数学建模之【遗传算法】

原创 TwinkelStar 已于 2023-10-02 16:49:21 修改 阅读量5.4k 收藏 28 点赞数 5 版权

分类专栏： 数学建模算法 文章标签： python

遗传算法

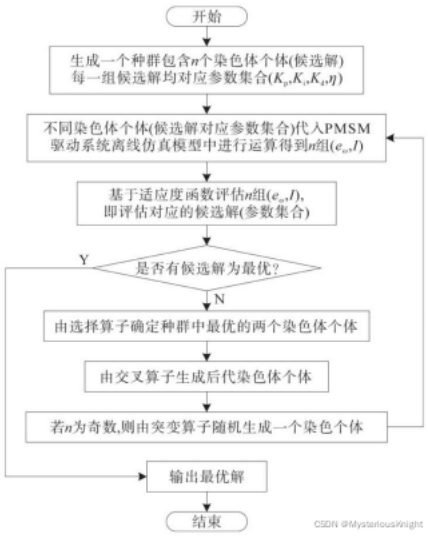
一、介绍

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。

二、基本原理

1）程序流程

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（适应度）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：



2）物竞天择

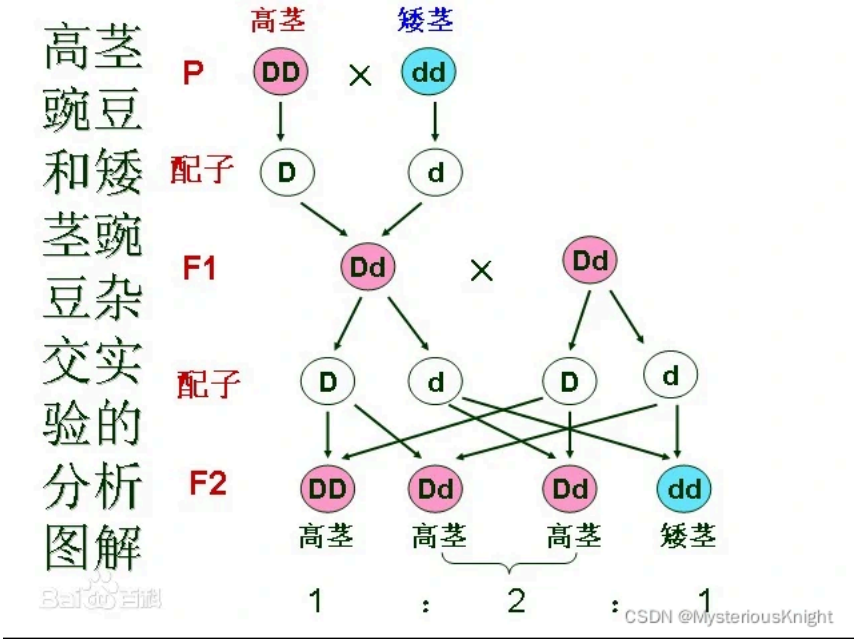
我们可以假设有一个种群，这个种群存在的目标是帮我寻找一个函数F（x）的极值（可能是max or min），首先我们随机生成一组种群，假设我对变量x的取值范围为【0，8】，这是我们的变量域，因为我们要玩一些不一样的操作，先跟着我，后面你会恍然大悟！我们会有一个二进制域【0，2^8】,我们的二进制域对应的就是我们的变量域，我们将数字8看成8位的二进制数，可以得到2的八次方为：256，但因为我们在二进制当中我们是从0开始的，0-255，有256个数,我们的八位二进制数最多只能取到255,取不到256。

```
"二进制" 0 0 0 0  0 0 0 0
"二进制数 x-> " 1 1 1 1  1 1 1 1
"x转化为十进制" 1*2^0 + 1*2^1 + 1*2^2 + 1*2^3 + 1*2^4 + 1*2^5 + 1*2^6 + 1*2^7 = 255
```

