

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

分类专栏：

数学建模算法

 文章标签：

python

版权

C

数学建模算法 专栏收录该内容

1 订阅 2 篇文章

订阅专栏

遗传算法

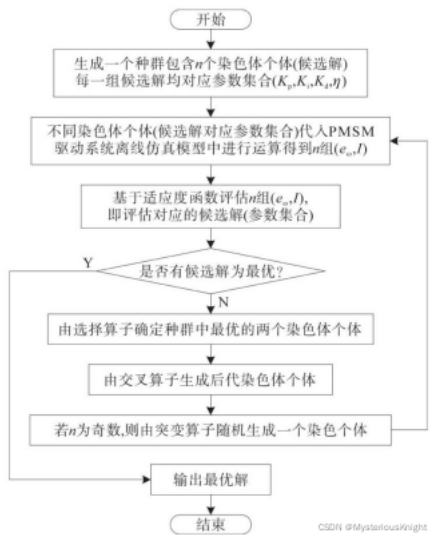
一、介绍

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。

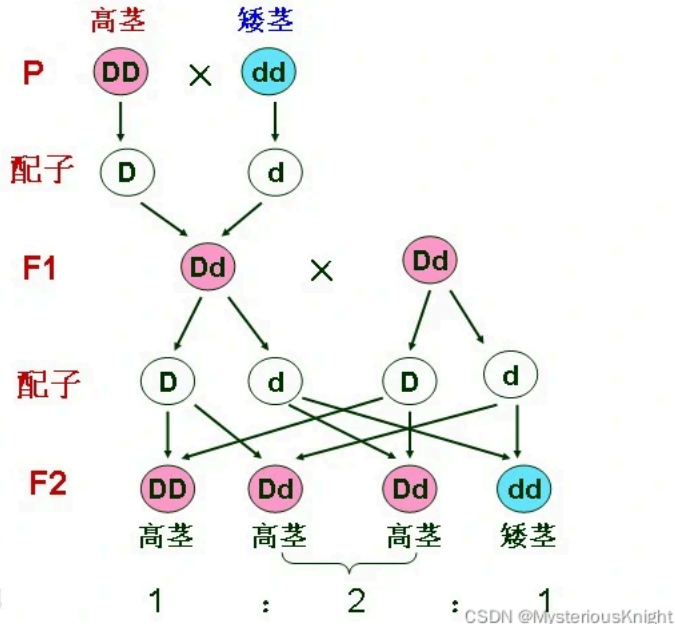
```
1 | "二进制" 0 0 0 0   0 0 0 0
2 | "二进制数 x->" 1 1 1 1   1 1 1 1
3 | "x转化为十进制" 1*2^0 + 1*2^1 + 1*2^2 + 1*2^3 + 1*2^4 + 1*2^5 + 1*2^6 + 1*2^7 = 255
```

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：

```
1 | "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：

高茎豌豆和矮茎豌豆杂交实验的分析图解



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

```
1 # 定义我们的染色体
2 x1 = [0 1 0 0 1 0 0 1]
3 x2 = [1 1 0 1 0 0 0 1]
4 #OK 我们现在有两个种群了（抽象一点）
5 # 第一个种群 x1 的染色体为 0 1 0 0 1 0 0 1
6 # 第一个种群 x2 的染色体为 1 1 0 1 0 0 0 1
7
8 #我们尝试做一下杂交实验？学一学孟德尔的八年抗战
9 "第一代:" 0 1 0 0 1 0 0 1 * 1 1 0 1 0 0 0 1
10           |                   |
11           v                   v
12
13 "配子:" [0 1 0 0] [1 0 0 1] [1 1 0 1] [0 0 0 1]
14                                     #用x1的第一个配子和x2的第二个配子进行基因重组
15                                     #用x1的第二个配子和x2的第一个配子进行基因重组
16
17 "F1代:" [0 1 0 0][0 0 0 1] [1 1 0 1][1 0 0 1]
18
19 new_x1 = [0 1 0 0 0 0 0 1]
20 new_x2 = [1 1 0 1 1 0 0 1]
21
22 # 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: $f(x) = 9\sin(5x) + 7\cos(4x)$ 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。

```
1 def initpop(popsizel, chromlength):
2     """
3     Parameters
4     -----
5     popsize : TYPE
6         种群的数量.
7     chromlength : TYPE
8         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
9
10    Returns
11    -----
12    pop : TYPE
13        随机种群
14    """
15    pop=np.round(np.random.rand(popsizel,chromlength))
16    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。

