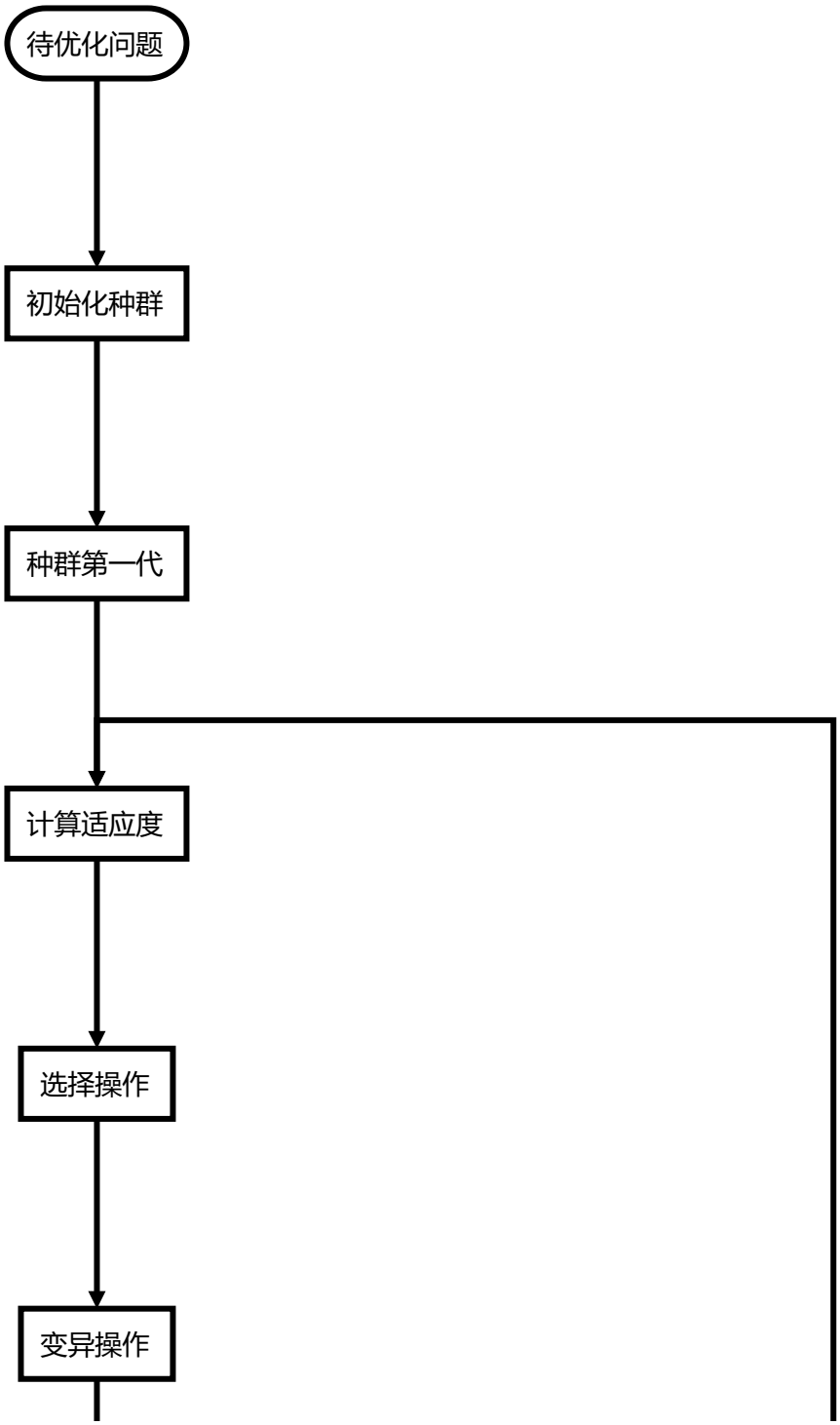
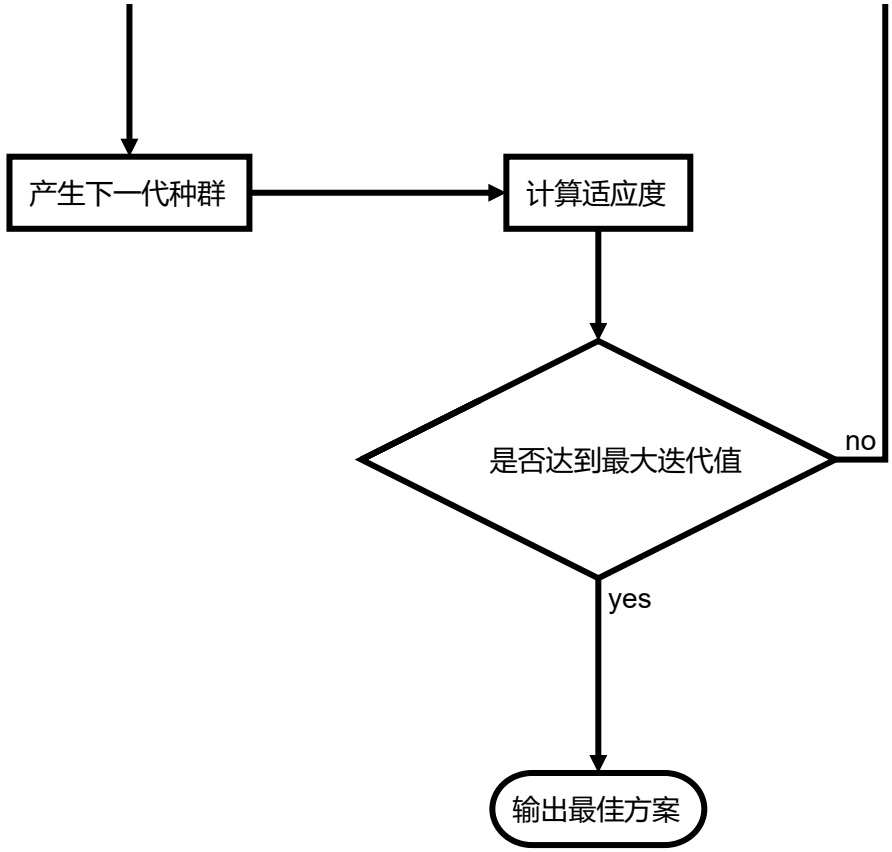
实验六_遗传算法求函数最小值实验

