函数接口及进化算法模板

在“快速入门”章节中我们展示了利用 Geatpy 简单进化算法模板解决多元单峰函数最值的搜索问题。在那时，我们把所有的代码都放在一个脚本文件里。对于解决简单的问题这种方式能够轻松胜任，但非常不利于重构和把 Geatpy 与其他算法或项目进行融合。本章我们将介绍如何通过编写函数接口以及精简的进化算法模板来实现低耦合的编程。

1. 函数接口

函数接口是目标函数和罚函数的统称，一般写在独立的 Python 源文件里，你也可以直接把它们定义在脚本中。目标函数传入种群表现型矩阵 $Phen$ 以及种群可行性列向量 $LegV$，计算各个个体的目标函数值，若有约束条件，此时将对不满足约束条件的个体在 $LegV$ 上对应的值设置为 0。最后返回种群的目标函数值列向量 $ObjV$ 以及修改后的种群可行性列向量 $LegV$；罚函数传入种群可行性列向量 $LegV$ 和适应度值列向量 $FitnV$，此时对于 $LegV$ 中值为 0 的个体 (即非可行解个体)“加以惩罚”，降低其适应度，最后返回新的适应度列向量 $NewFitnV$。

(注：上述变量的命名只是惯用命名，你可以修改其命名。)

在 Geatpy 有两种方式来对非可行解的个体进行惩罚：

1) 在目标函数 aimfuc 中，当找到非可行解个体时，当即对该个体的目标函数值进行惩罚。若为最小化目标，则惩罚时设置一个很大的目标函数值；反之设置一个很小的目标函数值。

2) 在目标函数 aimfuc 中，当找到非可行解个体时，并不当即对该个体的目标函数值进行惩罚，而是修改其在 $LegV$ 上对应位置的值为 0，同时编写罚函数 punishing，对 $LegV$ 为 0 的个体加以惩罚——降低其适应度。事实上，在 Geatpy 内置的算法模板中，已经对 $LegV$ 为 0 的个体的适应度加以一定的惩罚，因此若是使用内置模板，则不需要编写罚函数 punishing。此时若仍要编写罚函数 punishing 的话，起到的是辅助性的适应度惩罚。

(这里的“罚函数 punishing”跟数学上的“罚函数”含义是不一样的。后者是纯粹的数学公式，而前者是值使用 Geatpy 时自定义的一个名为’punishing’ 的根据可行性列向量 $LegV$ 来对非可行解的适应度 $FitnV$ 进行惩罚的一个函数。)

特别注意：如果采取上面的方法 1 对非可行解进行惩罚，在修改非可行解的目标函数值时，必须设置一个“极大或极小”的值，即当为最小化目标时，要给非可行解设置一个绝对地比所有可行解还要大的值；反之要设置一个绝对比所有可行解小的值。否则会容易出现“被欺骗”的现象：即某一代的所有个体全是非可行解，而此时因为惩罚力度不足，在后续的进化中，再也没有可行解比该非可行解修改后的目标函数值要优秀 (即更大或更小)。此时就会让进化算法“被欺骗”，得出一个非可行解的“最优”搜索结果。因此，在使用方法 1 的同时，可以同时对 $LegV$ 加以标记为 0，两种方法结合着对非可行解进行惩罚。

例：设计目标函数和罚函数，分别命名为 aimfuc 和 punishing。其中目标函数实现的是 2 个变量的平方和，罚函数实现的是惩罚值为 0 的变量。

注意：若使用 Geatpy 的内置算法模板，目标函数和罚函数必须按照下述的输入输出格式进行定义。

```
"""目标函数aimfuc.py"""
import numpy as np
def aimfuc(Phen, LegV): # 传入种群染色体矩阵解码后的基因表现型矩阵
    x1 = Phen[:, [0]] # 从Phen中片取得到x1变量
    x2 = Phen[:, [1]]
    ObjV = x1*x1 + x2*x2
    exIdx = np.where(Phen == 0)[0] # 得到非可行解在种群中的位置
    # 采用方法2对非可行解进行标记，在punishing中对其进行惩罚
    LegV[exIdx] = 0 # 标记非可行解为0
    return [ObjV,LegV]
```

