

awGA 参考资料

摘要: 适应性权重法多目标聚合函数。

描述:

该函数实现了 Gen-Cheng 的适应性权重法 (adaptive-weight GA: awGA)[GC98]，利用在各代种群中获得的正向的理想点，通过调整权重使 Pareto 最优解靠近理想点来进行解的搜索，最终返回合成的单目标函数值以及各目标的权重。

语法:

```
[CombinObjV, weight] = awGA(ObjV)
[CombinObjV, weight] = awGA(ObjV, LegV)
```

详细说明:

ObjV 是一个保存着种群个体对应的多目标函数值的矩阵，每一列对应一个个体的目标函数值。

LegV 为可选参数，是一个保存着种群个体对应的可行性的列向量，0 表示该个体是非可行解，1 表示是可行解。

CombinObjV 是一个保存着将多目标加权合成为单目标后的目标函数列向量。

weight 一个保存着各目标函数值的 array 类型行向量。

在计算多目标权重前，该函数会根据 LegV 把非可行解排除在外，以避免非可行解对理想点选取的影响。计算权重后，所有个体的多目标函数值一并乘上权重，得到加权的聚合单目标函数值。

此外，该函数遵循“最小化目标”的约定，因此传入 ObjV 前要乘上最大最小化标记 maxormin，函数返回 CombinObjV 后，需要再乘上 maxormin 以复原目标函数值。

特别注意:

本函数是根据传入参数 ObjV 来计算多目标聚合权重的，遵循“最小化模板”约定，因此在调用本函数前，需要对传入的 ObjV 乘上'maxormin'(最大最小化标记)，同时，对于返回的 CombinObjV，也需要乘上'maxormin' 进行还原。

应用实例:

考虑一个两个目标的优化问题，设种群规模为 4，这 4 个个体的目标函数值如下：

(1,2),(2,3),(2,3),(3,3)

使用适应性权重聚合法 awGA 使每个个体的两个目标函数值合成为 1 个目标函数值：

```
ObjV = np.array([[1,2],[2,3],[2,3],[3,3]])
[CombinObjV, weight] = awGA(ObjV)
```

结果如下：

$$\text{CombinObjV} = \begin{pmatrix} 7.35930736 \\ 11.99134199 \\ 11.99134199 \\ 13.8961039 \end{pmatrix}$$

$$\text{weight} = \begin{pmatrix} 1.9047619 & 2.72727273 \end{pmatrix}$$

bs2int 参考资料

摘要: 二进制串到整数值的转换。

描述:

该函数把二进制种群解码成十进制整数种群（无论它是标准的二进制编码还是格雷码），并且能够支持大数运算。

语法: Phen = bs2int(Chrom, FieldD)

详细说明:

Phen = bs2int(Chrom, FieldD) 根据区域描述器（又称译码矩阵）将用二进制/格雷码编码的种群矩阵 Chrom 解码成十进制的整数表示的种群矩阵 Phen。

二进制/格雷码种群 Chrom 是诸如下图所示的矩阵，矩阵的每一行代表种群中的一个个体的染色体。

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

译码矩阵 FieldD 具有下面的结构：

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

其中，*lens* 包含染色体的每个子染色体的长度。 $\text{sum}(lens)$ 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、*lbin* 和 *ubin* 的含义详见 bs2rv 函数的参考资料。在本函数中，这三个量并无实际用途，仅为了兼容 bs2rv 函数而设。因为本函数规定解码时使用算术尺度，并且包含变量的两个边界。至于要使用不包含变量的边界的使用场合，需要调用 crtflid 函数以生成符合规格的译码矩阵 FieldD。

应用实例:

调用 crtbp 函数生成一个二进制种群 Chrom，代表 2 个变量，范围分别是 [-4,2] 和 [-2, 7]。用 bs2int 函数将 Chrom 解码转换成整数表现型。

```
Chrom = crtbp(3, 5) # 调用crtbp创建一个3行5列的二进制种群矩阵
```

$$\text{Chrom} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

```
# 创建译码矩阵
```

```
FieldD = np.array([[2,3], [-4,-2], [2,7], [1,1], [1,1], [1,1], [1,1]])  
Phen = bs2int(Chrom, FieldD) # 进行解码
```

解码后结果如下：

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

解释：对 Chrom 进行解码时，bs2int 函数是先把二进制矩阵转换成十进制自然数矩阵，然后把结果均匀映射到变量的区间上，得到解码结果。当使用格雷码进行解码时，bs2int 函数先将格雷码矩阵转换成标准二进制编码矩阵，然后再按上述方式转换。

注：因为采用的是均匀的区间映射的方式，因此，当编码空间比解空间大时，会出现多个不同的染色体解码后得到的值是一样的情况；另外，若编码空间比解空间小，则会出现解空间中有些值无法通过解码得到（此时出现了较为明显的“汉明悬崖”）。

译码矩阵的结构比较复杂，但作为一个开放式框架，你可以手写比较复杂的译码矩阵 FieldD，也可以调用 crtflid 函数来自动生成。详见“crtflid 参考资料”。

描述:

雷码

语法: Phen = bs2rv(Chrom, FieldD)

编码的种群矩阵 Chrom 解码成十进制

不不体的来

$$\begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

译码矩阵 FieldD 具有下面的结构:

lens
lb
ub
codes
scales

染色体的长
和下界。

解码公式：设三进制进位值的某个进段为 $b_1 b_2 \dots b_k$ ，它解码后表示

范围在 $[lb, ub]$ 上的实数，设解码得到的值为

$$X = lb + \left(\sum_{i=0}^k b_i \cdot 2^{i-1} \right) \cdot \frac{ub - lb}{2^k - 1}$$

应用实例:
调用 crtbp 函数生成一个二进制种群 Chrom，代表 2 个变量，范围分别是 [2,10] 和

```
Chrom = crtbp(3, 5) # 调用crtbp创建一个3行6列的二进制种群矩阵
```

$$\text{Chrom} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

创建

```
FieldD = np.array([[3,3], [2,2], [10,10], [0,0], [0,0], [1,1], [1,1]])  
Phen = bs2rv(Chrom, FieldD) # 进行解码
```

$$\begin{pmatrix} 6.57142857 & 2.0 \end{pmatrix}$$

， 则会解码得到不同的表现型

```
Phen = bs2rv(Chrom, FieldD) # 进行解码
```

解码后结果如下：

$$\text{Phen} = \begin{pmatrix} 5.46872706 & 2.0 \\ 2.44568909 & 8.17765434 \end{pmatrix}$$

特别说明，当值

较复杂，但作为一个开放式框架，你可以使用 crtfld 函数来自动生成。详见“crtfld”。

[1] R. B. Holstien, Artificial Genetic Adaptation in Computer Communication Networks, Department of Computer and Communication Sciences, University of Illinois at Urbana-Champaign, 1990.

[2] R. A. Caruana and J. D. Schaffer, “Representation and Hidden Bias: Gray vs. Binary

[3] W. E. Schmitendorf, O. Shaw, R. Benson and S. Forrest, “Using Genetic Algorithms for Controller Design: Simultaneous Stabilization and Eigenvalue Placement in a Region”, Technical Report No. CS92-9, Dept. Computer Science, College of Engineering, University of New Mexico, 1992.

crtbase 参考资料

概要: 创建基因座。

描述:

该函数创建一个行矩阵，其元素对应染色体结构的基因座。对于离散数据建立种群时该函数可以与 crtbp 配合使用。

语法:

```
BaseV = crtbase(Lind)
BaseV = crtbase(Lind, Base)
BaseV = crtbase(SegLen, Base)
```

详细说明:

Lind 是一个整数，代表基因片段数，每个基因片段的长度为 1。

SegLen 是一个 1 行 Lind 列的行矩阵，代表染色体各个片段的长度。

Base 是整数或者是一个 1 行 Lind 列的行矩阵，代表染色体上各个片段的等位基因的基因座。

函数返回染色体的基因座的行矩阵。这里的基因座的概念与生物学中的有些异同，实际上是指变量的“可能情况”的个数，这与变量的范围又是不一样的。结合 crtbp 函数可以更清晰地理解这个概念。

应用实例:

下面展示 crtbase 函数所有用法以及输出结果，从中也可以更好地理解上面所述的概念：

```
BaseV=crtbase(4) # 只输入Lind，则默认各个基因座是2
```

$$\text{BaseV} = \begin{pmatrix} 2 & 2 & 2 & 2 \end{pmatrix}$$

```
BaseV=crtbase(4, 3) # 生成包含4个基因片段，各片段的基因座均为3的基因座
```

$$\text{BaseV} = \begin{pmatrix} 3 & 3 & 3 & 3 \end{pmatrix}$$

```
BaseV=crtbase(3, np.array([[4,2,3]])) # 3个基因片段，基因座分别是4,2和3
```

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

```
# crtbase结合crtbp创建一个有2个基本字符{0,1,2,3}、1个基本字符{0,1}
```

```
# 和3个基本字符{0,1,2}的随机种群
```

```
BaseV=crtbase(np.array([[2,1,3]]), np.array([[4,2,3]]))
```

```
Chrom=crtbp(4, BaseV) # 生成一个包含4个个体，基因座为BaseV的随机种群
```

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

```
BaseV=crtbase(np.array([[4,2,3]]), 3) # 3个基因片段，各个基因座均为3
```

$$\text{BaseV} = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

crtbp 参考资料

概要: 创建简单离散种群。

描述:

该函数创建一个简单的种群矩阵，矩阵的每一行代表一个个体的染色体串，染色体串是由随机的非负整数构成的。

语法:

- 1) Chrom = crtbp(Nind, Lind)
- 2) Chrom = crtbp(Nind, BaseV)

详细说明:

该函数使用随机函数 rand 生成简单的种群矩阵，该矩阵的元素均是非负的。

Nind 是一个整数，代表种群的大小，即种群包含的个体数。

Lind 是一个整数，代表染色体的长度。

BaseV 是 1 行 Lind 列的行矩阵，代表染色体结构的基因座，指定染色体中每个元素的基本字符。

格式 1) 没有传入 BaseV 参数，此时算法将自动生成一个元素均为 2 的 BaseV。因此函数返回一个大小为 Nind×Lind 的随机二元矩阵。

格式 2) 传入了 BaseV 参数，此时染色体长度 Lind 由 BaseV 的长度得出。函数根据 BaseV 来控制生成矩阵的元素范围。如 BaseV=np.array([[2, 3]])，表示生成的 Chrom 中有 2 列，每一列控制一个变量的表现型，并且每一列的元素的可能取值分别是 0,1 和 0,1,2。

实际上，我们可以把该函数创建的种群看成是变量下界固定为 0 的整数值种群。

应用实例:

```
Chrom = crtbp(3,4) # 创建一个有3个个体，染色体长度为4的二元种群
```

$$\text{Chrom} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

```
# crtbase结合crtbp创建一个有2个基本字符{0,1,2,3}、1个基本字符{0,1}
```

```
# 和3个基本字符{0,1,2}的随机种群
```

```
BaseV=crtbase(np.array([[2,1,3]]),np.array([[4,2,3]]))
```

```
Chrom=crtbp(4,BaseV) # 生成一个包含4个个体，基因座为BaseV的随机种群
```

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

crtfld 参考资料

概要: 区域描述器生成函数。

描述:

该函数根据输入的参数生成区域描述器 FieldD 或 FieldDR。

语法:

- 1) FieldDR = crtfd(ranges)
- 2) FieldDR = crtfd(ranges, borders)
- 3) FieldD = crtfd(ranges, borders, precisions)
- 4) FieldD = crtfd(ranges, borders, precisions, codes)
- 5) FieldD = crtfd(ranges, borders, precisions, codes, scales)

详细说明:

区域描述器的结构较为复杂，该函数提供了一个自动化的方法来生成区域描述器。

ranges 是一个 2 行 n 列的矩阵 (注意是 numpy 的 array 类型)，代表 n 个变量的边界范围。

其中第 0 行代表各个变量的下界；第 1 行是代表各个变量的上界。

borders 是一个 2 行 n 列的矩阵，代表 n 个变量是否包含区间的边界，0 表示不包含该边界，1 表示包含。

其中第 0 行代表是否包含各个变量的下界；第 1 行是代表是否包含各个变量的上界。

precisions 是可选参数，是一个一维 list，表示变量的编码精度，其元素必须是非负的，缺省或为 None 时默认元素全为 0。例如其中一个元素是 4，表示对应变量的编码可以精确到小数点后 4 位。

注意：该“精度”仅是对采用二进制或格雷编码而言的，若使用的是实值编码，则 precisions 不再表示其精度，而是作为表示实值连续型变量的边界精度。例如：执行 FieldDR = crtfd(ranges, borders)，由于 precisions 缺省，因此 FieldDR 表示的控制变量为离散实值。若想要表示连续型的实值控制变量，则需要设置 precisions 的元素为任意正整数即可。例如：precisions = [1, 1]。

上面所说的“边界精度”是指当控制变量范围不包含边界时，crtfd 函数会根据 precisions 把范围收缩一定的程度。上面所说的“设置 precisions 的元素为任意正整数”就体现在这里。比如精度设为 1 时，而边界为 0 (即不包含范围边界)，则 crtfd 函数会把范围往里收缩 0.1。

codes 是可选参数，是一个一维 list，表示变量是用什么方式编码的，例如其中一个元素为 0 时表示对应的变量是采用标准二进制编码，1 表示格雷编码，当 codes 缺省或为 None 时，函数将生成只有 2 行的区域描述器 FieldDR。

codes 是可选参数，是一个一维 list，指明变量用的是算术刻度还是对数刻度，其元素为 0 或 1。例如其中一个元素是 0，表示对应变量是采用算术刻度；1 表示采用对数刻度。缺省或为 None 时默认其元素全为 0。

因此特别注意：crtfd 函数会自动对变量的 ranges 范围以及 borders 边界进行处理，最终返回一个规范的区域描述器。

有关区域描述器的概念详见 bs2int 以及 crtfd 的参考资料。

应用实例:

例 1：下面欲创建包含变量 x_1, x_2 的整数值种群，2 个变量的区间范围分别是 [-3,5) 和 [2,10]，分别使用对数刻度的标准二进制编码和算术刻度的格雷编码，创建一个区域描述器来描述它。

```
x1 = [-3, 5]          # 自变量1的范围
x2 = [2, 10]          # 自变量2的范围
b1 = [1, 0]           # 自变量1的边界
b2 = [1, 1]           # 自变量2的边界
codes = [0, 1]         # 变量编码方式，分别采用二进制和格雷编码
precisions = [0, 0]      # 各变量的精度，0表示精确到个位
scales = [1, 0]         # 采用算术刻度
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
# 调用crtfd函数生成区域描述器
FieldD = crtfd(ranges, borders, precisions, codes, scales)
```

$$\text{FieldD} = \begin{pmatrix} 3 & 4 \\ -3 & 2 \\ 4 & 10 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \end{pmatrix}$$

解析：crtfd 函数对变量的区间范围以及边界进行了处理，返回的区域描述器中 x_1 的边界值已经被合理地调整为 [-3,4]，相应地，原本应该为 [1,0] 的 lbin 已经被修改为 [1,1]。

