快速入门

1. 安装

你可以通过联网安装或本地安装来安装 Geatpy。在安装前，你需要安装 Numpy。

联网安装：pip install geatpy

本地安装：在 github 下载 Geatpy 安装包，在本地控制台执行以下代码安装：

python setup.py install

2. 入门示例

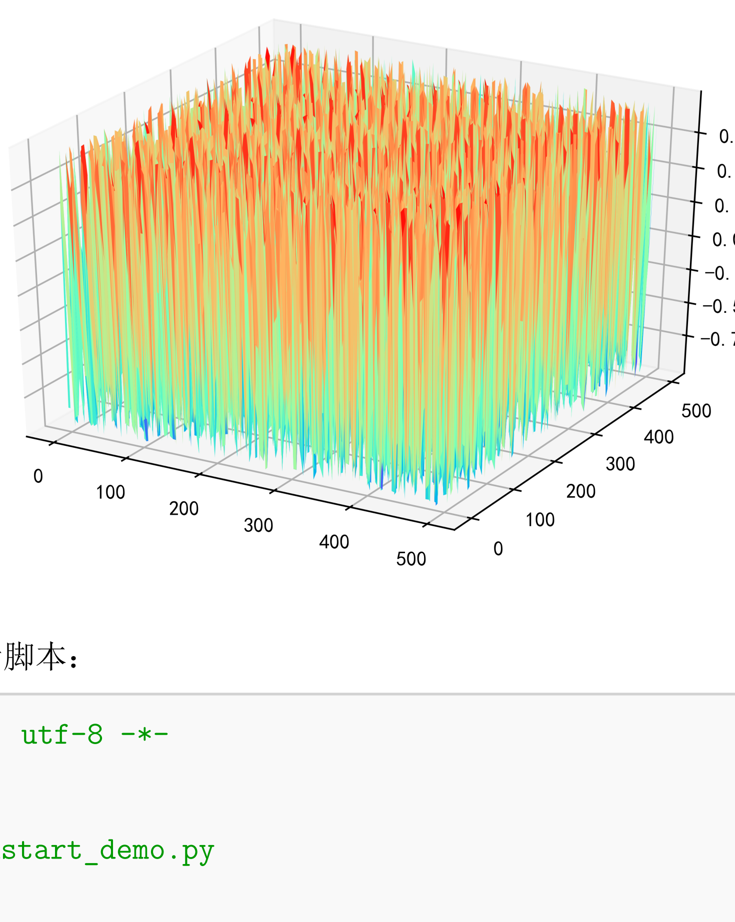
Geatpy 可以用最基础的脚本方式编写代码求解问题，wo 我们将用它来快速入门，其代码风格与 Matlab 极为类似 (有兴趣可以跟 Matlab 的 gatbx 和 GEATbx 工具箱用法进行比较)。