传入的 $Phen$ 代表种群染色体的表现型，它是一个矩阵，$LegV$ 代表种群个体的可行性列向量 (其中元素值为 0 表示对应个体是非可行解，为 1 表示对应的是可行解)。注意返回的是 $ObjV$ 和 $LegV$ 两个参数，前者是一个 numpy 的 array 类型列向量，每一列对应一个个体的目标函数值。

假设传入 aimfuc 函数的 Phen 的值如下：

$$\text{Phen} = \begin{pmatrix} 1 & 2 \\ 4 & 0 \\ 3 & 4 \end{pmatrix}$$

LegV 的值如下：

$$\text{LegV} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

那么调用 aimfuc(Phen, LegV) 后，将会得到种群个体对应的目标函数值为：

$$\text{ObjV} = \begin{pmatrix} 5 \\ 16 \\ 25 \end{pmatrix}$$

新的 LegV 为：

$$\text{LegV} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

进一步地理解 numpy 的 array 类型变量的维度，我们执行 print(Phen.shape) 和 print(ObjV.shape) 输出它们的维度信息，可看到结果分别 (3, 2) 和 (3, 1)。说明 Phen 是 3 行 2 列的矩阵 (实际上是数组类型)；ObjV 是 3 行 1 列的列向量。

下面继续编写罚函数 punishing，对 $LegV$ 标记了 0 的变量进行惩罚，使其适应度比当前种群最小适应度值还要小至少 50%：

```
"""罚函数punishing.py"""
import numpy as np
def punishing(LegV, FitnV):
    FitnV[np.where(LegV == 0)[0]] = np.min(FitnV) // 2 # 取整除法
    return FitnV
```

下面是完整执行上述例子的代码：

```
"""test.py"""
import numpy as np
from punishing import punishing
from aimfuc import aimfuc
import geatpy as ga
# 创建Phen代表种群的基因表现型矩阵，注意array的维度
Phen = np.array([
    [1, 2],
    [4, 0],
    [3, 4]])
LegV = np.ones((3, 1)) # 初始化种群可行性列向量
[ObjV, LegV] = aimfuc(Phen, LegV)
# 调用ranking计算适应度(因为若给ranking传入LegV参数时，ranking函数会自动
# 对LegV标记为0的非可行解进行惩罚，这里为了展示punishing的作用，因此调用
# ranking时不传入LegV)
FitnV = ga.ranking(ObjV)
FitnV=punishing(LegV,FitnV) # 得到新的适应度以及非可行解的下标
print(FitnV) # 输出结果
```

输出结果为：

$$\text{FitnV} = \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}$$

若不调用 punishing，将会得到：

$$\text{FitnV} = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix}$$

可见种群第二个个体成功地被“惩罚”，其适应度变成了 0。

注意事项：

在采用方法 1 来惩罚非可行解时，**要注意一个很容易出错的地方**，请看下面的例子：

$$\min f_1(x) = -25(x_1-2)^2 - (x_2-2)^2 - (x_3-1)^2 - (x_4-5)^2 - (x_5-1)^2$$
$$\min f_2(x) = (x_1-1)^2 + (x_2-1)^2 + (x_3-1)^2 + (x_4-1)^2 + (x_5-1)^2$$

$$s.t.\begin{cases} g_1(x) = x_1 + x_2 - 2 \ge 0 \\ g_2(x) = 6 - x_1 - x_2 \ge 0 \\ g_3(x) = 2 + x_1 - x_2 \ge 0 \\ g_4(x) = 2 - x_1 + 3x_2 \ge 0 \\ g_5(x) = 4 - (x_3-3)^3 - x_4 \ge 0 \\ g_6(x) = (x_5-3)^3 + x_4 - 4 \ge 0 \end{cases}$$
$$0 \le x_i \le 10\,(i = 1, 2, \cdots 5)$$

这是一个双目标优化问题，其中约束条件不再是简单的范围区间，而是多个变量的多个约束不等式。此时我们想让不符合约束条件的解对应的目标函数值 f_1 变大，同时 f_2 变小，于是可以设计以下罚函数：

$$f_i = [p_1 f_1(x), p_2 f_2(x)]$$

其中，若满足约束条件 $g_i(x)(i = 1, 2, ..., 6)$，取 $p_1 = p_2 = 1$，反之，取 $p_1 < 0$ 和 $p_2 > 0$ 的随机数。

