更多进化算法模板

本章介绍 Geatpy 内置的几个进化算法模板的功能，详细的代码可以阅读相应的源码，源码中有包括程序逻辑、输入输出的解析等非常详尽的注释。

这些模板内置在 Geatpy 中，因此不需要专门找到并导入。只需统一导入 Geatpy 库 (如 import geatpy as ga)，便可以通过 (ga. 函数名) 来直接调用。

你也可以把“进化算法模板”称作是“内置函数”。但封装成函数只是其存在形式，它们更多的是提供了一些利用进化算法解决问题实际问题的通用模板。你可以参照模板来实现自己的“模板”，比如一些改进型的遗传算法。

值得注意的是，为保持模板的简洁，内置模板并不进行对相关数据的合法性检查，Geatpy 核心函数会发现并捕获这些异常，并提供相关的错误信息。

Geatpy 中有以下几个内置的进化算法模板：

- sga_real_templet(单目标进化算法模板 (实值编码))
- sga_code_templet(单目标进化算法模板 (二进制/格雷编码))
- sga_permut_templet(单目标进化算法模板 (排列编码))
- sga_new_real_templet(改进的单目标进化算法模板 (实值编码))
- sga_new_code_templet(改进的单目标进化算法模板 (二进制/格雷编码))
- sga_new_permut_templet(改进的单目标进化算法模板 (排列编码))
- sga_mpc_real_templet(基于多种群竞争进化单目标编程模板 (实值编码))
- sga_mps_real_templet(基于多种群独立进化单目标编程模板 (实值编码))
- awGA_templet(基于适应性权重法 (awGA) 的多目标优化进化算法模板)
- i_awGA_templet(基于交互式适应性权重法 (i-awGA) 的多目标优化进化算法模板)
- nsga2_templet(基于改进 NSGA-II 算法的多目标优化进化算法模板)
- q_sorted_new_templet(基于改进的快速非支配排序法的多目标优化进化算法模板)
- q_sorted_templet(基于快速非支配排序法的多目标优化进化算法模板)

下面简要介绍其中几个模板的使用。

1. 改进的单目标进化算法模板 (二进制/格雷编码)(sga_new_code_templet)

功能解释：

该模板是单目标进化算法模板 (sga_code_templet) 的改进版，用改进的遗传算法解决控制变量是整数或实数的单目标优化问题 (染色体采用二进制或格雷编码)。

该模板没有采用标准的遗传算法流程，而是先进行交叉和变异生成子代，然后父子两代合并，再从合并的种群中择优保留，从而实现了精英保留，使算法收敛速度更快。

下面以一个经典的多元多峰函数单目标优化例子来展示如何调用该进化算法模板来解决问题，为了书写方便我们直接把目标函数写在执行脚本内。

$$\max f\left(x_1, x_2\right) = 21.5 + x_1 \sin\left(4\pi x_1\right) + x_2 \sin\left(4\pi x_1\right)$$
$$s.t. \begin{cases} -3.0 \leq x_1 \leq 12.1 \\ 4.1 \leq x_2 \leq 5.8 \end{cases}$$

编写如下函数接口和执行脚本：

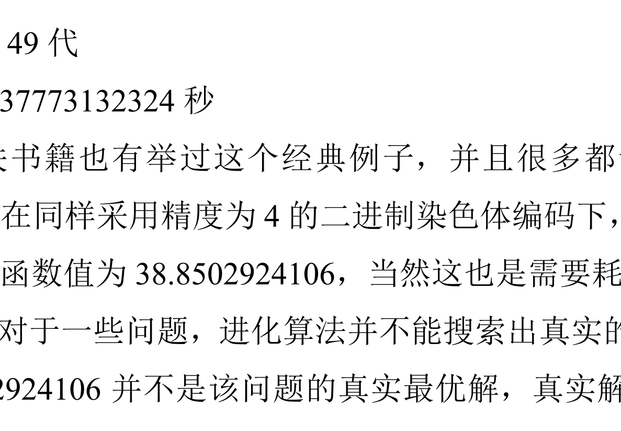
```python
"""函数接口aimfuc.py"""
import numpy as np


def aimfuc(Phen):
   x1 = Phen[:, [0]]
   x2 = Phen[:, [1]]
   f=21.5+x1*np.sin(4*np.pi*x1)+x2*np.sin(20*np.pi*x2)
   return f
```

```python
"""执行脚本main.py"""
import numpy as np
import geatpy as ga
from aimfuc import aimfuc


# 获取函数接口地址
AIM_M = __import__('aimfuc')
# 变量设置
x1 = [-3, 12.1] # 自变量1的范围
x2 = [4.1, 5.8] # 自变量2的范围
b1 = [1, 1] # 自变量1是否包含下界
b2 = [1, 1] # 自变量2是否包含上界
codes = [0, 0] # 自变量的编码方式
precisions = [4, 4] # 自变量的精度(精度不宜设置太高)
scales = [0, 0] # 是否采用对数刻度
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
# 生成区域描述器
FieldD = ga.crtfld(ranges, borders, precisions, codes, scales)
# 调用进化算法模板
[pop_trace, var_trace, times] = ga.sga_new_code_templet(AIM_M,
   'aimfuc', None, None, FieldD, problem = 'R', maxormin = -1, MAXGEN
   = 1000, NIND = 100, SUBPOP = 1, GGAP = 0.8, selectStyle = 'sus',
   recombinStyle = 'xovdp', recopt = None, pm = None)
```

