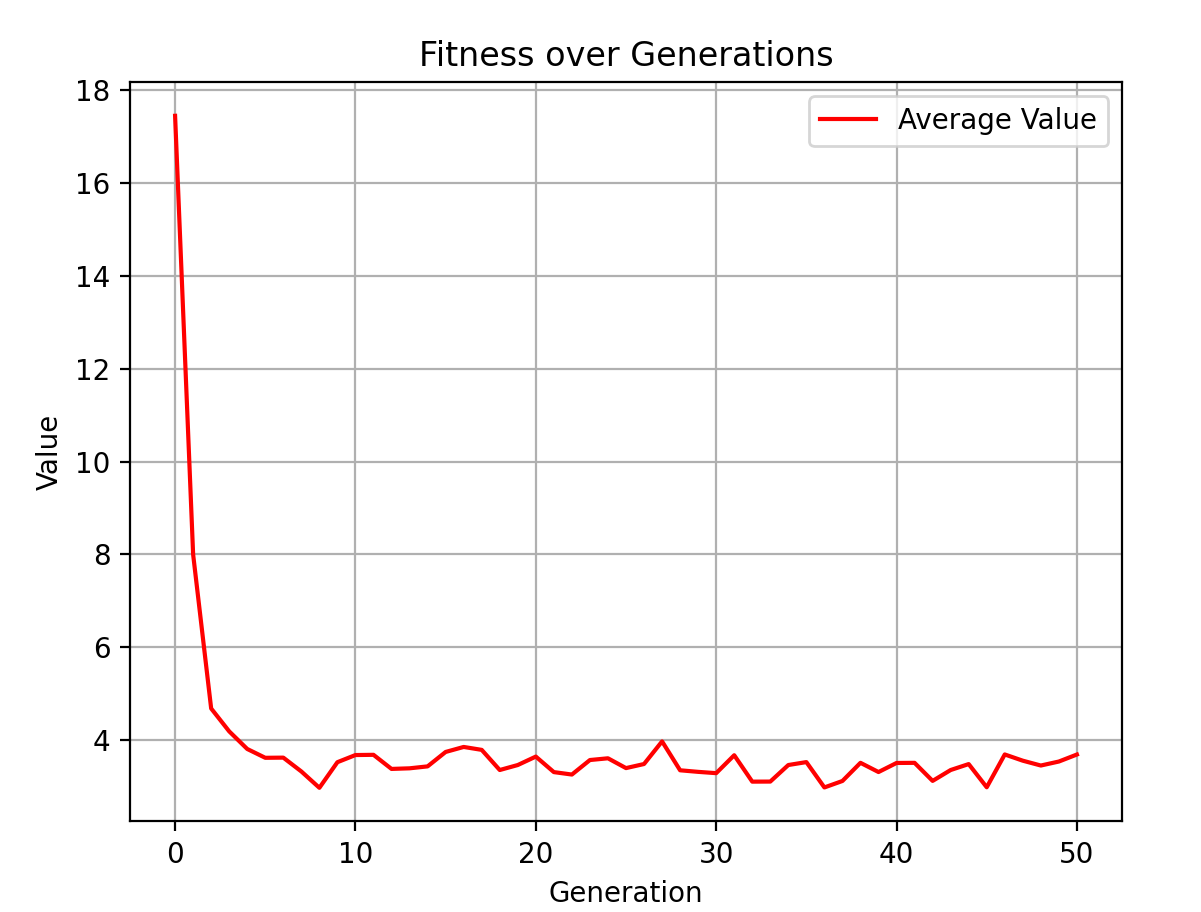
# 重复运行结果

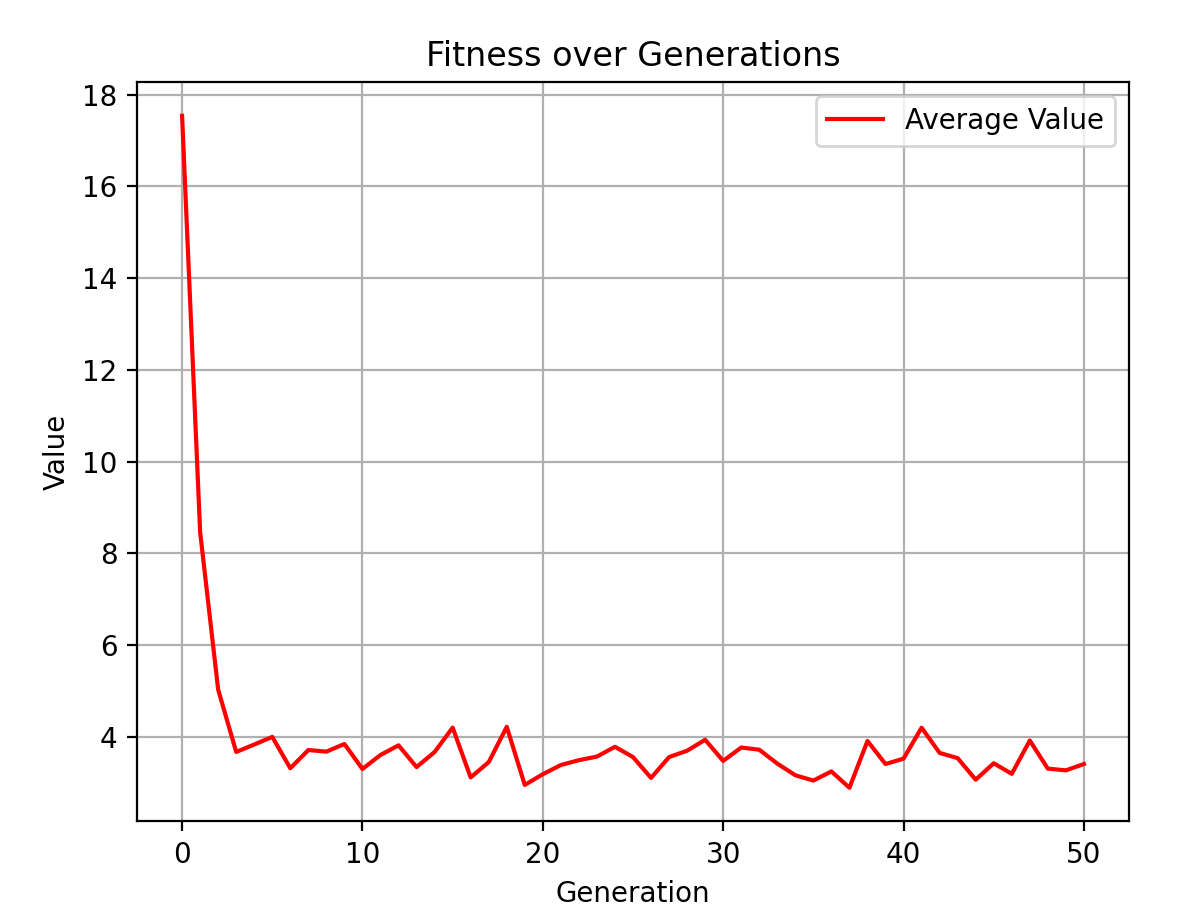