FieldD 中，第一行的 lens 参数是根据变量的范围计算得到的。本例中修正后的变量范围是 [-3,4] 和 [2,10]，意味着分别至少需要用 3 位和 4 位的二进制数来进行编码，因此 lens 参数的值是 [3 4]。

例 2：欲创建一个包含变量 x_1, x_2, x_3 的实数值种群，3 个变量的区间范围分别是 (-2.5,2), [3,5], [-4.8,3.6]，分别精确到小数点后 2、3、4 位。创建一个描述它的区域描述器：

```
x1 = [-2.5, 2]          # 自变量1的范围
x2 = [3, 5]              # 自变量2的范围
x3 = [-4.8, 3.6]         # 自变量3的范围
b1 = [0, 0]               # 自变量1的边界
b2 = [1, 1]               # 自变量2的边界
b3 = [1, 0]               # 自变量3的边界
precisions =[3, 3, 4]      # 各变量的精度，3表示精确到小数点后3位
ranges = np.vstack([x1, x2, x3]).T # 生成自变量的范围矩阵
borders = np.vstack([b1, b2, b3]).T # 生成自变量的边界矩阵
# 调用crtfd函数生成区域描述器
FieldDR = crtfd(ranges, borders, precisions)
```

$$\text{FieldDR} = \begin{pmatrix} -2.99 & 3.0 & -4.8 \\ 1.99 & 5.0 & 3.5999 \end{pmatrix}$$

crtip 参考资料

概要: 创建整数值初始种群。

描述:

该函数创建一个整数值的初始种群，矩阵的每一行代表一个个体的染色体串，染色体串是由随机的十进制整数构成的。

语法:

```
Chrom = crtip(Nind, FieldDR)
```

详细说明:

该函数利用随机函数 `rand` 和四舍五入法生成一个由十进制整数组成的种群矩阵，在遗传算法中，这种矩阵是不需要进行解码的。矩阵的每一列控制着一个变量的表现型。

`Nind` 是一个整数，代表种群的大小，即种群包含的个体数。

`FieldDR` 是一个 2 行 `Nvar` 列的矩阵 (`Nvar` 为种群中每个个体的变量个数)，称为区域描述器，但它不是译码矩阵，因为整数值种群不需要进行译码。它描述了变量的边界范围，第一行代表变量的下界，第二行代表变量的上界，并且不考虑变量是否包含边界的情况。它在变异函数里也有应用。

区域描述器 `FieldDR` 具有下面的结构：

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

应用实例:

```
# 定义边界范围变量
FieldDR=np.array([[-3.1, -2, 0, 3], # 下界
                  [4.2, 2, 1, 3]]) # 上界
crtip(4, FieldDR) # 创建一个包含4个个体的随机整数值种群。
```

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

crtpp 参考资料

概要: 创建排列编码初始种群。

描述:

该函数创建一个具有排列编码的初始种群矩阵，矩阵的每一行都是一个无重复数字的随机排列。

语法:

```
Chrom = crtpp(Nind, Lind, VarLen)
```

详细说明:

Nind 是一个整数，代表种群的大小，即种群包含的个体数，实际上它也是染色体数。

Lind 是一个整数，代表种群染色体长度。

VarLen 是一个整数，代表排列编码种群矩阵的元素的最大可能值，VarLen 必须不小于 Lind。

生成的种群矩阵，每一行都是一个从 [1, 2, 3,...,VarLen] 中抽取 Lind 个数组成的排列。

应用实例:

从 1-8 中抽取 6 个数构成染色体，并且生成含有 4 条染色体的排列种群：

```
Nind = 4 # 染色体数  
Lind = 6 # 染色体长度  
VarLen = 8 # 排列集合大小  
OldChrom = crtpp(Nind, Lind, VarLen)
```

$$\text{Chrom} = \begin{pmatrix} 3 & 4 & 2 & 7 & 8 & 6 \\ 5 & 4 & 7 & 1 & 2 & 3 \\ 8 & 7 & 1 & 3 & 4 & 2 \\ 2 & 6 & 5 & 1 & 3 & 8 \end{pmatrix}$$

crtrp 参考资料

概要: 创建实数值初始种群。

描述:

该函数利用随机函数 rand 生成一个实数值的种群矩阵，矩阵的每一行代表一个个体的染色体串，染色体串是由随机的十进制实数构成的。

语法:

```
Chrom = crtrp(Nind, FieldDR)
```

详细说明:

该函数生成一个由十进制实数组成的随机种群矩阵，在遗传算法中，这种矩阵是不需要进行解码的。矩阵的每一列控制着一个变量的表现型。

Nind 是一个整数，代表种群的大小，即种群包含的个体数。

FieldDR 是一个 2 行 Nvar 列的矩阵 (Nvar 为种群中每个个体的变量个数)，其结构详见 crtpr 参考资料，称为区域描述器，但它不是译码矩阵，因为实数值种群不需要进行译码。它描述了变量的边界范围，第一行代表变量的下界，第二行代表变量的上界，并且不考虑变量是否包含边界的情况。它在变异函数里也有应用。

区域描述器 FieldDR 具有下面的结构：

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

应用实例:

```
# 定义边界范围变量
FieldDR=np.array([[ -3, -4, 0, 2], # 下界
                  [2, 3, 2, 2]]) # 上界
crtrp(4, FieldDR) # 创建一个包含4个个体的随机实数值种群。
```

$$\text{Chrom} = \begin{pmatrix} 1.21326674 & 0.42325289 & 0.31157707 & 2.0 \\ 0.33649818 & 0.98672591 & 0.47582162 & 2.0 \\ 0.076081 & 2.21511576 & 0.46025118 & 2.0 \\ 1.87598638 & 0.31419139 & 0.53891627 & 2.0 \end{pmatrix}$$

etour 参考资料

概要: 精英锦标赛选择(低级选择函数)。

描述:

该函数利用精英保留策略的锦标赛选择法对种群进行选择，并返回所选择的个体在种群中的索引值。

语法:

NewChrIx = etour(FitnV, Nsel)

详细说明:

与传统锦标赛选择不同的是，传统锦标赛选择是通过随机选取个体参加锦标赛的，而精英锦标赛选择则采用精英保留策略，确保了精英个体一定能够被选中参与锦标赛。

FitnV 是一个列向量，代表种群中各个个体的适应度值。

Nsel 是一个正整数，代表被选择的个体数(可以比父代的个体数多)。

锦标赛的竞赛规模的值 tour(算法中的一个变量，使用函数时无需考虑)是根据 FitnV 最大值的向上取整来确定的。

比如：

$$\text{FitnV} = \begin{pmatrix} 1.2 \\ 0.8 \\ 2.1 \\ 3.2 \\ 0.6 \end{pmatrix}$$

那么竞赛规模 tour=4。

竞赛规模 tour 的值必须在 [1, Nind] 之间(其中 Nind 为种群的个体数)。当 tour>Nind 时，取 FitnV 平均值的向上取整，若 tour 仍大于 Nind，则默认取 tour=2。

当传入的 FitnV 是由 ranking 函数生成时，实际上 tour 等价于 ranking 函数里面的择优差 SP。

应用实例:

现有一个种群，其个体的适应度如下：

$$\text{FitnV} = \begin{pmatrix} 1.2 \\ 0.8 \\ 2.1 \\ 3.2 \\ 0.6 \\ 2.2 \\ 1.7 \\ 0.2 \end{pmatrix}$$

用锦标赛选择法从中选出 6 个个体。

```
FitnV = np.array([[1.2],[0.8],[2.1], [3.2],[0.6],[2.2],[1.7],[0.2]])
NewChrIx = etour(FitnV, 6)
```

得到所选择个体的索引值为：

$$\text{NewChrIx} = \begin{pmatrix} 5 & 3 & 4 & 6 & 3 & 0 \end{pmatrix}$$

frontplot 参考资料

概要: 多目标优化帕累托前沿绘图函数。

描述:

该函数根据帕累托最优集目标函数值矩阵 NDSetObjV 绘制 2 维或 3 维的目标函数值散点图和动画。

语法:

```
newAx = frontplot(NDSetObjV, saveFlag)
newAx = frontplot(NDSetObjV, saveFlag, ax)
newAx = frontplot(NDSetObjV, saveFlag, ax, gen)
newAx = frontplot(NDSetObjV, saveFlag, ax, gen, interval)
newAx = frontplot(NDSetObjV, saveFlag, ax, gen, interval, title)
newAx = frontplot(NDSetObjV, saveFlag, ax, gen, interval, title, save_path)
```

详细说明:

NDSetObjV 是一个 numpy 的 array 类型的矩阵，用于存储帕累托最优解的目标函数函数值。每一列对应一个目标函数值，其列数必须为 2 或 3。

saveFlag 是布尔类型的标记，表示是否要保存图片。当要绘制动画时，必须设为 False。

ax 是可选参数，在绘制动画的时候需要传入。其代表上一帧的动画。当画第一帧时，其值为 None。

gen 是可选参数，表示当前进化代数，默认为 None。当该参数为非 None 时，图片将在图例部分显示当前 gen 的值以及前沿点数。

interval 是可选参数，表示两帧动画之间的间隔时间，默认为 0.1，单位为'秒'。

title 是可选参数，表示图形的标题名称。

save_path 是 string 类型的可选参数，表示保存图片的路径。

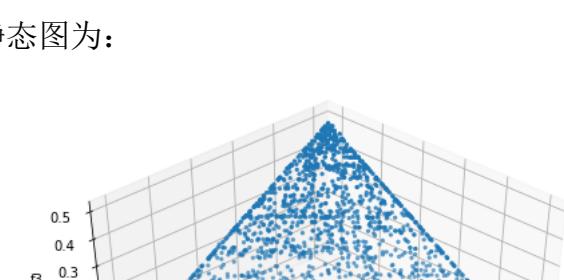
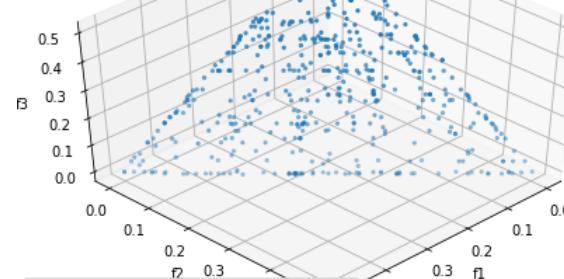
newAx 代表新的图形，是更新后的 ax。

应用实例:

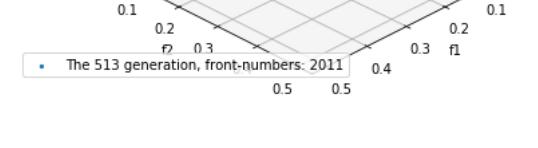
在解决 3 目标优化问题时绘制进化过程中的帕累托前沿的动画，以观察搜索到的帕累托前沿的动态变化。并且在进化结束后绘制一个最终的帕累托前沿图。

```
...
ax = None
# 开始进化
for gen in range(MAXGEN)
...
# 进化操作
...
# 更新帕累托最优集目标函数值NDSetObjV
...
# 绘图
ax = ga.frontplot(NDSetObjV, False, ax, gen + 1)
# 进化结束
ga.frontplot(NDSetObjV, True) # 绘制最终的帕累托前沿图
# end
```

运行结果动画部分截图如下：



最终的前沿面静态图为：



indexing 参考资料

概要: 基于指数变换的适应度计算。

描述:

该函数对目标函数值 ObjV 作指数变换，使其变成受 β 影响的指数尺度的适应度值。

变换公式: $Fit = e^{-\beta * ObjV} + 1$

该函数遵循“最小适应度为 0”的约定（特殊情况除外）。

语法:

```
FitnV = indexing(ObjV)
FitnV = indexing(ObjV, LegV)
FitnV = indexing(ObjV, LegV, Beta)
FitnV = indexing(ObjV, LegV, Beta, SUBPOP)
```

详细说明:

该函数先将个体的目标值 ObjV 进行 0-1 标准化，然后进行指数变换，最终返回一个代表种群适应度的列向量 FitnV。

LegV 是一个可选参数，保存着个体对应的可行性的列向量，0 表示该个体是非可行解，1 表示是可行解。

Beta 是一个正实数，其值影响指数变换。缺省情况下默认 Beta 为 1。

SUBPOP 是一个正整数，代表子种群的数量。SUBPOP 必须能够被种群个体数整除。关于子种群的概念详见 migrate 参考资料。

注：Geatpy 的适应度遵循“种群目标函数值越大，适应度越小”的原则。

并且当子种群的所有个体的目标函数值相等时，其对应的适应度值均为 1。

特别注意:

本函数是根据传入参数 ObjV 来计算适应度的，且遵循“种群目标函数值越大，适应度越小”的原则，因此在调用本函数前，需要对传入的 ObjV 乘上'maxormin'(最大最小化标记)。但是，由于返回的是 FitnV，它与 ObjV 在含义上无关了，因此不需要对其乘上'maxormin' 进行还原。

应用实例:

现有一个拥有 10 个个体的种群，每个个体的目标函数值为 1, 2, 2, 4, 5, 10, 9, 8, 7, 6，若该种群包含 2 个子种群，求其指数尺度变换的适应度值。

```
ObjV = np.array([[1], [2], [2], [4], [5], [10], [9], [8], [7], [6]])
LegV = np.array([[1], [1], [1], [1], [1], [1], [1], [1], [1], [1]])
FitnV = indexing(ObjV, LegV, 2, 2) # 设定Beta值为2, SUBPOP为2
```

$$FitnV = \begin{pmatrix} 2 \\ 1.60653066 \\ 1.60653066 \\ 1.22313016 \\ 1.13533528 \\ 1.13533528 \\ 1.22313016 \\ 1.36787944 \\ 1.60653066 \\ 2 \end{pmatrix}$$

解析：该种群拥有 10 个个体，但因为有 2 个子种群，所以前 5 个为第一个子种群，后 5 个为第二个子种群。由结果可见，目标函数值越大的个体适应度值越小。

meshrng 参考资料

概要: 网格化变量范围。

描述:

该函数对变量范围进行等长均分，再排列所有情况得到网格化的变量范围，并存储在 list 类型的变量中。

语法:

```
newRanges = meshrng(ranges)
newRanges = meshrng(ranges, gridnum)
```

详细说明:

`ranges` 是一个 2 行 n 列的矩阵 (注意是 numpy 的 array 类型)，代表 n 个变量的边界范围。其中第 0 行是代表各个变量的下界；第 1 行是代表各个变量的上界。

`gridnum` 是正整数，表示网格化时每个控制变量被均分的份数。

应用实例:

有两个控制变量 x_1 和 x_2 ，范围均是 [0, 1]，对其进行网格化：

```
import numpy as np
import geatpy as ga
x1 = [0, 1] # 自变量1的范围
x2 = [0, 1] # 自变量2的范围
ranges=np.vstack([x1, x2]).T # 生成自变量的范围矩阵
# 对控制变量范围进行网格化，网格边长为2
newRanges = ga.meshrng(ranges, gridnum = 1)
print(newRanges)
```

