華東理工大學

EAST CHINA UNIVERSITY OF SCIENCE AND TECHNOLOGY

《 人工智能 》

实验报告本

班　级：　计 203

学　号：　20002462

姓　名：　刘子言

指导教师：　陈志华

信息科学与工程学院

2022 年 12 月

《人工智能》实 验 报 告 三

实验 3 名称：遗传算法求最值问题实验	**实验地点**：信息楼 215

所使用的工具软件及环境：

Python 版本：Python 3 及以上

一、实验目的

熟悉和掌握遗传算法的原理、流程和编码策略，并利用遗传求解函数优化问题，理解求解流程并测试主要参数对结果的影响。

二、实验原理

遗传算法的基本思想正是基于模仿生物界遗传学的遗传过程。它把问题的参数用基因代表，把问题的解用染色体代表（在计算机里用二进制码表示），从而得到一个由具有不同染色体的个体组成的群体。这个群体在问题特定的环境里生存竞争，适者有最好的机会生存和产生后代。后代随机化地继承了父代的最好特征，并也在生存环境的控制支配下继续这一过程．群体的染色体都将逐渐适应环境，不断进化，最后收敛到一族最适应环境的类似个体，即得到问题最优的解。

三、实验要求

1. **用遗传算法求解下列函数的最大值，设定求解精度到 15 位小数。**

$$f(x,y) = \frac{6.452(x+0.125y)(\cos(x)-\cos(2y))^2}{0.8+(x-4.2)^2+2(y-7)^2}+3.226y$$

$$x\in[0,10), y\in[0,10)$$

1）设计及选择上述问题的编码、选择操作、交叉操作、变异操作以及控制参数等，填入表 1,并画出最佳适应度(Best fitness)和最佳个体（Best individual）图。

表 1 遗传算法参数的选择

编码	编码长度（population Length）	

种群参数	种群规模（population size）	
迭代次数	多种次数实验	
交叉操作	交叉概率	
	交叉方式	
变异操作	变异率	
参数区间	X 区间范围	
	Y 区间范围	

2）设置不同的种群规模，例如，求得相应的最佳适应度，并给出算法的运行时间，分析种群规模对算法性能的影响。

2.**用遗传算法求解下面一个 Rastrigin 函数的最小值，设定求解精度到 15 位小数。**

$$f(x,y) = 20 + x^2 + y^2 - 10(\cos 2\pi x + \cos 2\pi y)$$

$$x \in [1,2], y \in [1,2]$$

1）设计及选择上述问题的编码、选择操作、交叉操作、变异操作以及控制参数等，填入表 1,并画出最佳适应度(Best fitness)和最佳个体（Best individual）图。

表 2 遗传算法参数的选择

编码	编码长度（population Length）	
种群参数	种群规模（population size）	
迭代次数	多种次数实验	
交叉操作	交叉概率	
	交叉方式	
变异操作	变异率	
参数区间	X 区间范围	
	Y 区间范围	