因为我们实数x的范围只有【0，8】，我们将二进制转化为十进制之后，还有求解二进制域到变量域值，**二进制域**到**变量域**之间的映射关系为：

```
"变量域x_"  x_ = x*8/2^8
```

如此，我们将我们的8看作是染色体的长度，这样我们就可以做一些基因重组和基因变异的操作了！！！，我们先来一组基因重组图：



是不是很熟悉，这个不就是高中生物的杂交实验吗？如果我们把**DD**和**dd**看成我们染色体的**0**和**1**呢？我们再形象一点

```
# 定义我们的染色体
x1 = [0 1 0 0 1 0 0 1]
x2 = [1 1 0 1 0 0 0 1]
#OK 我们现在有两个种群了（抽象一点）
# 第一个种群 x1 的染色体为 0 1 0 0  1 0 0 1
# 第一个种群 x2 的染色体为 1 1 0 1  0 0 0 1

#我们尝试做一下杂交实验？学一学孟德尔的八年抗战
"第一代:" 0 1 0 0    1 0 0 1     *     1 1 0 1    0 0 0 1
                  |                            |
                  v                            v

"配子:" [0 1 0 0]    [1 0 0 1 ]       [1 1 0 1 ]   [0 0 0 1]
                                                    #用x1的第一个配子和x2的第二个配子进行基因重组
                                                    #用x1的第二个配子和x2的第一个配子进行基因重组

"F1代:" [0 1 0 0][0 0 0 1]             [1 1 0 1 ][1 0 0 1]

new_x1 = [0 1 0 0 0 0 0 1]
new_x2 = [1 1 0 1 1 0 0 1]

# 不知道你有没有恍然大悟
```

有了二进制的操作，我们做变异的时候也是同样的道理，我们只需要把0变异为1，1变异为0，就能完成我们的变异算法。

3）算法步骤

求解函数:f(x) = 9*sin(5x)* + 7cos(4*x)的最大值，x的取值范围为【0，8】

3.1初始化种群

我们先初始化我们的种群，先初始化一些随机解，因为我们要在搜索空间内寻找我们的最优解，可参考粒子群算法的思想。

```
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
```

3.2二进制编码转化为十进制数

设计我们的算法，将二进制的矩阵转化为十进制数，为了得到我们的二进制域。

```
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

def encode(bestpop):
    """
    解码x的值

    Parameters
    ----------
    bestpop : TYPE
        最好的个体.

    Returns
    -------
    x : TYPE
        最优X的取值.

    """

    temp = decodebinary(pop)
    x=temp*15/32767
    return x
```

我们的pop2是一个转化为十进制数之后的种群，可以理解为种群的**表现型**，我们的0和1的操作反应的是种群的**基因型**，因为我们的变量不可能只有一种，扩展我们的思维，我们先设置的染色体长度，因为我们的变量只有一个，它的值域在【0，8】，如果我们有两个变量，不同变量对应不同值域，我们的染色体的长度是否也要切片计算？仔细思考。

```
def decodechrom(pop,spoint,length):
    """
    将二进制编码转化为十进制数
    Parameters
    ----------
    pop : TYPE
        DESCRIPTION.
    spoint : TYPE
        染色体的起始位.
    length : TYPE
        DESCRIPTION.
    Returns
    -------
    pop2 : TYPE
        DESCRIPTION.

    """
    #对于多个变量而言，如有两个变量，采用20为表示，每个变量10位，则第一个变量从1开始，另一个变量从11开始。本例为一个变量
    #这句话的意思就是加入目标函数需要两个变量，则我可将染色体的数量拆为两个，然后遗传迭代
    #值得注意的是，我的染色体的长度也要跟着变量的数量改变，呈倍数关系
    pop1=pop[:,spoint:spoint+length]
    pop2=decodebinary(pop1)

    return pop2
```