我们研究复杂的 MoCormick 函数的最小化搜索，它是这样的一个函数：

$$f(x, y) = sin(x + y) + (x - y)^2 - 1.5x + 2.5y + 1$$

它是一个二元多峰函数，它具有一个全局极小点：

$f = (-0.54719, -1.54719) = -1.9133$，极难进行全局优化，函数图像如下：



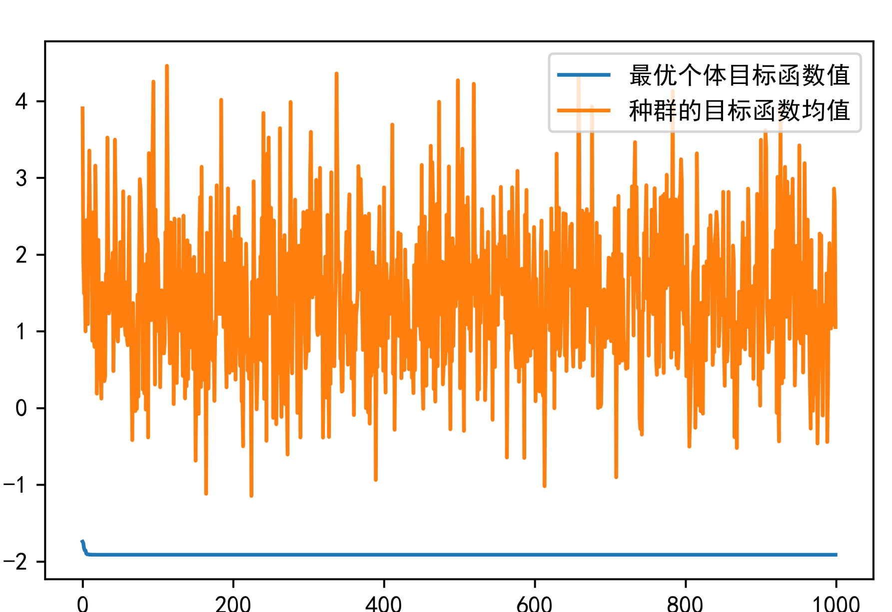
编写如下执行脚本：

```python
# -*- coding: utf-8 -*-
"""
执行脚本quickstart_demo.py
描述:
本demo展示了如何不使用Geatpy提供的进化算法框架、
纯粹地写编程脚本来解决一个无约束的单目标优化问题。
优化目标: f = min(np.sin(x + y) + (x - y) ** 2 - 1.5 * x + 2.5 * y +1)
注意目标函数aimfuc的输入输出参数的写法
"""

import numpy as np
import geatpy as ga # 导入geatpy库
import time

"""============================函数定义============================"""
# 传入种群基因表现型矩阵Phen以及种群个体的可行性列向量LegV
def aimfuc(Phen, LegV):
  x = np.array([Phen[:, 0]]).T # 取出Phen第一列得到种群所有个体的x值
  y = np.array([Phen[:, 1]]).T # 取出Phen第二列得到种群所有个体的y值
  # 计算MoCormick函数值
  f = np.sin(x + y) + (x - y) ** 2 - 1.5 * x + 2.5 * y +1
  return [f, LegV]
"""============================变量设置============================"""
x1 = [-1.5, 4]; x2 = [-3, 4]   # 自变量范围
b1 = [1, 1];  b2 = [1, 1]      # 自变量边界
codes = [1, 1]                 # 变量的编码方式，2个变量均使用格雷编码
precisions =[6, 6]             # 变量的精度，10表示精确到小数点后10位
scales = [0, 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
"""========================遗传算法参数设置========================="""
NIND = 50             # 种群规模
MAXGEN = 1000         # 最大遗传代数
GGAP = 0.8            # 代沟：子代与父代个体不相同的概率为0.8
selectStyle = 'sus';  # 遗传算法的选择方式设为"sus"——随机抽样选择
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 = 0.1              # 变异概率
SUBPOP = 1            # 设置种群数为1
maxormin = 1          #
   设置最大最小化目标标记为1，表示是最小化目标，-1则表示最大化目标
"""========================开始遗传算法进化========================"""
FieldD = ga.crtfld(ranges,borders,precisions,codes,scales) #
   调用函数创建区域描述器
Lind = np.sum(FieldD[0, :]) # 计算编码后的染色体长度
Chrom = ga.crtbp(NIND, Lind) # 根据区域描述器生成二进制种群
Phen = ga.bs2rv(Chrom, FieldD) #对初始种群进行解码
LegV = np.ones((NIND, 1)) # 初始化种群的可行性列向量
[ObjV,LegV] = aimfuc(Phen, LegV) # 计算初始种群个体的目标函数值
# 定义进化记录器，初始值为nan
pop_trace = (np.zeros((MAXGEN, 2)) * np.nan)
# 定义种群最优个体记录器，记录每一代最优个体的染色体，初始值为nan
ind_trace = (np.zeros((MAXGEN, Lind)) * np.nan)
# 开始进化！！
start_time = time.time() # 开始计时
for gen in range(MAXGEN):
  FitnV = ga.ranking(maxormin * ObjV, LegV) #
     根据目标函数大小分配适应度值
  SelCh=ga.selecting(selectStyle, Chrom, FitnV, GGAP, SUBPOP) # 选择
  SelCh=ga.recombin(recombinStyle, SelCh, recopt, SUBPOP) #交叉
  SelCh=ga.mutbin(SelCh, pm) # 二进制种群变异
  Phen = ga.bs2rv(SelCh, FieldD) # 对育种种群进行解码(二进制转十进制)
  LegVSel=np.ones((SelCh.shape[0], 1))#初始化育种种群的可行性列向量
  [ObjVSel,LegVSel] = aimfuc(Phen, LegVSel) # 求育种个体的目标函数值
  [Chrom,ObjV,LegV] =
     ga.reins(Chrom,SelCh,SUBPOP,1,1,maxormin*ObjV,maxormin*ObjVSel
  ,ObjV,ObjVSel,LegV,LegVSel) # 重插入得到新一代种群
  # 记录
  pop_trace[gen, 1] = np.sum(ObjV) / ObjV.shape[0] #
     记录当代种群的目标函数均值
  if maxormin == 1:
    best_ind = np.argmin(ObjV) # 计算当代最优个体的序号
  elif maxormin == -1:
    best_ind = np.argmax(ObjV)
  pop_trace[gen, 0]=ObjV[best_ind]#记录当代种群最优个体目标函数值
  ind_trace[gen, :] = Chrom[best_ind, :] #
     记录当代种群最优个体的变量值
# 进化完成
end_time = time.time() # 结束计时
"""============================绘图================================"""
ga.trcplot(pop_trace, [['最优个体目标函数值','种群的目标函数均值']],
   ['demo_result'])
```

