更浅地谈遗传算法

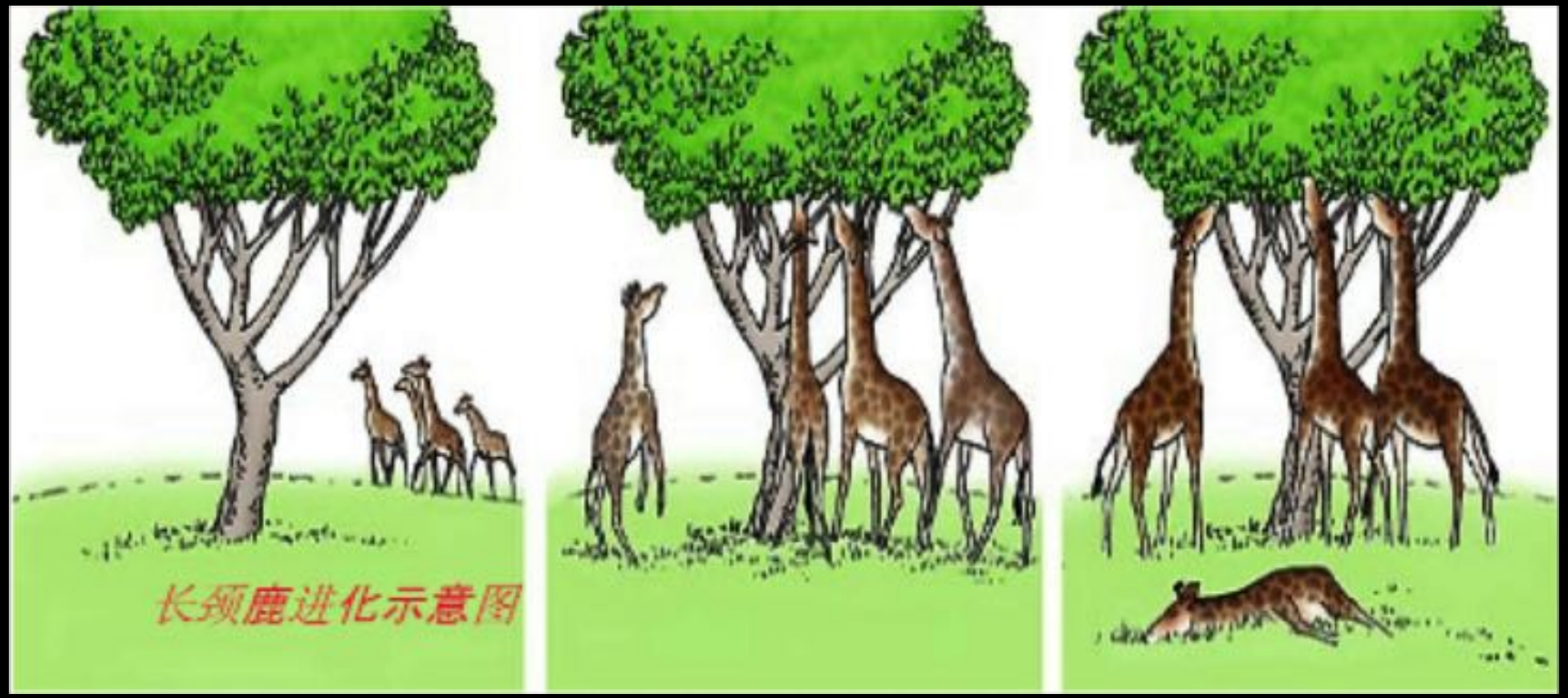
冀锐 ~~可能是本课最帅的助教子~~

什么是遗传算法？

遗传算法是一种解决问题的方法。它模拟大自然中种群在选择压力下的演化，从而得到问题的一个近似解。

核心思想：

遗传变异、自然选择（~~这与革命的李森科环境学说不符！~~）



现在遗传算法讲得已经很清楚了

你们应该都明白了

没明白的也不要再问了

下课

三角形拟合图片（纯娱乐项目）

寻找函数区间内最值（有点实用性？）

三角形拟合图片

从一个故事讲起

从前有一群跑得没有记者快的扇贝

总有一个山东大汉来挖走它们之中的一部分

这厮的夫人不喜欢Firefox，因为她用360浏览器

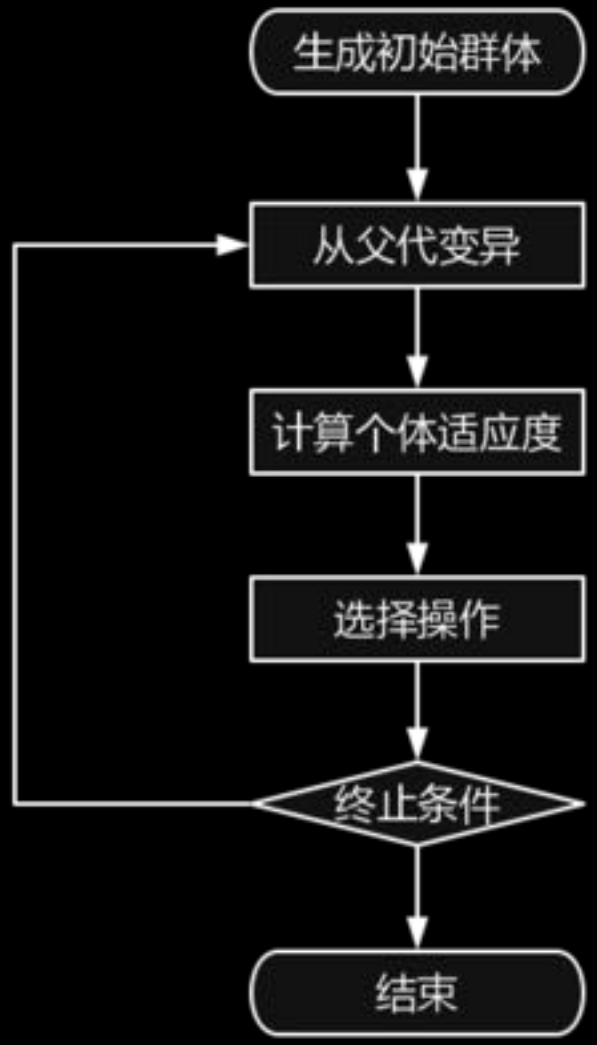
~~花纹长得比较不像Firefox图标的扇贝~~

几十万代之后……