2）设置不同的种群规模，例如，求得相应的最佳适应度，并给出算法的运行时间，分析种群规模对算法性能的影响。

四、实验步骤

1、编写相关遗传算法的代码，求取最大、最小值。

改进点：将各参数的设置方式改为了键盘输入；添加了算法运行时间的计算公式，更直观地反映算法性能。

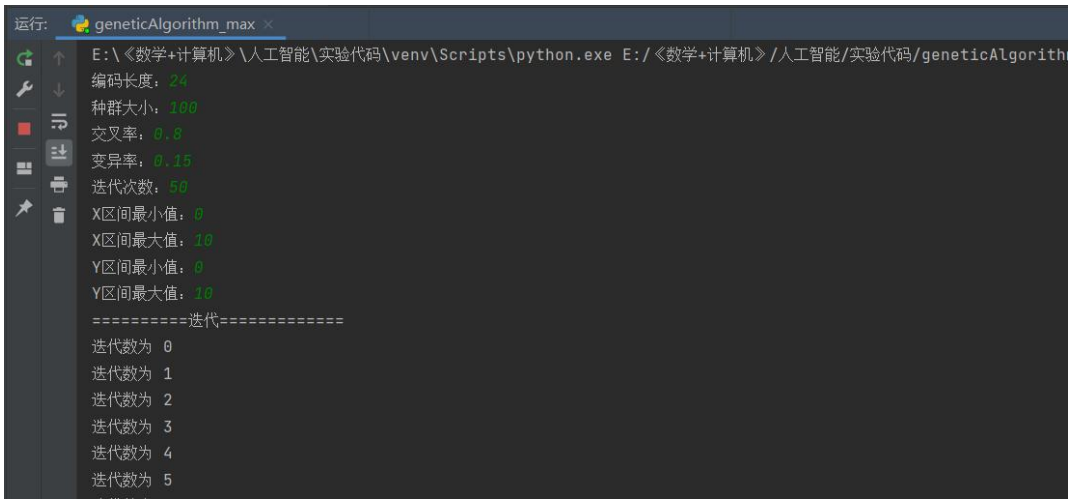
2、用遗传算法求解函数 f(x,y)的最大值，设定求解精度到 15 位小数

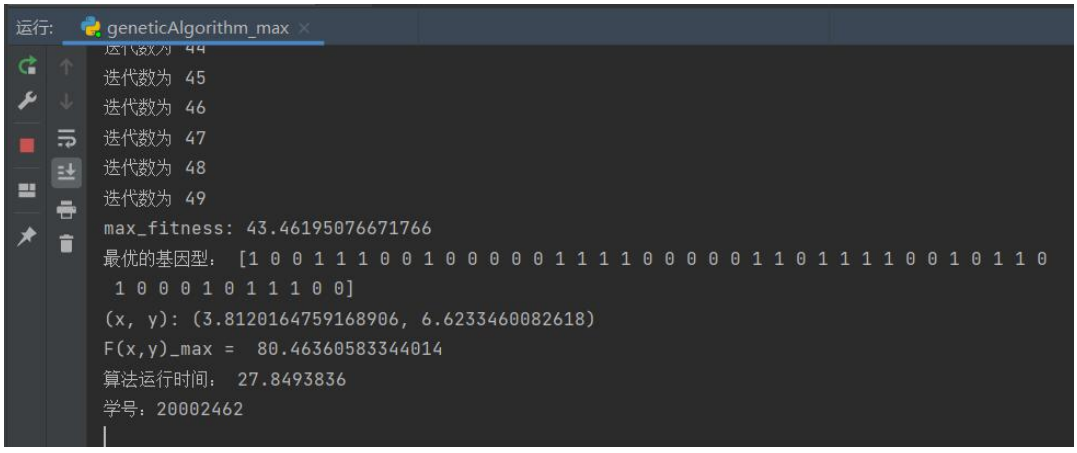
（1）设计及选择上述问题的编码、选择操作、交叉操作、变异操作以及控制参数等，填入表 1,并画出最佳适应度(Best fitness)和最佳个体（Best individual）图。

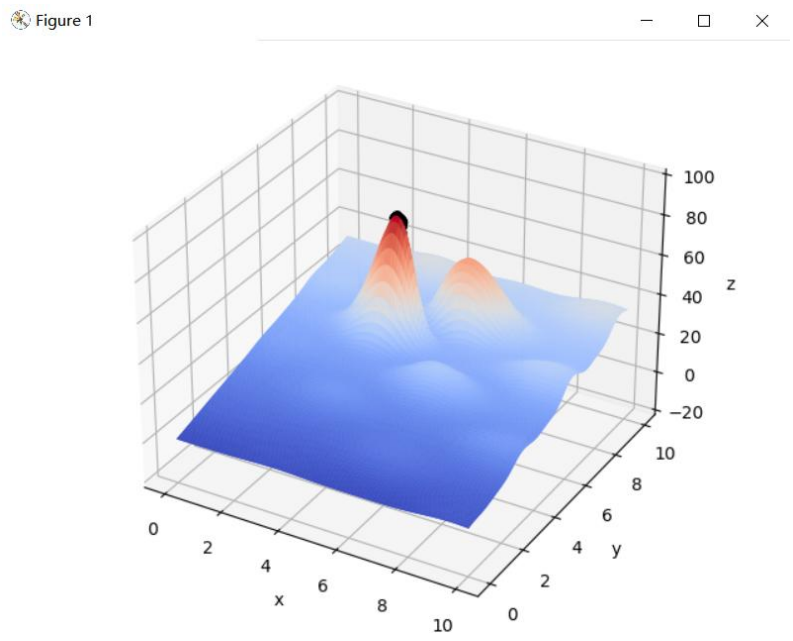
表 1 遗传算法参数的选择

编码	编码长度（population Length）	24
种群参数	种群规模（population size）	100
迭代次数	多种次数实验	50
交叉操作	交叉概率	0.8
	交叉方式	两点交叉
变异操作	变异率	0.15
参数区间	X 区间范围	[0,10]
	Y 区间范围	[0,10]