```
1 def decodebinary(pop):
2     """
3     二进制数转十进制数函数
4     Parameters
5     -----
6     pop : TYPE
7         初始化种群.
8
9     Returns
10    -----
11    pop2 种群的染色体二进制数转十进制数
12
13    """
14    px,py = pop.shape
15    pop1 = np.zeros((px,py))
16    for i in range(py):
17        pop1[:,i] = (2**(py-i-1))*pop[:,i]
18    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
19
20    return pop2
21
22 def encode(bestpop):
23     """
24     解码x的值
25
26     Parameters
27     -----
28     bestpop : TYPE
29         最好的个体.
30
31     Returns
32     -----
33     x : TYPE
34         最优x的取值.
35
36     """
37
38     temp = decodebinary(pop)
39     x=temp*15/32767
40     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】，如果我们有二个变量，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。

```
1 def decodechrom(pop,spoint,length):
2     """
3     将二进制编码转化为十进制数
4     Parameters
5     -----
6     pop : TYPE
7         DESCRIPTION.
8     spoint : TYPE
9         染色体的起始位.
10    length : TYPE
11        DESCRIPTION.
12    Returns
13    -----
14    pop2 : TYPE
15        DESCRIPTION.
16
17    """
18    #对于多个变量而言，如有二个变量，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
19    #这句话的意思就是加入目标函数需要二个变量，则我可将染色体的数量拆为二个，然后遗传迭代
20    #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
21    pop1=pop[:,spoint:spoint+length]
22    pop2=decodebinary(pop1)
23
24    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。

```
1 def calobjvalue(pop):
2     """
3     实现目标函数的计算
4     Parameters
5     -----
6     pop : TYPE
7         种群.
8
9     Returns
10    -----
11    objvalue : TYPE
12        返回目标函数值.
13
14    """
15    temp = decodechrom(pop, 0, 15)
16    x=temp*15/32767
17    objvalue = 9*np.sin(5*x)+7*np.cos(4*x)
18    objvalue = np.reshape(objvalue, (-1,1))#以行的形式输出
19    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。

```
1 def calfitvalue(objvalue):
2     """
3     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
4
5     Parameters
6     -----
7     objvalue : TYPE
8         目标函数值.
9
10    Returns
11    -----
12    fitvalue : TYPE
13        个体适应值.
14
15    """
16    global Cmin
17    Cmin = 0
18    fitvalue = np.zeros((objvalue.shape[0],objvalue.shape[1]))
19
20    px, py = objvalue.shape
21    for i in range(px):
22        if objvalue[i,0] + Cmin > 0:
23            temp = objvalue[i,0] + Cmin
24        else:
25            temp = 0
26
27        fitvalue[i,0]=temp
28    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

```
1 def selection(pop, fitvalue):
2     """
3     选择函数 选择复制，决定哪些个体可以进入下一代
4     采用轮盘赌选择
5
6     Parameters
7     -----
8     pop : TYPE
9         种群的个体.
10    fitvalue : TYPE
11        个体适应值.
12
13    Returns
14    -----
15    newpop : dict
16        新的种群.
17
```

