Guide of Python

内容概要： 数学建模算法

创建时间： 2022/4/7 13:41 **更新时间：** 2022/4/17 15:27

作者： **TwinkelStar**

Genetic 遗传算法
Genetic Algorithm

1、操作系统相关环境

1）硬件环境：

- 电脑

2）软件环境：

- Python3.7(向下兼容 Python3)(程序设计语言)
- Numpy1.19.5(兼容大部分版本)(科学计算库)

3）操作系统(2 选 1)：

- Windows7
- Windows10
- Windows11

2、遗传算法

遗传算法是用于解决最优化问题的一种搜索算法。遗传算法借用了生物学里达尔文的进化理论：“适者生存，优胜劣汰”，将该理论以算法的形式表现出来就是遗传算法的过程。

1）基本原理

遗传算法是通过大量备选解的变换、迭代和变异，在解空间中并

行动态地进行全局搜索的最优化方法，是模拟生物基因遗传的做法，算法的基本原理通过编码组成的群体组成初始群体后，遗传操作的任务就算对群体的个体按照他们对环境适应度（也就我们所求的函数的极值）评估，施加了一定的操作之后从而实现优胜劣汰的进化过程，算法的基本程序框图如图 1 所示：

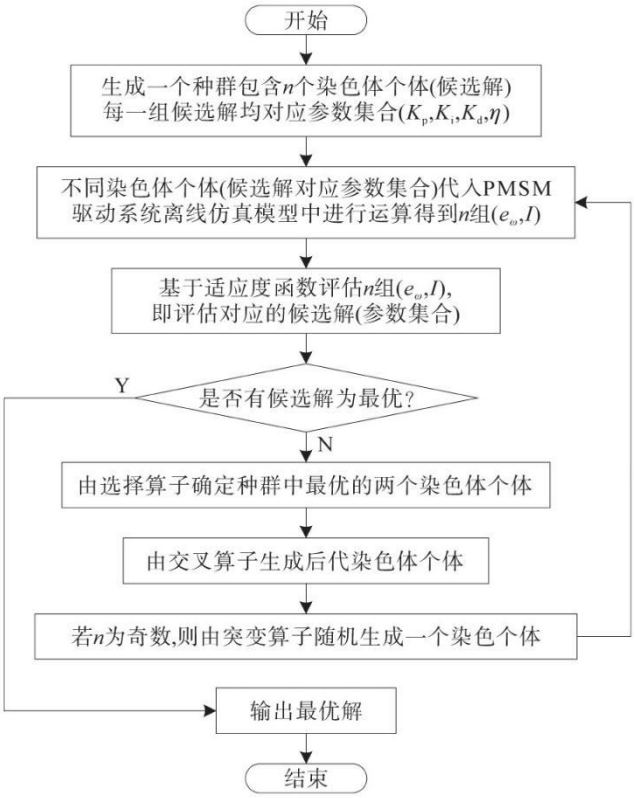

图 1.遗传算法程序框图
fig1.Genetic Algorithm

2）初始化种群

图 1 中所提到的个体概念，也就是一组可能解，在计算机程序中被编码为一个向量表示，而在我们这个问题中，个体是 x , y 的取值，是两个实数，所以问题就可以转化为如何将实数编码为一个向量表示，为了方便变异和交叉的方便，需要将实数编码为二进制数。

初始化种群的代码如图 2 所示：

```
def initpop(popsize, chromlength):
    """
    Parameters
    ----------
    popsize :  TYPE
        种群的数目.
    chromlength : TYPE
        表示染色体的长度（二值数的长度），  长度的大小取决于变量的二进制编码的长度

    Returns
    -------
    pop : TYPE
        随机种群

    """
    pop=np.round(np.random.rand(popsize,chromlength))
    return pop
```

图 2.初始化种群
fig2.Init population

我们创建一组随机的矩阵，染色体的长度作为矩阵的列，种群的数量作为矩阵的行。

我们可以假设这个种群存在的目的，是帮我寻找一个函数 F(x)的极值（可能是最大值或者最小值）。通过我们自己人为的筛选，将不满足条件的个体（也就是矩阵的行）给淘汰，直到进化出符合条件的个体为止，图 3 为随机生成的种群矩阵，矩阵的值用随机的 0，1 填充：

