Geatpy 数据结构

Geatpy 的大部分数据都是存储在 numpy 的 array 数组里的，numpy 中另外还有 matrix 的矩阵类型，但我们不适用它，于是我们默认 array 就是存储“矩阵”(也可以存储一维向量，接下来会谈到)。其中有一些细节需要特别注意：numpy 的 array 在表示行向量时会有 2 种不同的结构，一种是 1 行 n 列的矩阵，它是二维的；一种是纯粹的一维行向量。因此，在 Geatpy 教程中会严格区分这两种概念，我们称前者为“行矩阵”，后者为“行向量”。Geatpy 中不会使用超过二维的 array。

例如有一个行向量 x，其值为 1 2 3 4 5 6，那么，用 print(x.shape) 输出其规格，可以得到 (6,)，若 x 是行矩阵而不是行向量，那么 x 的规格就变成是 (1,6) 而不再是 (6,)。

在 numpy 的 array 类型中，实际上没有“列向量”的概念。所谓“向量”是指一维的，但用 numpy 的 array 表示列向量时，它实际上是二维的，只不过只有 1 列。我们不纠结于这个细节，统一仍用“列向量”来称呼这种只有 1 列的矩阵。

在编程中，如果对 numpy 的 array 感到疑惑，你可以用“变量.shape”语句来输出其维度信息，以确定其准确的维度。

1. 种群染色体

Geatpy 中，种群染色体是一个二维矩阵，简称“种群矩阵”。一般所说的“种群”是特指种群染色体矩阵。

种群矩阵一般用 Chrom 命名，是一个是 numpy 的 array 类型的，每一行对应一条染色体，同时也对应着一个个体。染色体的每个元素是染色体上的基因。

我们一般把种群的规模 (即种群的个体数) 用 $Nind$ 命名；把种群个体的染色体长度用 $Lind$ 命名。

$$\text{Chrom} = \begin{pmatrix} g_{1,1} & g_{1,1} & g_{1,1} & \cdots & g_{1,1} \\ g_{1,1} & g_{1,2} & g_{1,3} & \cdots & g_{1,Lind} \\ g_{2,1} & g_{2,2} & g_{2,3} & \cdots & g_{2,Lind} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ g_{Nind,1} & g_{Nind,2} & g_{Nind,3} & \cdots & g_{Nind,Lind} \end{pmatrix}$$

对于多种群，Geatpy 有 2 种设计，一种是将子种群全部放在一个大的种群内，称为“基于种群 Chrom 的子种群划分”，此时所有子种群的染色体编码是一样的；另一种是采用多个 Chrom 变量来表示多个子种群。此时子种群的染色体编码可以不一样。

对于基于种群染色体矩阵 Chrom 的子种群，子种群之间必须有相同数量的个体，并且按照下列方案进行有序排列：

$$\text{Chrom} = \begin{pmatrix} subchrom_{1\_ind_1} \\ subchrom_{1\_ind_2} \\ \vdots \\ subchrom_{1\_ind_m} \\ subchrom_{2\_ind_1} \\ subchrom_{2\_ind_2} \\ \vdots \\ subchrom_{2\_ind_m} \\ \vdots \\ subchrom_{SUBPOP\_ind_1} \\ subchrom_{SUBPOP\_ind_2} \\ \vdots \\ subchrom_{SUBPOP\_ind_m} \end{pmatrix}$$

比如如下种群：

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

假设它要表示 2 个子种群，那么，前两个个体 (前两行) 就是 1 号子种群，后两个个体 (后两行) 就是 2 号子种群。种群个体的染色体为：1234, 2341, 3142 和 4231。

2. 种群表现型

种群表现型的数据结构跟种群染色体基本一致，也是 numpy 的 array 类型。我们一般用 Phen 来命名。它是种群矩阵 Chrom 经过解码操作后得到的基因表现型矩阵，每一行对应一个个体，每行中每个元素都代表着一个变量，并用 $Nvar$ 表示变量的个数。如下图：

$$\text{Phen} = \begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \cdots & x_{1,Nvar} \\ x_{2,1} & x_{2,2} & x_{2,3} & \cdots & x_{2,Nvar} \\ x_{3,1} & x_{3,2} & x_{3,3} & \cdots & x_{3,Nvar} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{Nind,1} & x_{Nind,2} & x_{Nind,3} & \cdots & x_{Nind,Nvar} \end{pmatrix}$$

Phen 的值与采用的解码方式有关。Geatpy 提供二进制/格雷码编码转十进制整数或实数的解码方式。另外，在 Geatpy 也可以使用不需要解码的“实值编码”种群，这种种群的染色体的每个基因就对应变量的实际值，即 Phen 等价于 Chrom。如上面的例子中，Chrom 是种群的染色体矩阵，其第一个个体的染色体为 1234，若该种群是实值种群，那么这条染色体就代表了 4 个控制变量，值分别为 1,2,3 和 4。

这里需要注意的是：我们可以用不同的方式去解码一个种群染色体，得到的结果往往是不同的。

对于混合编码，可以使用多种群去表现不同的编码，运算效率实际上会远高于把多种编码方式融合在同一个种群中。因此我们一般采用多种群去应对复杂的混合编码。

3. 目标函数值

Geatpy 采用 numpy 的 array 类型变量来存储种群的目标函数值。一般命名为 $ObjV$，每一行对应种群矩阵的每一个个体。因此它拥有与 $Chrom$ 相同的行数。每一列代表一个目标函数值。因此对于单目标函数，ObjV 会只有 1 列；而对于多目标函数，ObjV 会有多列。