这里看上去并无问题，但实际上，我们不建议取 $p_1 < 0$ 的随机数，因为它会改变数值的符号。因为目标函数 f_1 是恒不大于 0 的，如果罚函数遍历所有约束条件来依次让 x_i 不满足约束条件的目标函数值乘上 p_1，那么此时稍有不慎就会极有可能出现目标函数值多次被修改的情况，因为不满足各个约束条件的非可行解可能会有交集，而 $p_1 < 0$，所以此时着意味着这些目标函数值会被反复变换正负号，这将导致罚函数反而将目标函数值变得更小的错误（“惩罚出错”）。对此，一定不能依次地对不满足各个约束条件的非可行解进行惩罚，而是要取不满足各个约束条件的非可行解的全集，对这个全集中的非可行解进行惩罚，这样能避免上述问题的“惩罚出错”的情况出现。另一种解决办法是像上文所说的，让非可行解设置一个绝对比所有可行解要大的值。这个需要数学上证明才可去像这样设值。这样也能避免“惩罚出错”的情况出现。

因此，更建议利用惩罚方法 2 来对可行解进行惩罚。

2. 进化算法模板

前面我们已经介绍过进化算法模板的概念和重要性。下面我们从头开始自定义一个遗传算法模板，命名为 mintemp1，并编写测试脚本，调用该遗传算法模板来解决搜索 $y = x_1^2 + x_2^2$ 的最小值，其中 x_1 与 x_2 分别是 [-5, 0)U(0, 5] 以及 (2, 10] 的整数。

模板中调用的 Geatpy 内置函数的相关用法可以在”Geatpy 函数” 章节中找到详细的讲解。

```
# -*- coding: utf-8 -*-
"""自定义进化算法模板mintemp1.py"""
import numpy as np
import geatpy as ga # 导入geatpy库
import time

def mintemp1(AIM_M, AIM_F, PUN_M, PUN_F, ranges, borders, MAXGEN,
     NIND, SUBPOP, GGAP, selectStyle, recombinStyle, recopt, pm,
     maxormin):
    """=========================初始化配置==========================="""
    # 获取目标函数和罚函数
    aimfuc = getattr(AIM_M, AIM_F) # 获得目标函数
    punishing = getattr(PUN_M, PUN_F) # 获得罚函数
    FieldDR = ga.crtfld(ranges, borders) # 初始化区域描述器
    NVAR = ranges.shape[1] # 得到控制变量的个数
    # 定义进化记录器，初始值为nan
    pop_trace = (np.zeros((MAXGEN ,3)) * np.nan).astype('int64')
    # 定义变量记录器，记录控制变量值，初始值为nan
    var_trace = (np.zeros((MAXGEN ,NVAR)) * np.nan).astype('int64')
    """========================开始遗传算法进化======================="""
    Chrom = ga.crtip(NIND, FieldDR) #
        根据区域描述器FieldDR生成整数型初始种群
    LegV = np.ones((NIND, 1)) #
        生成可行性列向量，元素为1表示对应个体是可行解，0表示非可行解
    [ObjV, LegV] = aimfuc(Chrom, LegV) #
        计算种群目标函数值，同时更新LegV
    start_time = time.time() # 开始计时
    # 开始进化！！
    for gen in range(MAXGEN):
        FitnV = ga.ranking(maxormin * ObjV, LegV) # 计算种群适应度
        FitnV = punishing(LegV, FitnV) # 调用罚函数
        # 记录进化过程
        bestIdx = np.argmax(FitnV)
        if LegV[bestIdx] != 0:
            feasible = np.where(LegV != 0)[0] # 排除非可行解
            # 记录当代种群的适应度均值
            pop_trace[gen, 1] = np.sum(FitnV[feasible]) /
                FitnV[feasible].shape[0]
            # 记录当代种群最优个体的目标函数值
            pop_trace[gen, 0] = ObjV[bestIdx]
            # 记录当代种群的最优个体的适应度值
            pop_trace[gen, 2] = FitnV[bestIdx]
            # 记录当代种群最优个体的变量值
            var_trace[gen, :] = Chrom[bestIdx, :]
        # 进行遗传操作！！
        SelCh=ga.selecting(selectStyle, Chrom, FitnV, GGAP, SUBPOP) #
            选择
        SelCh=ga.recombin(recombinStyle, SelCh, recopt, SUBPOP) #交叉
        SelCh=ga.mutint(SelCh, FieldDR, pm) # 实值变异
        LegVSel = np.ones((SelCh.shape[0], 1)) #
            创建育种个体的可行性列向量
        [ObjVSel, LegVSel] = aimfuc(SelCh, LegVSel) #
            求育种个体的目标函数值
        FitnVSel = punishing(LegVSel, FitnV) # 调用罚函数
        [Chrom,ObjV,LegV] =
            ga.reins(Chrom,SelCh,SUBPOP,1,1,FitnV,FitnVSel,ObjV,ObjVSel,
        LegV,LegVSel) #重插入
    end_time = time.time() # 结束计时
    # 后处理进化记录器
    delIdx = np.where(np.isnan(pop_trace))[0]
    pop_trace = np.delete(pop_trace, delIdx, 0)
    var_trace = np.delete(var_trace, delIdx, 0)
    # 返回进化记录器、变量记录器以及执行时间
    return [pop_trace, var_trace, end_time - start_time]
```