	0	1	2	3	4	5	6	7	8
0	1	0	0	1	1	0	0	1	0
1	1	0	0	0	0	1	0	0	1
2	1	1	0	0	1	0	0	0	0
3	1	0	0	1	1	1	0	0	0
4	0	0	1	0	1	1	0	0	0
5	0	0	1	1	0	0	0	0	0
6	1	0	1	1	1	0	0	0	0
7	0	1	0	0	1	1	0	1	0
8	0	1	0	1	0	0	1	1	1
9	0	0	1	0	0	0	0	0	1
10	0	0	0	1	1	1	1	0	1

图 3.种群矩阵
fig3.Population matrix

从图 3 可以看到我们生成的随机矩阵，行为 20，列为 8，表示我们有 20 个种群，染色体的长度为 8，通过将二进制数转化为十进制数之后再映射到变量域，带入我们的函数 F(x)后可以得到目标函数值，将得到的目标函数值存储，寻找函数的极值。

3）函数值的编码

对于函数 F(x)，可能存在多个变量决定 F(x)的值，也可能只有一

个实数 x 的输入，决定着 F(x)的输出值。

假设变量 x 的取值范围为[0，8]，这是变量域，与之对应会有一个二进制域[0，2^8]，二进制域对应的就是我们的变量域，将数字 8 看成 8 位的二进制数，可以得到 2 的八次方为：256，但因为在二进制当中是从 0 开始的，0-255，有 256 个数，由于八位二进制数最多只能取到 255,取不到 256。

实数 x 的范围只有[0，8]，我们将二进制转化为十进制之后，还要求解二进制域到变量域的值，需要先将我们二进制数转化为十进制数，然后用十进制数映射到变量域中，二进制域到变量域之间的映射关系为:

$$x' = x * y_{len} / 2^{y_{len}} \quad (1)$$

在公式（1）中，x` 表示的二进制数转化为十进制数之后，对应变量域的值，x 是二进制数转化为十进制的数，y_{len} 表示的是染色体的长度。

二进制数转化为十进制数的代码如图 4 所示：

```
def decodebinary(pop):
    """
    二进制数转十进制数函数
    Parameters
    ----------
    pop : TYPE
          初始化种群.

    Returns
    -------
    pop2 种群的染色体二进制数转十进制数

    """
    px,py = pop.shape
    pop1 = np.zeros((px,py))
    for i in range(py):
        pop1[:,i] = (2**(py-i-1))*pop[:,i]
    pop2=np.sum(pop1, 1)#对pop1向量的每行求和 标识位0代表每列求和，标识位1代表每行求和

    return pop2
```

图 4.编码与解码
fig4.decode and encode

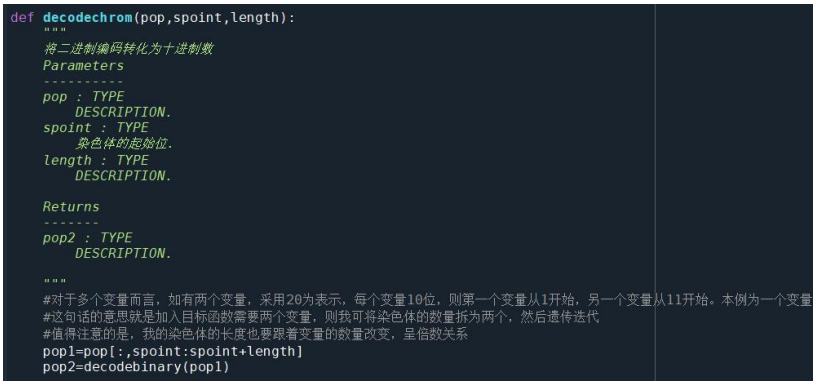

图 5.多变量的编码与解码
fig5.more value of decode and encode

4）目标函数值的计算

得到了一个种群，现在要根据适者生存的规则把优秀的个体保存下来，换句话来说，就是将个体所对应变量域的值代入函数 F(x)，同时淘汰掉那些不适应环境的个体。在我们的求最大值的问题中可以直接用可能解（个体）对应的函数的函数值的大小来评估，这样可能解对应的函数值越大越有可能被保留下来。