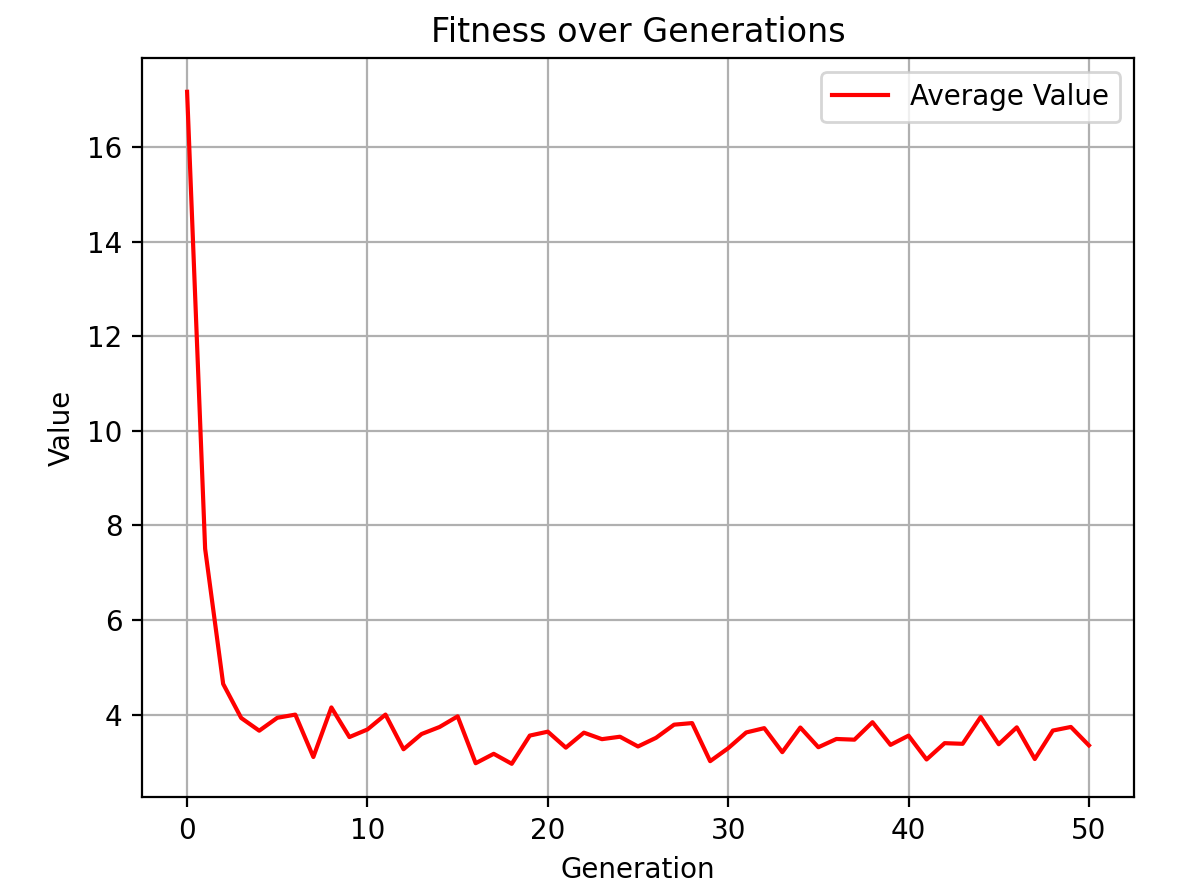
## 结果展示

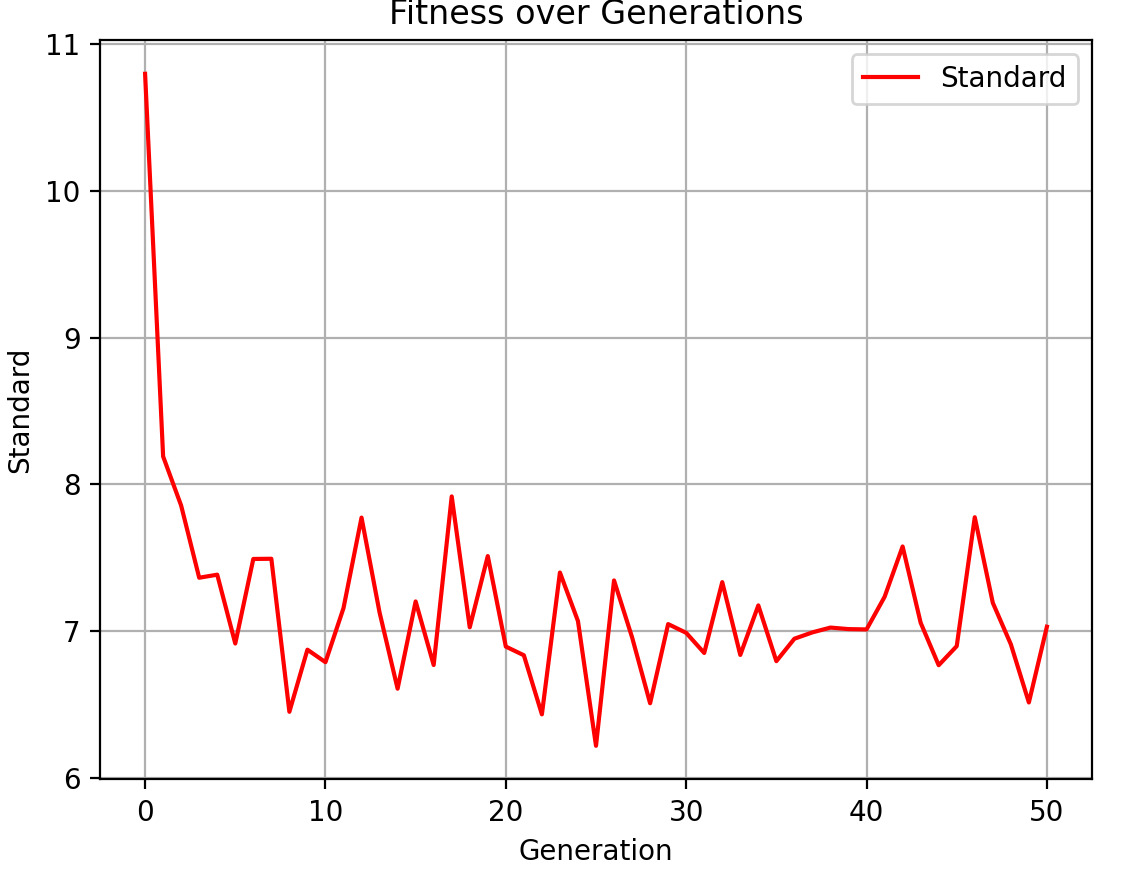