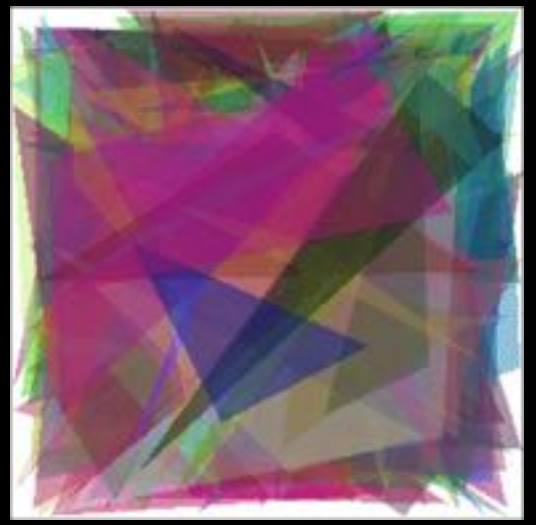
Firefox 身上纹 大家都是社会人（贝？

这些扇贝生活在电脑的内存中。它们是一个遗传算法程序的一部分，这个程序的目的就是用256个半透明三角形把Firefox的图标尽可能像地画出来

图像拟合



目标图像



生成图像

Talk is Cheap
Show me the code

扇贝花纹的颜色

```
class Color(object):
    '''
    定义颜色的类，这个类包含r,g,b,a表示颜色属性
    '''
    def __init__(self):
        self.r = r.randint(0, 255)
        self.g = r.randint(0, 255)
        self.b = r.randint(0, 255)
        self.a = r.randint(95, 115)
```