运行得到一个存储着网格化后所有各个网格范围的列表，结果如下：

```
[array([[0. , 0. ],[0.5, 0.5]]),
array([[0. , 0.5],[0.5, 1. ]]),
array([[0.5, 0. ],[1. , 0.5]]),
array([[0.5, 0.5],[1. , 1. ]])]
```

migrate 参考资料

摘要: 子种群间迁移个体。

描述:

该函数在当前种群 Chrom 的子种群之间实现个体迁移，并返回迁移后的种群 Chrom。

语法:

```
Chrom = migrate(Chrom, SUBPOP)
Chrom = migrate(Chrom, SUBPOP, MIGR)
Chrom = migrate(Chrom, SUBPOP, MIGR, Select)
Chrom = migrate(Chrom, SUBPOP, MIGR, Select, Structure)
Chrom = migrate(Chrom, SUBPOP, MIGR, Select, Structure, FitnV)
[Chrom, ObjV] = migrate(Chrom, SUBPOP, MIGR, Select, Structure, FitnV, ObjV)
[Chrom, ObjV, LegV] = migrate(Chrom, SUBPOP, MIGR, Select, Structure, FitnV, ObjV,
LegV)
```

详细说明:

Geatpy 中子种群的染色体矩阵是基于种群染色体矩阵 Chrom 的，子种群之间拥有相同数量的个体，并且按照下列方案进行有序排列：

$$\text{Chrom} = \begin{pmatrix} subchrom_1_ind_1 \\ subchrom_1_ind_2 \\ \vdots \\ subchrom_1_ind_n \\ subchrom_2_ind_1 \\ subchrom_2_ind_2 \\ \vdots \\ subchrom_2_ind_n \\ \vdots \\ subchrom_{SUBPOP}_ind_1 \\ subchrom_{SUBPOP}_ind_2 \\ \vdots \\ subchrom_{SUBPOP}_ind_n \end{pmatrix}$$

Chrom 是一个种群矩阵，矩阵的每一行代表种群中的一个个体的染色体。本函数没对种群的编码方式作特殊要求。

SUBPOP 是一个正整数，代表子种群的数量。SUBPOP 必须能够被种群个体数整除。

MIGR 是一个在 [0,1] 间的实数，代表种群间的迁移概率。缺省情况下默认为 0.2，即子种群中将会有 20% 的个体参与迁移。

Select 是一个正整数，表示种群迁移时所选择的方式。Select 为 0 时表示均匀迁移，此时会对子种群的个体进行随机排序，然后选出排序较前的一些个体进行迁移；Select 为 1 时表示基于适应度的迁移，此时会根据子种群个体的目标函数值 ObjV 或适应度 FitnV 进行排序，并选出该值较小的一些个体进行迁移。

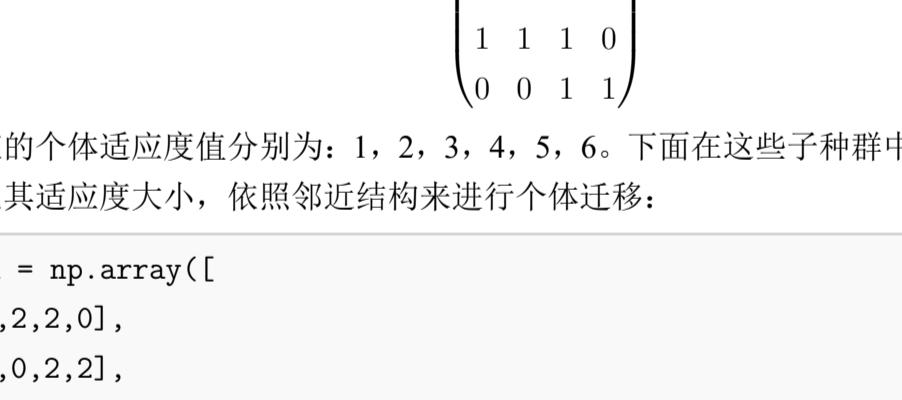
LegV 和 FitnV 都是列向量，LegV 代表种群中各个个体的可行性 (0 为非可行解，1 为可行解)；FitnV 代表种群中各个个体的适应度。

ObjV 是种群个体的目标函数值组成的矩阵，每一列对应一个目标，因此该函数适用于多目标的种群。

Structure 是一个正整数，表示种群迁移的结构。为 0 时表示完全网状结构；为 1 时表示邻近结构；为 2 时表示环状结构。

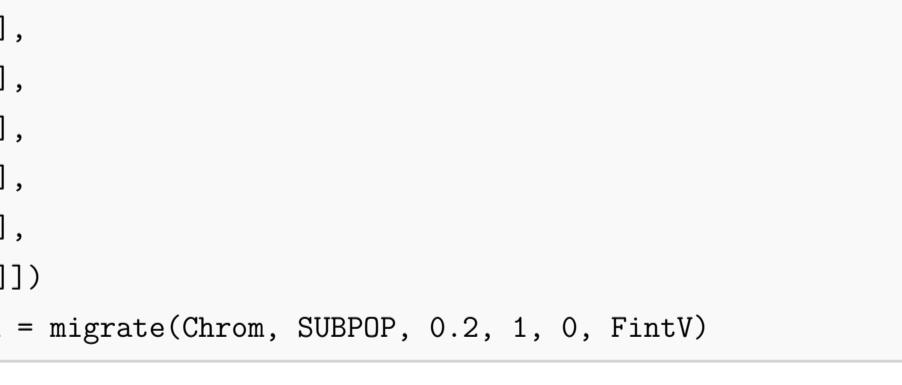
1) 完全网状结构：

每个子种群都会向其他子种群迁移个体。



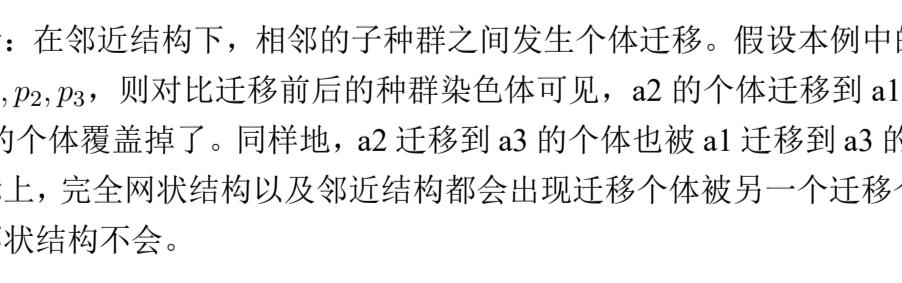
2) 邻近结构：

将子种群首尾相接，相邻的两个子种群之间发生个体迁移。



3) 环状结构：

将子种群首尾相接，1 号子种群的个体向 2 号子种群迁移，2 号子种群个体向 3 号子种群迁移……最后一个子种群的个体向 1 号子种群迁移。



特别注意:

本函数是根据随机方法或根据 FitnV 来进行种群迁移的，与 ObjV 无关，因此在调用本函数前，不需要对传入的 ObjV 乘上'maxmin'（最大最小化标记），对于返回的 ObjV，也不需要乘上'maxmin' 进行还原。

应用实例:

现有一个拥有 3 个子种群的种群，其染色体矩阵如下：

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

对应的个体适应度值分别为：1, 2, 3, 4, 5, 6。下面在这些子种群中选择 20% 的个体按照其适应度大小，依照邻近结构来进行个体迁移：

```
Chrom = np.array([
    [2,2,2,0],
    [1,0,2,2],
    [0,1,1,0],
    [0,0,1,0],
    [1,1,1,0],
    [0,0,1,1]])
```

```
SUBPOP = 3
```

```
FintV = np.array([
    [1],
    [2],
    [3],
    [4],
    [5],
    [6]])
```

```
Chrom = migrate(Chrom, SUBPOP, 0.2, 1, 0, FintV)
```

迁移后的种群染色体矩阵如下：

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

解析：在邻近结构下，相邻的子种群之间发生个体迁移。假设本例中的 3 个子种群分别是 p_1, p_2, p_3 ，则对比迁移前后的种群染色体可见， a_2 的个体迁移到 a_1 后，被 a_3 迁移到 a_1 的个体覆盖掉了。同样地， a_2 迁移到 a_3 的个体也被 a_1 迁移到 a_3 的个体所覆盖。

实际上，完全网状结构以及邻近结构都会出现迁移个体被另一个迁移个体覆盖的情况，而环状结构不会。

参考文献:

[1] H. Mühlenbein, M. Schomisch and J. Born, “The Parallel Genetic Algorithm as a Function Optimizer”, Parallel Computing, No. 17, pp.619-632, 1991.

[2] T. Starkweather, D. Whitley and K. Mathias, “Optimization using Distributed Genetic Algorithms”, In Parallel Problems Solving from Nature, Lecture Notes in Computer Science, Vol. 496, pp. 176-185, Springer, 1991.

[3] R. Tanese, “Distributed Genetic Algorithms”, Proc. ICGA 3, pp. 434-439, Morgan Kaufmann Publishers, 1989.

[4] H.-M. Voigt, J. Born and I. Santibanez-Koref, “Modelling and Simulation of Distributed Evolutionary Search Processes for Function Optimization”, Parallel Problems Solving from Nature, Lecture Notes in Computer Science, Vol. 496, pp. 373-380, Springer Verlag, 1991.

mut 参考资料

摘要: 简单离散变异算子。

描述:

该函数用给定的概率对种群的染色体进行变异，并返回变异后的结果。

语法:

```
NewChrom = mut(OldChrom)
NewChrom = mut(OldChrom, BaseV)
NewChrom = mut(OldChrom, BaseV, Pm)
NewChrom = mut(OldChrom, BaseV, Pm, params3)
NewChrom = mut(OldChrom, BaseV, Pm, params3, params4)
```

详细说明:

所谓离散种群即的种群矩阵的每个元素都是离散值，如：二进制/格雷码编码的种群、整数值种群等。简单变异是指仅根据染色体基因座矩阵 BaseV 而并非区域描述器来进行变异。

OldChrom 即变异前的简单离散种群矩阵，其概念详见 crtbp 参考资料。

BaseV 代表染色体基因座，其概念详见 crtbp 参考资料。

Pm 是一个在 [0,1] 上的实数，代表变异的概率。缺省时默认 $Pm = 0.7/Lind$ ，其中 Lind 为种群个体的染色体长度。

params3 和 params4 是无用参数，目的是为了与其他变异函数兼容，以便被高级变异函数调用。

应用实例:

根据 BaseV 使用 crtbp 创建一个有 3 个个体的简单离散种群，然后用 mut 函数进行简单离散变异。

```
BaseV = np.array([[4,4,2,5]])
OldChrom = crtbp(3, BaseV)
NewChrom = mut(OldChrom, BaseV, 0.01) # 变异概率为0.01
```

变异前种群矩阵如下：

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

变异后，种群矩阵如下：

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

参考文献:

[1] Jürgen Hesser and Reinhard Männer, “Towards an Optimal Mutation Rate Probability for Genetic Algorithms”, In Parallel Problem Solving from Nature, Lecture Notes in Computer Science, Vol. 496, pp23-32, 1990.

mutate 参考资料

概要: 高级变异函数。

描述:

该函数接收一个种群矩阵，并分别对各个子种群调用低级变异函数进行个体变异，最终返回新的种群矩阵。

语法:

```
NewChrom = mutate(MUT_F, SUBPOP, OldChrom)
NewChrom = mutate(MUT_F, SUBPOP, OldChrom, params3)
NewChrom = mutate(MUT_F, SUBPOP, OldChrom, params3, params4)
NewChrom = mutate(MUT_F, SUBPOP, OldChrom, params3, params4, params5)
```

详细说明:

MUT_F 是一个字符串，其值是低级变异函数的函数名。低级变异函数有：mut, mut-bga, mutbin, mutint。

SUBPOP 是一个正整数，代表子种群的数量。SUBPOP 必须能够被种群个体数整除。

OldChrom 是变异前的种群矩阵，每一行对应着一个个体。

params3, params4, params5 均是要传入低级变异函数的一些参数，其具体含义与 MUT_F 指定的低级变异函数所需参数含义一致

应用实例:

现有一个拥有 2 个子种群的整数值种群：

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

该种群所代表的四个变量的范围分别是 [-3,4], [-3, 2], [-3,2], [3,4]。调用 mutate 函数对该种群进行概率为 0.01 的变异。

```
OldChrom = np.array([
    [-2, -2, 0, 3],
    [1, 2, 0, 4],
    [1, -3, -2, 3],
    [3, 1, 1, 3]])
FieldDR = np.array([
    [-3, -3, -3, 3],
    [4, 2, 2, 4]])
SUBPOP = 2
NewChrom = mutate('mutint', SUBPOP, OldChrom, FieldDR, 0.01)
```

变异后，种群矩阵如下：

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

mutbga 参考资料

摘要: 实值突变。

描述:

该函数用给定的概率对实数值种群的染色体进行变异，并返回变异后的结果。

语法:

```
NewChrom = mutbga(OldChrom, FieldDR)
NewChrom = mutbga(OldChrom, FieldDR, Pm)
NewChrom = mutbga(OldChrom, FieldDR, Pm, MutShrink)
NewChrom = mutbga(OldChrom, FieldDR, Pm, MutShrink, Gradient)
```

详细说明:

所谓实数值种群即种群矩阵的每个元素都是实数。

OldChrom 即变异前的实数值种群矩阵。

FieldDR 是区域描述器，其概念详见 crtrp 参考资料。

Pm 是一个在 [0,1] 上的实数，代表变异的概率。缺省时默认 $Pm = 0.7/Lind$ ，其中 **Lind** 为种群个体的染色体长度。

MutShrink 是一个在 [0,1] 上的实数，代表压缩率，用于压缩变异结果，缺省情况下默认为 1。

当 **MutShrink** 为 1 时，该函数会增强对控制变量边界区域的变异。即变异结果更容易出现在控制变量的范围边界。

Gradient 是变异距离的梯度划分个数，表示将变异距离划分多少个梯度，函数将根据梯度来变异。例如：控制变量的范围为 0-4，**Gradient** = 4，那么划分梯度为：1., 0.5, 0.25, 0.125，变异时，从这 4 个数中随机选择 1 个到 4 个求和后乘上变量范围，得变异距离，然后进行变异。当超出变量范围时，取变量的边界值。默认情况下，**Gradient** 的值为 20。

应用实例:

根据 **FieldDR** 使用 **crtrp** 创建一个有 3 个个体的简单离散种群，然后用 **mutbga** 函数进行实数值变异（变异概率设为 0.1，压缩率设为 1）。

```
FieldDR = np.array([
    [8, 7],
    [10, 10]])
OldChrom = crtrp(3, FieldDR)
NewChrom = mutbga(OldChrom, FieldDR, 0.1, 1)
```

变异前种群矩阵如下：

$$\text{OldChrom} = \begin{pmatrix} 9.28458271 & 7.0 \\ 8.0 & 7.0 \\ 9.67620623 & 9.99391841 \end{pmatrix}$$

变异后，种群矩阵如下：

$$\text{NewChrom} = \begin{pmatrix} 9.28458271 & 7.0 \\ 8.00000763 & 7.0 \\ 9.67620623 & 9.98796749 \end{pmatrix}$$