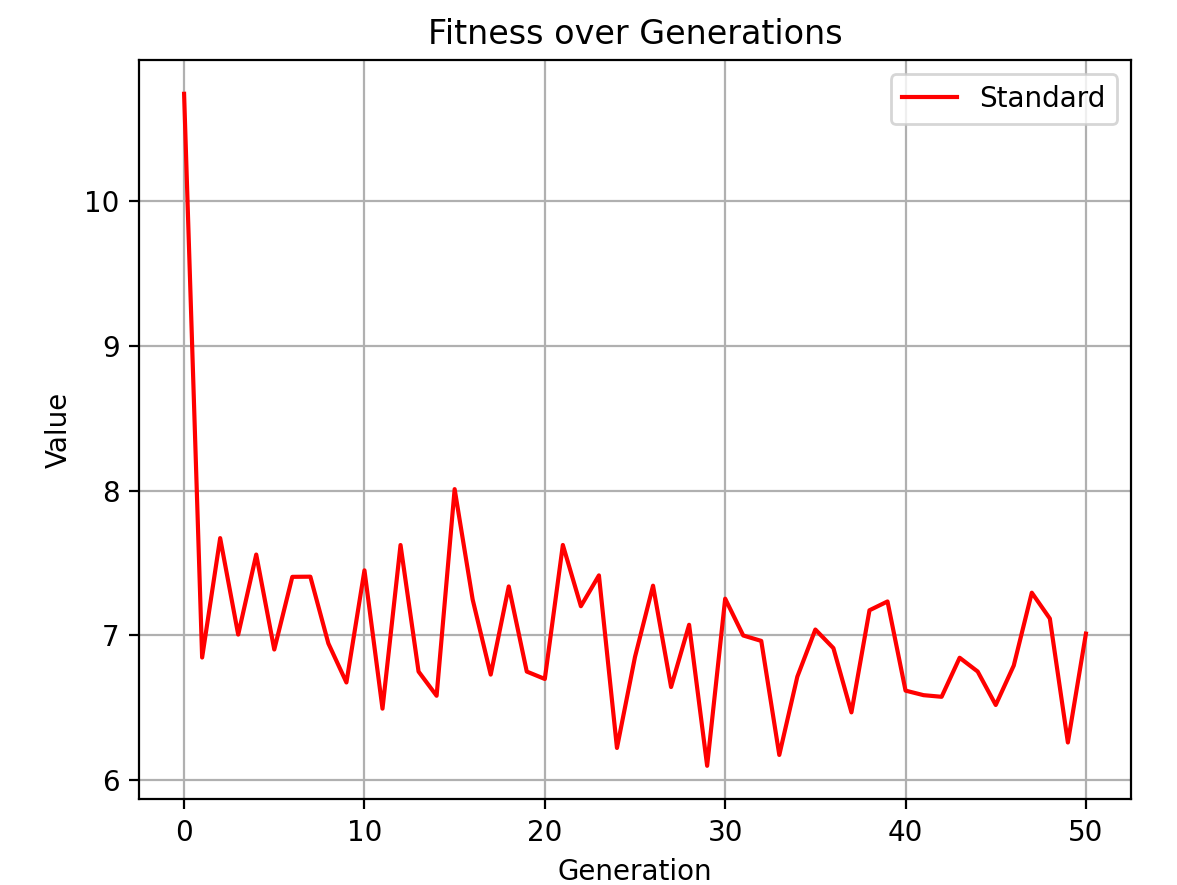
 

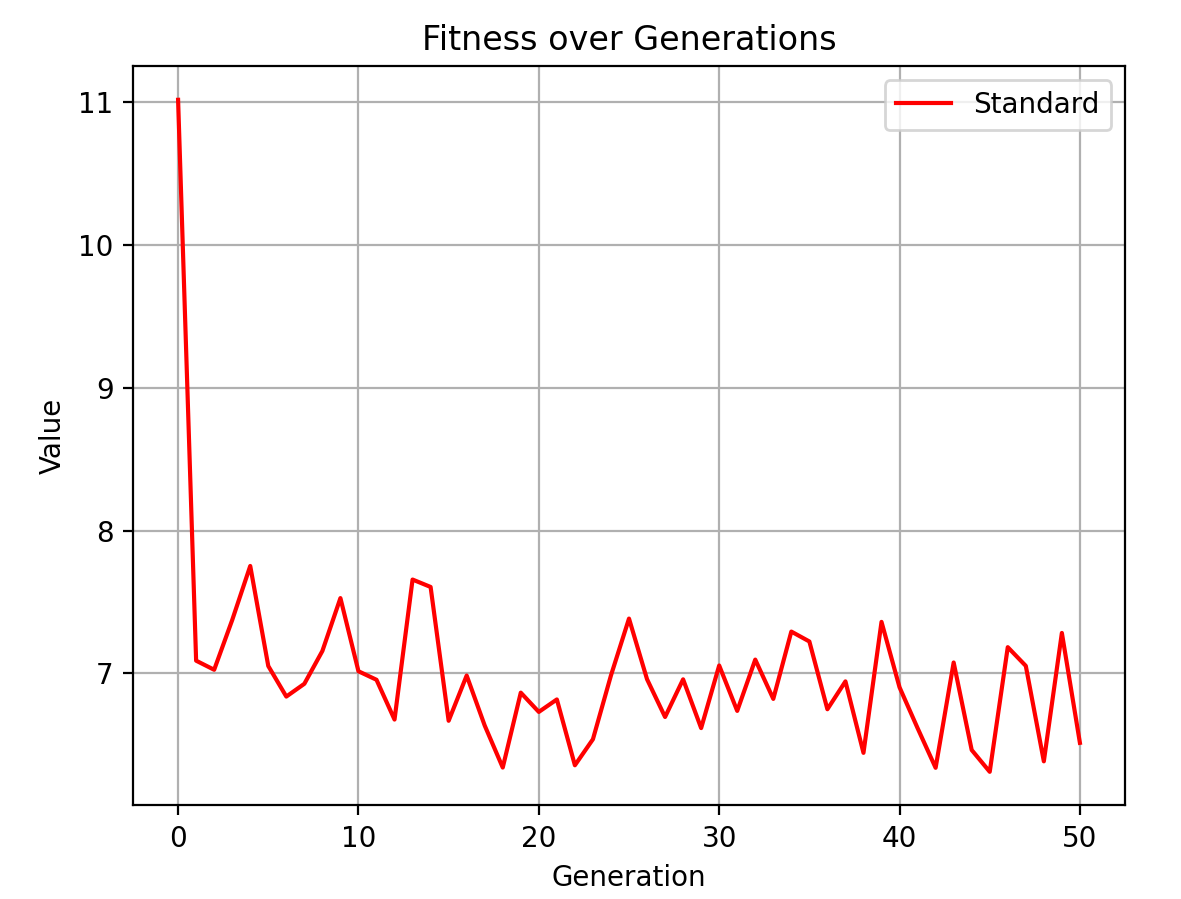