详细解析：

上面我们自定义了一个遗传算法模板 mintemp1, 用于进行带约束的整数变量的目标函数最小化搜索。mintemp1 函数传入了好多参数，下面来一一解析这些参数的含义：

1) AIM_M 和 AIM_F：前者是自定义目标函数接口所在的模块名，后者是该接口的函数名。

2) PUN_M 和 PUN_F：前者是自定义罚函数接口所在的模块名，后者是该接口的函数名。

3) ranges 和 borders：控制变量的范围及是否包含边界。这里设计相关的数据结构，规定 ranges 和 borders 均是 2 行 Nvar 列的矩阵 (Nvar 表示变量的个数)，第一行分别表示第一个变量的范围下界及是否包含下界；第二行分别表示第二个变量的范围上界及是否包含上界。

4) MAXGEN：遗传算法最大进化代数。

5) NIND：种群规模，即种群的个体数。

6) SUBPOP：种群中包含的子种群数量。要求 SUBPOP 必须能被 NIND 整除。

7) GGAP：进化代沟，即子代种群与父代种群个体不相同的概率。

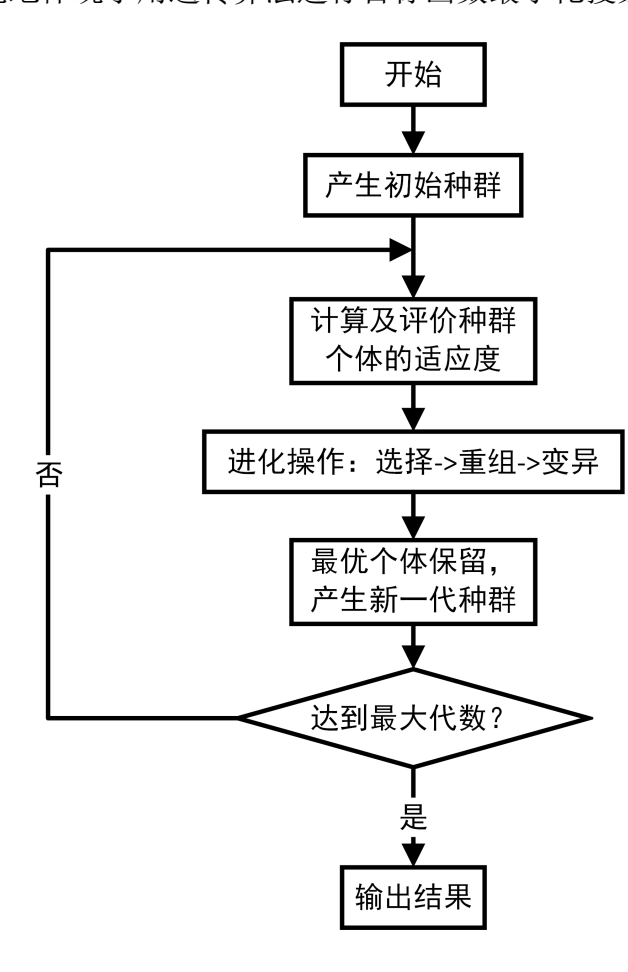
8) selectStyle：低级选择算子的字符串，如’sus’, ’rws’, ’tour’ 等。