需要将求解的函数写成代码的形式，objvalue 就是目标函数的返回值，目标函数值的计算代码如图 6 所示：

图 6.个体函数值的计算
fig6.Fitvalue Algorithm function value calculate

5）筛选

在目标函数中，x 对应的取值范围是[0,15]，所以我们要将小于 0

的数进行筛除，做第一步的筛选操作，筛选代码如图 7 所示：

```
def calfitvalue(objvalue):
    """
    计算个体的适应值，在calobjvalue已经计算好，需要将小于0的个体删除，方便后续的概率计算

    Parameters
    ----------
    objvalue : TYPE
        目标函数值.

    Returns
    -------
    fitvalue : TYPE
        个体适应值.

    """
    global Cmin
    Cmin = 0
    fitvalue = np.zeros((objvalue.shape[0],objvalue.shape[1]))
    px, py = objvalue.shape
    for i in range(px):
        if objvalue[i,0] + Cmin > 0:
            temp = objvalue[i,0] + Cmin
        else:
            temp = 0

        fitvalue[i,0]=temp
    return fitvalue
```

图 7.个体函数值
fig7.Fitvalue Algorithm function value

6）选择复制

在遗传学中，选择复制是种群繁衍后代的表现，在一些特定的环境下，种群会不断的迭代更新，直到选出最好的个体为止。对等的，在遗传算法中，选择复制是种群迭代的关键，我们将种群所对应的函数值计算出来之后，将函数值作为我们的适应条件，然后计算每个种群所占的比例，最后通过轮盘赌的方式进行选择，如图 8 所示：

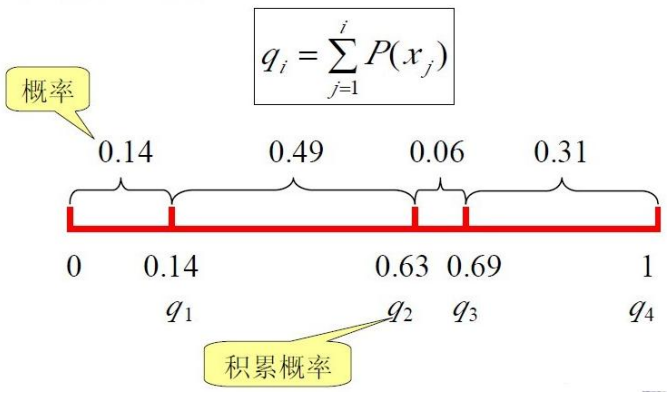

图 8.轮盘赌算法
fig8.Roulette Algorithm

我们计算出所有种群的函数值之后，把各个概率累加，这样就得到长度为 1 的线段，每一段长度代表相应值的概率，概率越大，该线

段占整个现电的比例越大。然后随机选取 0~1 之间的一个数，插入整个线段中。落入哪一段就选那个值。并且落入哪一段的概率与该段的长度（概率）成正相关，这样就相当于转轮盘选数了。这就是遗传算法的核心所在。

选择复制的代码如图 9 所示：

```
def selection(pop, fitvalue):
    """
    选择函数 选择复制，决定哪些个体可以进入下一代
    采用轮盘赌选择

    Parameters
    ----------
    pop : TYPE
        种群的个体.
    fitvalue : TYPE
        个体适应值.

    Returns
    -------
    newpop : dict
        新的种群.

    """
    totalfit = np.sum(fitvalue)
    fitvalue_pro = fitvalue / (totalfit + 0.000001)
    fitvalue_pro_cumnsum = np.cumsum(fitvalue_pro)
    fitvalue_pro_cumnsum = np.reshape(fitvalue_pro_cumnsum,(-1,1))#以行的形式

    px, py = pop.shape
    #轮盘随机概率 从小到大排序
    ms = np.sort(np.random.rand(px,1),0)

    fitin = 1 - 1   #pop种群 第几代个体 因为python的下标从0开始。所以是第一代的索引是0 故用1-1
    newin = 1 - 1  #pop种群 第几代个体
    newpop_dict = {}
    #我愿称为[适者生存，优胜劣汰] while循环
    while newin <= px-1:
        if ms[newin, 0] < fitvalue_pro_cumnsum[fitin, 0]:
            newpop_dict[newin] = pop[fitin,:]
            newin += 1
        else:
            fitin += 1

    newpop = np.zeros((newin,py))
    for i in range(newin):
        newpop[i,:] = newpop_dict[i]

    return newpop
```