扇贝的小花纹

```
class Triangle(object):
    '''
    定义三角形的类
    属性:
            ax,ay,bx,by,cx,cy: 表示每个三角形三个顶点的坐标
            color            : 表示三角形的颜色
            img_t            : 三角形绘制成的图，用于合成图片
    方法:
            mutate_from(self, parent):       从父代三角形变异
            draw_it(self, size=(256, 256)): 绘制三角形
    '''
```

扇贝

```
class Canvas(object):
    '''
    定义每一张图片的类
    属性:
            mutate_rate   : 变异概率
            size          : 图片大小
            target_pixels: 目标图片像素值
    方法:
            add_triangles(self, num=1)        : 在图片类中生成num个三角形
            mutate_from_parent(self, parent): 从父代图片对象进行变异
            calc_match_rate(self)             : 计算环境适应度
            draw_it(self, i)                  : 保存图片
    '''
```

图像拟合

瞅瞅你长得像
不像Firefox

```
def calc_match_rate(self):
        do something...
         # 分解为RGBA四通道
        arrs = [np.array(x) for x in list(self.img.split())]
        for i in range(3):
            # 对RGB通道三个矩阵分别与目标图片相应通道作差取平方加和
            # 评估相似度
            self.match_rate += np.sum(
              np.square(arrs[i]-self.target_pixels[i]))[0]
```

图像拟合

```
def main():
        读入目标图片
        生成一系列的Canvas作为父本
        选择其中最好的一个进行遗传，删掉其它Canvas
        # 进入遗传算法的循环
        i = 0
        while i < 30000:
            每一代从父代中变异出10个个体
            选择其中适应度最好的一个个体
            如果子代比父代更适应环境，那么子代成为新的父代
            否则保持原样
            if i % LOOP == 0:
                # 每隔LOOP代保存一次图片
            i += 1
```