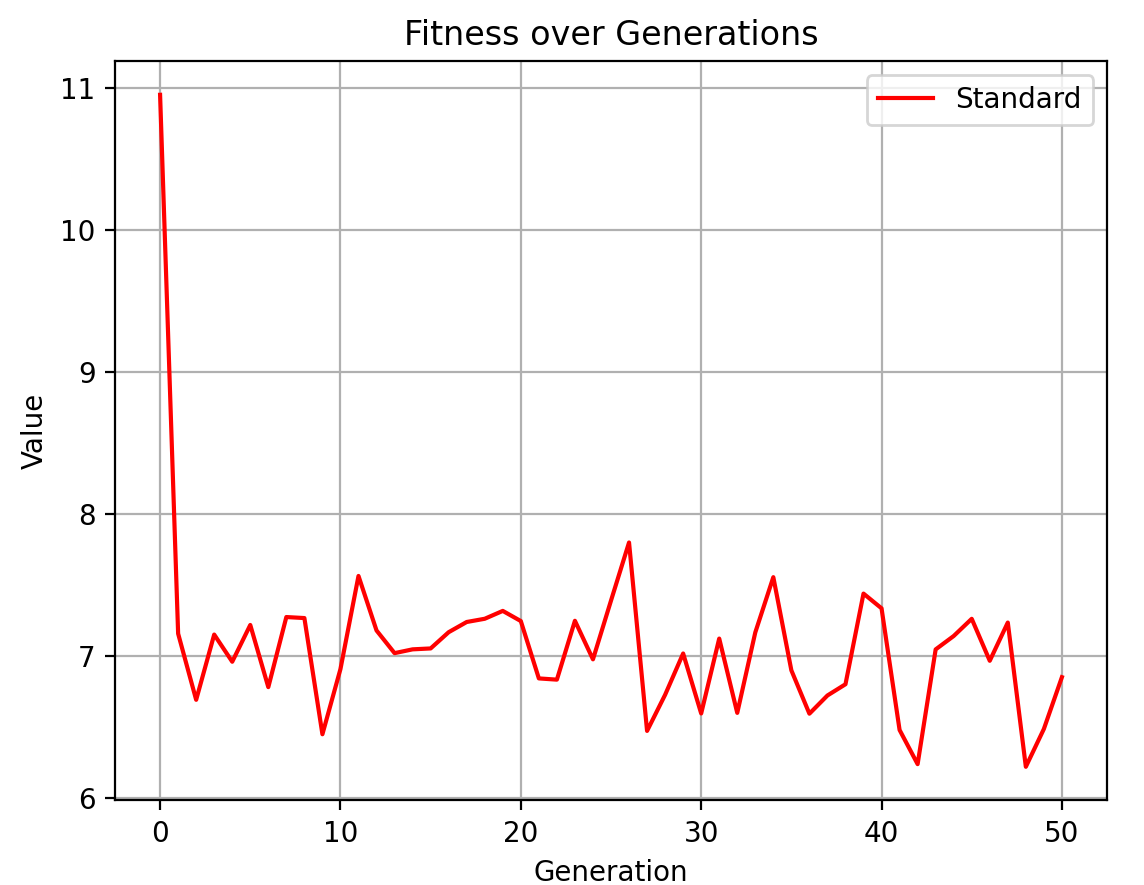
## 平均值

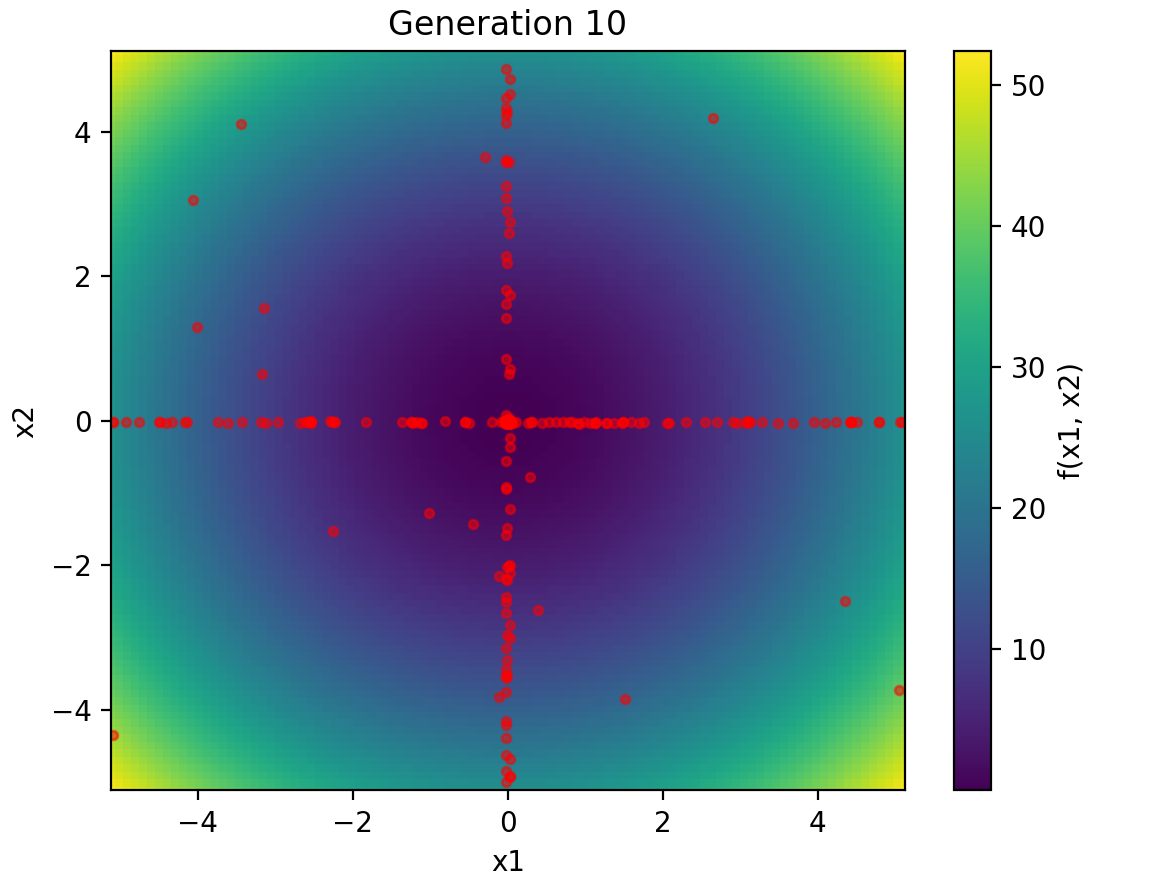
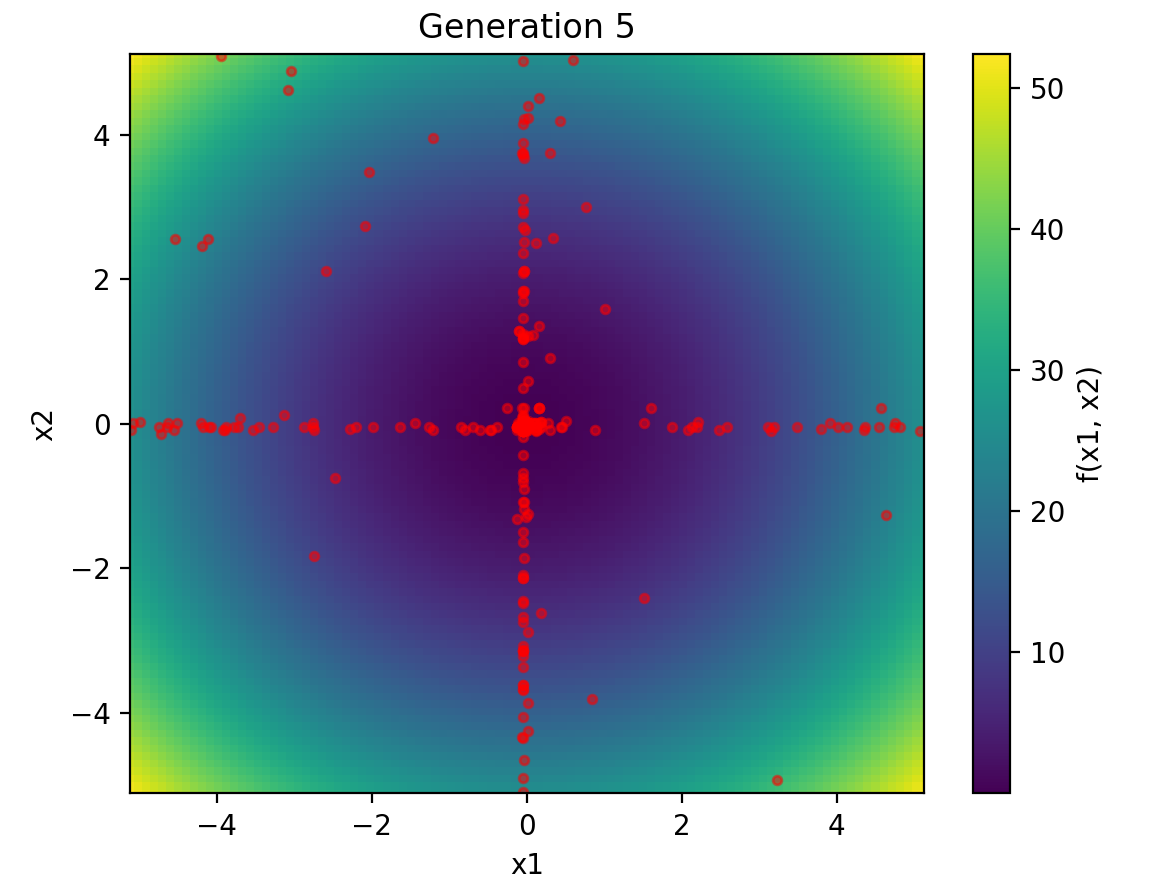