参考文献:

[1] H. Mühlenbein and D. Schlierkamp-Voosen, “Predictive Models for the Breeder Genetic Algorithm: I. Continuous Parameter Optimization”, Evolutionary Computation, Vol. 1, No. 1, pp.25-49, 1993.

mutbin 参考资料

概要: 二进制变异算子。

描述:

该函数让一个二进制种群矩阵根据其突变率让个体的每个变量发生突变，并返回一个新的种群。

语法:

```
NewChrom = mutbin(OldChrom, Pm)  
NewChrom = mutbin(OldChrom, Pm, params2)  
NewChrom = mutbin(OldChrom, Pm, params2, params3)  
NewChrom = mutbin(OldChrom, Pm, params2, params3, params4)
```

详细说明:

所谓二进制种群即种群矩阵的每个元素都是 0 或 1。

OldChrom 即变异前的实数值种群矩阵。

Pm 是一个在 [0,1] 上的实数，代表变异的概率。缺省时默认 $Pm = 0.7/Lind$ ，其中 Lind 为种群个体的染色体长度。

params2, params3 和 params4 都是无用参数，目的是为了与其他变异函数兼容，以便被高级变异函数调用。

应用实例:

现有二进制种群如下：

$$\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

调用 mutbin 函数对其进行变异 (变异概率设为 0.1)：

```
OldChrom = np.array([  
    [1, 0, 0, 1],  
    [0, 0, 1, 1],  
    [0, 0, 1, 0]])  
  
NewChrom = mutbin(Chrom, 0.1)
```

变异后结果如下：

$$NewChrom = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

mutgau 参考资料

摘要: 实数值高斯突变。

描述:

该函数用给定的概率对实数值种群的染色体进行高斯变异，并返回变异后的结果。

该函数与 mutbga 类似，不同的是，高斯变异会产生更强的变异效果，这种效果是提高变异概率也无法达到的。因此可以在恰当的时候进行一次高斯变异来提高种群的多样性。

语法:

```
NewChrom = mutgau(OldChrom, FieldDR)
NewChrom = mutgau(OldChrom, FieldDR, Pm)
NewChrom = mutgau(OldChrom, FieldDR, Pm, MutShrink)
NewChrom = mutgau(OldChrom, FieldDR, Pm, MutShrink, params4)
```

详细说明:

所谓实数值种群即种群矩阵的每个元素都是实数。

OldChrom 即高斯变异前的实数值种群矩阵。

FieldDR 是区域描述器，其概念详见 crtrp 参考资料。

Pm 是一个在 [0,1] 上的实数，代表变异的概率。缺省时默认 $Pm = 0.7/Lind$ ，其中 Lind 为种群个体的染色体长度。

MutShrink 是一个在 [0,1] 上的实数，用于压缩变异结果，缺省情况下默认为 1。

params4 是无用参数，目的是为了与其他变异函数兼容，以便被高级变异函数调用。

应用实例:

根据 FieldDR 使用 crtrp 创建一个有 3 个个体的简单离散种群，然后用 mutgau 函数进行实数值高斯变异（变异概率设为 0.1）。

```
FieldDR = np.array([
    [8,7],
    [10,10]])
OldChrom = crtrp(3, FieldDR)
NewChrom = mutgau(OldChrom, FieldDR, 0.1)
```

变异前种群矩阵如下：

$$\text{OldChrom} = \begin{pmatrix} 9.76735356 & 8.0661952 \\ 9.69070615 & 7.0 \\ 8.0 & 7.21464982 \end{pmatrix}$$

变异后，种群矩阵如下：

$$\text{NewChrom} = \begin{pmatrix} 10.0 & 8.37431304 \\ 9.69070615 & 7.05298091 \\ 8.0 & 9.27708553 \end{pmatrix}$$

mutint 参考资料

摘要: 整数值变异算子。

描述:

该函数让一个整数值种群矩阵根据其突变率让个体的每个变量发生突变，并返回一个新的种群。

语法:

```
NewChrom = mutint(OldChrom, FieldDR)
NewChrom = mutint(OldChrom, FieldDR, Pm)
NewChrom = mutint(OldChrom, FieldDR, Pm, params3)
NewChrom = mutint(OldChrom, FieldDR, Pm, params3, params4)
```

详细说明:

所谓整数值种群即种群矩阵的每个元素都是整数值。

OldChrom 即变异前的整数值种群矩阵。

FieldDR 是区域描述器，其概念详见 crtsp 参考资料。

Pm 是一个在 [0,1] 上的实数，代表变异的概率。缺省时默认 $Pm = 0.7/Lind$ ，其中 Lind 为种群个体的染色体长度。

params3 和 params4 是无用参数，目的是为了与其他变异函数兼容，以便被高级变异函数调用。

应用实例:

现有整数值种群如下：

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

区域描述器为：

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

调用 mutint 函数对其进行变异(设置变异概率为 0.1)：

```
Chrom=np.array([
    [-2, 1, 0, 3],
    [1, 2, 0, 4],
    [1, 2, -2, 3],
    [3, 1, 1, 3]])

FieldDR=np.array([
    [-3, -1, -3, 3],
    [4, 3, 2, 4]])

Chrom=mutint(Chrom, FieldDR, 0.1)
```

变异后结果如下：

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

mutpp 参考资料

概要: 排列编码种群变异算子。

描述:

该函数对排列编码的种群进行变异，每个个体的染色体最多只对其中一位进行变异。当排列集合大小 VarLen 等于种群染色体长度时，该函数实际上是进行 2 点互换变异。

语法:

```
NewChrom = mutpp(OldChrom, VarLen)
NewChrom = mutpp(OldChrom, VarLen, Pm)
NewChrom = mutpp(OldChrom, VarLen, Pm, params3)
NewChrom = mutint(OldChrom, FieldDR, Pm, params3, params4)
```

详细说明:

排列编码种群即种群染色体是无重复数字的随机排列。

OldChrom 即变异前的排列编码种群矩阵。

VarLen 是一个整数，代表排列编码种群矩阵的元素的最大可能值，VarLen 必须不小于 Lind。

Pm 是一个在 [0,1] 上的实数，代表变异的概率。缺省时默认 $Pm = 0.7/Lind$ ，其中 Lind 为种群个体的染色体长度。

params3 和 params4 是无用参数，目的是为了与其他变异函数兼容，以便被高级变异函数调用。

应用实例:

使用 crtpp 创建一个有 4 个个体的排列编码种群，然后用 mutpp 函数进行变异。

```
Nind = 4 # 染色体数
Lind = 6 # 染色体长度
VarLen = 8 # 排列集合大小
OldChrom = crtpp(Nind, Lind, VarLen) # 创建排列编码种群
NewChrom = mutpp(OldChrom, VarLen, 0.1) # 突变率设为0.1，进行变异
```

变异前种群矩阵如下：

$$\text{Chrom} = \begin{pmatrix} 5 & 3 & 8 & 1 & 6 & 4 \\ 4 & 1 & 6 & 3 & 5 & 7 \\ 6 & 2 & 1 & 4 & 7 & 3 \\ 3 & 6 & 5 & 4 & 8 & 2 \end{pmatrix}$$

变异后：

$$\text{NewChrom} = \begin{pmatrix} 5 & 3 & 2 & 1 & 6 & 4 \\ 4 & 3 & 6 & 1 & 5 & 7 \\ 6 & 2 & 1 & 5 & 7 & 3 \\ 3 & 1 & 5 & 4 & 8 & 2 \end{pmatrix}$$

ndomin 参考资料

概要: 简单非支配排序 (nondominated-sorting)。

描述:

该函数利用简单非支配排序法构造种群的非支配集，即简单地遍历种群所有个体，计算每个个体被多少个其他个体支配，根据支配该个体的个体数计算并返回种群个体适应度及非支配个体的索引。若传入 exIdx 参数，则会对非可行解的个体进行排除。

语法:

```
[FitnV, frontIdx] = ndomin(ObjV)
[FitnV, frontIdx] = ndomin(ObjV, LegV)
```

详细说明:

ObjV 是种群个体的目标函数矩阵，每一行对应一个个体，每一列对应一个目标。

LegV 是一个保存着个体对应的可行性的列向量，0 表示该个体是非可行解，1 表示是可行解。

函数返回经过简单非支配排序后的种群个体适应度列向量 FitnV 以及非支配个体在种群中的索引：frontIdx，它是一个 numpy 的 array 类型行向量。

简单非支配排序的算法流程如下：

1. 遍历种群所有个体，计算各个个体被多少个种群中的其他个体支配 (保存在 snp 集合中)，
2. 根据第 2 步得到的 snp 集合，其中为 0 的元素对应的个体即为当代种群的非支配个体，
3. 个体的适应度 = 种群个体数 / (支配该个体的个体数 + 1)。

注意: Geatpy 的非支配排序均遵循最小化目标的约定。

特别注意:

本函数是根据传入参数 ObjV 来进行非支配排序的，且遵循“最小化目标”的约定，因此在调用本函数前，需要对传入的 ObjV 乘上'maxormin'(最大最小化标记)来让其符合约定。但是，由于返回的是 FitnV，它与 ObjV 在含义上无关了，因此不需要对其乘上'maxormin' 进行还原。

应用实例:

考虑一个两个目标的优化问题，设种群规模为 20，这 20 个个体的目标函数值如下：

```
(9,1),(7,2),(5,4),(4,5),(3,6),(2,7),(1,9),(10,3),(8,5),(7,6),(5,7),(4,8),(3,9),(10,5),(9,6),(8,7),
(7,9),(10,6),(9,7),(8,9)
```

使用简单非支配排序法计算该种群的非支配个体：

```
ObjV =
np.array([[9,1],[7,2],[5,4],[4,5],[3,6],[2,7],[1,9],[10,3],[8,5],
[7,6],[5,7],[4,8],[3,9],[10,5],[9,6],[8,7],[7,9],[10,6],[9,7],[8,9]])
[FitnV, frontIdx] = ndomin(ObjV)
```

得到的非支配个体索引 frontIdx 为： 0 1 2 3 4 5 6

即前 7 个个体是种群中的非支配个体。

再次提醒的是，frontIdx 是 numpy 的 array 类型的行向量，行向量和行矩阵的关系在“Geatpy 数据结构”章节中有详细描述。可以使用 print(frontIdx.shape) 来查看 frontIdx 的规格，可以发现结果为 (7,)，若是行矩阵的话，输出的是 (1,7) 而不是 (7,)。

ndomindeb 参考资料

概要: Deb 非支配排序 (nondominated-sorting of Deb)。

描述:

该函数利用 Deb 提出的非支配排序法 (Deb et al,2000) 来对种群个体按非支配关系进行分级，函数返回种群个体的适应度以及分级情况 (1,2,3,4……)，其中处在第 1 级的个体即为该种群的帕累托最优个体。若传入 exIdx 参数，则会对非可行解的个体进行排除。

语法:

```
[FitnV, levels] = ndomindeb(ObjV)
[FitnV, levels] = ndomindeb(ObjV, needLevel)
[FitnV, levels] = ndomindeb(ObjV, needLevel, LegV)
```

详细说明:

ObjV 是种群个体的目标函数矩阵，每一行对应一个个体，每一列对应一个目标。

needLevel 是一个可选参数，表示要对种群个体最大分多少级，缺省或为 None 时默认是 5。

LegV 是一个保存着个体对应的可行性的列向量，0 表示该个体是非可行解，1 表示是可行解。

返回参数中，FitnV 是一个 array 类型的列向量，每一行对应于一个个体的适应度。

levels 是一个 array 类型的行向量，每一行对应于一个个体所在的等级，如 1,2 或 3 等等。

注意: Geatpy 的非支配排序均遵循最小化目标的约定。

特别注意:

本函数是根据传入参数 ObjV 来进行非支配排序的，且遵循“最小化目标”的约定，因此在调用本函数前，需要对传入的 ObjV 乘上'maxormin'(最大最小化标记)来让其符合约定。但是，由于返回的是 FitnV，它与 ObjV 在含义上无关了，因此不需要对其乘上'maxormin' 进行还原。

应用实例:

考虑一个两个目标的优化问题，设种群规模为 20，这 20 个个体的目标函数值如下：

```
(9,1),(7,2),(5,4),(4,5),(3,6),(2,7),(1,9),(10,3),(8,5),(7,6),(5,7),(4,8),(3,9),(10,5),(9,6),(8,7),
(7,9),(10,6),(9,7),(8,9)
```

使用 Deb 非支配排序法计算该种群个体进行分级：

```
ObjV =
np.array([[9,1],[7,2],[5,4],[4,5],[3,6],[2,7],[1,9],[10,3],[8,5],
[7,6],[5,7],[4,8],[3,9],[10,5],[9,6],[8,7],[7,9],[10,6],[9,7],[8,9]])
[FitnV, levels] = ndomindeb(ObjV)
```

得到的个体分级情况如下：

```
1 1 1 1 1 1 2 2 2 2 2 3 3 3 3 4 4 4
```

再次提醒的是，levels 是 numpy 的 array 类型的行向量，行向量和行矩阵的关系在“Geatpy 数据结构”章节中有详细描述。可以使用 print(levels.shape) 来查看 levels 的规格，可以发现结果为 (20,)，若是行矩阵的话，输出的是 (1,20) 而不是 (20,)。

ndominfast 参考资料

摘要: 快速非支配排序 (nondominated-sorting fast)。

描述:

快速非支配排序算法与简单非支配排序法类似，但不同的是它不会遍历种群的全部个体，而是遍历排除了被支配个体后的种群，每趟遍历都对整个种群中的被支配个体进行淘汰。函数最后根据种群个体的非支配情况计算并返回种群个体适应度及非支配个体的索引。若传入 exIdx 参数，则会对非可行解的个体进行排除。

语法:

```
[FitnV, frontIdx] = ndominfast(ObjV)
[FitnV, frontIdx] = ndominfast(ObjV, LegV)
```

详细说明:

ObjV 是种群个体的目标函数矩阵，每一行对应一个个体，每一列对应一个目标。

LegV 是一个保存着个体对应的可行性的列向量，0 表示该个体是非可行解，1 表示是可行解。

函数返回经过快速非支配排序后的种群个体适应度列向量 FitnV 以及非支配个体在种群中的索引：frontIdx，它是一个 numpy 的 array 类型行向量。

注意： Geatpy 的非支配排序均遵循最小化目标的约定。

特别注意:

本函数是根据传入参数 ObjV 来进行非支配排序的，且遵循“最小化目标”的约定，因此在调用本函数前，需要对传入的 ObjV 乘上'maxormin'(最大最小化标记)来让其符合约定。但是，由于返回的是 FitnV，它与 ObjV 在含义上无关了，因此不需要对其乘上'maxormin' 进行还原。

应用实例:

考虑一个两个目标的优化问题，设种群规模为 20，这 20 个个体的目标函数值如下：

```
(9,1),(7,2),(5,4),(4,5),(3,6),(2,7),(1,9),(10,3),(8,5),(7,6),(5,7),(4,8),(3,9),(10,5),(9,6),(8,7),
(7,9),(10,6),(9,7),(8,9)
```