运行结果如下：



最优的目标函数值为：38.8439131372

最优的控制变量值为：

11.6281808021

5.72503128147

最优的一代是第 49 代

时间已过 3.929537773132324 秒

提示：很多相关书籍也有举过这个经典例子，并且很多都说目标函数最大值为 38.818208。实际上，在同样采用精度为 4 的二进制染色体编码下，增大进化代数，算法能搜索出最大的目标函数值为 38.8502924106，当然这也是需要耗费更多的时间的。

这也提醒我们，对于一些问题，进化算法并不能搜索出真实的全局最优解，只能无限逼近 (当然 38.8502924106 并不是该问题的真实最优解，真实解是一个超越数，我们只能无限逼近它)。

2. 单目标进化算法模板 (实值编码)(sga_real_templet)

功能解释：

该模板用于解决控制变量是整数或实数的单目标优化问题 (染色体采用实值编码)。

该模板并没有对种群使用精英保留策略，而是使用一个进化追踪器来记录各代种群的最优个体。关于实现了精英保留的同类模板，可以参考改进的单目标进化算法模板 (实值编码)(sga_new_real_templet)。

实值编码即染色体不需要进行解码即表达了控制变量的真实值。下面同样以一个经典的整数规划问题来介绍该模板的用法：

$$\max z = 18x_1 + 10x_2 + 12x_3 + 8x_4$$
$$s.t. \begin{cases} 12x_1 + 6x_2 + 10x_3 + 4x_4 \\ x_3 + x_4 \leq 1 \\ x_3 \leq x_1 \\ x_4 \leq x_2 \\ x_j \in \{0, 1\}, (j = 1, 2, 3, 4) \end{cases}$$

这个整数规划问题是源于一个公司选址问题。下面展示不把约束条件写在罚函数里而是和目标函数写在一起的示例：

编写目标函数：

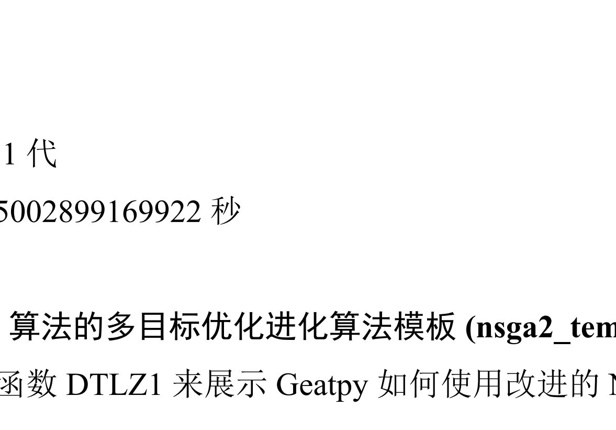
```python
"""函数接口aimfuc.py"""
import numpy as np
def aimfuc(Phen):
   x1 = Phen[:, 0]
   x2 = Phen[:, 1]
   x3 = Phen[:, 2]
   x4 = Phen[:, 3]
   f = 18 * x1 + 10 * x2 + 12 * x3 + 8 * x4
   # 约束条件(危险的写法，建议使用罚函数)
   f[np.where(12 * x1 + 6 * x2 + 10 * x3 + 4 * x4 > 20)[0]] = 0
   f[np.where(x3 + x4 > 1)[0]] = 0
   f[np.where(x3 - x1 > 0)[0]] = 0
   f[np.where(x4 - x2 > 0)[0]] = 0
   return np.array([f]).T
```