1、画出遗传算法的算法流程图；





2、根据实验内容，给出相应结果以及结果分析；

源代码

```
import math

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
import datetime

DNA_SIZE = 24  # 编码长度
POP_SIZE = 5  # 种群大小
CROSS_RATE = 0.8  # 交叉率
MUTA_RATE = 0.15  # 变异率
Iterations = 1000  # 代次数
X_BOUND = [1, 2]  # X区间
Y_BOUND = [1, 2]  # Y区间
```

```
16
17
18  def F(x, y):  # 适应度函数
19      return 20 + x ** 2 + y ** 2 - 10 * (np.cos(2 * math.pi * x) + np.cos(2 * math.pi * y))
20
21
22  def decodeDNA(pop):  # 解码
23      x_pop = pop[:, 1::2]  # 奇数列表示X
24      y_pop = pop[:, ::2]  # 偶数列表示y
25      x = x_pop.dot(2 ** np.arange(DNA_SIZE)[::-1]) / float(2 ** DNA_SIZE - 1) *
    (X_BOUND[1] - X_BOUND[0]) + X_BOUND[0]
26      y = y_pop.dot(2 ** np.arange(DNA_SIZE)[::-1]) / float(2 ** DNA_SIZE - 1) *
    (Y_BOUND[1] - Y_BOUND[0]) + Y_BOUND[0]
27      return x, y
28
29
30  def getfitness(pop):
31      x, y = decodeDNA(pop)
32      temp = F(x, y)
33      print("平均适应度：", np.average(temp), "最小适应度：", np.min(temp))
34      return (temp - np.min(temp)) + 0.0001  # 减去最小的适应度是为了防止适应度出
    现负数
35
36
37  def select(pop, fitness):  # 根据适应度选择
38      fitness = fitness.max() - fitness
39      if fitness.sum() == 0:
40          temp = np.random.choice(np.arange(POP_SIZE), size=POP_SIZE,
    replace=True)
41      else:
42          temp = np.random.choice(np.arange(POP_SIZE), size=POP_SIZE,
    replace=True, p=fitness / fitness.sum())
43      return pop[temp]
44
45
46  def crossmuta(pop, CROSS_RATE):
47      new_pop = []
48      for i in pop:  # 遍历种群中的每一个个体，将该个体作为父代
49          temp = i  # 子代先得到父亲的全部基因
50          if np.random.rand() < CROSS_RATE:  # 以交叉概率发生交叉
51              j = pop[np.random.randint(POP_SIZE)]  # 从种群中随机选择另一个个体，
    并将该个体作为母代
```