9) recombinStyle：低级重组函数的字符串，如’xovsp’, ’xovdp’ 等。

10) recopt 和 Pm：分别代表重组概率和变异概率。

11) maxormin: 最小最大化标记，1 表示是最小化目标,-1 表示是最大化目标。

该算法模板直观地体现了用遗传算法进行目标函数最小化搜索的流程：



下面创建测试脚本进行问题的求解：

```
# -*- coding: utf-8 -*-
"""
执行脚本main.py
描述：
    该demo是展示如何计算带约束的单目标优化问题
    本案例通过自定义算法模板"mintemp1.py"来解决该问题
    其中目标函数写在aimfuc.py文件中，约束条件写在罚函数文件punishing.py中
"""
import numpy as np
import geatpy as gea # 导入geatpy库
from mintemp1 import mintemp1 # 导入自定义的编程模板

# 获取函数接口地址
AIM_M = __import__('aimfuc')    # 目标函数
PUN_M = __import__('punishing') # 罚函数
"""============================变量设置============================"""
x1 = [-5, 5]; x2 = [2, 10]      # 自变量的范围
b1 = [1, 1] ; b2 = [0, 1]       # 自变量的边界
ranges=np.vstack([x1, x2]).T    # 生成自变量的范围矩阵
borders=np.vstack([b1, b2]).T   # 生成自变量的边界矩阵
"""=======================遗传算法参数设置========================="""
NIND = 50             # 种群规模
MAXGEN = 500           # 最大遗传代数
GGAP = 0.8            # 代沟：子代与父代的重复率为(1-GGAP)
selectStyle = 'rws'   # 遗传算法的选择方式设为"rws"——轮盘赌选择
recombinStyle = 'xovdp' # 遗传算法的重组方式，设为两点交叉
recopt = 0.9          # 交叉概率
pm = 0.01             # 变异概率
SUBPOP = 1            # 设置种群数为1
maxormin = 1          # 设置标记表明这是最小化目标
"""======================调用编程模板进行种群进化==================="""
# 调用编程模板进行种群进化，得到种群进化和变量的追踪器以及运行时间
[pop_trace, var_trace, times] = mintemp1(AIM_M, 'aimfuc', PUN_M,
    'punishing', ranges, borders, MAXGEN, NIND, SUBPOP, GGAP,
    selectStyle, recombinStyle, recopt, pm, maxormin)
"""=========================绘图及输出结果========================="""
# 传入pop_trace进行绘图
gea.trcplot(pop_trace, [['各代种群最优目标函数值'],
    ['各代种群个体平均适应度值', '各代种群最优个体适应度值']],
    ['demo_result1', 'demo_result2'])
# 输出结果
best_gen = np.argmin(pop_trace[:, 0]) # 记录最优种群是在哪一代
print('最优的目标函数值为：', np.min(pop_trace[:, 0]))
print('最优的控制变量值为：')
for i in range(var_trace.shape[1]):
    print(var_trace[best_gen, i])
print('最优的一代是第',best_gen + 1,'代')
print('用时：', times, '秒')
```