使用快速非支配排序法计算该种群的非支配个体：

```
ObjV =
np.array([[9,1],[7,2],[5,4],[4,5],[3,6],[2,7],[1,9],[10,3],[8,5],
[7,6],[5,7],[4,8],[3,9],[10,5],[9,6],[8,7],[7,9],[10,6],[9,7],[8,9]])
[FitnV, frontIdx] = ndominfast(ObjV)
```

得到的非支配个体索引 frontIdx 为： 0 1 2 3 4 5 6

即前 7 个个体是种群中的非支配个体。

再次提醒的是，frontIdx 是 numpy 的 array 类型的行向量，行向量和行矩阵的关系在“Geatpy 数据结构”章节中有详细描述。可以使用 print(frontIdx.shape) 来查看 frontIdx 的规格，可以发现结果为 (7,)，若是行矩阵的话，输出的是 (1,7) 而不是 (7,)。

powing 参考资料

概要: 幂尺度变换适应度计算

描述:

该函数对目标函数值 ObjV 进行幂尺度变换，使其变成由 k 影响的幂尺度尺度。计算公式: $F' = (-\text{ObjV})^k + 1$:

遵循“最小适应度为 0”的约定（特殊情况除外）。

语法:

$\text{FitnV} = \text{powing}(\text{ObjV})$

$\text{FitnV} = \text{powing}(\text{ObjV}, \text{LegV})$

$\text{FitnV} = \text{powing}(\text{ObjV}, \text{LegV}, k)$

$\text{FitnV} = \text{powing}(\text{ObjV}, \text{LegV}, k, \text{SUBPOP})$

详细说明:

ObjV 为一个保存着个体对应的目标函数值的列向量。

LegV 是一个可选参数，保存着个体对应的可行性的列向量，0 表示该个体是非可行解，1 表示是可行解。

k (可选参数) 为 $(0, +\infty]$ 上的正实数，若缺省或为 `None`，则默认值为 1。

$k < 1$ 时，适应度大的个体经过幂指数变换后适应度相差较小，此时为“保持种群多样性策略”；

$k > 1$ 时，适应度大的个体经过幂指数变换后适应度相差会较大，此时为“精英策略”。

SUBPOP (可选参数) 表示子种群的数量，要求能够被种群个体数整除。缺省时默认为 1。

FitnV 是记录着种群个体适应度值的列向量。

该函数遵循“目标函数值越大适应度越小”的约定。

特别注意:

本函数是根据传入参数 ObjV 来计算适应度的，且遵循“种群目标函数值越大，适应度越小”的原则，因此在调用本函数前，需要对传入的 ObjV 乘上`'maxormin'`(最大最小化标记)。但是，由于返回的是 FitnV ，它与 ObjV 在含义上无关了，因此不需要对其乘上`'maxormin'` 进行还原。

应用实例:

考虑有 10 个个体的种群，其当前目标值 ObjV 如下情况。

```
ObjV = np.array([[ 1], [ 2], [ 3], [ 4], [ 5], [10], [ 9], [ 8], [ 7], [ 6]])
LegV = np.array([[ 1], [ 1], [ 1], [ 1], [ 1], [ 1], [ 1], [ 1], [ 1], [ 1]])
FitnV = powing(ObjV, LegV, 2, 2) #幂尺度变换适应度计算
```

得到 FitnV :

$$\text{FitnV} = \begin{pmatrix} 2 \\ 1.5625 \\ 1.5625 \\ 1.0625 \\ 1 \\ 1 \\ 1.0625 \\ 1.25 \\ 1.5625 \\ 2 \end{pmatrix}$$

ranking 参考资料

概要: 根据目标函数值排序的适应度分配

描述:

该函数实现了“基于等级划分的适应度分配”算法，返回包含种群个体适应度的列向量。

遵循“目标函数值越大适应度越小”的约定。

语法:

```
FitnV = ranking(ObjV)
FitnV = ranking(ObjV, LegV)
FitnV = ranking(ObjV, LegV, RFun)
FitnV = ranking(ObjV, LegV, RFun, SUBPOP)
```

详细说明:

ObjV 是一个保存着个体对应的目标函数值的列向量。

LegV 是一个可选参数，保存着个体对应的可行性的列向量，0 表示该个体是非可行解，1 表示是可行解。

RFun 是一个可选参数：

1. 如果 RFun 是一个在 [1, 2] 范围内的 1*1 的矩阵，则函数采用线性排序，此时 RFun 代表选择压力；

2. 如果 RFun 是一个 1*2 的 array，则：

RFun[0]: SP 是一标量，指定了选择压力；

RFun[1]: RM 指定排序方式：

当 RM = 0 为线性排序，此时 SP 要在 [1,2] 范围内；

当 RM = 1 为非线性排序，此时 SP 要在 [1,len(ObjV)-2] 范围内。

3. 如果 RFun 是一个长度等于 ObjV 长度的行矩阵，则表示对 ObjV 每一行的适应度值计算，此时 RFun 通常是一个元素递增的行向量；

4. 如果 RFun 是缺省或者设为 None，则默认采用线性排序且选择压力为 2。

SUBPOP 表示子种群的数量，是一个可选参数。SUBPOP 要求能够被种群个体数整除。当该参数缺省时，子种群的数量默认值为 1。关于 Geatpy 子种群的有关概念详见 migrate 函数的参考资料。

算法说明:

该函数实现了基于等级划分的适应度分配算法（算法描述详见“进化算法介绍”中的“适应度计算”章节）。

值得注意的是，由于矩阵 FitnV 未被排序，所以可以反映初始输入矩阵 ObjV 的顺序。

另外，如果种群某些个体的目标函数值为 nan 或 None（即不合法），而可行性列向量又没有作标记，此时函数将对这些额外的非可行解做出标记，更新 LegV。

特别注意:

本函数是根据传入参数 ObjV 来计算适应度的，且遵循“种群目标函数值越大，适应度越小”的原则，因此在调用本函数前，需要对传入的 ObjV 乘上'maxormin'(最大最小化标记)。但是，由于返回的是 FitnV，它与 ObjV 在含义上无关了，因此不需要对其乘上'maxormin' 进行还原。

应用实例:

考虑有 10 个个体的种群，其当前目标值 ObjV 以及可行性列向量 LegV 如下情况。

```
ObjV=np.array([[ 1],[ 2],[ 3],[ 4],[ 5],[10],[ 9],[ 8],[ 7],[ 6]])
LegV=np.array([[ 1],[ 1],[ 1],[ 1],[ 1],[ 1],[ 1],[ 1],[ 1],[ 1]])
```

(1) 使用线性排序和选择压力为 2，求适应度：

```
RFun = np.array([[2,0]])
FitnV = ranking(ObjV, LegV, RFun)
```

得到 FitnV：

$$FitnV = \begin{pmatrix} 2 \\ 1.77777778 \\ 1.55555556 \\ 1.33333333 \\ 1.11111111 \\ 0 \\ 0.22222222 \\ 0.44444444 \\ 0.66666667 \\ 0.88888889 \end{pmatrix}$$

(2) 采用非线性排序和选择压力为 2，求适应度：

```
RFun = np.array([[2,1]])
FitnV = ranking(ObjV, LegV, RFun, 1)
```

得到 FitnV：

$$FitnV = \begin{pmatrix} 2 \\ 1.66331464 \\ 1.38330779 \\ 1.15043805 \\ 0.95677023 \\ 0.38065341 \\ 0.45770463 \\ 0.55035244 \\ 0.66175385 \end{pmatrix}$$

参考文献:

[1] D. Whitley, “The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best”, Proc. ICGA 3, pp. 116-121, Morgan Kaufmann Publishers, 1989.

recdis 参考资料

摘要: 离散重组 (低级重组函数)

描述:

该函数完成对当前种群的一对个体的离散重组，并返回新的种群。

语法:

```
NewChrom = recdis(OldChrom)
NewChrom = recdis(OldChrom, XOVR)
```

详细说明:

recdis 在当前种群 OldChrom 中将各对个体进行离散重组，在种群重组后返回新种群 NewChrom。交配时按序配对，即其奇数行与下一个偶数行进行配对。如果矩阵 OldChrom 总行数为奇数行，则最后一个奇数行不进行交配并加入 NewChrom 的最后一行。OldChrom 和 NewChrom 中的每一行均对应一个个体的染色体。

OldChrom 为表示种群的矩阵，其每一行对应一个个体的一条染色体。其元素可以是任何值，包括实数值、二进制值等等。

XOVR 在本函数中为无用参数 (为了兼容同类的其他低级重组函数)。

NewChrom 为重组后的种群矩阵。

应用实例:

考虑有 5 个个体的种群进行离散重组。

```
OldChrom = np.array([
    [1, 0, 1, 1, 1, 0],
    [0, 1, 0, 1, 1, 1],
    [0, 0, 1, 1, 0, 1],
    [0, 0, 1, 1, 1, 1],
    [1, 1, 0, 0, 1, 0]])
NewChrom = recdis(OldChrom, 1)
```

得到 NewChrom:

$$\text{NewChrom} = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

参考文献:

[1] H. Mühlenbein and D. Schlierkamp-Voosen, “Predictive Models for the Breeder Genetic Algorithm: I. Continuous Parameter Optimization”, Evolutionary Computation, Vol. 1,

recint 参考资料

概要: 中间重组 (低级重组函数)

描述:

该函数完成对当前种群的一对个体的中间重组，并返回新的种群。交配的一对是有序的，奇数行和它的下一个偶数行配对。

语法:

```
NewChrom = recint(OldChrom)  
NewChrom = recint(OldChrom, XOVR)
```

详细说明:

OldChrom 为一个代表种群的矩阵，每一行对应一个个体的一条染色体。其元素可以是任何值，包括实数值、二进制值等等。

XOVR 在本函数中为无用参数 (为了兼容同类的其他函数)。

NewChrom 为重组后的种群矩阵。

算法说明: 中间重组的详细算法说明详见“进化算法介绍”中的“适应度计算”章节。

应用实例:

考虑有 5 个个体的种群进行中间重组。

```
OldChrom = np.array([  
    [1, 0, 1, 1, 1, 0],  
    [0, 1, 0, 1, 1, 1],  
    [0, 0, 1, 1, 0, 1],  
    [0, 0, 1, 1, 1, 1],  
    [1, 1, 0, 0, 1, 0]])  
  
NewChrom = recint(OldChrom, 1)
```

得到 NewChrom:

$$\text{NewChrom} = \begin{pmatrix} 0.11816188 & -0.04513522 & 1.04318166 & 1 & 1 & 1 & 0.38659477 \\ 1.17989411 & 0.56798459 & 0.05292328 & 1 & 1 & 1 & 0.67095383 \\ 0 & 0 & 1 & 1 & 1 & 0.6941655 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0.26856461 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

参考文献:

[1] H. Mühlenbein and D. Schlierkamp-Voosen, “Predictive Models for the Breeder Genetic Algorithm: I. Continuous Parameter Optimization”, Evolutionary Computation, Vol. 1, No. 1, pp.25-49, 1993.

reclin 参考资料

概要: 线性重组 (低级重组函数)

描述:

该函数完成对当前种群的一对个体的线性重组，并返回新的种群。交配的一对是有序的，奇数行和它的下一个偶数行配对。

语法:

```
NewChrom = reclin(OldChrom)  
NewChrom = reclin(OldChrom, XOVR)
```

详细说明:

OldChrom 为一个代表种群的矩阵，每一行对应一个个体的一条染色体。其元素可以是任何值，包括实数值、二进制值等等。

XOVR 在本函数中为无用参数 (为了兼容同类的其他函数)。

NewChrom 为重组后的种群矩阵。

应用实例:

考虑有 5 个个体的种群进行线性重组。

```
OldChrom = np.array([  
    [1, 0, 1, 1, 1, 0],  
    [0, 1, 0, 1, 1, 1],  
    [0, 0, 1, 1, 0, 1],  
    [0, 0, 1, 1, 1, 1],  
    [1, 1, 0, 0, 1, 0]])  
  
NewChrom = reclin(OldChrom) #进行线性重组
```

得到 NewChrom:

$$\text{NewChrom} = \begin{pmatrix} 0.99726619 & 0.00273381 & 0.99726619 & 1 & 1 & 0.00273381 & 1 \\ -0.21956057 & 1.21956057 & -0.21956057 & 1 & 1 & 1.21956057 & \\ 0 & 0 & 1 & 1 & 1 & 0.9782408 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0.04145496 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

参考文献:

[1] H. Mühlenbein and D. Schlierkamp-Voosen, “Predictive Models for the Breeder Genetic Algorithm: I. Continuous Parameter Optimization”, Evolutionary Computation, Vol. 1, No. 1, pp.25-49, 1993.

recombin 参考资料

概要: 实现个体之间染色体的重组 (重组函数)

描述:

该函数在种群中实现染色体重组并返回重组后的种群。该函数可以处理多个种群，并调用低级的重组函数对各个子种群进行重组操作。

语法:

```
NewChrom = recombin(REC_F, Chrom)
NewChrom = recombin(REC_F, Chrom, RecOpt)
NewChrom = recombin(REC_F, Chrom, RecOpt, SUBPOP)
```

详细说明:

recombin 完成种群 Chrom 中个体的重组，返回重组后的新种群 NewChrom。Chrom 和 NewChrom 中的每行对应一个个体。

REC_F 为包含低级重组函数名的字符串，比如'xovdp'。

Chrom 为用于重组的旧种群，每行对应着一个个体。

RecOpt(可选参数) 指明了种群发生重组的概率，若为缺省值或 None，则默认概率为 0.7。

SUBPOP(可选参数) 为子种群的数量，若为缺省值或 None，则默认为 1。

应用实例:

考虑由 2 个子种群组成的种群，一共有 4 个个体：

```
Chrom = np.array([
    [0,0,0,0,0,1,1,1],
    [1,0,0,0,1,0,0,1],
    [0,0,1,0,1,0,0,0],
    [1,1,0,1,1,0,1,1]]).astype('float')
```

实现个体之间染色体的单点交叉：

```
NewChrom = recombin('xovsp', Chrom, 0.7, 2)
```

得到 NewChrom：

$$\text{NewChrom} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

redisNDSet 参考资料

概要: 基于拥挤距离的帕累托最优子集筛选 (re-choose base on crowding-distance)。

描述:

该函数实现了基于拥挤距离计算的帕累托最优子集的筛选，筛选出符合数量的分布性较好的帕累托最优解子集。

语法:

```
[NDSetSub, NDSetObjVSub] = redisNDSet(NDSet, NDSetObjV)  
[NDSetSub, NDSetObjVSub] = redisNDSet(NDSet, NDSetObjV, NUM)
```

详细说明:

NDSet 是筛选前的帕累托最优解集，它是 numpy array 类型的矩阵，每一行代表一个帕累托最优解的控制变量的值。有多少个控制变量，NDSet 就有多少列。

NDSetObjV 是筛选前的帕累托最优集的目标函数值矩阵，它也是 numpy array 类型的，每一行代表各个帕累托最优解的目标函数值。有多少个目标，NDSetObjV 就有多少列。