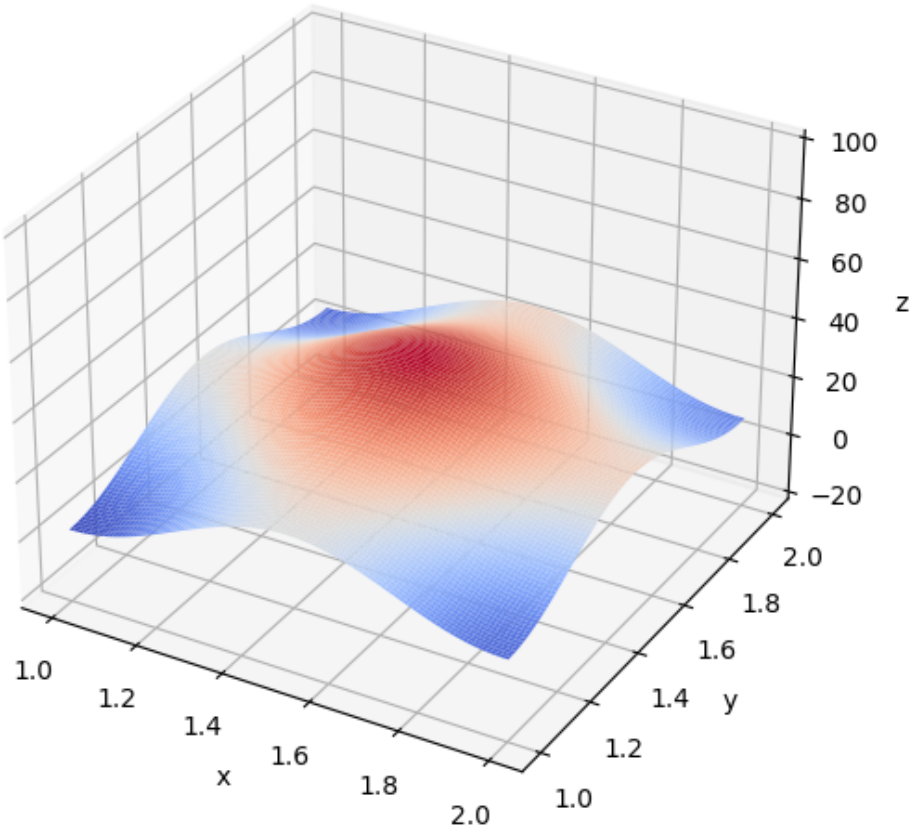
```
            cpoints1 = np.random.randint(0, DNA_SIZE * 2 - 1)  # 随机产生交叉的点
            cpoints2 = np.random.randint(cpoints1, DNA_SIZE * 2)
            temp[cpoints1:cpoints2] = j[cpoints1:cpoints2]  # 子代得到位于交叉点后的母代的基因
        mutation(temp, MUTA_RATE)  # 后代以变异率发生变异
        new_pop.append(temp)
    return new_pop


def mutation(temp, MUTA_RATE):
    if np.random.rand() < MUTA_RATE:  # 以MUTA_RATE的概率进行变异
        mutate_point = np.random.randint(0, DNA_SIZE)  # 随机产生一个实数，代表要变异基因的位置
        temp[mutate_point] = temp[mutate_point] ^ 1  # 将变异点的二进制为反转


def print_info(pop):  # 用于输出结果
    fitness = getfitness(pop)
    minfitness = np.argmin(fitness)  # 返回最小值的索引值
    print("min_fitness:", fitness[minfitness])
    x, y = decodeDNA(pop)
    print("最优的基因型：", pop[minfitness])
    print("(x, y):", (x[minfitness], y[minfitness]))
    print("F(x,y)_min = ", F(x[minfitness], y[minfitness]))


def plot_3d(ax):
    X = np.linspace(*X_BOUND, 100)
    Y = np.linspace(*Y_BOUND, 100)
    X, Y = np.meshgrid(X, Y)
    Z = F(X, Y)
    ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.coolwarm)
    ax.set_zlim(-20, 100)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')
    plt.pause(3)
    plt.show()


start_t = datetime.datetime.now()
if __name__ == "__main__":
```

```
    fig = plt.figure()
    ax = Axes3D(fig, auto_add_to_figure=False)
    fig.add_axes(ax)
    plt.ion()
    plot_3d(ax)

    pop = np.random.randint(2, size=(POP_SIZE, DNA_SIZE * 2))
    for _ in range(Iterations):  # 迭代N代
        x, y = decodeDNA(pop)
        if 'sca' in locals():
            sca.remove()
        sca = ax.scatter(x, y, F(x, y), c='black', marker='o')
        plt.show()
        plt.pause(0.1)
        pop = np.array(crossmuta(pop, CROSS_RATE))
        fitness = getfitness(pop)
        pop = select(pop, fitness)  # 选择生成新的种群
end_t = datetime.datetime.now()
print('花费时间：', (end_t - start_t).seconds)
print_info(pop)
plt.ioff()
plot_3d(ax)