例如 ObjV 是一个二元函数值矩阵：

$$\text{ObjV} = \begin{pmatrix} f_1(x_{1,1}, x_{1,2}, \cdots x_{1,Nvar}), f_2(x_{1,1}, x_{1,2}, \cdots x_{1,Nvar}) \\ f_1(x_{2,1}, x_{2,2}, \cdots x_{2,Nvar}), f_2(x_{2,1}, x_{2,2}, \cdots x_{2,Nvar}) \\ f_1(x_{3,1}, x_{3,2}, \cdots x_{3,Nvar}), f_2(x_{3,1}, x_{3,2}, \cdots x_{3,Nvar}) \\ \vdots \\ f_1(x_{Nind,1}, x_{Nind,2}, \cdots x_{Nind,Nvar}), f_2(x_{Nind,1}, x_{Nind,2}, \cdots x_{Nind,Nvar}) \end{pmatrix}$$

4. 个体适应度

Geatpy 采用**列向量**来存储种群个体适应度。一般命名为 $FitnV$，它同样是 numpy 的 array 类型，每一行对应种群矩阵的每一个个体。因此它拥有与 $Chrom$ 相同的行数。

$$\text{FitnV} = \begin{pmatrix} fit_1 \\ fit_2 \\ fit_3 \\ \vdots \\ fit_{Nind} \end{pmatrix}$$

5. 区域描述器

Geatpy 使用区域描述器来描述种群染色体的特征，比如染色体中基因所表达的控制变量的范围、是否包含范围的边界、采用什么编码方式，是否使用对数刻度等等。

1) 对于二进制/格雷编码的种群，使用 7 行 n 列的矩阵 FieldD 来作为区域描述器，n 是染色体所表达的控制变量个数。FieldD 的结构如下：

$$\begin{pmatrix} lens \\ lb \\ ub \\ codes \\ scales \\ lbin \\ ubin \end{pmatrix}$$

其中，$lens$ 包含染色体的每个子染色体的长度。sum($lens$) 等于染色体长度。

lb 和ub 分别代表每个变量的上界和下界。

$codes$ 指明染色体子串用的是标准二进制编码还是格雷编码。$codes[i] = 0$ 表示第i个变量使用的是标准二进制编码；$codes[i] = 1$ 表示使用格雷编码。

$scales$ 指明每个子串用的是算术刻度还是对数刻度。$scales[i] = 0$ 为算术刻度，$scales[i] = 1$ 为对数刻度。对数刻度可以用于变量的范围较大而且不确定的情况，对于大范围的参数边界，对数刻度让搜索可用较少的位数，从而减少了遗传算法的计算量。

$lbin$ 和$ubin$ 指明了变量是否包含其范围的边界。0 表示不包含边界；1 表示包含边界。

2) 对于实值编码 (即前面所说的不需要解码的编码方式) 的种群，使用 2 行 n 列的矩阵 FieldDR 来作为区域描述器，n 是染色体所表达的控制变量个数。FieldDR 的结构如下：

$$\begin{pmatrix} x_1\text{下界} & \cdots & x_n\text{下界} \\ x_1\text{上界} & \cdots & x_n\text{上界} \end{pmatrix}$$

区域描述器 FieldD 和 FieldDR 都是 numpy 的 array 类型。可以直接用代码创建，比如：

```
FieldDR=np.array([-3, -4, 0, 2],
                 [2, 3, 2, 2])
```

也可以用 Geatpy 内置的 crtfld 函数来方便地快速生成区域描述器，其详细用法参见“Geatpy 函数”的“crtfld 参考资料”。

6. 进化追踪器

在使用 Geatpy 进行进化算法编程时，常常建立一个进化追踪器 (如 pop_trace) 来记录种群在进化的过程中各代的最优个体，尤其是采用无精英保留机制时，进化追踪器帮助我们记录种群在进化的“历史长河”中产生过的最优个体。待进化完成后，再从进化追踪器中挑选出“历史最优”的个体。这种进化记录器也是 numpy 的 array 类型，结构如下：

$$\text{trace} = \begin{pmatrix} a_1 & b_1 & c_1 & \cdots & \omega_1 \\ a_2 & b_2 & c_2 & \cdots & \omega_2 \\ a_3 & b_3 & c_3 & \cdots & \omega_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{MAXGEN} & b_{MAXGEN} & c_{MAXGEN} & \cdots & \omega_{MAXGEN} \end{pmatrix}$$

其中 MAXGEN 是种群进化的代数。trace 的每一列代表不同的指标，比如第一列记录各代种群的最佳目标函数值，第二列记录各代种群的平均目标函数值……trace 的每一行对应每一代，如第一行代表第一代，第二行代表第二代……

7. 全局最优集

在使用 Geatpy 进行多目标进化优化编程时，常常建立一个全局的帕累托最优集 (NDSet) 来记录帕累托最优解。它也是 numpy 的 array 类型，结构如下：

$$\text{NDSet} = \begin{pmatrix} f_1 & g_1 & h_1 & \cdots & \varphi_1 \\ f_2 & g_2 & h_2 & \cdots & \varphi_2 \\ f_3 & g_3 & h_3 & \cdots & \varphi_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f_n & g_n & h_n & \cdots & \varphi_n \end{pmatrix}$$

$f, g, h, \ldots, \varphi$ 表示不同的目标。NDSet 的每一行都是一个帕累托非支配解。