NUM 是可选参数，表示需要筛选出的子集包含多少个帕累托最优解。当缺省或数值不大于当前帕累托最优解集 NDSet 的行数 (即帕累托最优解的个数) 时，不进行筛选，直接返回传入的 NDSet 和 NDSetObjv。

NDSetSub 和 NDSetObjVSub 分别是筛选后的 NDSet 和 NDSetObjV。

调用 redisNDSet 函数是实现 NSGA2 算法必不可少的一步。当然，在其他多目标优化算法中，也可以调用该函数，以增强帕累托最优解的分布性。

特别注意:

本函数是根据传入参数 NDSetObjV 来计算拥挤距离的，且遵循“最小化目标”的约定，但是，在对 NDSetObjV 进行排序时，无论是从小到大排序还是从大到小排序，对拥挤距离的计算是无影响的，因此，不需要对传入的 NDSetObjV 乘上'maxormin'(最大最小化标记)。也不需要对返回的 NDSetObjVSub 其乘上'maxormin' 进行还原。

参考文献:

[1] ARAVIND SESHADRI. A FAST ELITIST MULTIOBJECTIVE GENETIC ALGORITHM: NSGA-II.

reins 参考资料

概要：在种群中重插入育种个体

描述：

该函数将育种个体重插入到父代种群中，生成新一代种群。

语法：

```
Chrom = reins(Chrom, SelCh)
Chrom = reins(Chrom, SelCh, SUBPOP)
Chrom = reins(Chrom, SelCh, SUBPOP, Select)
Chrom = reins(Chrom, SelCh, SUBPOP, Select, INSR)
Chrom = reins(Chrom, SelCh, SUBPOP, Select, INSR, FitnVCh)
Chrom = reins(Chrom, SelCh, SUBPOP, Select, INSR, FitnVCh, FitnVSel)
[Chrom, ObjV] = reins(Chrom, SelCh, SUBPOP, Select, INSR, FitnVCh, FitnVSel, ObjVCh,
ObjVSel)
[Chrom, ObjV, LegV] = reins(Chrom, SelCh, SUBPOP, Select, INSR, FitnVCh, FitnVSel,
ObjVCh, ObjVSel, LegVCh, LegVSel)
```

详细说明：

reins 将育种个体插入到当前种群中，用代替父代某些个体并返回重插入后的新一代种群。

Chrom 为父代种群矩阵，其每行代表一个个体的染色体。

SelCh 为选择、交叉、变异等操作后得到的育种种群矩阵，术语上称作“育种种群”，其每行对应一个育种个体。

SUBPOP (可选参数) 表示子种群的数量，若缺省或设为 None，则默认是 1。

Select (可选参数) 指明育种个体替代父代个体的选择方法：

0 为均匀选择；

1 为基于适应度的选择；

如果 Select 缺省或为 None，则默认为 0。

在基于适应度的选择中，适应度强的个体被用于替换父代的适应度差的个体。

INSR (可选参数) 表示选择重插入的育种个体数占全部育种个体数的比率 (即选择了百分之多少的育种个体)。如果缺省或设为 None，则默认为 1.0。

FitnVCh 是一个保存着父代种群的个体对应的适应度值的列向量。

对基于适应度的重插入 (即当 Select 为 1 时)，FitnVCh 发挥作用。

FitnVSel 为一个保存着育种种群的个体对应的适应度值的列向量。

如果所有育种个体的数量大于重插入到种群中的育种个体数量，则 FitnVSel 发挥作用。此时将按育种个体的适应度从大到小的顺序选择插入。

ObjVCh 是一个保存着父代种群的个体对应目标函数值的矩阵 (可以是多目标)。

ObjVSel 为一个保存着育种种群的个体对应的目标函数值的矩阵 (可以是多目标)。

LegVCh 是一个保存着父代种群的个体对应的可行性的列向量，0 表示该个体是非可行解，1 表示是可行解。

LegVSel 是一个保存着育种种群的个体对应的可行性的列向量，0 表示该个体是非可行解，1 表示是可行解。

注意：当给 reins 函数传入 ObjVCh 参数时，也要传入 ObjVSel，即不能缺省。此时函数将不仅返回重插入后的种群矩阵，还会返回重插入后种群的目标函数值矩阵。

同理，当给 reins 函数传入 LegVCh 参数时，也要传入 LegVSel，即不能缺省。此时函数将不仅返回重插入后的种群矩阵，还会返回重插入后种群的目标函数值矩阵。

一旦传入 LegVCh 和 LegVSel，要求也要传入合法的 ObjVCh 和 ObjVSel。

此外，可以将目标函数值当作适应度传入本函数中，但传入前要乘上 maxormin，(maxormin 为最大最小化标记，它为 1 时表示这是个最小化目标，为 -1 时表示这是个最大化目标)。

特别注意：

本函数是根据 FitnVCh 和 FitnVSel 来进行重插入的，与 ObjVCh 和 ObjVSel 无关，因此在调用本函数前，不需要对传入的 ObjVCh 和 ObjVSel 乘上‘maxormin’(最大最小化标记)，对于返回的 ObjV，也不需要乘上‘maxormin’进行还原。

应用实例：

现有四个变量，范围分别是 [-10,10]、[-5,5]、[-3,3]、[-1,1]。创建一个含有这 4 个变量的 6 个个体的实数值种群 Chrom，同时再创建一个含有 2 个个体的实数值种群 SelCh 来重插入到 Chrom 中。

```
FieldDR = np.array([[-10, -5, -3, -1], [10, 5, 3, 1]]) # 创建区域描述器
Chrom = crtrp(6, FieldDR) # 创建含有6个个体的种群，把它看作父代种群
# 创建列向量来存储父代种群个体的目标函数值
FitnVCh = np.array([[21, 22, 23, 16, 15, 24]]).T
SelCh=crtrp(2, FieldDR) #
# 创建含有2个个体的种群，看成是待重插入的育种种群
# 把育种个体重插入到父代种群中
Chrom = reins(Chrom, SelCh, 1, 1, 1, FitnVCh)
```

插入前父代种群如下：

Chrom =
$$\begin{pmatrix} 1.46122027e-01 & 3.45234379e+00 & 2.31583857e+00 & 6.91123313e-01 \\ 5.67085869e+00 & 3.52743074e-01 & 1.57330911e+00 & 2.97135778e-03 \\ 5.19726307e+00 & 2.67165148e+00 & 7.85408841e-01 & 1.89785150e-01 \\ 2.25048911e+00 & 4.45678441e+00 & 1.92489047e+00 & 6.05092404e-01 \\ 1.49160644e+00 & 3.71236655e+00 & 2.69765077e+00 & 3.12855563e-01 \\ 5.00117627e+00 & 4.05129548e+00 & 2.78183093e+00 & 1.33010496e-01 \end{pmatrix}$$

待插入的育种种群如下：

Selch =
$$\begin{pmatrix} 1.08190019 & 3.99550597 & 0.72815683 & 0.31596068 \\ 3.49844636 & 0.00448962 & 0.63786374 & 0.94370521 \end{pmatrix}$$

重插入得到的新一代种群如下：

Chrom =
$$\begin{pmatrix} 1.46122027e-01 & 3.45234379e+00 & 2.31583857e+00 & 6.91123313e-01 \\ 5.67085869e+00 & 3.52743074e-01 & 1.57330911e+00 & 2.97135778e-03 \\ 5.19726307e+00 & 2.67165148e+00 & 7.85408841e-01 & 1.89785150e-01 \\ 3.49844636e+00 & 4.48961622e-03 & 6.37863745e-01 & 9.43705208e-01 \\ 1.08190019e+00 & 3.99550597e+00 & 7.28156829e-01 & 3.15960684e-01 \\ 5.00117627e+00 & 4.05129548e+00 & 2.78183093e+00 & 1.33010496e-01 \end{pmatrix}$$

对比重插入前后的 Chrom 矩阵，可以看出重插入前目标函数值最大 (遵循“目标函数值越大适应度越小”的约定) 的两个个体在重插入过程中被育种个体替换了。

rwGA 参考资料

摘要: 随机权重法多目标聚合函数。

描述:

该函数实现了随机权重法 (weighted-sum approach)[98IM]，是给多目标函数加以各种随机的权重并合成单目标函数而求得帕累托最优解的方法，该函数最终返回合成的单目标函数值的以及各目标的权重。

语法:

```
[CombinObjV, weight] = rwGA(ObjV)
[CombinObjV, weight] = rwGA(ObjV, LegV)
```

详细说明:

ObjV 是一个保存着种群个体对应的多目标函数值的矩阵，每一列对应一个个体的目标函数值。

LegV 为可选参数，是一个保存着种群个体对应的可行性的列向量，0 表示该个体是非可行解，1 表示是可行解。

CombinObjV 是一个保存着将多目标加权合成为单目标后的目标函数列向量。

weight 一个保存着各目标函数值的 array 类型行向量。

在计算多目标权重前，该函数会根据 LegV 把非可行解排除在外，以避免非可行解对理想点选取的影响。计算权重后，所有个体的多目标函数值一并乘上权重，得到加权的聚合单目标函数值。

此外，该函数遵循“最小化目标”的约定，因此传入 ObjV 前要乘上最大最小化标记 maxormin，函数返回 CombinObjV 后，需要再乘上 maxormin 以复原目标函数值。

特别注意:

本函数是根据传入参数 ObjV 来计算多目标聚合权重的，遵循“最小化模板”约定，因此在调用本函数前，需要对传入的 ObjV 乘上'maxormin'(最大最小化标记)，同时，对于返回的 CombinObjV，也需要乘上'maxormin' 进行还原。

应用实例:

考虑一个两个目标的优化问题，设种群规模为 4，这 4 个个体的目标函数值如下：

(1,2),(2,3),(2,3),(3,3)

使用随机权重聚合法 rwGA 使每个个体的两个目标函数值合成为 1 个目标函数值：

```
ObjV = np.array([[1,2],[2,3],[2,3],[3,3]])
[CombinObjV, weight] = rwGA(ObjV)
```

结果如下：

$$\text{CombinObjV} = \begin{pmatrix} 1.04586331 \\ 2.04586331 \\ 2.04586331 \\ 3.0 \end{pmatrix}$$

$$\text{weight} = \begin{pmatrix} 0.95413669 & 0.04586331 \end{pmatrix}$$

rws 参考资料

概要: 轮盘赌选择 (低级选择函数)

描述:

rws 函数在当前种群中根据的适应度 FitnV 选择出数量为 Nsel 的个体。

rws 函数是一低级函数，通常被高级选择函数 selecting 调用。

语法:

```
NewChrIx = rws(FitnV, Nsel)
```

详细说明:

rws 函数返回一个行向量 NewChrIx，代表与所选的个体在种群中的索引。可以通过 OldChrom[NewChrIx, :] 来获得这些选择出来的个体。

FitnV 是一包含种群中每个个体的适应度值的列向量，它可以通过使用函数 ranking 或 scaling 等计算得到。

Nsel 为被选择个体的数目，可以比父代多。

rws 函数是一低级函数，通常被 selectting 调用。

算法说明:

轮盘赌选择的算法详见“进化算法介绍”的“选择”章节。

应用实例:

考虑有 8 个个体的种群并假设有下面的适应度 FitnV。

```
FitnV = np.array([[1.5, 1.35, .21, 1.07, 0.92, 0.78, 0.64, 0.5]]).T
```

选择 6 个个体，得到它们在种群中的索引：

```
NewChrIx = rws(FitnV, 6)      #进行轮盘赌选择
```

得到 NewChrIx：

$$\text{NewChrIx} = \begin{pmatrix} 1 & 3 & 1 & 5 & 3 & 0 \end{pmatrix}$$

参考文献:

[1] J. E. Baker, “Reducing bias and inefficiency in the selection algorithm”, Proc ICGA 2, pp. 14-21, Lawrence Erlbaum Associates, Publishers, 1987.

[2] David E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison Wesley, 1989.

scaling 参考资料

概要：线性尺度变换适应度计算

描述：

该函数对目标函数值 ObjV 进行线性变换，遵循“最小适应度为 0”的约定（特殊情况除外）。

语法：

```
FitnV = scaling(ObjV)
FitnV = scaling(ObjV, LegV)
FitnV = scaling(ObjV, LegV, Smul)
FitnV = scaling(ObjV, LegV, Smul, SUBPOP)
```

详细说明：

ObjV 为一个保存着个体对应的目标函数值的列向量。

LegV 是一个保存着个体对应的可行性的列向量，0 表示该个体是非可行解，1 表示是可行解。

Smul(可选参数) 线性变换的决定上界的常数。若缺省或为 None，则默认为 2。

SUBPOP(可选参数) 为子种群的数量，要求能够被种群个体数整除。若缺省或为 None，默认值为 1。

FitnV 为记录着种群个体适应度值的列向量。

算法：

该函数计算出由 Smul 的值决定上界的适应度值即 $F' = aF + b$ 。

其中 F 为直接把目标函数值 ObjV 作为适应度值的函数： $F(\text{个体}) = \text{ObjV}$

F' 为对 F 进行线性尺度变换后的适应度值函数，且 F' 要满足以下两个条件：

1. 为了保证适应度在平均值的个体在下一代的期望复制数为 1，因此要使得原适应度的平均值 Favg = 新适应度的平均值 $F'\text{avg}$ 。

2. 为了控制适应度最大的个体在下一代中的复制数，因此设定变换后的适应度最大值等于原适应度平均值的。指定倍数，即： $F'\text{max} = \text{Smul} * \text{Favg}$

经过线性变换后，原适应度高的少数个体的适应度等比例缩小，同时适应度较差的个体的适应度等比例扩大，从而有利于种群的多样性。

注意：为遵循“目标函数值越大适应度越小”的约定，算法中先对 ObjV 取相反数。

根据上述条件进行线性变换后，原适应度最小的个体可能因为这种变换而导致适应度小于 0，此时我们要特殊处理这种情况，采用另一种变换方式：

1. 变换后适应度再加 1 使最小适应度为 1，即： $F'\text{min} = 1$

2. 不再要求 $F'\text{avg} = \text{Favg}$ ，而是设置斜率 $dF'/dF = \text{Favg} / (\text{Favg} - \text{Fmin})$

算法处理了以下 3 种特殊情况：

1. 当种群的所有个体的目标函数值都相等时，此时算法中作为分母的 delta 会为 0，因此假如遇到这种情况，则将 FitnV 设置为全 1 的列向量

2. 有可能上述情况下 delta 的值相当的小，但不为 0，此时也要将 FitnV 设置为 1 的列向量。

3. 对于传入的 ObjV 均小于 0 的情况，要先进行平移变换，使所有的 ObjV 都 ≥ 0 。

4. 如果种群某些个体的目标函数值为 nan 或 None (即不合法)，而可行性列向量又没有作标记，此时函数将对这些额外的非可行解做出标记，更新 LegV。

特别注意：

本函数是根据传入参数 ObjV 来计算适应度的，且遵循“种群目标函数值越大，适应度越小”的原则，因此在调用本函数前，需要对传入的 ObjV 乘上'maxormin'(最大最小化标记)。但是，由于返回的是 FitnV，它与 ObjV 在含义上无关了，因此不需要对其乘上'maxormin' 进行还原。