运行结果如下：



若要输出相关的数据结果，则可以在上面脚本最下方添加以下代码：

```python
"""============================输出结果============================"""
best_gen = np.argmin(pop_trace[:, 0]) # 计算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：')
# 最优个体记录器存储的是各代种群最优个体的染色体，
# 此处需要解码得到对应的基因表现型
variables = ga.bs2rv(ind_trace, FieldD) # 解码
for i in range(variables.shape[1]):
  print(variables[best_gen, i])
print('最优的一代是第',best_gen + 1,'代')
print('用时：', end_time - start_time, '秒')
```

对应的输出结果为：

最优的目标函数值为：-1.91322295498

最优的控制变量值为：

-0.547197228336

-1.54719704952

最优的一代是第 600 代

用时：2.879918336868286 秒

进化过程中各代种群最优个体的目标函数值和均值如下图：

寻优结果与理论值非常接近。

脚本详细解析：

编写 Geatpy 脚本风格与 Matlab 的 gatbx 以及 GEATbx 相近，对于新的问题，你可以复制上面的脚本，结合实际问题修改函数定义和变量设置，以及绘图和输出便可以对新问题进行求解。实际上你会发现脚本中大部分基本上可以固定不变的 (前提是多次使用时采用的基本模型没变)，因此在后面的章节中，我们会讲解如何用进化算法模板来代替编写脚本，这样会大大节省开发时间。

下面详细讲解编写类似脚本的流程，你需要做以下步骤：

1. 导入相关库，如 numpy、Geatpy 等。

2. 定义了优化目标函数。(在后面的章节中，我们会详细介绍如何用函数接口来取代这种把函数定义写在脚本内部的方法)。

3. 定义了优化问题的变量约束条件。其中 ranges、borders、codes、precisions、scales 是根据实际问题而编写的，但写法格式是固定的。假如优化函数中包含 3 个变量，那么我们可以这么写：

x1 = [下界, 上界]

x2 = [下界, 上界]

x3 = [下界, 上界]

b1 = [0, 1] # (这里 0 表示变量 x1 不包含下界，1 表示包含上界)

b2 = [1, 0]

b2 = [1, 1]

ranges = np.vstack([x1, x2, x3]).T # 根据 x1,x2,x3 生成一个矩阵来表示所有变量的范围

borders = np.vstack([b1, b2, b3]).T

...

4. 对遗传算法的运行参数进行设置，比如确定种群规模 (即种群包含多少个个体)、是否包含子种群 (SUBPOP 设为 1 表示种群只包含 1 个种群，没有划分子种群)、最大遗传代数、代沟、重组及变异概率等。

5. 创建用于描述种群特征的“区域描述器”。我们可以发现，在 Matlab 的 gatbx 和 GEATbx 工具箱中，区域描述器是需要手动创建的，而且格式相当复杂。在 Geatpy 中，我们采用 crtfld 函数来自动化创建 (当然你也可以手动创建)。crtfld 函数会根据变量的边界和精度自动地调整变量的范围，返回一个符合规范的区域描述器。

6. 根据创建的区域描述器计算染色体的长度。FieldD 区域描述器的结构参见“Geatpy 函数”章节的 crtfld 函数解析，其中讲解了区域描述器 FieldD 的第一行是代表控制各个变量的染色体基因数目。因此对它求和便可算出染色体的长度。

7. 创建进化记录器，它的写法也是固定的，2 表示该进化记录器有 2 列，每一列存放不同的数据。本例中，它的第一列是存放各代种群最优个体的目标函数值；第二列是存放各代种群的平均目标函数值。

8. 创建最优个体记录器，记录各代种群的最优个体的染色体。

9. 做完这些基本步骤后，后面都是遗传算法的基本流程了。注释中有详细解释，这里不再重复讲解。

10. 绘图和输出寻优结果。Geatpy 中提供 trcplot 和 resplot 两个绘图函数 (后续会添加更多的绘图函数)。关于其具体用法详见“Geatpy 函数的”相关函数解析文档。

特别注意：

Geatpy 的适应度计算均遵循”目标函数越大，适应度越小”的约定，与之密切相关的函数有：ranking、reins 等。这些函数都要传种群个体的目标函数值，此时若研究的问题是最大化问题，那么要对目标函数值矩阵取相反数 (乘上-1)。因此在 Geatpy 中常常设置一个最大最小化目标的标记：maxormin。maxormin=1 表示是最小化目标;maxormin=-1 表示是最大化目标。注意，当函数的传入参数与返回参数均含有与目标函数值相关的变量时，在传入前要乘以 maxormin，同时也要对该函数所返回的含义等价或相似的变量乘上 maxormin 进行数值的还原。(否则目标函数值就会变成相反数了)。这些函数有：awGA、rwGA、upNDSet（详见这些函数的参考资料）。

3. 总结

至此，你已经可以模仿着上面的脚本来用遗传算法求解一些简单的优化问题了。在后面的章节我们会讲解如何使用更简单、灵活的进化算法模板和函数接口来快速编程以及融合到你正在进行的其他 Python 项目中。