注意细则

本章介绍 Geatpy 的几点注意细则，以提高编程效率。

1. 目标函数正确编写是关键

在使用 Geatpy 求解进化算法问题时，目标函数的正确编写是关键。目标函数可以是简单的或复杂的，只要能够把目标函数值求出来即可。这里需要注意 Geatpy 的目标函数矩阵的编写规范（详见 Geatpy 数据结构章节），目标函数矩阵一般在 Geatpy 中命名为 $ObjV$ 或与之类似的名字，它是 Numpy array 类型的列向量（单目标）或矩阵（多目标）。$ObjV$ 的每一列对应一个目标，每一行对应种群的一个个体。

因此在编写目标函数的时候，返回的目标函数矩阵必须是拥有与种群一样的行数。一般可以利用 Numpy 的矩阵运算来快速求出种群各个个体对应的目标函数值。当然，当目标函数比较复杂的时候，可以用循环的方法把各个个体的目标函数值求出来。

2. 处理约束条件的两种方法

在前面的“函数接口及进化算法模板”章节中详细介绍了 Geatpy 处理约束条件的两种方法，并介绍了 Geatpy 中用于标记个体是否满足约束条件的变量 $LegV$。这里再总结一下：

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。

算法注意：若采用了上述的方法 2 对非可行解个体进行了标记，则在种群进化过程中，Geatpy 的内置算法模板采用“遗忘策略”来排除这些个体。这里的“排除”并不是指不让这些非可行解个体保留到下一代，这个“排除”是对于进化记录器而言的：在进化过程中，当使用进化记录器记录各代种群的最优解时，需要排除非可行解个体对记录的影响。它是不会影响种群进化的。根据进化算法的原理非可行解个体保留到下一代的概率只会被降低，而并非将它们完全排除。(“遗忘策略”详见以'sga_real_templet' 等进化算法模板的源代码。

在多目标优化中，假如采取上面的惩罚方法 2，则在调用'ndomin'、'ndomindeb'、'ndominfast' 等计算种群非支配个体及种群个体适应度时，这个记录着种群个体是否是可行解的变量 $LegV$ 就起到非常重要的作用了。它会让算法排除这些个体对非支配排序的影响。

3. 关于 maxormin 的使用

Geatpy 是遵循最小化目标原则的，即认为目标函数值越小越好。因此，在需要最大化目标时，需要设置'maxormin' 为-1。在调用内核函数时，假如函数的输入或输出参数列表中包含与目标函数值有关的变量时，需要注意是否要对这些输入、输出参数乘上 maxormin。比如使用'ranking' 函数计算种群个体的适应度时，传入的 ObjV 就需要乘上'maxormin'（详见“ranking 参考资料”)。当使用'upNDSet' 函数来更新多目标优化的全局帕累托最优解集时，不但要对输入的'NDSetObjV' 乘上'maxormin'，对于计算后返回的'NDSetObjV'，也需要乘上'maxormin' 进行还原。需不需要乘'maxormin'，本质上取决于算法是否是根据目标函数值进行相关的计算，因此，当碰到输入输出参数列表中包含相关类型的变量时，查看以下该函数的详细文档，或使用'help' 命令查看帮助文档即可。掌握规律后就可以不再依赖于文档了。

4. 注意区分'percisions' 的含义

'percisions' 是 Geatpy 的 demo 示例代码中经常出现的一个变量，它事实上是调用'crtfld' 函数来创建“区域描述器”（又称“译码矩阵”）时的一个传入参数。根据“crtfld 参考资料”（或可用'help' 命令查看相应的帮助文档)，该'percisions' 是有两重含义的：

1) 当使用'crtfld' 创建一个表示二进制/格雷码编码的区域描述器 $FieldD$ 时，'percisions' 用于控制二进制/格雷码的编码精度。

注意：这个“精度”并不是平常所说的“精确到小数点后多少位”，而是值用二进制/格雷码编码后所能够表达小数点后多少位的控制变量。

2) 当使用'crtfld' 创建一个表示实值编码的种群时，'percisions' 并不代表“精度”，而是用于控制变量的边界。比如：当某个变量范围为 [-1,1) 时，假如传入'crtfld' 的 percisions 对应的值为 2，那么，由于该变量不包含上界，因此，'crtfld' 会对 [-1,1) 进行适当的调整，使得返回的区域描述器中，该变量的范围被修正为：[-1,0.99]。

5. 关于 Geatpy 输出动画的问题

使用 Geatpy 的内置算法模板可以绘制动画。相关的方法详见内置算法模板的源代码或“Geatpy 教程”的“数据可视化”章节。这里要注意的是，若要绘制动画，需要运行前在控制台中输入"matplotlib qt5" 这条命令。

这里，用普通的 Python 控制台是无法执行这条命令的。因此建议使用 ipython，或者一个简单的方法是安装 Anaconda，利用里面的 Spyder 来进行 Python 代码的编写与执行。在 Spyder 的控制台中，就可以执行"matplotlib qt5" 并顺利绘制 Geatpy 的动画了。

6. 关于强精英策略与增强种群个体的多样性

在 Geatpy 中，'etour' 是一个增强的精英保留锦标赛选择算子。采用该算子进行个体的选择，可以保证最优个体被保留下来。另外，在进化算法模板中，可以采用“父子合并”共同选择出下一代的方法来替代传统的“重插入”生成下一代，以增强精英保留的效果。因为如果采用“重插入”的方法生成下一代，若代沟设置不合理，则会容易丢失种群中较优甚至是最优解。

另外，可以采用'indexing' 或'powing' 算子来代替'ranking' 算子进行种群适应度的计算，也可以在一定程度上增强精英策略的效果，加速算法的收敛速度。

如何增强种群个体的多样性？Geatpy 的内置进化算法模板提供增强种群多样性的机制（详见进化算法模板源码)。此时设置 distribute 为 True，内置模板就会调用该机制。该机制适当地修改了种群个体的适应度，在一定程度上增强了目标函数值分布稀疏的个体的适应度，从而改善了种群的分布性。这个改善效果在多目标优化中尤为明显，可增强帕累托前沿的分布性。