```

17
18
19 """
20 totalfit = np.sum(fitvalue)
21 fitvalue_pro = fitvalue / (totalfit + 0.000001)
22 fitvalue_pro_cumsum = np.cumsum(fitvalue_pro)
23 fitvalue_pro_cumsum = np.reshape(fitvalue_pro_cumsum, (-1,1))#以行的形式
24
25 px, py = pop.shape
26 #轮盘随机概率 从小到大排序
27 ms = np.sort(np.random.rand(px,1),0)
28
29 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
30 newin = 1 - 1 #pop种群 第几代个体
31 newpop_dict = {}
32 #我愿称为[适者生存, 优胜劣汰] while循环
33 while newin <= px-1:
34     if ms[newin, 0] < fitvalue_pro_cumsum[fitin, 0]:
35         newpop_dict[newin] = pop[fitin,:]
36         newin += 1
37     else:
38         fitin += 1
39
40 newpop = np.zeros((newin,py))
41 for i in range(newin):
42     newpop[i,:] = newpop_dict[i]
43
44 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：

```

1 def crossover(pop, pc):
2     """
3     交叉算法 实现基因重组
4     Parameters
5     -----
6     pop : TYPE
7         种群.
8     pc : TYPE
9         交叉概率.
10    Returns
11    -----
12    newpop : TYPE
13        DESCRIPTION.
14
15    """
16    px, py = pop.shape
17    newpop = np.zeros((px, py))
18    # seletion_list = [i for i in range(px)]
19    # sl = seletion_list[0:px:2]
20    for i in range(px-1):
21        #是否能够进行基因重组
22        if pc > np.random.rand(1)[0]:
23            cpoint = int(np.round(np.random.rand(1)[0] * py) - 1)
24            if cpoint == 0:
25                cpoint = 1
26
27            newpop[i, :][0:cpoint] = pop[i, 0:cpoint]
28            newpop[i, :][cpoint+1:py] = pop[i+1, cpoint+1:py]
29
30            newpop[i+1, :][0:cpoint] = pop[i+1, 0:cpoint]
31            newpop[i+1, :][cpoint+1:py] = pop[i, cpoint+1:py]
32
33        else:
34            newpop[i,:] = pop[i,:]
35            newpop[i+1,:] = pop[i+1,:]
36
37    return newpop

```

3.7变异算法 (基因突变)

0变1，1变0 你上你也行

```

1 def mutation(pop, pm):
2     """

```

```

3  变异算法 实现基因突变
4
5  Parameters
6  -----
7  pop : TYPE
8      种群.
9  pm : TYPE
10     变异概率.
11
12 Returns
13 -----
14 newpop : TYPE
15     新的变异种群.
16
17 """
18 px, py = pop.shape
19 newpop = np.zeros((px, py))
20
21 for i in range(px):
22     if pm > np.random.rand(1)[0]:
23         mpoint = int(np.round(np.random.rand(1)[0] * py) - 1)
24         if mpoint == 0:
25             mpoint = 1
26         newpop[i,:] = pop[i,:]
27         if newpop[i, mpoint] == 0:
28             newpop[i, mpoint] = 1
29
30     newpop[i,:] = pop[i,:]
31 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）

```

1  def best(pop, fitvalue):
2      """
3      最优的个体及其适应值
4      Parameters
5      -----
6      pop : TYPE
7          种群.
8      fitvalue : TYPE
9          适应值.
10
11 Returns
12 -----
13 bestindividual : 最大适应值
14                 DESCRIPTION.
15 bestfit : TYPE
16         最大适应值的个体.
17
18 """
19 px, py = pop.shape
20 bestindividual = pop[0,:]
21 bestfit = fitvalue[0]
22
23 for i in range(1,px):
24     if fitvalue[i] > bestfit:
25         bestindividual = pop[i,:]
26         bestfit = fitvalue[i]
27
28 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。

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  # 遗传算法
5  def initpop(popsizes, chromlength):
6      """
7      Parameters
8      -----

```

免登录复制

```
9      popsize : TYPE
10          种群数目.
11      chromlength : TYPE
12          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
13
14      Returns
15      -----
16      pop : TYPE
17          随机种群
18
19      """
20      pop=np.round(np.random.rand(popsize,chromlength))
21      return pop
22
23 def decodebinary(pop):
24     """
25     二进制数转十进制数函数
26     Parameters
27     -----
28     pop : TYPE
29         初始化种群.
30
31     Returns
32     -----
33     pop2 种群的染色体二进制数转十进制数
34
35     """
36     px,py = pop.shape
37     pop1 = np.zeros((px,py))
38     for i in range(py):
39         pop1[:,i] = (2**(py-i-1))*pop[:,i]
40     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
41
42     return pop2
43
44 def decodechrom(pop,spoint,length):
45     """
46     将二进制编码转化为十进制数
47     Parameters
48     -----
49     pop : TYPE
50         DESCRIPTION.
51     spoint : TYPE
52         染色体的起始位.
53     length : TYPE
54         DESCRIPTION.
55
56     Returns
57     -----
58     pop2 : TYPE
59         DESCRIPTION.
60
61     """
62     #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
63     #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
64     #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
65     pop1=pop[:,spoint:spoint+length]
66     pop2=decodebinary(pop1)
67
68     return pop2
69
70 def encode(bestpop):
71     """
72     解码x的值
73
74     Parameters
75     -----
76     bestpop : TYPE
77         最好的个体.
78
79     Returns
80     -----
81     x : TYPE
82         最优X的取值.
83
84     """
85
```