```

设置不同的初始范围

编码	编码方式(population type)	二进制编码，长度为24位
种群参数	种群规模(population size)	100
	初始种群的个体取值范围(initial range)	[1,1.1]
选择操作	个体选择概率分配策略(对应 fitness scaling)	按照适应度随机选择
	个体选择方法(selection function)	计算适应度，选择适应度最高的
最佳个体保存	优良个体保存数量(elite count)	1
交叉操作	交叉概率(crossover fraction)	0.8
	交叉方式(crossover function)	随机选择另一个作为母代
变异操作	变异方式(mutation function)	二进制反转
停止参数	最大迭代步数(generations)	1000
	最大运行时间限制(time limit)	无
	最小适应度限制(fitness limit)	子代中最小的适应度
	停滞代数(stall generations)	无
	停滞时间限制(stall time limit)	无

[1,1.1]



花费时间： 115

平均适应度： 4.980282465401942 最小适应度： 4.980282465401942

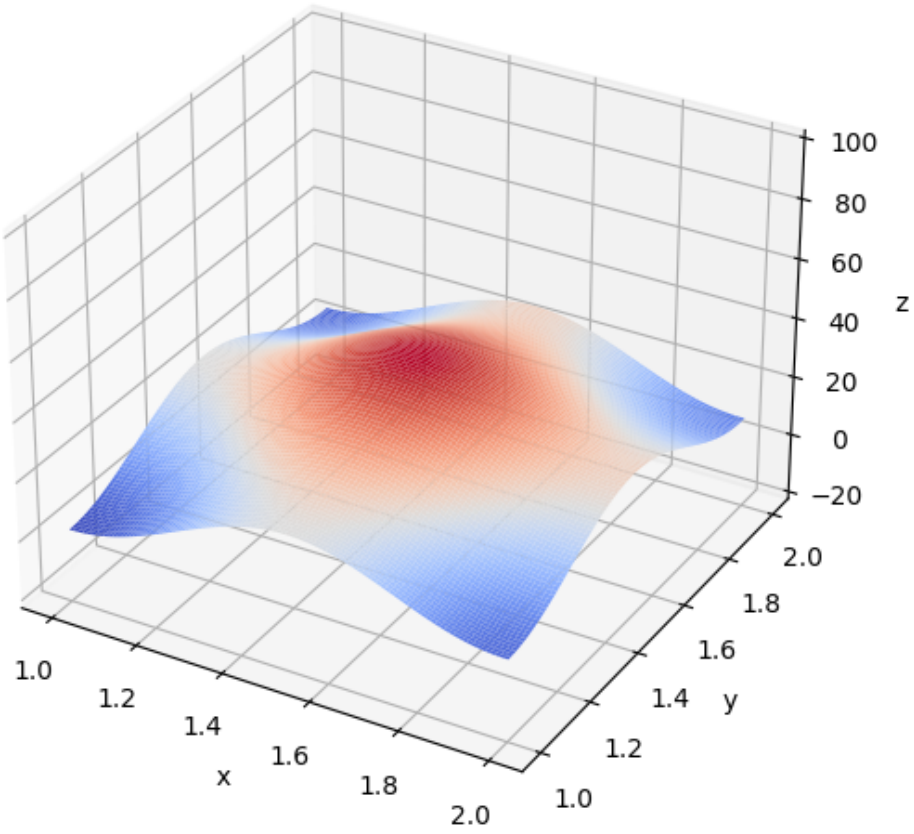
min_fitness: 0.0001

最优的基因型： [1 0 1 0 1 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 1 0 1 0 1 1 0 0 1 1 1 1 1 0 1 1 0 0 1 0 1 0 0 0 0 0 1 1]

(x, y): (1.0001717209918333, 1.9891945117231913)

F(x,y)_min = 4.980282465401942

[1,100]



花费时间： 115

平均适应度： 15.952031719172627 最小适应度： 13.397163667909066

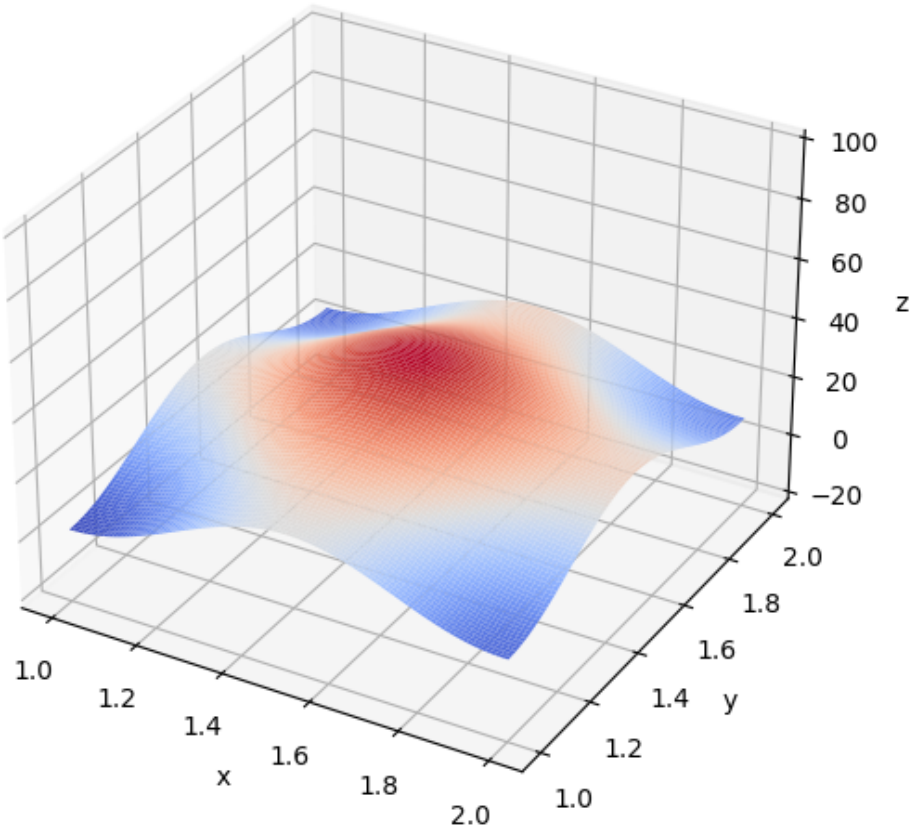
min_fitness: 0.0001

最优的基因型： [0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 1 1 1 1 0 0 0 0 1 0 0 1 1 0 1 0 1 0 0 0 1 1 1 0 0 0 0 1]