第一代遭到
毒手的扇贝

不知疲倦的
山东大汉

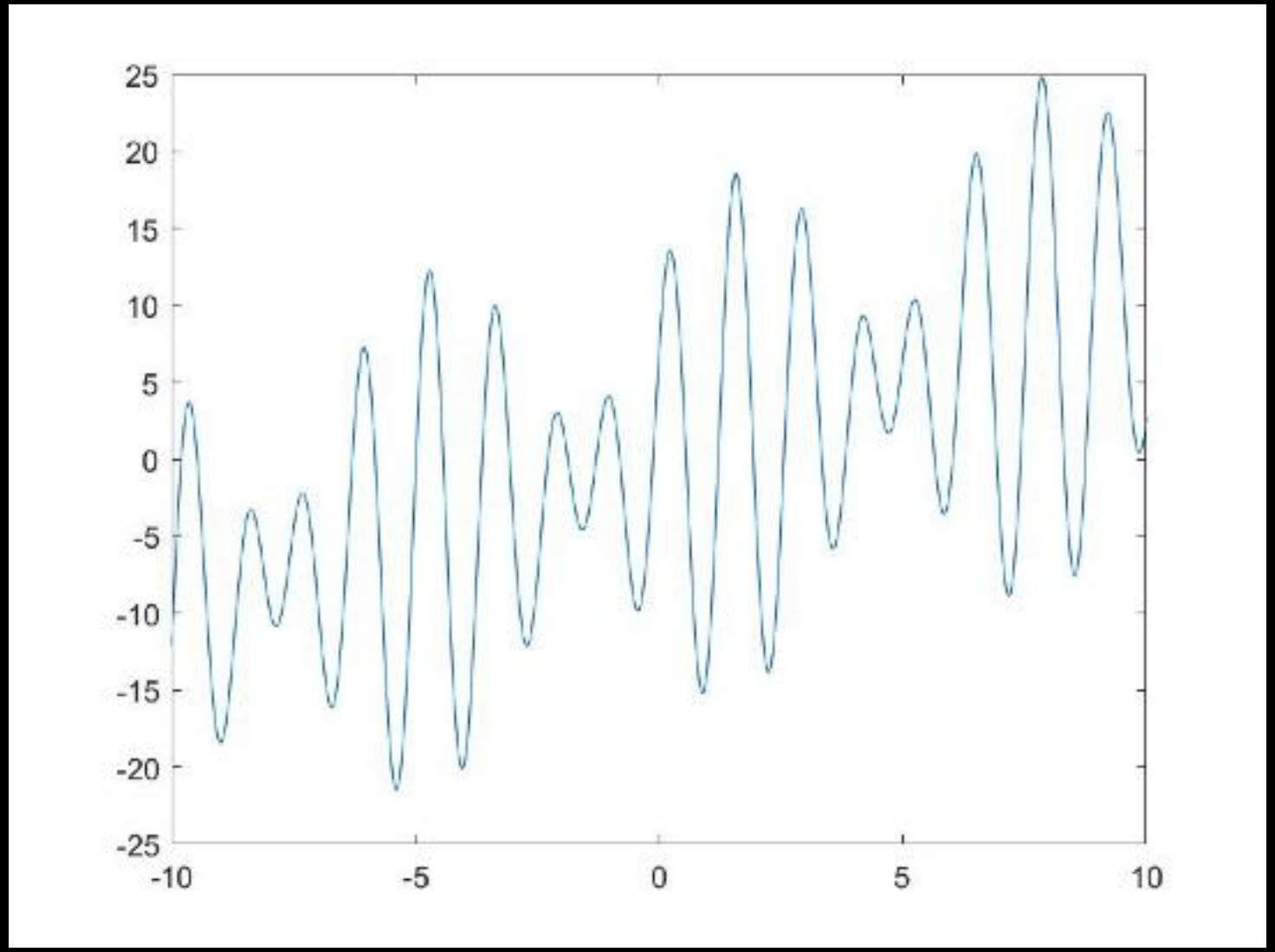
图像拟合

故事的结局

夕阳下海岸边，经常有人看到，一个暗搓搓的身影孤独地打磨着贝壳

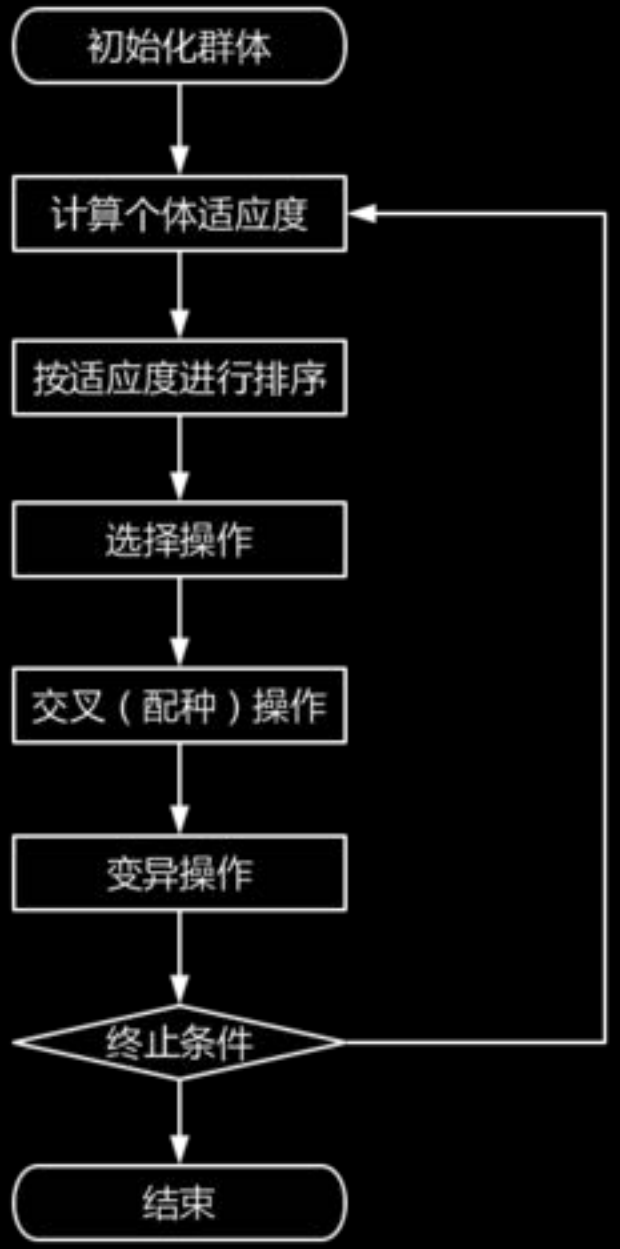
寻找函数区间内最值

函数最值



$$f(x) = x + 10 * \sin(5 * x) + 7 * \cos(4 * x)$$

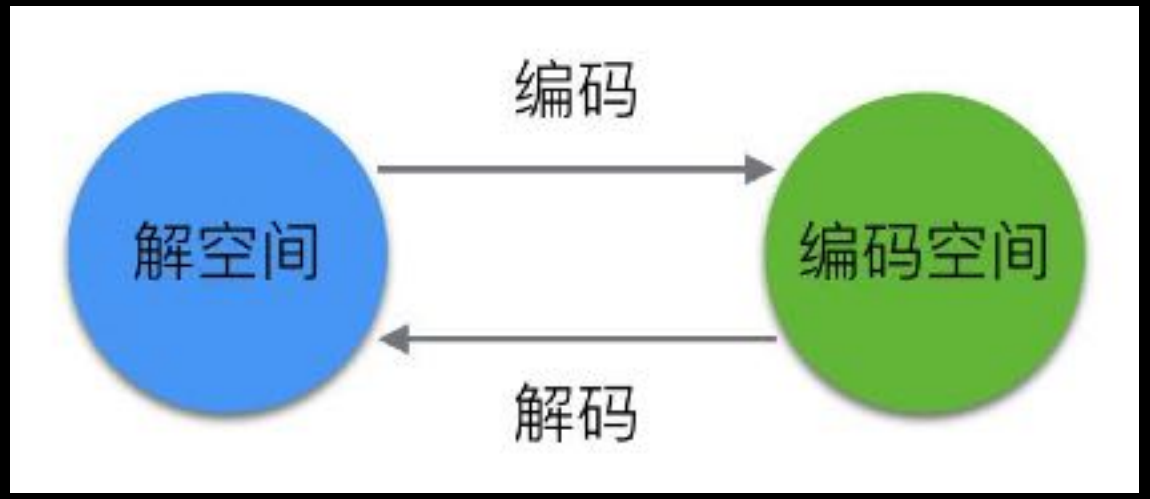
函数最值



对于函数优化问题，一般有两种编码方式，各具优缺点

实数编码：直接用实数表示基因，容易理解且不需要解码过程，但容易过早收敛，从而陷入局部最优

二进制编码：稳定性高，种群多样性大，但需要的存储空间大，需要解码且难以理解



函数最值

```python
# 对个体进行评价
def calobjValue(pop, chrom_length, max_value):
    temp1 = []
    obj_value = []
    temp1 = decodechrom(pop, chrom_length)
    for i in range(len(temp1)):
        x = temp1[i] * max_value / (math.pow(2, chrom_length) - 1)
        obj_value.append(10 * math.sin(5 * x) + 7 * math.cos(4 * x))
    return obj_value
```

目标函数

函数最值

```
# 淘汰（去除负值）
def calfitValue(obj_value):
    fit_value = []
    c_min = 0
    for i in range(len(obj_value)):
        if(obj_value[i] + c_min > 0):
            temp = c_min + obj_value[i]
        else:
            temp = 0.0
        fit_value.append(temp)
    return fit_value
```

函数最值

```
# 交配
def crossover(pop, pc):
    pop_len = len(pop)
    for i in range(pop_len - 1):
        if(random.random() < pc):
            cpoint = random.randint(0,len(pop[0]))
            temp1 = []
            temp2 = []
            temp1.extend(pop[i][0:cpoint])
            temp1.extend(pop[i+1][cpoint:len(pop[i])])
            temp2.extend(pop[i+1][0:cpoint])
            temp2.extend(pop[i][cpoint:len(pop[i])])
            pop[i] = temp1
            pop[i+1] = temp2
```

