设计一个遗传算法来求解dejong1和dejong2在给定范围内的最小值

1. 在实现中明确指出以下步骤：

- 初始化；

  初始化了相关数据，比如：

  - 最大的迭代次数
  - 变异的概率和交叉的概率
  - 最开始种群的规模
  - 最好的适应度，所有适应度，某一代的适应度

  ```
  max_epoches = 400  # 最大的迭代次数
  cmp = 0.95  # 种群交叉的概率
  mop = 0.05  # 种群变异的概率

  fun_one_bound = [-5.12, 5.12]
  fun_two_bound = [-2.048, 2.048]

  best_fitness = []  # 每一代的最好的适应度
  all_fitness = []  # 所有代所有个体的适应度
  one_fitness = []  # 某一代所有个体的适应度

  # 初始化最开始的种群规模
  first_population = np.random.randint(low=0, high=2, size=(200, 2, 20))
  ```

- 适度值评估；

  ```
  best_fitness = []  # 每一代的最好的适应度
  all_fitness = []  # 所有代所有个体的适应度
  one_fitness = []  # 某一代所有个体的适应度
  ```

- 交叉和变异的选择；

  轮盘交叉方法。

- 环境选择。

2.请根据你自己的喜好来设置以下组件：

- 交叉、变异概率； $cmp = 0.95$
- 精英主义的比例；

  ```
  best_idx=fitness.index(np.min(fitness)) # 找到最小的那个
  best_best=parent[best_idx].copy() # 找到最好的那个值，下标，当作是精英
  ```

- 群体的大小，交叉和变异的类型；

  ```
  # 初始化最开始的种群规模
  ```

```python
first_population = np.random.randint(low=0, high=2, size=(200, 2, 20))
# 轮盘赌交叉方式
def rws_algoritm(first_population, fitness, n):  # 定义轮盘赌算法
    next_population = []  # 定义下一个子代
    sum_ = sum(fitness)  # 获取所有的适应度的和

    p_ = ((sum_-fitness)/sum_) / (len(fitness)-1) # 获得概率

    idx = np.random.choice(np.arange(len(first_population)), size=n,
replace=True, p=p_)

    for i in idx:
        next_population.append(first_population[i])
    return next_population
```

交叉：我们随机定义一个数，$np.random.rand()$ 。如果这个值比我们自己设置的 cmp （交叉编译概率）来得大，就不会发生，反之，会发生交叉变异。我们设置一个随机的长度 $random_length$ ，模拟在一个长度为 n 的染色体上随机选取一段长度 x，然后把两个染色体的这个片段进行交换，就可以得到新的子代。

```python
def cross_mutation(first_population,cmp):  # 交叉与变异
    next_population=[] # 定义下一代
    for each in first_population: # 遍历
        res=each # 子代变成父代
        x=np.random.rand()
        if x>=cmp continue
        else:
            # 随机初始化一个数 比他大不发生变化 比他小就发生交叉交换
            # 随机从first_population中生成一段，然后进行交换

 random_length=first_population[np.random.randint(0,len(first_population))]
# 另一个

            # 定义随机生成的x交叉点和y交叉点
            x_c_m_pos=np.random.randint(0,len(random_length[0]))
            y_c_m_pos=np.random.randint(0,len(random_length[1]))

            for i in range(x_c_m_pos,len(random_length[0])):
                res[0][i]=random_length[0][i] # 赋值,剪切之后交换
            for j in range(y_c_m_pos,len(random_length[1])):
                res[1][j]=random_length[1][j] # 赋值,剪切之后交换

        next_population.append(res)
    return next_population
```