应用实例：

根据目标函数值 ObjV，利用线性尺度变换求其对应的适应度：

```
ObjV = np.array([[1], [2], [3], [4], [5], [10], [9], [8], [7], [6]])
LegV = np.array([[1], [1], [1], [1], [1], [1], [1], [1], [1], [1]])
FitnV = scaling(ObjV, LegV) # 进行线性尺度变换适应度计算
```

得到适应度如下：

$$\text{FitnV} = \begin{pmatrix} 10 \\ 9 \\ 8 \\ 7 \\ 6 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix}$$

参考文献：

[1] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning,

Addison Wesley Publishing Company, January 1989.

selecting 参考资料

概要: 高级选择函数

描述:

此函数执行通用选择，处理多个种群，选择过程实际上调用了低级选择函数。

语法:

```
SelCh = selecting(SEL_F, Chrom, FitnV)
SelCh = selecting(SEL_F, Chrom, FitnV, GGAP)
SelCh = selecting(SEL_F, Chrom, FitnV, GGAP, SUBPOP)
[SelCh, ObjVSel] = selecting(SEL_F, Chrom, FitnV, GGAP, SUBPOP, ObjV)
[SelCh, ObjVSel, LegVSel] = selecting(SEL_F, Chrom, FitnV, GGAP, SUBPOP, ObjV,
LegV)
```

详细说明:

函数 select 从种群 Chrom 中选择优良个体，并将选择的个体返回到新种群 SelCh 中。

SEL_F 为包含低级选择函数名称的字符串如 ‘rws’。

Chrom 为包含当前种群的染色体矩阵，每一行对应一个个体的一条染色体。

FitnV 为包含种群 Chrom 中个体的适应度值的列向量。

GGAP (可选参数) 表示代沟，表示被选择的种群的比率，默认值为 1.0。允许代沟大于 1，表示子代种群个体数大于父代种群个体数。

SUBPOP (可选参数) 表示子种群的数量。如果 SUBPOP 缺省或设为 None 时，默认的 SUSPOP=1。Chrom 中的所有子种群个体数量是相等的。

ObjV 是一个保存着父代种群的个体对应目标函数值的矩阵 (可以是多目标)。

LegV 是一个保存着父代种群的个体对应的可行性的列向量，0 表示该个体是非可行解，1 表示是可行解。

SelCh, ObjVSel, LegVSel 的含义分别和 Chrom, ObjV, LegV 相同，前者是后者经过选择后得出的。

算法说明:

selecting 检测输入参数的一致性并调用低级选择函数，遍历所有子种群进行选择操作。

特别注意:

本函数是根据 FitnV 来进行选择的，与 ObjV 无关，因此在调用本函数前，不需要对传入的 ObjV 乘上‘maxormin’(最大最小化标记)，对于返回的 ObjVSel，也不需要乘上‘maxormin’进行还原。

应用实例:

考虑以下具有 8 个个体的种群 Chrom，其适应度为 FitnV。

```
Chrom=np.array([[1,11,21],
                [2,12,22],
                [3,13,23],
                [4,14,24],
                [5,15,25],
                [6,16,26],
                [7,17,27],
                [8,18,28]])
```

```
FitnV = np.array( [[1.50,1.35,1.21,1.07,0.92,0.78,0.64,0.5]] ).T
SelCh = selecting('sus',Chrom,FitnV) # 使用随机遍历抽样sus选择个体
```

得到 SelCh:

$$SelCh = \begin{pmatrix} 1 & 11 & 21 \\ 4 & 14 & 24 \\ 3 & 13 & 23 \\ 1 & 11 & 21 \\ 2 & 12 & 22 \\ 6 & 16 & 26 \\ 4 & 14 & 24 \\ 7 & 17 & 27 \end{pmatrix}$$

sgaplot 参考资料

概要: 单目标进化动态绘图函数。

描述:

该函数根据传入的数据集和进化代数 gen 绘制动态图或进化结束后的静态图。

语法:

```
newAx = sgaplot(ValueSet, Label, saveFlag)
newAx = sgaplot(ValueSet, Label, saveFlag, ax)
newAx = sgaplot(ValueSet, Label, saveFlag, ax, gen)
newAx = sgaplot(ValueSet, Label, saveFlag, ax, gen, interval)
newAx = sgaplot(ValueSet, Label, saveFlag, ax, gen, interval, title)
newAx = sgaplot(ValueSet, Label, saveFlag, ax, gen, interval, title, save_path)
```

详细说明:

ValueSet 是一个 numpy 的 array 类型的列向量，一般传入该函数前是进化记录器 pop_trace 的某一列数据。其实际含义由 Label 确定。

Label 是一个字符串，代表数据集 ValueSet 的含义

saveFlag 是布尔类型的标记，表示是否要保存图片。当要绘制动画时，必须设为 False。

ax 是可选参数，在绘制动画的时候需要传入。其代表上一帧的动画。当画第一帧时，其值为 None。

gen 是可选参数，表示当前进化代数，默认为 None。该参数没有缺省或为非 None 时，图片将绘制动态图。

interval 是可选参数，表示两帧动画之间的间隔时间，默认为 0.1，单位为‘秒’。

title 是可选参数，表示图形的标题名称。

save_path 是 string 类型的可选参数，表示保存图片的路径。

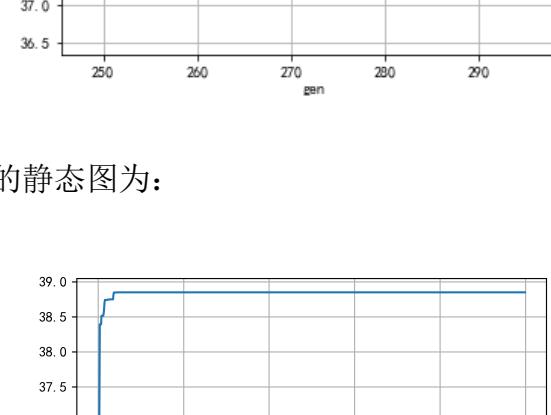
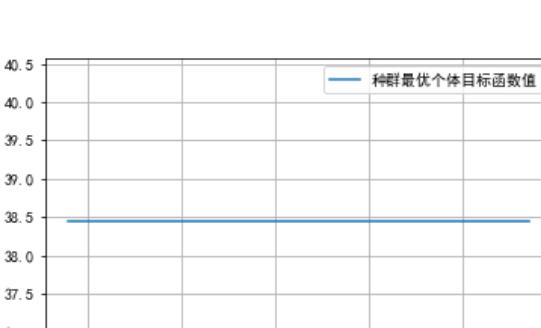
newAx 代表新的图形，是更新后的 ax。

应用实例:

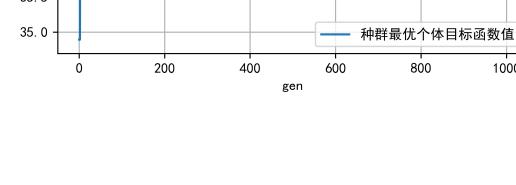
在解决一个单目标优化问题时绘制进化过程中的各代种群最优个体的目标函数值的变化动态图。并且在进化结束后绘制一个进化全过程的各代最优个体的目标函数值静态图。

```
...
ax = None # 存储上一帧图片
# 开始进化
for gen in range(MAXGEN)
    ...
    # 进化操作
    ...
    # 记录最优个体
    bestIdx = np.max(FitnV)
    pop_trace[gen,1] = ObjV[bestIdx] # 记录当代目标函数的最优值
    ...
    # 绘图
    ax = ga.sgaplot(pop_trace[:,[1]], '种群最优个体目标函数值',
                     False, ax, gen)
# 进化结束
ga.sgaplot(pop_trace[:,[1]], '种群最优个体目标函数值',
            True) # 绘制最终的帕累托前沿图
# end
```

运行结果动画部分截图如下：



进化结束后绘制的静态图为：



sus 参考资料

摘要: 随机抽样选择(低级选择函数)

描述:

该函数根据种群中个体的适应度 FitnV, 有概率地选择 Nsel 个个体。

语法:

NewChrIx = sus(FitnV, Nsel)

详细说明:

FitnV 为种群各个个体的适应度值矩阵列向量, 其可以通过 ranking 或 scaling 函数获得。

Nsel 为被选中个体的数目, 可以比父代多。NewChrIx 是为培养选择的个体索引值, 是按它们选择的顺序排列的。这个选择的个体可以通过评估 Chrom(NewChrIx,:) 恢复。

sus 为低级选择函数, 通常被 selectting 调用。

函数返回记录着被选个体所在种群位置的索引 NewChrIx, 它是一个行向量。

算法说明:

随机抽样选择的算法详见“进化算法介绍”的“选择”章节。

应用实例:

考虑以下具有 8 各个个体的种群 Chrom, 其适应度为 FitnV。

```
FitnV = np.array([[1.5, 1.35, 1.21, 1.07, 0.92, 0.78, 0.64, 0.5]]).T
```

```
NewChrIx = sus(FitnV, 6) #选择6个个体的索引
```

得到 NewChrIx:

$$\text{NewChrIx} = \begin{pmatrix} 5 \\ 0 \\ 1 \\ 0 \\ 3 \\ 4 \end{pmatrix}$$

参考文献:

[1] J. E. Baker, “Reducing bias and inefficiency in the selection algorithm”, Proc. ICGA 2, pp. 14-21, Lawrence Erlbaum Associates, Publishers, 1987.

tour 参考资料

摘要: 锦标赛选择 (低级选择函数)。

描述:

该函数利用锦标赛选择法对种群进行选择，并返回所选择的个体在种群中的索引值。

语法:

NewChrIx = tour(FitnV, Nsel)

详细说明:

该函数实现的是传统锦标赛选择，通过随机抽取若干个体参与竞赛 (无精英保留策略)，因此，最优个体有可能会因没被选中参加锦标赛而最后没有被保留下。

FitnV 是一个列向量，代表种群中各个个体的适应度值。

Nsel 是一个正整数，代表被选择的个体数 (可以比父代的个体数多)。

锦标赛的竞赛规模 tour(算法中的一个变量，使用函数时无需考虑) 是根据 FitnV 最大值的向上取整来确定的。

比如：

$$\text{FitnV} = \begin{pmatrix} 1.2 \\ 0.8 \\ 2.1 \\ 3.2 \\ 0.6 \end{pmatrix}$$

那么竞赛规模 tour=4。

竞赛规模 tour 的值必须在 [1, Nind] 之间 (其中 Nind 为种群的个体数)。当 tour>Nind 时，取 FitnV 平均值的向上取整，若 tour 仍大于 Nind，则默认取 tour=2。

当传入的 FitnV 是由 ranking 函数生成时，实际上 tour 等价于 ranking 函数里面的择压差 SP。

应用实例:

现有一个种群，其个体的适应度如下：

$$\text{FitnV} = \begin{pmatrix} 1.2 \\ 0.8 \\ 2.1 \\ 3.2 \\ 0.6 \\ 2.2 \\ 1.7 \\ 0.2 \end{pmatrix}$$

用锦标赛选择法从中选出 6 个个体。

```
FitnV = np.array([[1.2], [0.8], [2.1], [3.2], [0.6], [2.2], [1.7], [0.2]])
NewChrIx = tour(FitnV, 6)
```

得到所选择个体的索引值为：

$$\text{NewChrIx} = \begin{pmatrix} 5 & 3 & 4 & 6 & 3 & 0 \end{pmatrix}$$

trcplot 参考资料

摘要: 单目标进化优化绘图。

描述:

该函数用给定的单目标进化追踪器来绘制相关图形。

语法:

```
trcplot(pop_trace, labels)
trcplot(pop_trace, labels, titles)
trcplot(pop_trace, labels, titles, save_path)
```

详细说明:

`pop_trace` 是一个 numpy 的 array 类型的进化追踪器，每一列代表一个参数，如第一列代表个体最优目标函数值等。每一行对应一“代”，比如第一行对应的是第一代种群的最优个体。

`labels` 是一个 list 类型的二维列表，表示各图片中的图例，其每一列的含义与 `pop_trace` 是对应的。例如：

1. 假设 `pop_trace` 有 2 列，含义分别是'a' 和'b'，则 `labels = [['a'], ['b']]`，表示要画 2 张图，每张图画 2 个变量，图例分别是'a' 和'b'。
2. 假设 `pop_trace` 有 2 列，含义分别是'a' 和'b'，则 `labels = [['a', 'b']]`，表示要画 1 张图，图中有 2 个变量，图例分别是'a' 和'b'。
3. 假设 `pop_trace` 有 3 列，含义分别是'a', 'b' 和 'c'，则 `labels = [['a'], ['b', 'c']]`，表示要画 2 张图，第一张图有 1 个变量，图例是'a'；第二张图有 2 个变量，图例是'b' 和 'c'。

应用实例:

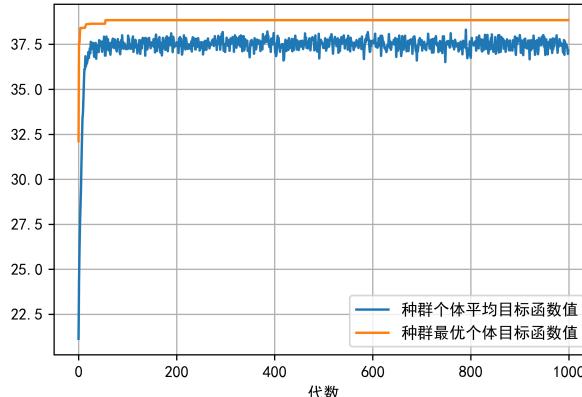
在 `sga_code_templat` 模板中，有这样的一行代码：

```
ga.trcplot(pop_trace, [['种群个体平均目标函数值',
    '种群最优个体目标函数值']])
```

其中 `pop_trace` 是一个 n 行 2 列的矩阵，第一列代表种群个体平均目标函数值，第二列代表种群最优个体目标函数值。因此传入 `trcplot` 绘图函数的参数 `labels` 设为 `[['种群个体平均目标函数值', '种群最优个体目标函数值']]`，表示要画 1 张图，这张图中同时绘制“种群个体平均目标函数值”以及“种群最优个体目标函数值”。

`titles` 参数是缺省的，因此绘图将不显示标题。

绘图结果如下：



upNDSet 参考资料

摘要: 更新帕累托最优集 (Update NDSet)。

描述:

该函数根据传入的个体多目标函数值更新当前的帕累托最优集 (一般是全局的), 并根据个体是否被当前帕累托最优集的解所支配来修改个体的适应度。最后返回修改后的种群个体的适应度以及更新后的帕累托最优解集及其对应的目标函数值矩阵。若传入 exIdx 参数, 则会对非可行解的个体进行排除。

语法:

```
[newFitnV, newNDSet, newNDsetObjV, repnum] = upNDSet(Phen, ObjV, FitnV, NDSet,  
NDSetObjV)
```

```
[newFitnV, newNDSet, newNDsetObjV, repnum] = upNDSet(Phen, ObjV, FitnV, NDSet,  
NDSetObjV, frontIdx)
```

```
[newFitnV, newNDSet, newNDsetObjV, repnum] = upNDSet(Phen, ObjV, FitnV, NDSet,  
NDSetObjV, frontIdx, LegV)
```