```

86     temp = decodebinary(pop)
87     x=temp*15/32767
88     return x
89
90
91 def calobjvalue(pop):
92     """
93     实现目标函数的计算
94     Parameters
95     -----
96     pop : TYPE
97         种群.
98
99     Returns
100    -----
101    objvalue : TYPE
102        返回目标函数值.
103
104    """
105    temp = decodechrom(pop, 0, 15)
106    x=temp*15/32767
107    objvalue = 9*np.sin(5*x)+7*np.cos(4*x)
108    objvalue = np.reshape(objvalue, (-1,1))#以行的形式输出
109    return objvalue
110
111 def calfitvalue(objvalue):
112     """
113     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
114
115     Parameters
116     -----
117     objvalue : TYPE
118         目标函数值.
119
120     Returns
121     -----
122     fitvalue : TYPE
123         个体适应值.
124
125     """
126     global Cmin
127     Cmin = 0
128     fitvalue = np.zeros((objvalue.shape[0],objvalue.shape[1]))
129     px, py = objvalue.shape
130     for i in range(px):
131         if objvalue[i,0] + Cmin > 0:
132             temp = objvalue[i,0] + Cmin
133         else:
134             temp = 0
135
136         fitvalue[i,0]=temp
137     return fitvalue
138
139 def selection(pop, fitvalue):
140     """
141     选择函数 选择复制，决定哪些个体可以进入下一代
142     采用轮盘赌选择
143
144     Parameters
145     -----
146     pop : TYPE
147         种群的个体.
148     fitvalue : TYPE
149         个体适应值.
150
151     Returns
152     -----
153     newpop : dict
154         新的种群.
155
156     """
157     totalfit = np.sum(fitvalue)
158     fitvalue_pro = fitvalue / (totalfit + 0.000001)
159     fitvalue_pro_cumsum = np.cumsum(fitvalue_pro)
160     fitvalue_pro_cumsum = np.reshape(fitvalue_pro_cumsum, (-1,1))#以行的形式
161
162

```



```

162 px, py = pop.shape
163 #轮盘随机概率 从小到大排序
164 ms = np.sort(np.random.rand(px,1),0)
165
166
167 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
168 newin = 1 - 1 #pop种群 第几代个体
169 newpop_dict = {}
170 #我愿称为[适者生存, 优胜劣汰] while循环
171 while newin <= px-1:
172     if ms[newin, 0] < fitvalue_pro_cumsum[fitin, 0]:
173         newpop_dict[newin] = pop[fitin,:]
174         newin += 1
175     else:
176         fitin += 1
177
178 newpop = np.zeros((newin,py))
179 for i in range(newin):
180     newpop[i,:] = newpop_dict[i]
181
182 return newpop
183
184
185 def crossover(pop, pc):
186     """
187     交叉算法 实现基因重组
188     Parameters
189     -----
190     pop : TYPE
191         种群.
192     pc : TYPE
193         交叉概率.
194
195     Returns
196     -----
197     newpop : TYPE
198         DESCRIPTION.
199
200     """
201     px, py = pop.shape
202     newpop = np.zeros((px, py))
203     # seletion_litst = [i for i in range(px)]
204     # sl = seletion_litst[0:px:2]
205     for i in range(px-1):
206         #是否能够进行基因重组
207         if pc > np.random.rand(1)[0]:
208             cpoint = int(np.round(np.random.rand(1)[0] * py) - 1)
209             if cpoint == 0:
210                 cpoint = 1
211
212             newpop[i, :][0:cpoint] = pop[i, 0:cpoint]
213             newpop[i, :][cpoint+1:py] = pop[i+1, cpoint+1:py]
214
215             newpop[i+1, :][0:cpoint] = pop[i+1, 0:cpoint]
216             newpop[i+1, :][cpoint+1:py] = pop[i, cpoint+1:py]
217
218         else:
219             newpop[i,:] = pop[i,:]
220             newpop[i+1,:] = pop[i+1,:]
221
222     return newpop
223
224 def mutation(pop, pm):
225     """
226     变异算法 实现基因突变
227
228     Parameters
229     -----
230     pop : TYPE
231         种群.
232     pm : TYPE
233         变异概率.
234
235     Returns
236     -----
237     newpop : TYPE
238

```