3、请独立运行算法至少30次，最后报告平均值和标准值

- $maxEpoches = 400$

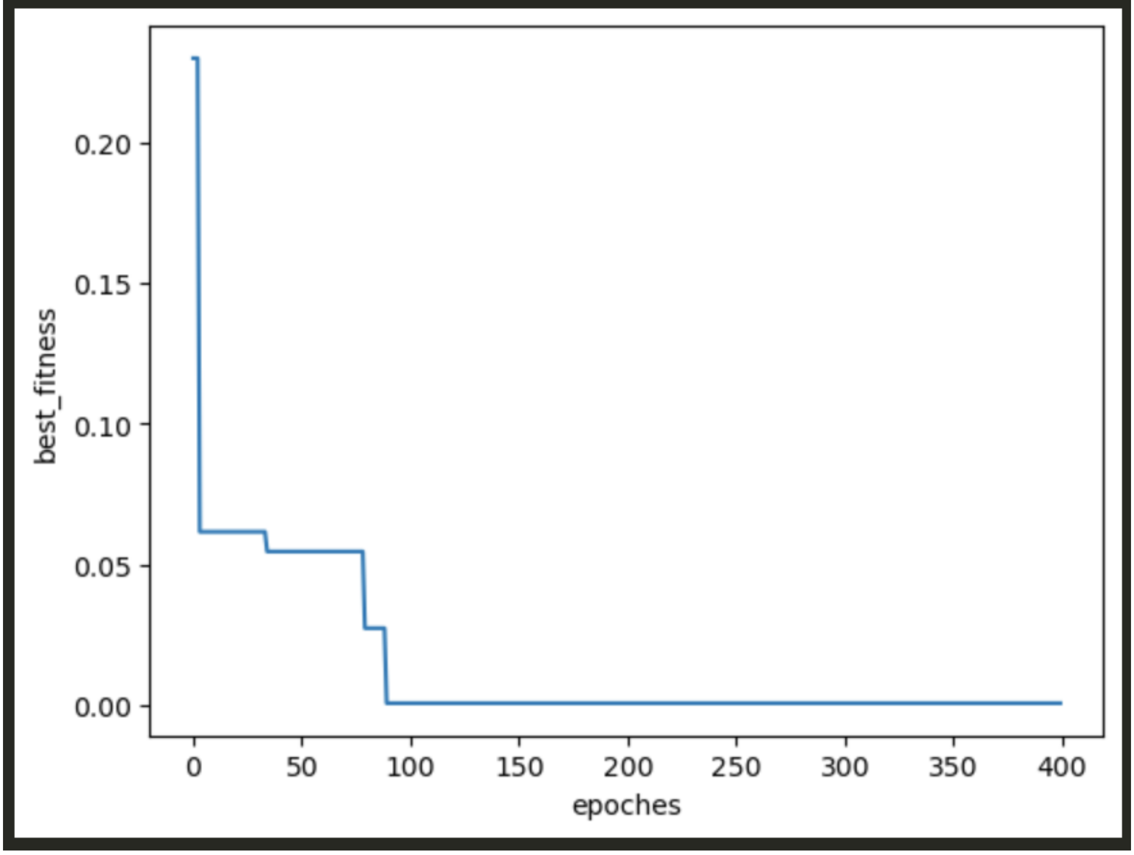
- ```
  # 画图
  print("average:{}".format(sum(best_fitness)/len(best_fitness)))
  print("max_element:{}".format(max(best_fitness)))
  print("min_element:{}".format(min(best_fitness)))

  plt.plot(np.arange(max_epoches),best_fitness)
  plt.xlabel("epoches")
  plt.ylabel("best_fitness")
  plt.show()
  ```
- ```
  average:0.013822483856634643
  max_element:0.22989880184877856
  min_element:0.0006512713933393305
  ```

4、请用任何一次运行的数据画出进化过程

- 在某次运行中收集相应的数据（代数以及相应的适度值），例如第五次运行中，适度值为多少...；
- 将数据绘制在一个二维轴上；
- 横轴表示代数，纵轴表示适度值；
- 试着从这个图中得出一些结论。

```
average:0.013822483856634643
```




```
for i in range(max_epoches):    # 最大的迭代次数

 first_population=nature_selection(function_one,first_population,cmp,mop,fun
_one_bound)
    one_fitness=get_fitness(first_population,function_one,fun_one_bound)
    all_fitness.append(one_fitness)
    best_fitness.append(np.min(one_fitness))
```

```python
print("finish")

# 画图
print("average:{}".format(sum(best_fitness)/len(best_fitness)))
print("max_element:{}".format(max(best_fitness)))
print("min_element:{}".format(min(best_fitness)))

plt.plot(np.arange(max_epoches),best_fitness)
plt.xlabel("epoches")
plt.ylabel("best_fitness")
plt.show()
```

结论：在100之前就已经迭代完毕，说明在100次之前，这个自然选择已经选出了最优解。

```
.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305,
.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305,
.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305,
.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305,
.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305,
.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305,
```