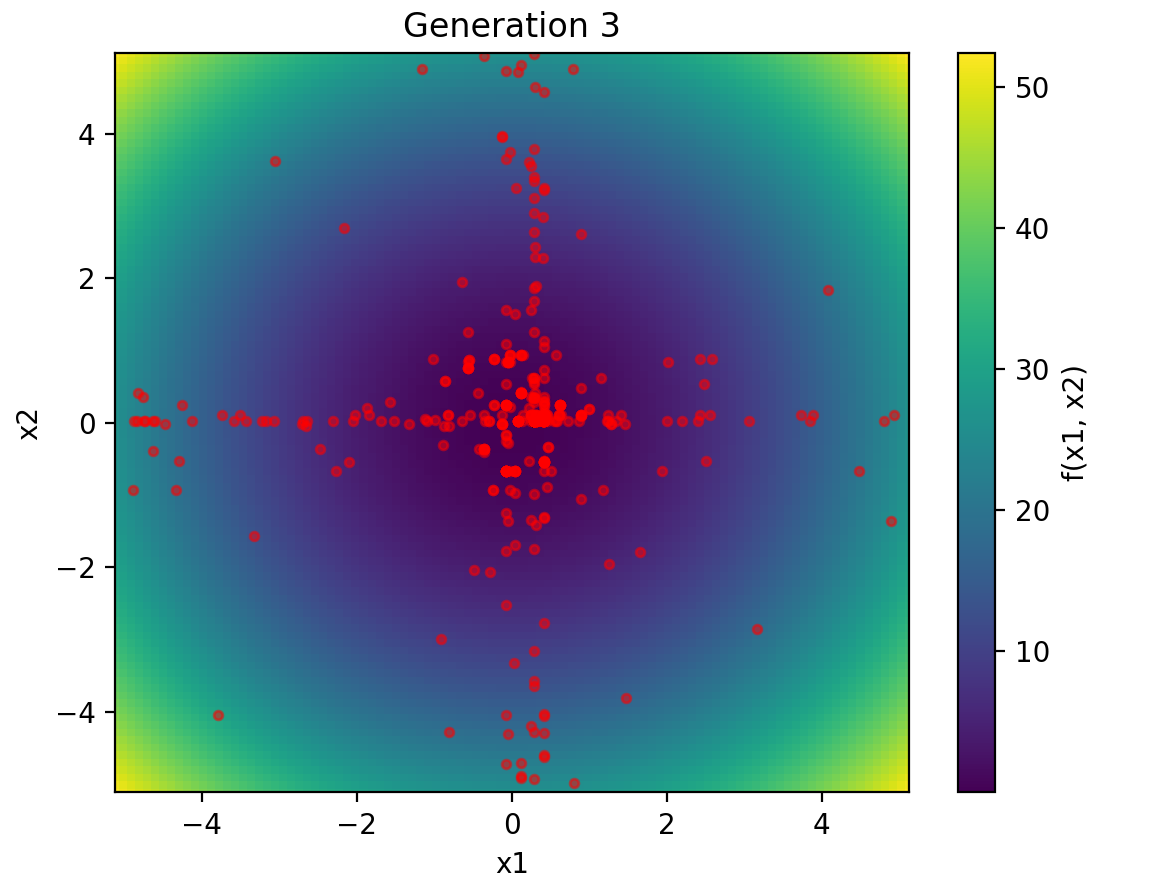
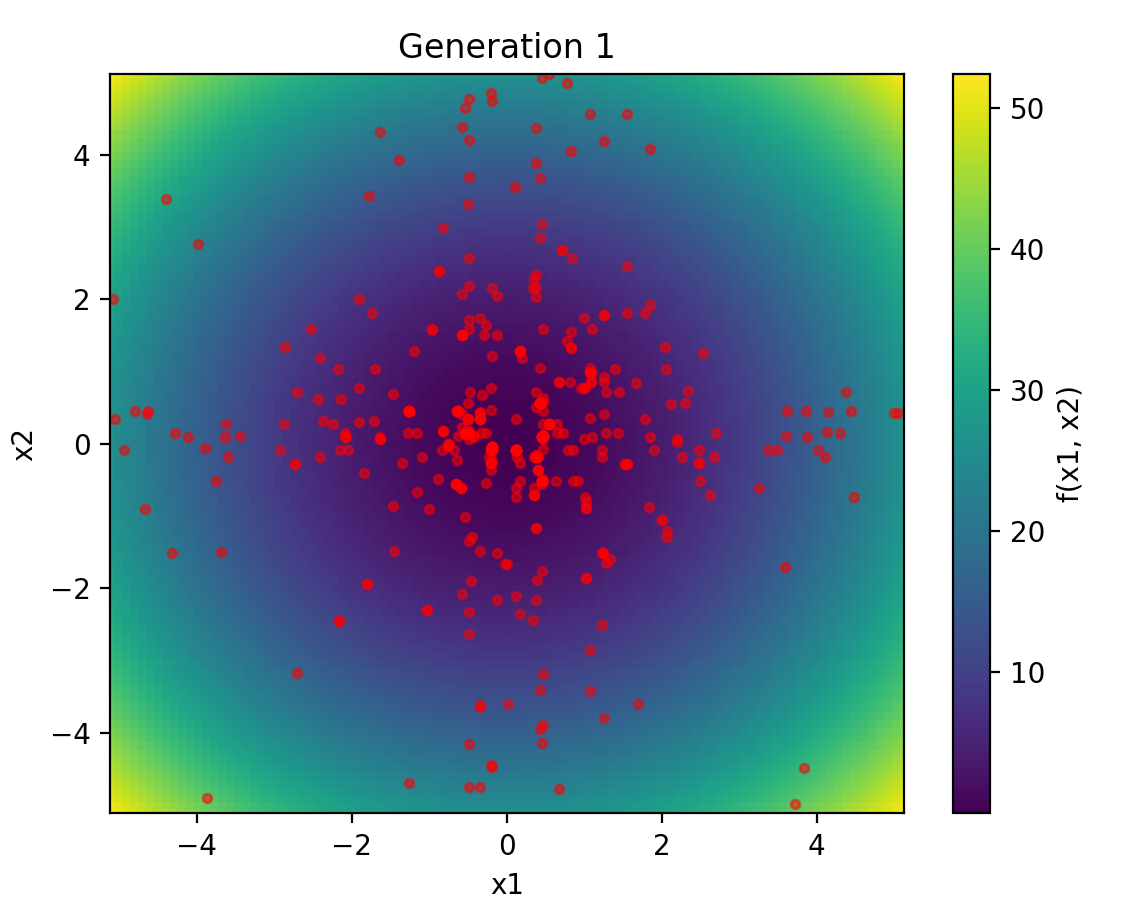