(x, y): (3.0076673631469824, 1.947040852727941)

F(x,y)_min = 13.397163667909066

[1,2]



花费时间： 115

平均适应度： 4.979950801847235 最小适应度： 4.9799499159127265

min_fitness: 0.0001

最优的基因型： [1 0 1 0 1 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 1 1 1 0 1 1 1 1 0 0 0 0 1 0 1 1 1 0 0]

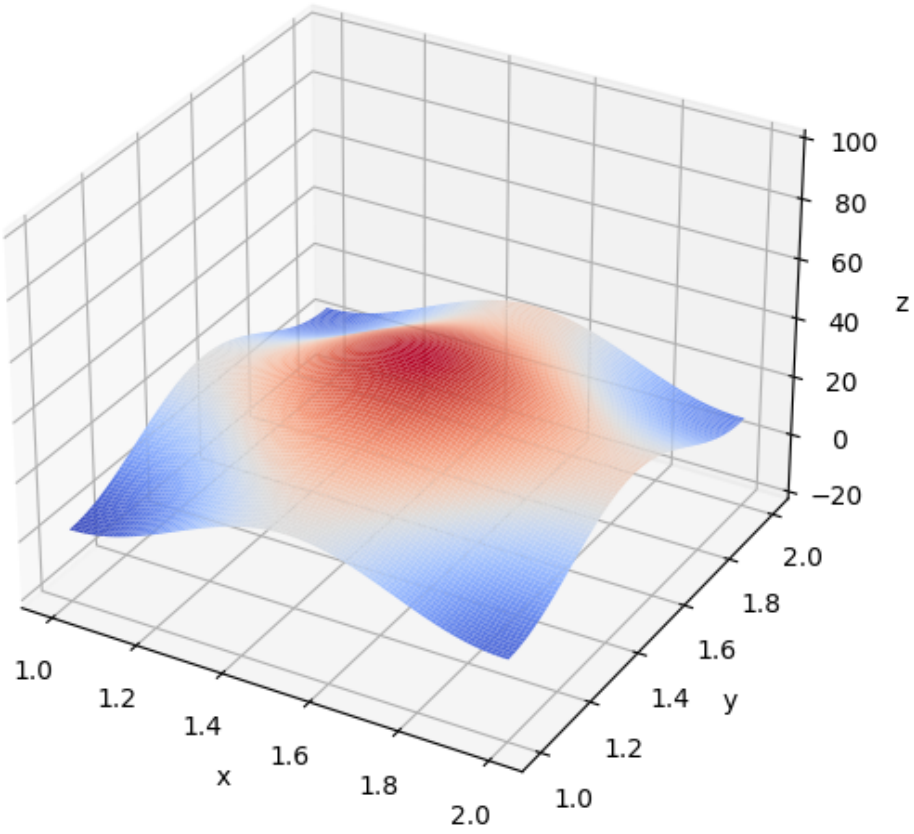
(x, y): (1.0000580549274716, 1.990011393428528)

F(x,y)_min = 4.9799499159127265

设置不同的交叉概率

编码	编码方式(population type)	二进制编码，长度为24位
种群参数	种群规模(population size)	100
	初始种群的个体取值范围(initial range)	[1,2]
选择操作	个体选择概率分配策略(对应 fitness scaling)	按照适应度随机选择
	个体选择方法(selection function)	计算适应度，选择适应度最高的
最佳个体保存	优良个体保存数量(elite count)	1
交叉操作	交叉概率(crossover fraction)	0
	交叉方式(crossover function)	随机选择另一个作为母代
变异操作	变异方式(mutation function)	二进制反转
停止参数	最大迭代步数(generations)	1000
	最大运行时间限制(time limit)	无
	最小适应度限制(fitness limit)	子代中最小的适应度
	停滞代数(stall generations)	无
	停滞时间限制(stall time limit)	无