图 9 选择复制算法

fig9.Selection Clone Algorithm

7）基因重组

在自然界生物进化过程中起核心作用的是生物遗传基因的重组（加上变异）。同样。在遗传算法中起核心作用的就是遗传操作的交叉算子。从函数值的编码我们可以很容易的得到一组二进制数，我们可以实现对种群的某一对或者多对染色体进行基因重组（实质上就是二进制数的交换），通过这样的方法，我们的种群矩阵的对于环境的适应度（也就是目标函数值）会增强，会得到一些优秀的个体收敛我们的模型。

染色体交叉算法如图 10 所示：

```
def crossover(pop, pc):
    """
    交叉算法 实现基因重组
    Parameters
    ----------
    pop : TYPE
        种群.
    pc : TYPE
        交叉概率.

    Returns
    -------
    newpop : TYPE
        DESCRIPTION.

    """
    px, py = pop.shape
    newpop = np.zeros((px, py))
    # seletion_litst = [i for i in range(px)]
    # sl = seletion_litst[0:px:2]
    for i in range(px-1):
        #是否能够进行基因重组
        if pc > np.random.rand(1)[0]:
            cpoint = int(np.round(np.random.rand(1)[0] * py) - 1)
            if cpoint == 0:
                cpoint = 1

            newpop[i, :][0:cpoint] = pop[i, 0:cpoint]
            newpop[i, :][cpoint+1:py] = pop[i+1, cpoint+1:py]

            newpop[i+1, :][0:cpoint] = pop[i+1, 0:cpoint]
            newpop[i+1, :][cpoint+1:py] = pop[i, cpoint+1:py]

        else:
            newpop[i,:] = pop[i,:]
            newpop[i+1,:] = pop[i+1,:]

    return newpop
```

图 10.交叉算子的计算
fig10.Crossover Algorithm

8）基因突变

变异算子的基本内容是对群体的个体串的某些基因座上的基因值做变动。依据个体编码表示的方法不同，可有二进制变异，实数变异等算法。因为我们采用的是二进制的编码方式，所以我们使用二进制变异，其原理很简单，在设定变异概率的基础上，对于某个个体，使其染色体某个位置上的二进制数进行变化，将 0 变为 1，1 变为 0，实现基因的突变，因为在遗传迭代的过程中，如果种群在一个平面内进行收敛，可能只会得到一个局部最优解，而基因突变可以打破这种平衡，突变出适应度更高的个体。

基因突变的算法实现如图 11 所示：

```
def mutation(pop, pm):
    """
    变异算法 实现基因突变

    Parameters
    ----------
    pop : TYPE
        种群.
    pm : TYPE
        变异概率.

    Returns
    -------
    newpop : TYPE
        新的变异种群.

    """
    px, py = pop.shape
    newpop = np.zeros((px, py))

    for i in range(px):
        if pm > np.random.rand(1)[0]:
            mpoint = int(np.round(np.random.rand(1)[0] * py) - 1)
            if mpoint == 0:
                mpoint = 1
            newpop[i,:] = pop[i,:]
            if newpop[i, mpoint] == 0:
                newpop[i, mpoint] = 1

        newpop[i,:] = pop[i,:]
    return newpop
```

图 11.基因突变算法
fig11.Mutation Algorithm

9）最优值的计算

我们最后只需要将最好的结果保存下来即可，代码的实现如图 12 所示：

```
def best(pop, fitvalue):
    """
    最优的个体及其适应值
    Parameters
    ----------
    pop : TYPE
        种群.
    fitvalue : TYPE
        适应值.

    Returns
    -------
    bestindividual : 最大适应值
        DESCRIPTION.
    bestfit : TYPE
        最大适应值的个体.

    """
    px, py = pop.shape
    bestindividual = pop[0,:]
    bestfit = fitvalue[0]

    for i in range(1,px):
        if fitvalue[i] > bestfit:
            bestindividual = pop[i,:]
            bestfit = fitvalue[i]

    return bestindividual,bestfit
```