最佳适应度(Best fitness)和最佳个体(Best individual)的数据及图如下：



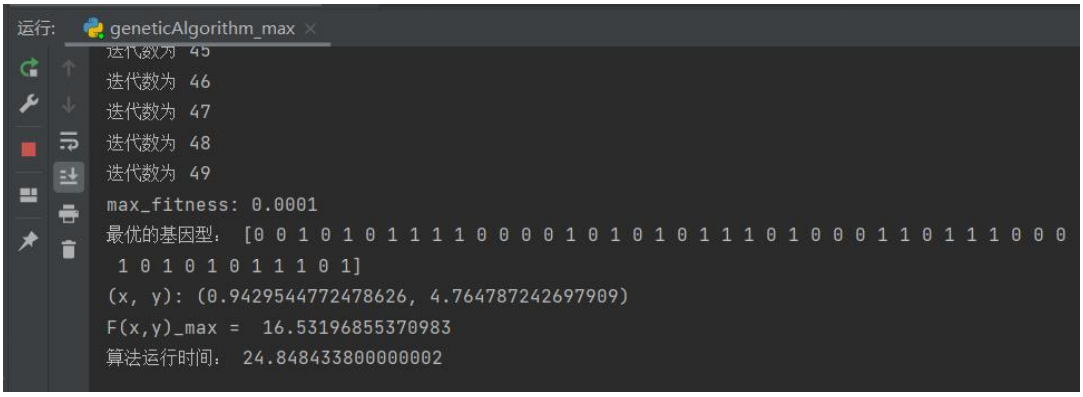




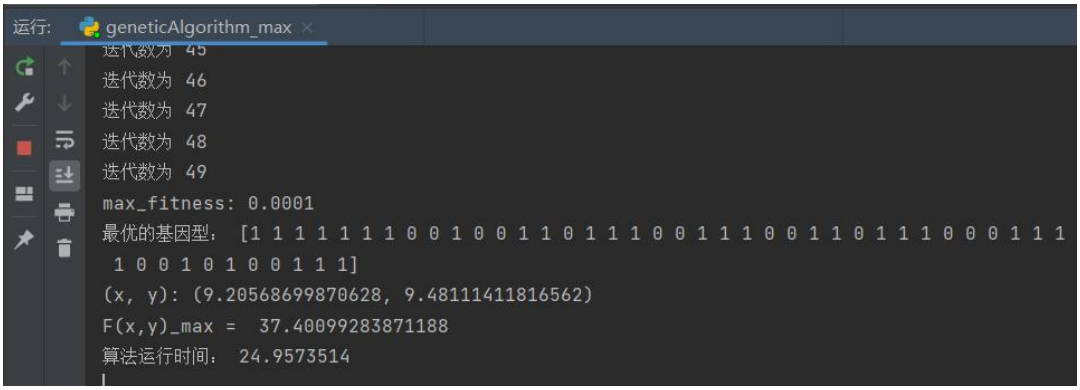
（2）设置不同的种群规模，求得相应的最佳适应度，并给出算法的运行时间，分析种群规模对算法性能的影响。