变异率为**0**，交叉率为**1**



花费时间： 115

平均适应度： 9.55797530690818 最小适应度： 9.55797530690818

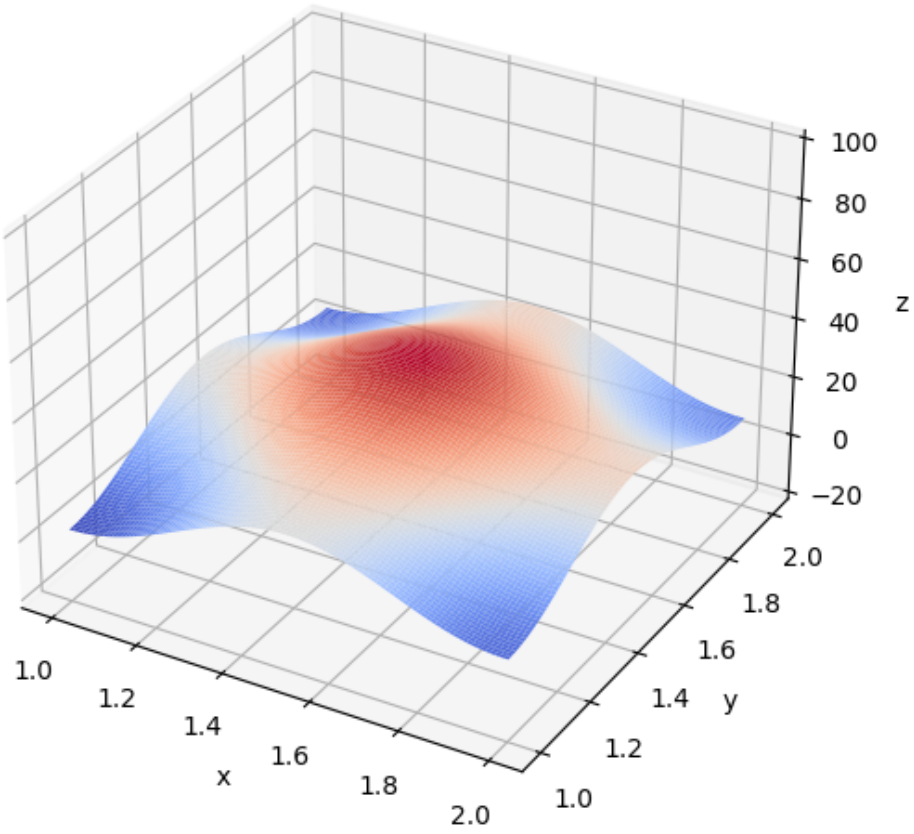
min_fitness: 0.0001

最优的基因型： [0 0 0 0 1 0 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 0 0 0 0 0 1 1 1 1 0 1 0 0 1 0 0 1 1 1 0 1 0 1]

(x, y): (1.1206330728908225, 1.1522457690385441)