详细说明:

Phen 是种群表现型矩阵, 每一行代表一个个体的控制变量值。

ObjV 是种群个体的目标函数矩阵, 它是 array 类型的, 每一行对应一个个体, 每一列对应一个目标。

FitnV 是 array 类型的列向量, 代表种群各个个体的适应度值。

NDSet 是更新前的帕累托最优解集, 它也是 array 类型的, 列的含义跟 Chrom 相同, 每一行代表各个帕累托最优解。

NDSetObjV 是更新前的帕累托最优集的目标函数值矩阵, 它也是 array 类型的, 列的含义跟 ObjV 相同, 每一行代表各个帕累托最优解。

frontIdx 是可选参数, 它是一个 array 类型的行向量, 存储种群中的非支配个体, 当缺省或为 None 时, 默认种群的全部个体就是非支配的。

LegV 是一个保存着个体对应的可行性的列向量, 0 表示该个体是非可行解, 1 表示是可行解。

newFitnV、newNDSet 和 newNDSetObjV 分别是更新后的 FitnV、NDSet 和 NDSetObjV。

repnum 代表种群中非支配个体在修改前的帕累托最优集中重复出现的次数。

upNDSet 常常在多目标优化的编程模板中使用, 目的是在找到当代种群的非支配个体后, 把这些非支配解与全局帕累托最优集作比较, 使这些非支配个体中不被全局帕累托最优集中的解所支配的个体加入到全局帕累托最优集中, 从而更新全局帕累托最优集。同时, 惩罚那些被全局帕累托最优集支配的种群非支配解 (降低其适应度)。另外, 统计这些非支配个体中有多少是在全局帕累托最优集中已存在的解, 若重复个数很多, 那么在后面就要考虑使用高斯变异来增强种群的多样性。

特别注意:

本函数是根据传入参数 ObjV 和 NDSetObjV 来进行非支配排序的, 且遵循“最小化目标”的约定, 因此在调用本函数前, 需要对传入的 ObjV 和 NDSetObjV 乘上'maxormin'(最大最小化标记) 来让其符合约定。对于返回参数, 由于返回参数均与 ObjV 在意义上不相关, 因此不需要对其乘上'maxormin' 进行还原。而对于 newNDSetObjV, 它是“更新后的帕累托最优解集的目标函数值矩阵”, 它的计算是直接与传入参数 NDSetObjV 有关的, 因此, 调用本函数后, 需要对其乘上'maxormin' 进行还原, 否则对于最大化问题, 得出的 newNDSetObjV 的正负性会相反。

xovdp 参考资料

概要: 两点交叉 (低级重组函数)

描述:

该函数调用多点交叉算子 xovmp 函数实现两点交叉，返回一个新的种群矩阵。

语法:

NewChrom = xovdp(OldChrom)

NewChrom = xovdp(OldChrom, XOVR)

详细说明:

xovdp 完成当前种群 OldChrom 中一对个体按两点交叉概率 XOVR 进行单点交叉。返回交叉后的新种群 NewChrom。

OldChrom 为表示种群的矩阵，每一行为一个个体的一条染色体。其元素可以是任何值，括实数值、二进制值等。

XOVR 为交叉概率，默认值为 0.7。

NewChrom 为单点交叉后生成的种群矩阵，其每行为一个个体的一条染色体。

应用实例:

调用 crtbp 函数生成一个二进制种群 OldChrom，完成种群 OldChrom 的两点交叉。

```
OldChrom=crtbp(5,6) #调用crtbp创建一个5行6列的二进制种群矩阵
```

$$\text{OldChrom} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

```
NewChrom = xovdp(OldChrom, 1) #交叉率为1
```

交叉结果如下：

$$\text{NewChrom} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

xovdprs 参考资料

概要: 减少代理的两点交叉

描述:

该函数调用多点交叉算子函数 xovmp 实现减少代理的两点交叉，返回一个新的种群矩阵。

语法:

```
NewChrom = xovdprs(OldChrom)  
NewChrom = xovdprs(OldChrom, XOVR)
```

详细说明:

xovdprs 在当前种群 OldChrom 一对个体间按交叉率 XOVR 进行减少代理的两点交叉并返回交配后的新种群 NewChrom。

OldChrom 为代表种群的矩阵。OldChrom 每行都表示一个个体的一条染色体。其元素可以是任何值，包括实数值、二进制值等。

XOVR 表示交叉概率，是可选参数。在缺省条件下，默认值为 0.7。

NewChrom 为减少代理的两点交叉后的种群矩阵，其每行表示一个个体的一条染色体。

算法说明:

有关减少代理的概念详见“进化算法介绍”的“重组”章节，它是通过限制交叉点的位置来完成的，使得交叉点只能发生在基因不同的地方，从而尽量地让交叉操作产生新的性状。

应用实例:

调用 xovdprs 函数生成较少代理的两点交叉后的种群矩阵 NewChrom。

```
OldChrom=crtbp(5,6) #调用crtbp创建一个5行6列的二进制种群矩阵
```

$$\text{OldChrom} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

```
NewChrom = xovdprs(OldChrom, 1) #交叉率为1
```

交叉结果如下：

交叉结果如下：

$$\text{NewChrom} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

参考文献:

[1] L. Booker, “Improving search in genetic algorithms,” In Genetic Algorithms and Simulated Annealing, L. Davis (Ed.), pp. 61-73, Morgan Kaufmann Publishers, 1987.

xovmp 参考资料

概要: 多点交叉

描述:

该函数把输入的 OldChrom 种群矩阵进行多点交叉，并返回新的种群矩阵。

语法:

```
NewChrom = xovmp(OldChrom)
NewChrom = xovmp(OldChrom, Px)
NewChrom = xovmp(OldChrom, Px, Npt)
NewChrom = xovmp(OldChrom, Px, Npt, Rs)
```

详细说明:

xovmp 在当前种群 OldChrom 成对个体间多点交叉并返回交配后的新种群 NewChrom。

OldChrom 为代表种群的矩阵。OldChrom 每行都表示一个个体的一条染色体。其元素可以是任何值，包括实数值、二进制值等。

Px 表示交叉概率，是可选参数。在缺省条件下，默认值为 0.7。

Npt 指明交叉点数，是可选参数。Npt=0 表示洗牌交叉；Npt=1 表示单点交叉；Npt=2 表示两点交叉。其默认为 Npt=0。

Rs 表示是否使用代理，是可选参数。Rs=0 表示不减少代理；Rs=1 表示减少代理。默认为 Rs=0。

NewChrom 为交叉后的种群矩阵，其每行表示一个个体的一条染色体。

单点交叉 xovsp、两点交叉 xovdp、洗牌交叉 xovsh、减少代理的单点交叉 xovsprs 等实际上是调用了多点交叉函数。

应用实例:

调用 xovmp 函数生成多点交叉后的种群矩阵 NewChrom。

```
OldChrom=crtbp(5,6) #调用crtbp创建一个5行6列的二进制种群矩阵
```

$$\text{OldChrom} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

```
NewChrom = xovmp(OldChrom, 0.9, 2, 0)
#交叉率为0.9,进行两点交叉,不使用减少代理
交叉结果如下:
```

NewChrom 结果如下：

$$\text{NewChrom} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

xovpm 参考资料

概要: 部分匹配交叉 (低级重组函数)

描述:

该函数实现了部分匹配交叉 (PMX) 算法，对排列种群进行交叉操作。

语法:

NewChrom = xovpm(OldChrom)

NewChrom = xovpm(OldChrom, XOVR)

详细说明:

所谓排列种群即种群染色体是一个无重复随机数的排列。

OldChrom 是交叉前的种群矩阵。

XOVR 是交叉概率，缺省或设为 None 时默认为 0.7。

应用实例:

使用 crtpp 创建一个有 4 个个体的排列编码种群，然后进行部分匹配交叉。

```
Nind = 4 # 染色体数
Lind = 6 # 染色体长度
VarLen = 8 # 排列集合大小
OldChrom = crtpp(Nind, Lind, VarLen) # 创建排列编码种群
NewChrom = xovpm(OldChrom, 0.9) # 设置交叉概率为0.9
```

交叉前种群矩阵如下：

$$\text{OldChrom} = \begin{pmatrix} 5 & 1 & 3 & 7 & 6 & 8 \\ 7 & 3 & 2 & 4 & 1 & 5 \\ 1 & 3 & 8 & 7 & 5 & 2 \\ 8 & 1 & 2 & 3 & 6 & 7 \end{pmatrix}$$

交叉后：

$$\text{NewChrom} = \begin{pmatrix} 8 & 3 & 2 & 4 & 1 & 5 \\ 4 & 1 & 3 & 7 & 6 & 8 \\ 3 & 1 & 2 & 7 & 5 & 8 \\ 2 & 3 & 8 & 1 & 6 & 7 \end{pmatrix}$$

xovsh 参考资料

摘要: 洗牌交叉

描述:

该函数调用多点交叉算子 xovmp 函数实现洗牌交叉，返回一个新的种群矩阵。

语法:

```
NewChrom = xovsh(OldChrom)  
NewChrom = xovsh(OldChrom, XOVR)
```

详细说明:

xovshrs 在当前种群 OldChrom 一对个体间按交叉率 XOVR 进行洗牌交叉并返回交配后的新种群 NewChrom。

OldChrom 为代表种群的矩阵。OldChrom 每行都表示一个个体的一条染色体。其元素可以是任何值，包括实数值、二进制值等。

XOVR 表示交叉概率，是可选参数。在缺省条件下，默认值为 0.7。

NewChrom 为洗牌交叉后的种群矩阵，其每行表示一个个体的一条染色体。

交配的对是有序的，即奇数行与下一偶数行进行配对。如果矩阵 OldChrom 为奇数行，则最后一行不参与交配，因此，种群将按要求组织成连续的对。这可以使用函数 ranking 计算每个染色体的适应度并用选择函数（select、sus 或 rws）用它在种群的适应度相关的概率选择个体来完成。

应用实例: 调用 xovsh 函数生成洗牌交叉后的种群矩阵 NewChrom。

```
OldChrom=crtbp(5,6) #调用crtbp创建一个5行6列的二进制种群矩阵
```

$$\text{OldChrom} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

```
NewChrom = xovsh(OldChrom, 1) #交叉率为1
```

交叉结果如下：

$$\text{NewChrom} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

参考文献:

[1] R. A. Caruana, L. A. Eshelman, J. D. Schaffer, “Representation and hidden bias II: Eliminating defining length bias in genetic search via shuffle crossover”, In Eleventh International Joint Conference on Artificial Intelligence, N. S. Sridharan (Ed.), Vol. 1, pp. 750-755, Morgan Kaufmann Publishers, 1989.

xovshrs 参考资料

概要: 减少代理的洗牌交叉

描述:

该函数调用多点交叉算子 xovmp 函数实现减少代理的洗牌交叉，返回一个新的种群矩阵。

语法:

```
NewChrom = xovshrs(OldChrom)  
NewChrom = xovshrs(OldChrom, XOVR)
```

详细说明:

xovshrs 在当前种群 OldChrom 一对个体间按交叉率 XOVR 进行减少代理的洗牌交叉并返回交配后的新种群 NewChrom。

OldChrom 为代表种群的矩阵。OldChrom 每行都表示一个个体的一条染色体。其元素可以是任何值，包括实数值、二进制值等。

NewChrom 为交叉后的种群矩阵，其每行表示一个个体的一条染色体。

算法说明:

有关减少代理的概念详见“进化算法介绍”的“重组”章节，它是通过限制交叉点的位置来完成的，使得交叉点只能发生在基因不同的地方，从而尽量地让交叉操作产生新的性状。

应用实例:

调用 xovshrs 函数生成较少代理的洗牌交叉后的种群矩阵 NewChrom。

```
OldChrom = crtbp(5,6) #调用crtbp创建一个5行6列的二进制种群矩阵
```

$$\text{OldChrom} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

```
NewChrom = xovshrs(OldChrom, 1) #交叉率为1
```

交叉结果如下：

$$\text{NewChrom} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

参考文献:

[1] L. Booker, “Improving search in genetic algorithms,” In Genetic Algorithms and Simulated Annealing, L. Davis (Ed.), pp. 61-73, Morgan Kaufmann Publishers, 1987.

xovsp 参考资料

摘要: 单点交叉

描述:

该函数调用多点交叉算子 xovmp 函数实现单点交叉，返回一个新的种群矩阵。

语法:

NewChrom = xovsp(OldChrom)

NewChrom = xovsp(OldChrom, XOVR)

详细说明:

xovsp 完成当前种群 OldChrom 中一对个体按交叉概率 XOVR 进行单点交叉。返回交叉后的新种群 NewChrom。

OldChrom 为表示种群的矩阵，每一行为一个个体的一条染色体。其元素可以是任何值，括实数值、二进制值等。

XOVR 为交叉概率，默认值为 0.7。

NewChrom 为单点交叉后生成的种群矩阵，其每行为一个个体的一条染色体。

应用实例:

调用 crtbp 函数生成一个二进制种群 OldChrom，完成种群 OldChrom 的单点交叉。

```
OldChrom=crtbp(5,6) #调用crtbp创建一个5行6列的二进制种群矩阵
```

$$\text{OldChrom} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

```
NewChrom = xovsp(OldChrom, 1) #交叉率为1
```

解码后结果如下：

$$\text{NewChrom} = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

xovsprs 参考资料

摘要: 减少代理的单点交叉。

描述:

该函数调用多点交叉算子 xovmp 函数实现减少代理的单点交叉，返回一个新的种群矩阵。

语法:

```
NewChrom = xovsprs(OldChrom)  
NewChrom = xovsprs(OldChrom, XOVR)
```

详细说明:

xovsprs 在当前种群 OldChrom 一对个体间按交叉率 XOVR 进行减少代理的两点交叉并返回交配后的新种群 NewChrom。

OldChrom 为表示种群的矩阵。OldChrom 每行都表示一个个体的一条染色体。其元素可以是任何值，包括实数值、二进制值等。

XOVR 表示交叉概率，是可选参数。在缺省条件下，默认值为 0.7。

NewChrom 为减少代理的两点交叉后的种群矩阵，其每行表示一个个体的一条染色体。

算法说明:

有关减少代理的概念详见“进化算法介绍”的“重组”章节，它是通过限制交叉点的位置来完成的，使得交叉点只能发生在基因不同的地方，从而尽量地让交叉操作产生新的性状。

应用实例:

调用 xovdprs 函数生成较少代理的两点交叉后的种群矩阵 NewChrom。

```
OldChrom=crtbp(5,6) #调用crtbp创建一个5行6列的二进制种群矩阵
```

$$\text{OldChrom} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

```
NewChrom = xovsprs(OldChrom, 1) #交叉率为1
```

交叉结果如下：

$$\text{NewChrom} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

参考文献:

[1] L. Booker, “Improving search in genetic algorithms,” In Genetic Algorithms and Simulated Annealing, L. Davis (Ed.), pp. 61-73, Morgan Kaufmann Publishers, 1987.