编写执行脚本：

```python
"""执行脚本main.py"""
import numpy as np
import geatpy as ga
from aimfuc import aimfuc


# 获取函数接口地址
AIM_M = __import__('aimfuc')
# 变量设置
ranges = np.vstack([np.zeros((1, 4)), np.ones((1, 4))]) #
   生成自变量的范围矩阵
borders = np.vstack([np.ones((1, 4)), np.ones((1, 4))]) #
   生成自变量的边界矩阵
FieldDR = ga.crtfld(ranges, borders) # 生成区域描述器
# 调用进化算法模板
[pop_trace, var_trace, times] = ga.sga_real_templet(AIM_M, 'aimfuc',
   None, None, FieldDR, problem = 'I', maxormin = -1, MAXGEN = 1000,
   NIND = 50, SUBPOP = 1, GGAP = 0.9, selectStyle = 'sus',
   recombinStyle = 'xovdp', recopt = 0.9, pm = 0.1)
```

运行结果如下：



最优的目标函数值为：28.0

最优的控制变量值为：

1.0

1.0

0.0

0.0

最优的一代是第 1 代

时间已过 0.05585002899169922 秒

3. 基于改进 NSGA-II 算法的多目标优化进化算法模板 (nsga2_templet)

下面以标准测试函数 DTLZ1 来展示 Geatpy 如何使用改进的 NSGA2 算法模板。

编写目标函数：

```python
"""函数接口aimfuc.py"""
# DTLZ1
def aimfuc(Chrom, M = 3): # M为问题维度，如2维、3维
   x = Chrom.T
   XM = x[M-1:]
   k = x.shape[0] - M + 1
   gx = 100 * (k + np.sum((XM - 0.5) ** 2 - np.cos(20 * np.pi * (XM -
      0.5)), 0))

   ObjV = (np.array([[]]).T) * np.zeros((1, Chrom.shape[0]))
   ObjV = np.vstack([ObjV, 0.5 * np.cumprod(x[:M-1], 0)[-1] * (1 +
      gx)])
   for i in range(2, M):
   ObjV = np.vstack([ObjV, 0.5 * np.cumprod(x[: M-i], 0)[-1] * (1 -
      x[M-i]) * (1 + gx)])
   ObjV = np.vstack([ObjV, 0.5 * (1 - x[0]) * (1 + gx)])
   return ObjV.T
```

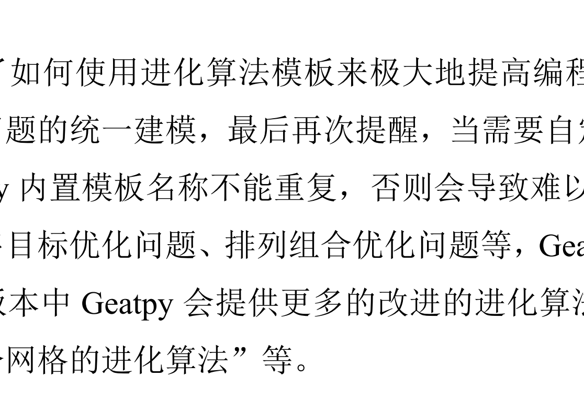
编写执行脚本：

```python
"""执行脚本main.py"""
"""main.py"""
import numpy as np
import geatpy as ga # 导入geatpy库

# 获取函数接口地址
AIM_M = __import__('aimfuc')

"""============================变量设置============================"""
ranges = np.vstack([np.zeros((1,7)), np.ones((1,7))]) # define the
   ranges of variables in DTLZ1
borders = np.vstack([np.ones((1,7)), np.ones((1,7))]) # define the
   borders of variables in DTLZ1
FieldDR = ga.crtfld(ranges, borders) # create the FieldDR
"""=======================use sga2_templet to find the Pareto
   front==================="""
[ObjV, NDSet, NDSetObjV, times] = ga.nsga2_templet(AIM_M, AIM_F,
   None, None, FieldDR, problem = 'R', maxormin = 1, MAXGEN = 1000,
   MAXSIZE = 2000, NIND = 50, SUBPOP = 1, GGAP = 1, selectStyle =
   'tour', recombinStyle = 'xovdprs', recopt = 0.9, pm = None,
   distribute = False, drawing = 1)
```

运行结果如下：



用时：11.241936683654785 秒

帕累托前沿点个数：2008 个

单位时间找到帕累托前沿点个数：178.6169106360056 个

4. 总结

本章再次回顾了如何使用进化算法模板来极大地提高编程效率。Geatpy 内置进化算法模板是对常见问题的统一建模，最后再次提醒，当需要自定义模板时，请确保定义的模板名称与 Geatpy 内置模板名称不能重复，否则会导致难以解决的冲突问题。

对于单目标和多目标优化问题、排列组合优化问题等，Geatpy 都给出了通用的进化算法模板。在后续版本中 Geatpy 会提供更多的改进的进化算法模板，比如“自适应进化算法”、“基于划分网格的进化算法”等。