F(x,y)_min = 9.55797530690818

变异率为**1**，交叉率为**0**



花费时间： 115

平均适应度： 5.724796559205404 最小适应度： 5.625833059631432

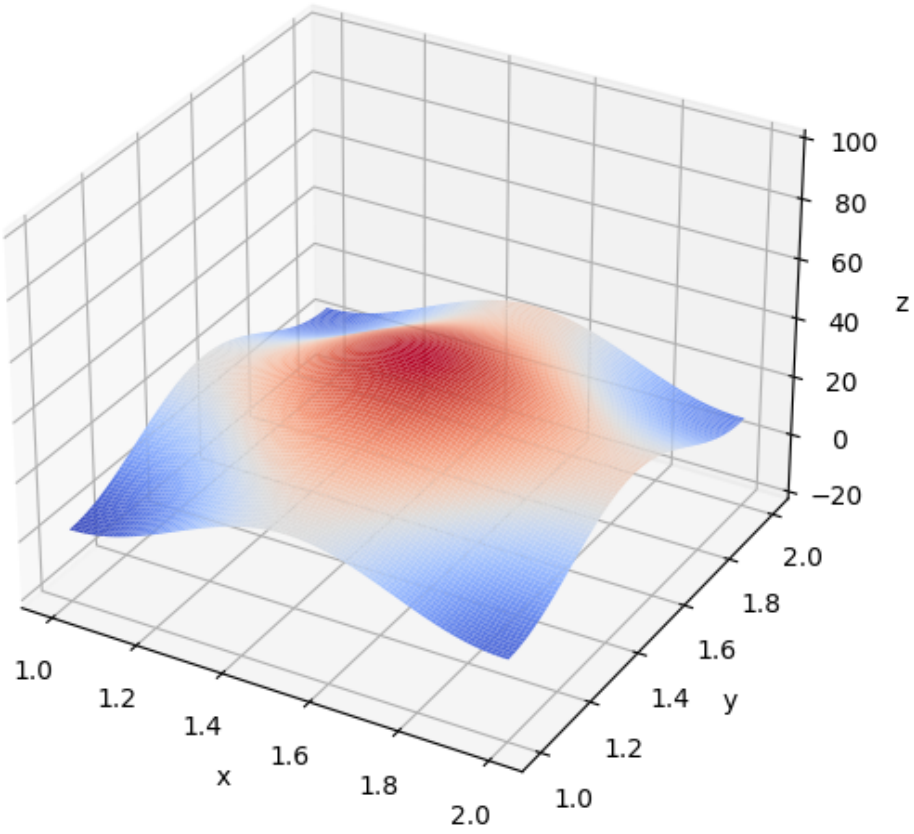
min_fitness: 0.0001

最优的基因型： [1 0 1 0 1 0 1 0 0 1 1 0 0 1 1 1 1 0 0 0 0 1 0 0 1 0 0 0 0 0 1 1 1 1 1 1 0 0 1 0 1 1 0 1 0 0 1 0]

(x, y): (1.043484451978472, 1.9591346954783617)

F(x,y)_min = 5.625833059631432

变异率为**0.15**，交叉率为**0.8**



花费时间： 115

平均适应度： 4.991693595426156 最小适应度： 4.991693595426156

min_fitness: 0.0001

最优的基因型： [0 1 0 1 0 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 1 1 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 0 0]

(x, y): (1.9823113073296135, 1.000213980687498)

F(x,y)_min = 4.991693595426156

3、总结遗传算法的特点，并说明适应度函数在遗传算法中的作用；

遗传算法的特点

- 遗传算法 从问题解的中集开始嫂索，而不是从单个解开始。 这是 遗传算法 与传统优化 算法 的极大区别。 ...
- 遗传算法 求解时使用特定问题的信息极少，容易形成通用 算法 程序。 ...
- 遗传算法 有极强的容错能力 ...
- 遗传算法 中的选择、交叉和变异都是随机操作，而不是确定的精确规则。 ...
- 遗传算法 具有隐含的并行性

适应度函数在遗传算法中的作用

在遗传算法中,适应度是描述个体性能的主要指标.根据适应度的大小,对个体进行优胜劣汰.适应度是驱动遗传算法的动力.从生物学角度讲,适应度相当于“生存竞争、适者生存”的生物生存能力,在遗传过程中具有重要意义.将优化问题的目标函数与个体的适应度建立映射关系,即可在群体进化过程中实现对优化问题目标函数的寻优.适应度函数也称评价函数,是根据目标函数确定的用于区分群体中个体好坏的标准,总是非负的,任何情况下都希望它的值越大越好.在选择操作中,会出现2个成为遗传算法欺骗的问题:

1. 在遗传算法初期,通常会产生一些超常个体,按照比例选择法,这些超常个体会因竞争力突出,而控制选择过程,影响到算法的全局优化性能;
2. 遗传算法后期,当算法趋于收敛时,由于种群中个体适应度差异较小,继续优化的潜能降低,可能获得某个局部最优解.因此,如果适应度函数选择不当就会产生以上的欺骗问题.可见适应度函数的选择对于遗传算法的意义重大.