设置种群规模为 5、10、20、50、100，初始种群的个体取值范围为[0,10]，其他参数与（1）中相同：

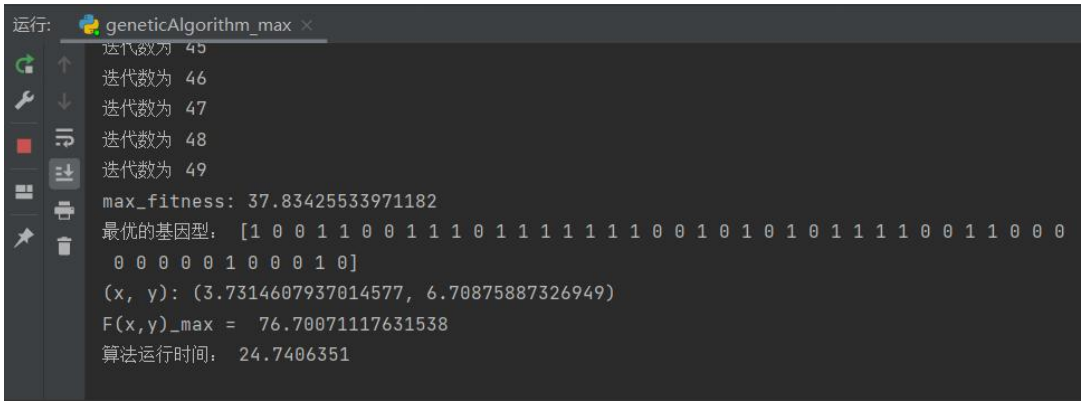
• 种群规模为 5 时



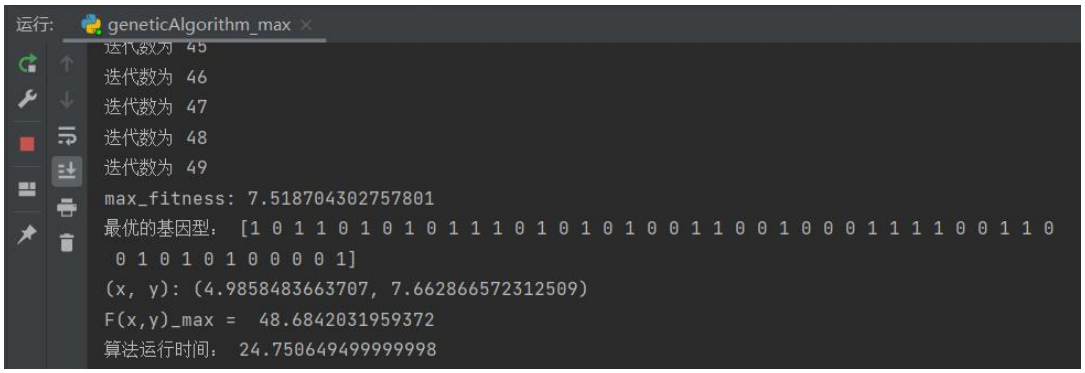
• 种群规模为 10 时



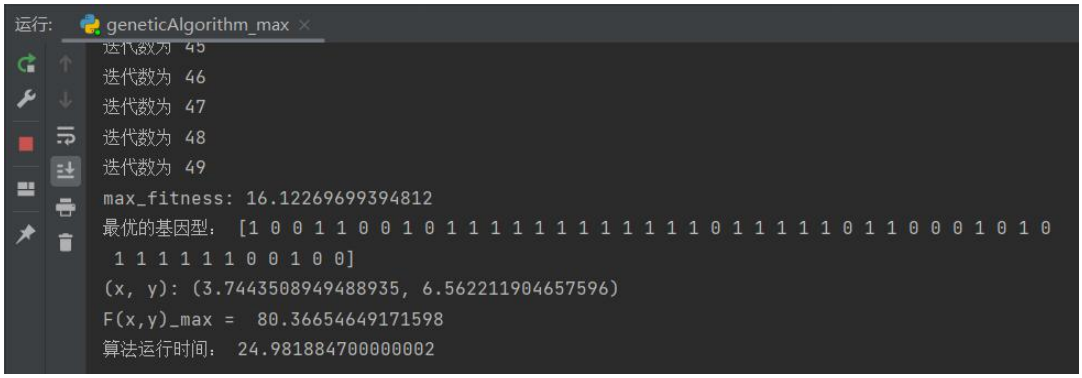
• 种群规模为 20 时



• 种群规模为 50 时



• 种群规模为 100 时



整理得到相应的最佳适应度、最佳个体以及算法运行时间如下表：

种群规模	最佳适应度	最佳个体		算法运行时间/s
		x	y	
5	16.53197	0.9430	4.7648	24.8484338
10	37.40099	9.2057	9.4811	24.9573514
20	76.70071	3.7315	6.7088	24.7406351
50	48.68420	4.9858	7.6629	24.7506495
100	80.36655	3.7444	6.5622	24.9818847