## 标准值

# 进化过程

## 进化过程图

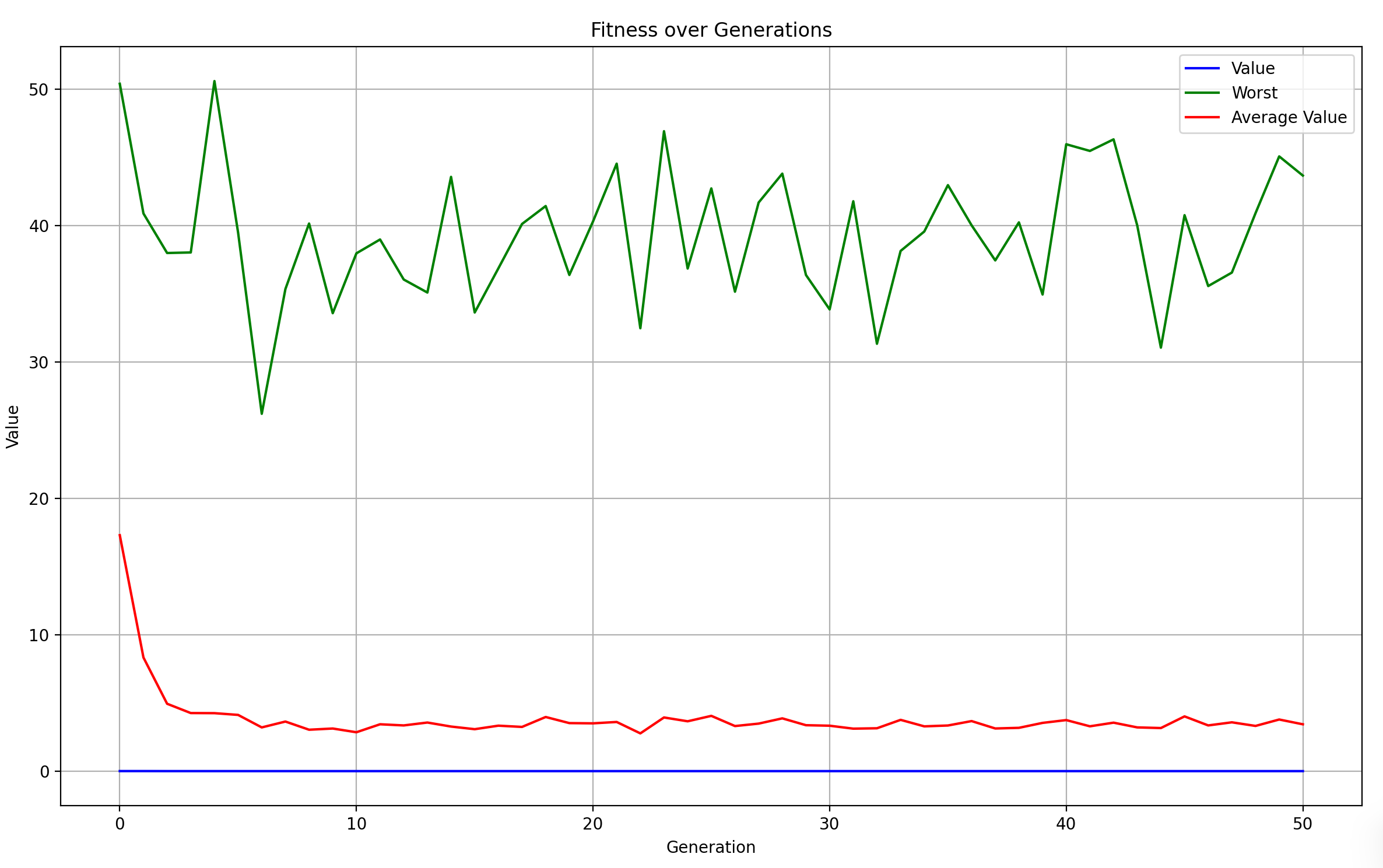


## 结论

多次实验下均能或得基本相同的结果，证明所编写的代码能够针对函数值进行优化。

# 同一代的平均值、最大值、最小值

## 每一代的平均值、最大值和最小值图



## 结论

本实验验证了遗传算法在连续优化问题中的有效性。最优解和平均解均随迭代次数增加而下降，符合优化目标；最差解虽波动较大，但整体趋势向好。算法在初期表现出较强的全局搜索能力，后期则更偏向局部优化，最终成功逼近理论最优解 (0,0)。

## 