注意：本例的函数接口为上文所定义的目标函数 aimfuc 和罚函数 punishing，在运行测试脚本前，要保证自定义的算法模板、和函数接口都在同一个目录下。

运行结果如下：

最优的目标函数值为：10

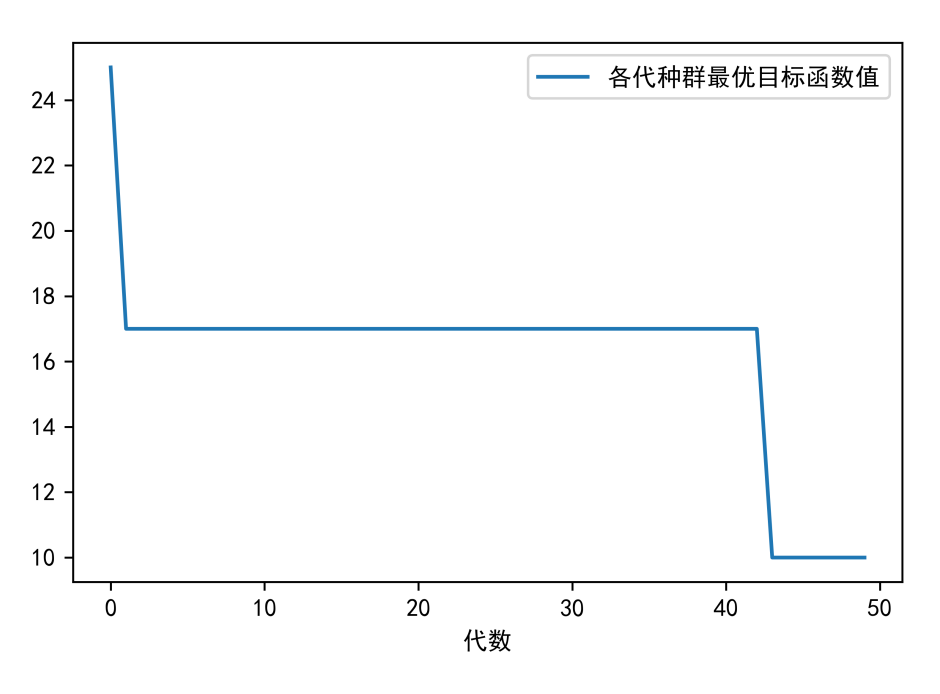
最优的控制变量值为：

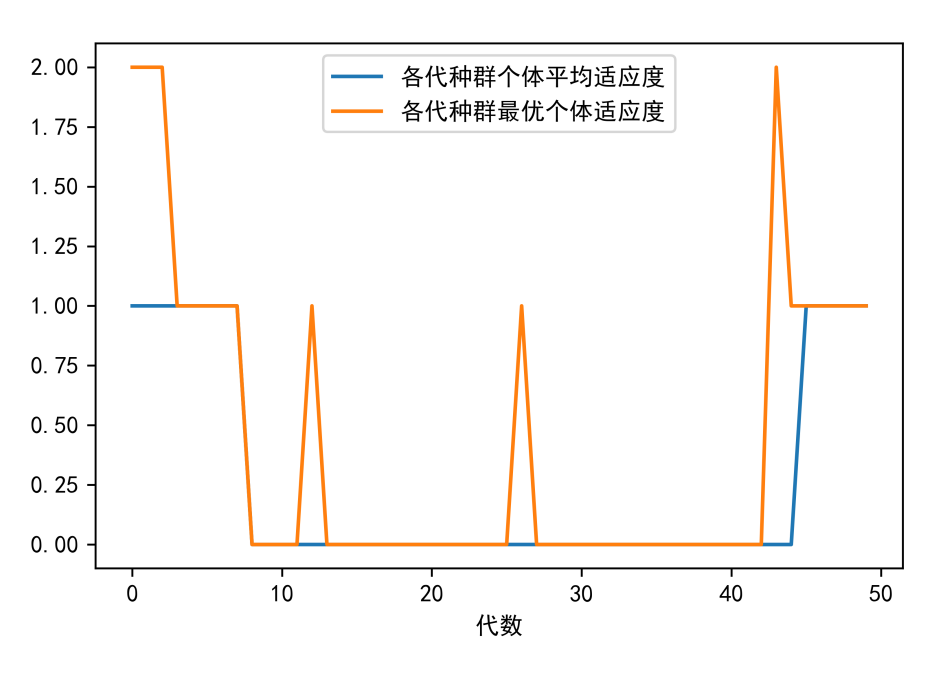
-1

3

最优的一代是第 44 代

用时：0.04921364784240723 秒





3. 总结

因此，在 Geatpy 中，你可以像“快速入门”章节中展示的用纯粹脚本的方式来编写遗传算法程序，也可以用进化算法模板 + 函数接口的方式。

推荐使用可自由自定义的进化算法模板来进行遗传算法编程。在融合其他项目代码时，把进化算法模板、函数接口放到项目目录下，就可以把 Geatpy 与你的其他项目相结合。

总结一下，使用进化算法模板解决遗传算法问题，你需要做 3 件事情：

1. 编写程序执行脚本 main.py

2. 编写函数接口文件，实际上是定义目标函数和罚函数 (如果有的话)，假设命名为 aimfuc.py 和 punishing.py。你也可以把函数接口直接写在程序执行脚本 main.py 中。

3. 准备好进化算法模板 (也可以用 Geatpy 自带的进化算法模板)。自定义的进化算法模板必须和函数接口文件以及程序执行脚本 main.py 放在同一个目录下 (以便 main.py 能够找到)。

在下一章中，我们将介绍几种 Geatpy 自带的几个实用进化算法模板。