由实验过程及上表分析可知，种群规模会影响遗传优化的结果和效率：

当种群规模太小时，遗传算法的优化性能一般不会太好，容易陷入局部最优解；而当种群规模太大时，则计算复杂，会增加算法运行的时间，降低算法的性能。因此，种群规模的选择需要适中。

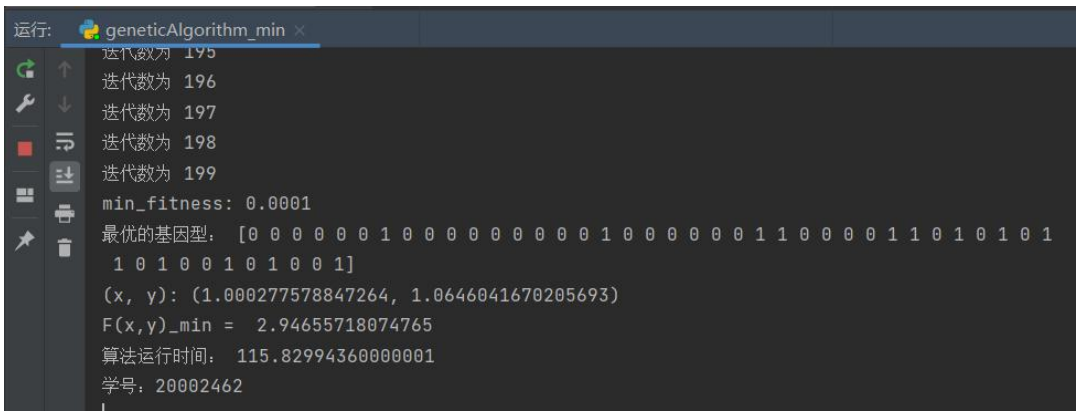
3、用遗传算法求解 Rastrigin 函数的最小值，设定求解精度到 15 位小数。

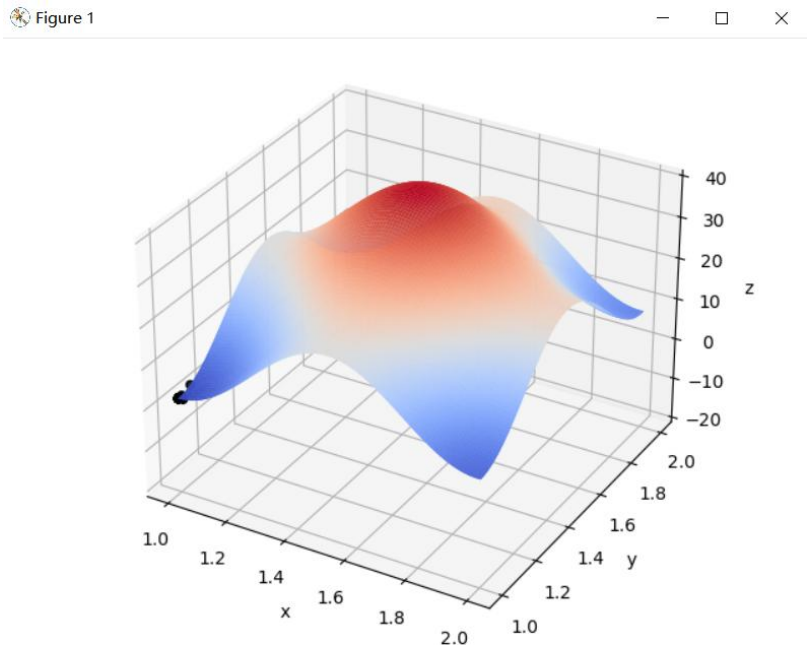
（1）设计及选择上述问题的编码、选择操作、交叉操作、变异操作以及控制参数等，填入表 2，并画出最佳适应度(Best fitness)和最佳个体（Best individual）图。

表 2 遗传算法参数的选择

编码	编码长度（population Length）	24
种群参数	种群规模（population size）	200
迭代次数	多种次数实验	200
交叉操作	交叉概率	0.5
	交叉方式	两点交叉
变异操作	变异率	0.015
参数区间	X 区间范围	[1,2]
	Y 区间范围	[1,2]