```

239     新的变异种群.
240
241     """
242     px, py = pop.shape
243     newpop = np.zeros((px, py))
244
245     for i in range(px):
246         if pm > np.random.rand(1)[0]:
247             mpoint = int(np.round(np.random.rand(1)[0] * py) - 1)
248             if mpoint == 0:
249                 mpoint = 1
250             newpop[i,:] = pop[i,:]
251             if newpop[i, mpoint] == 0:
252                 newpop[i, mpoint] = 1
253
254             newpop[i,:] = pop[i,:]
255     return newpop
256
257 def best(pop, fitvalue):
258     """
259     最优的个体及其适应值
260     Parameters
261     -----
262     pop : TYPE
263         种群.
264     fitvalue : TYPE
265         适应值.
266
267     Returns
268     -----
269     bestindividual : 最大适应值
270         DESCRIPTION.
271     bestfit : TYPE
272         最大适应值的个体.
273
274     """
275     px, py = pop.shape
276     bestindividual = pop[0,:]
277     bestfit = fitvalue[0]
278
279     for i in range(1,px):
280         if fitvalue[i] > bestfit:
281             bestindividual = pop[i,:]
282             bestfit = fitvalue[i]
283
284     return bestindividual,bestfit
285
286
287
288
289 if __name__ == "__main__":
290     popsize=30; #群体大小
291     chromlength=15; #字符串长度 (染色体的长度)
292     pc=0.7; #交叉概率
293     pm=0.005 #变异概率
294
295     #初始化种群
296     pop=initpop(popsize, chromlength)
297     poptest=pop
298     #开始迭代
299     epoch = 200
300
301     x_ = []
302     y_ = []
303
304     for i in range(epoch):
305         objvalue = calobjvalue(pop)      #计算目标函数值
306         fitvalue = calfitvalue(objvalue)  #计算群体中每个个体的适应度
307
308         newpop = selection(pop, fitvalue)
309         newpop1 = crossover(newpop, pc)
310         newpop2 = mutation(newpop1, pm)
311
312         objvalue = calobjvalue(newpop2)  #计算目标函数值
313         fitvalue = calfitvalue(objvalue)  #计算群体中每个个体的适应度
314
315

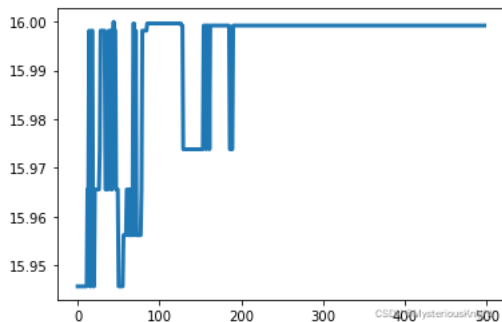
```

```

315     bestindividual,bestfit = best(newpop2, fitvalue) #求出群体中适应值最大的个体及其适应值
316     x_.append(bestfit)
317     pop = newpop2
318
319     plt.plot(x_, lw=3, label='funcotio_max_value_x')
320     plt.show()
321
322     reshape_best_infrivedual = np.reshape(bestindividual,(1,-1))#以行的形式
323     best_x_value = encode(reshape_best_infrivedual)
324     plt.plot(x_, lw=3, label='funcotio_max_value_x')
325     plt.legend()
326     plt.show()
327
328     print("最好的个体是 ",bestindividual)
    print("函数:  $f(x) = 9\sin(5x)+7\cos(4x)$  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.]

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

x的取值应为: 7.851802117984557

显示推荐内容



TwinkelStar

👍 5



🌟 28

💬 4