这里的spoint代表的是我们的一个终止位，可以对我们的变量进行一个切片，自行理解。

3.3实现目标函数的计算

实现目标函数的计算，就是把x的值带进去，很简单。

```
def calobjvalue(pop):
    """
    实现目标函数的计算
    Parameters
    ----------
    pop : TYPE
        种群.

    Returns
    -------
    objvalue : TYPE
        返回目标函数值.

    """
    temp = decodechrom(pop, 0, 15)
    x=temp*15/32767
    objvalue = 9*np.sin(5*x)+7*np.cos(4*x)
    objvalue = np.reshape(objvalue,(-1,1))#以行的形式输出
    return objvalue
```

3.4剔除0以外的值

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。

```
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
```

3.5选择算法

选择算法我们采用轮盘赌，决定哪些个体可以进入下一代，用轮盘赌选择复制

```
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
```

3.6交叉算法（基因重组）

物竞天择是遗传学的核心，相应的，选择和变异也是遗传算法的核心，我们对染色体进行一个基因重组的操作，具体代码如下：

```
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
```

3.7变异算法（基因突变）

0变1，1变0 你上你也行

```
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
```

3.7最优计算（基因突变）

简单的排序取最大（不要忘记我们粒子群算法里面的全局极值和个体极值）

```
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
```

3.完整代码

不要忘记导入我们的numpy 和matplotlib库，将上面的函数和导入的库，放在主函数的上面，然后运行代码，不过我这里的取值是【0，15】，上面说【0，8】便于读者理解。

免登录复制

```
import numpy as np
import matplotlib.pyplot as plt

# 遗传算法
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

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

def decodechrom(pop,spoint,length):
    """
    将二进制编码转化为十进制数
    Parameters
    ----------
    pop : TYPE
        DESCRIPTION.
    spoint : TYPE
        染色体的起始位.
    length : TYPE
        DESCRIPTION.

    Returns
    -------
    pop2 : TYPE
        DESCRIPTION.

    """
    #对于多个变量而言，如有两个变量，采用20为表示，每个变量10位，则第一个变量从1开始，另一个变量从11开始。本例为一个变量
    #这句话的意思就是加入目标函数需要两个变量，则我可将染色体的数量拆为两个，然后遗传迭代
    #值得注意的是，我的染色体的长度也要跟着变量的数量改变，呈倍数关系
    pop1=pop[:,spoint:spoint+length]
    pop2=decodebinary(pop1)

    return pop2

def encode(bestpop):
    """
    解码x的值

    Parameters
    ----------
    bestpop : TYPE
        最好的个体.

    Returns
    -------
    x : TYPE
        最优X的取值.

    """
```

```
    temp = decodebinary(pop)
    x=temp*15/32767
    return x

def calobjvalue(pop):
    """
    实现目标函数的计算
    Parameters
    ----------
    pop : TYPE
        种群.

    Returns
    -------
    objvalue : TYPE
        返回目标函数值.

    """
    temp = decodechrom(pop, 0, 15)
    x=temp*15/32767
    objvalue = 9*np.sin(5*x)+7*np.cos(4*x)
    objvalue = np.reshape(objvalue,(-1,1))#以行的形式输出
    return objvalue

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

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



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

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

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




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 = 200

    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
```

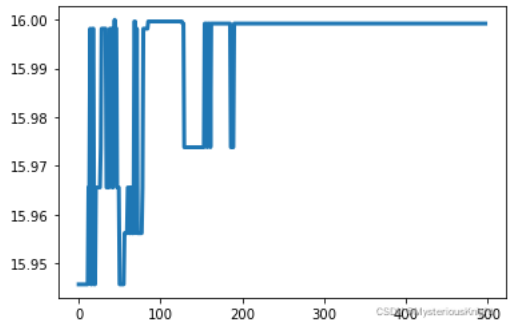
```
        bestindividual,bestfit = best(newpop2, fitvalue) #求出群体中适应值最大的个体及其适应值
        x_.append(bestfit)
        pop = newpop2

        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
```

4）运行结果



最好的个体是 [1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]

函数：f(x) = 9sin(5x)+7cos(4x)的极值为 15.999199583116948

x的取值应为： 7.851802117984557

显示推荐内容

TwinkelStar