最佳适应度(Best fitness)和最佳个体(Best individual)的数据及图如下：





（2）设置不同的种群规模，求得相应的最佳适应度，并给出算法的运行时间，分析种群规模对算法性能的影响。

设置种群规模为 10、50、100、200，初始种群的个体取值范围为[1,2]，其他的参数与（1）中相同：

• 种群规模为 10 时

```
运行:  geneticAlgorithm_min
迭代数为 196
迭代数为 197
迭代数为 198
迭代数为 199
min_fitness: 0.0001
最优的基因型： [0 1 0 1 0 1 1 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 1 0 0 1 0 1 1 0 1 0 0 0
 1 1 1 1 0 1 0 1 1 1 0]
(x, y): (1.990870654038826, 1.0625476278393047)
F(x,y)_min =  5.871370833416517
算法运行时间： 142.6225839
```

• 种群规模为 50 时

```
运行:  geneticAlgorithm_min
迭代数为 196
迭代数为 197
迭代数为 198
迭代数为 199
min_fitness: 0.0001
最优的基因型： [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 1 1 0 1
 0 1 1 0 0 1 0 0 1 1 0]
(x, y): (1.0000392198586, 1.0000985264836864)
F(x,y)_min =  2.0002777237355254
算法运行时间： 145.9278598
```

• 种群规模为 100 时

```
运行:  geneticAlgorithm_min
迭代数为 196
迭代数为 197
迭代数为 198
迭代数为 199
min_fitness: 0.0001
最优的基因型： [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0
 1 1 0 1 0 1 1 0 0 1 0]
(x, y): (1.0000059604648328, 1.0039690139275201)
F(x,y)_min =  2.0110750794421506
算法运行时间： 109.6899361
```

• 种群规模为 200 时

```
运行:  geneticAlgorithm_min
迭代数为 196
迭代数为 197
迭代数为 198
迭代数为 199
min_fitness: 0.0001
最优的基因型： [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 1 0 1 0 0 1 1 1
 1 1 0 0 0 0 0 0 1 1 0]
(x, y): (1.0000516176254521, 1.0101690894466095)
F(x,y)_min =  2.040950800498159
算法运行时间： 158.06644219999998
```

整理得到相应的最佳适应度、最佳个体以及算法运行时间见下表：

种群规模	最佳适应度	最佳个体		算法运行时间/s
		x	y	
10	5.87137	1.9909	1.0625	142.6225839
50	2.00027	1.0000	1.0001	145.9278598
100	2.01108	1.0000	1.0040	109.6899361
200	2.04095	1.0001	1.0102	158.0664422

由实验过程及上表分析，得到的结论与利用遗传算法求最大值时得到的结论基本相同，种群规模会影响遗传优化的结果和效率：当种群规模太小时，遗传算法的优化性能一般不会太好，容易陷入局部最优解；而当种群规模太大时，则计算复杂，会增加算法运行的时间，降低算法的性能。因此，种群规模的选择需要适中。