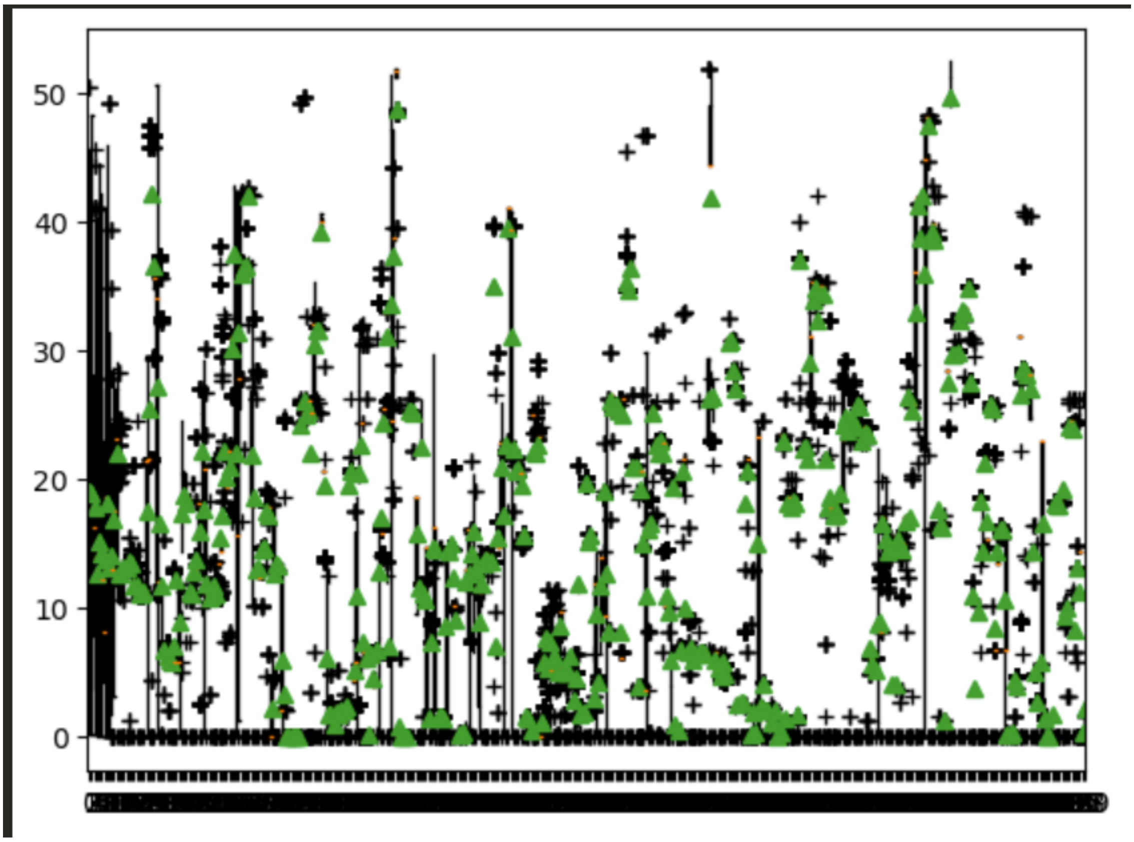
打印出所有的最好的适应度 $print(best_fitness)$

```
0.06157019557030884, 0.06157019557030884, 0.06157019557030884, 0.06157019557030884, 0.06157019557030884, 0.06157019557030884, 0.06157019557030884,
0.06157019557030884, 0.06157019557030884, 0.06157019557030884, 0.06157019557030884, 0.06157019557030884, 0.06157019557030884, 0.06157019557030884,
0.06157019557030884, 0.06157019557030884, 0.06157019557030884, 0.06157019557030884, 0.06157019557030884, 0.06157019557030884, 0.06157019557030884,
0.06157019557030884, 0.06157019557030884, 0.06157019557030884, 0.06157019557030884, 0.06157019557030884, 0.06157019557030884, 0.05455650364203947,
0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947,
0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947,
0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947,
0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947,
0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947,
0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947, 0.05455650364203947,
0.05455650364203947, 0.05455650364203947, 0.027303300720763552, 0.027303300720763552, 0.027303300720763552, 0.027303300720763552,
0.027303300720763552, 0.027303300720763552, 0.027303300720763552, 0.027303300720763552, 0.027303300720763552, 0.027303300720763552,
0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305,
0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305,
0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305,
0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305,
0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305,
0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305,
0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305,
0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305,
0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305, 0.0006512713933393305,
```

5.请画出同一代在不同运行中的平均值、最大值和最小值（boxplot）

- 收集所有运行中同一世代的适度值；
- 使用boxplot绘制从30次运行中收集的数据；
- 试着从这幅图中得出一些结论。

```python
# 画出同一代的图形
plt.boxplot(all_fitness, labels = np.arange(max_epoches),
                         sym = "+",
                         widths=0.6,     # 指定箱线图的宽度，默认为0.5；
                         patch_artist=True,     # 是否填充箱体的颜色；
                         showmeans=True
)
plt.show()
```



结论：每一代朝着最优的方向进行。用＋来表示每一次的结果。