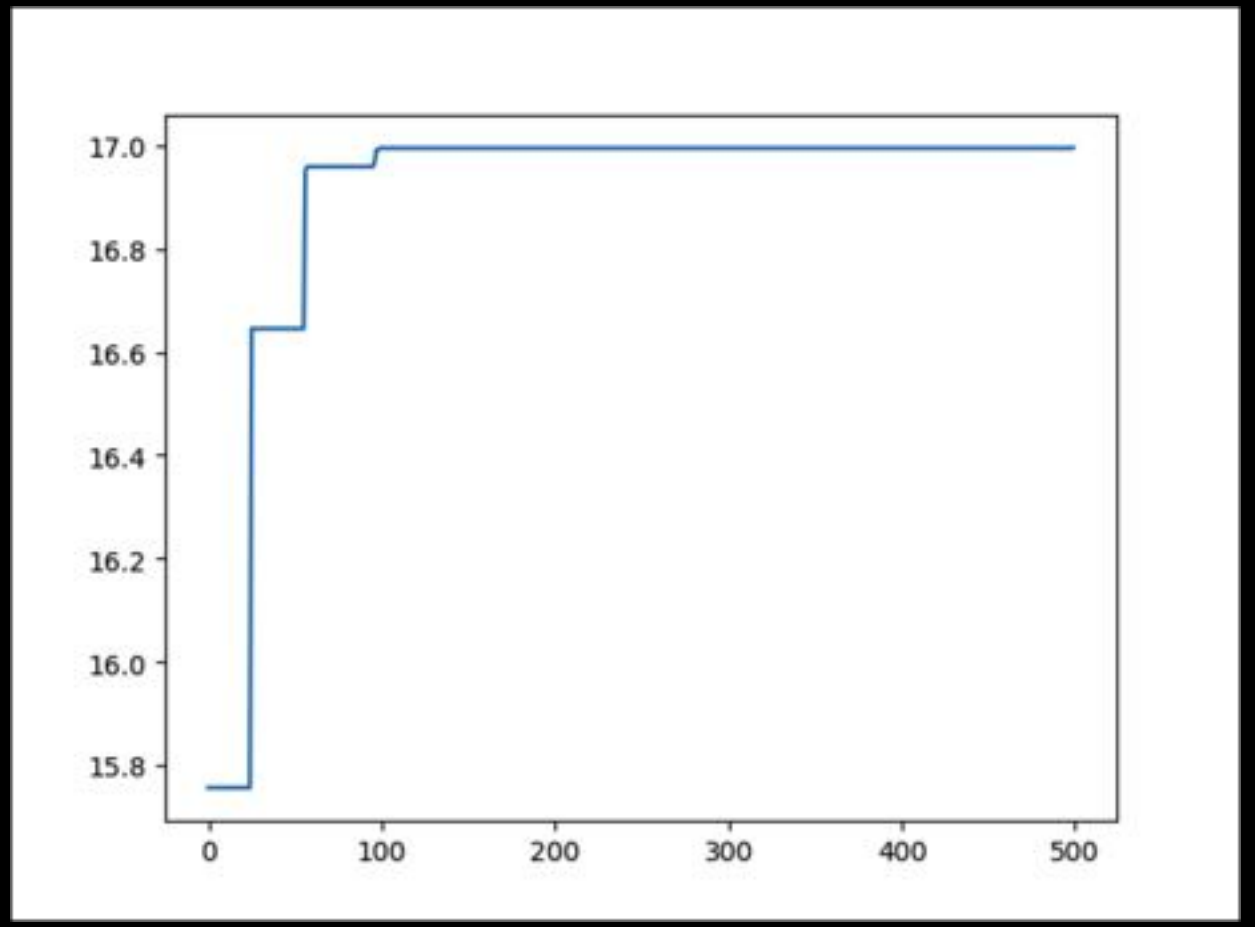
函数最值

```
# 变异
def mutation(pop, pm):
    px = len(pop)
    py = len(pop[0])

    for i in range(px):
        if(random.random() < pm):
            mpoint = random.randint(0, py-1)
            if(pop[i][mpoint] == 1):
                pop[i][mpoint] = 0
            else:
                pop[i][mpoint] = 1
```

函数最值

$$y = 16.996303, \quad x = 7.849462$$



GAME OVER

观看广告继续