五、 程序设计的核心代码

1、遗传算法的代码设计（**以求最大值的代码为例**）：

```
def F(x, y):    # 适应度函数
    return (6.452 * (x + 0.125 * y) * (cos(x) - cos(2 * y)) ** 2) / (
            0.8 + (x - 4.2) ** 2 + 2 * (y - 7) ** 2) + 3.226 * y

def decodeDNA(pop):    # 解码
    x_pop = pop[:, 1::2]    # 奇数列表示X
    y_pop = pop[:, ::2]    # 偶数列表示y
    x = x_pop.dot(2 ** np.arange(DNA_SIZE)[::-1]) / float(2 ** DNA_SIZE - 1) *
(X_BOUND[1] - X_BOUND[0]) + X_BOUND[0]
    y = y_pop.dot(2 ** np.arange(DNA_SIZE)[::-1]) / float(2 ** DNA_SIZE - 1) *
(Y_BOUND[1] - Y_BOUND[0]) + Y_BOUND[0]
    return x, y

def getfitness(pop):
    x, y = decodeDNA(pop)
    temp = F(x, y)
    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

def select(pop, fitness):    # 根据适应度选择
    temp = np.random.choice(np.arange(POP_SIZE), size=POP_SIZE, replace=True,
p=(fitness) / (fitness.sum()))
    return pop[temp]

def crossmuta(pop, CROSS_RATE):
    new_pop = []
    for i in pop:    # 遍历种群中的每一个个体，将该个体作为父代
```

```
            temp = i    # 子代先得到父亲的全部基因
            if np.random.rand() < CROSS_RATE:    # 以交叉概率发生交叉
                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)
        maxfitness = np.argmax(fitness)    # 返回最大值的索引值
        print("max_fitness:", fitness[maxfitness])
        x, y = decodeDNA(pop)
        print("最优的基因型：", pop[maxfitness])
        print("(x, y):", (x[maxfitness], y[maxfitness]))
        print("F(x,y)_max = ", F(x[maxfitness], y[maxfitness]))

def plot_3d(ax):    # 绘制3D图像
        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.show()

if __name__ == "__main__":
        ......
        fig = plt.figure()
        ax = Axes3D(fig)
        plt.ion()
```

```
plot_3d(ax)
......
pop = np.random.randint(2, size=(POP_SIZE, DNA_SIZE * 2))
print('==========迭代==========')
for _ in range(Iterations):  # 迭代N代
    print("迭代数为", _)
    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
print_info(pop)
plt.ioff()
......
plot_3d(ax)
```

2、将各参数的设置方式改为了键盘输入：

```
# 输入参数：
DNA_SIZE = int(input("编码长度："))  # 编码长度
POP_SIZE = int(input("种群大小："))  # 种群大小
CROSS_RATE = float(input("交叉率："))  # 交叉率
MUTA_RATE = float(input("变异率："))  # 变异率
Iterations = int(input("迭代次数："))  # 迭代次数
X_BOUND.append(int(input("X区间最小值：")))
X_BOUND.append(int(input("X区间最大值：")))
Y_BOUND.append(int(input("Y区间最小值：")))
Y_BOUND.append(int(input("Y区间最大值：")))
```

3、添加了算法运行时间的计算公式，反映算法性能：

```
start = time.perf_counter()  # 算法运行的开始时间
......
end = time.perf_counter()  # 算法运行的结束时间
print("算法运行时间：", end - start)
```

六、实验体会

通过本次实验，我对遗传算法的基本思想、原理和一般步骤有了更加深入的了解。在掌握了编码、群体设定、适应度函数、选择、交叉、变异等知识的基础之上，我学习了如何利用 Python 设计具体的算法步骤，并且最终通过编程实现了用遗传算法求具体函数的最大值最小值，求得相应的最佳适应度、最佳个体及算法的运行时间，收获颇丰。

在实验过程中，通过改变种群规模的大小，也观察到了种群规模对遗传优化的结果和效率、算法性能等的影响，总结了以下的思考和结论：

当种群规模太小时，遗传算法的优化性能一般不会太好，容易陷入局部最优解；而当种群规模太大时，则计算复杂，会增加算法运行的时间，降低算法的性能。因此，种群规模的选择需要适中。

种群规模的确定受遗传操作中选择操作的影响很大。模式定理表明：若种群规模为M，则遗传操作可从这 M 个个体中生成和检测 M^3个模式，并在此基础上能够不断形成和优化积木块，直到找到最优解。

一方面，种群规模越大，遗传操作所处理的模式就越多，产生有意义的积木块并逐步进化为最优解的机会就越高。种群规模太小，会使遗传算法的搜索空间范围有限，因而搜索有可能停止在未成熟阶段，出现未成熟收敛现象，使算法陷人局部最优解。因此，必须保持种群的多样性，即种群规模不能太小。

另一方面，种群规模太大会带来若干弊病：一是群体越大，其适应度评估次数增加，所以计算量也增加，从而影响算法效率；二是群体中个体生存下来的概率大多采用和适应度成比例的方法，当群体中个体非常多时，少量适应度很高的个体会被选择而生存下来，但大多数个体却被淘汰，这会影响配对库的形成，从而影响交叉操作。

所以根据经验，种群规模一般取为 20～100。

七、教师评语

该学生_____完成了实验任务，算法设计_____，实验结果_____，实验体会_____。

因此总体评价为_____。

教师签字：

年　月　日