图 12.最优值的计算
Fig12.best value of count

3、例题与代码实现

1）例题

求解下列函数的最小值，目标函数的取值范围为[-5，5]：

$$f(\mathrm{x}) = 4x^2 - 2.1x^4 + \frac{x^6}{3} + xy - 4y^2 + 4y^4 \qquad (2)$$

2）代码实现

Python 主函数代码如图 13 所示：

```python
if __name__ == "__main__":
    popsize=30; #群体大小
    chromlength=15; #字符串长度（染色体的长度）
    pc=0.7; #交叉概率
    pm=0.005 #变异概率

    #初始化种群
    pop=initpop(popsize, chromlength)
    poptest=pop
    #开始迭代
    epoch = 50

    x_ = []
    y_ = []

    for i in range(epoch):
        objvalue = calobjvalue(pop)     #计算目标函数值
        fitvalue = calfitvalue(objvalue)    #计算群体中每个个体的适应度

        newpop = selection(pop, fitvalue)
        newpop1 = crossover(newpop, pc)
        newpop2 = mutation(newpop1, pm)

        objvalue = calobjvalue(newpop2)     #计算目标函数值
        fitvalue = calfitvalue(objvalue)    #计算群体中每个个体的适应度

        bestindividual,bestfit = best(newpop2, fitvalue) #求出群体中适应值最大的个体及其适应值
        x_.append(bestfit)
        pop = newpop2

        plt.plot(x_, lw=3, label='funcotio_max_value_x')
        plt.show()

    reshape_best_infrivedual = np.reshape(bestindividual,(1,-1))#以行的形式
    best_x_value = encode(reshape_best_infrivedual)
    plt.plot(x_, lw=3, label='funcotio_max_value_x')
    plt.legend()
    plt.show()

    print("最好的个体是 ",bestindividual)
    print("函数：f(x) = 9sin(5x)+7cos(4x)的极值为 ",bestfit[0]) #最后拟合之后随便取一个值即可
    print("x的取值应为：  ",best_x_value[0]) #最后拟合之后随便取一个值即可
```

图 13.主函数
fig13.main

函数说明如表 1 所示：

表 1.符号描述图表
table1.Symbol Description

Symbol	Description
Initpop	初始化种群
Decodechrom	二进制数转化为十进制数
Decodebinary	二进制编码转为十进制数多变量
Calobvalue	计算目标函数
Calfitvalue	计算群体中每个个体的适应度
Selection	选择算法
Crossover	交叉算法
Mutation	变异算法
Best	计算适应度值最大的个体及其适应度
MainGenetic	主函数程序

3）代码调用方式

配置好 Python 环境，选择 Genetic.py 文件，直接运行代码，运行结果如图 14 所示：

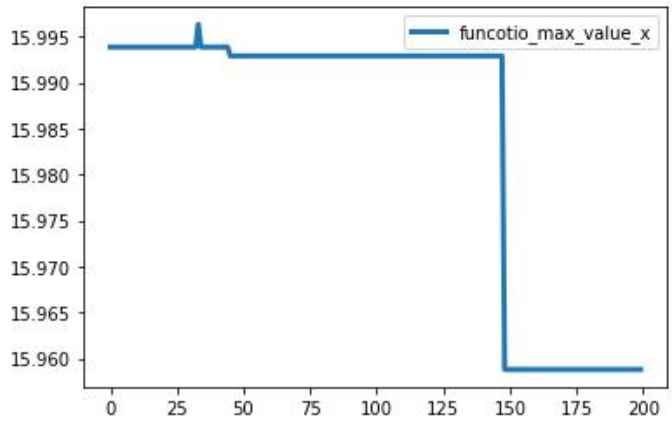

图 14.文件目录
fig14.file path

函数返回结果如图 15 所示：

```
最好的个体是  [1. 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
函数：f(x) = 9sin(5x)+7cos(4x)的极值为  15.958789992389079
x的取值应为：   14.121524704733421
```

图 15.变量区运行结果
fig15.value test result

需要注意的是，在遗传算法中，有四个参数需要提前设定：

- 群体的大小的选取范围：[20,100]；
- 遗传算法的迭代次数的选取范围:[100,500]；
- 交叉概率的选取范围：[20,100]；
- 变异概率的选取范围：[0.0001,0.1]。