遗传算法求解低维单目标优化问题

一、实验目的

1. 掌握遗传算法的基本原理及其实现方法
2. 了解各个超参数对遗传算法的影响
3. 掌握利用遗传算法求解单目标优化问题的方法
4. 熟悉 python 编程

二、实验环境

软件环境：python3.7
开发环境：Modelarts: pytorch1.4-cuda10.1-cudnn7-ubuntu18.04

三、待解决的问题

利用遗传算法对三个低维单目标优化问题进行求解。
三个测试函数分别为：
二维球形函数：$f_1(x,y) = x^2 + y^2 \quad x,y \in [-5.12,\ 5.12]$
De.Jong 函数：$f_2(x,y) = 100(y - x^2)^2 + (x-1)^2 \quad x,y \in [-2.048,\ 2.048]$
Six-hump cameback 函数：$f_3(x,y) = 4x^2 + 2.1x^4 + 1/3x^6 + xy - 4y^2 + 4y^4 \quad x,y \in [-5,\ 5]$

四、遗传算法

(i) 遗传算法的简介：

遗传算法是模拟达尔文的遗传选择和自然淘汰的生物进化过程的计算模型，是一种通过模拟自然进化过程搜索最优解的方法。遗传算法是从代表问题可能潜在的解集的一个种群开始的，而一个种群则由经过基因编码的一定数目的个体组成，每个个体实际上是染色体带有特征的实体。染色体作为遗传物质的主要载体，即多个基因的集合，其内部表现是某种基因组合，它决定了个体的形状的外部表现。因此，在一开始需要实现从表现型到基因型的映射，即编码工作。由于仿照基因编码的工作很复杂，我们往往进行简化，即二进制编码。初代种群产生之后，按照适者生存和优胜劣汰的原理，逐代演化产生出越来越好的近似解，在每一代，根据问题域中个体的适应度大小选择个体，并借助于自然遗传学的遗传算子进行组合交叉和变异，产生出代表新的解集的种群，这个过程将导致种群像自然进化一样，后生代种群比前代更适合于环境。末代种群中的最优个体经过解码，可以作为问题近似最优解。

(ii) 遗传算法的算法流程：

遗传算法的搜索特点是以编码空间代替问题的参数空间，以适应度函数为评估依据，将编码集作为遗传的基础，对个体进行遗传操作，建立一个迭代过程。首先，算法从一个由个体组成的种群开始，对每个种群个体进行适应度评价，其次，利用个体的适应度选择个体，并用交叉和变异等遗传算子作用其上，产生后代个体；最后，在原种群个体和后代个体中选择个体生成下一代种群。

经典遗传算法的基本步骤如下：

步骤一：将解空间的个体表示成遗传算法编码空间的基因型个体。

步骤二：定义适应度函数。适应度函数表明个体或解的优劣性。不同的问题，适应度函数定义的方式也不同。

步骤三：确定遗传策略，包括设置种群规模，确定选择，交叉，变异方法，以及确定交叉概率，变异概率等遗传参数。

步骤四：随机产生一组初始个体构成初始种群，并计算每一个个体的适应度值。

步骤五：判断算法收敛准则是否满足，若满足则输出搜索结果，否则执行以下步骤。

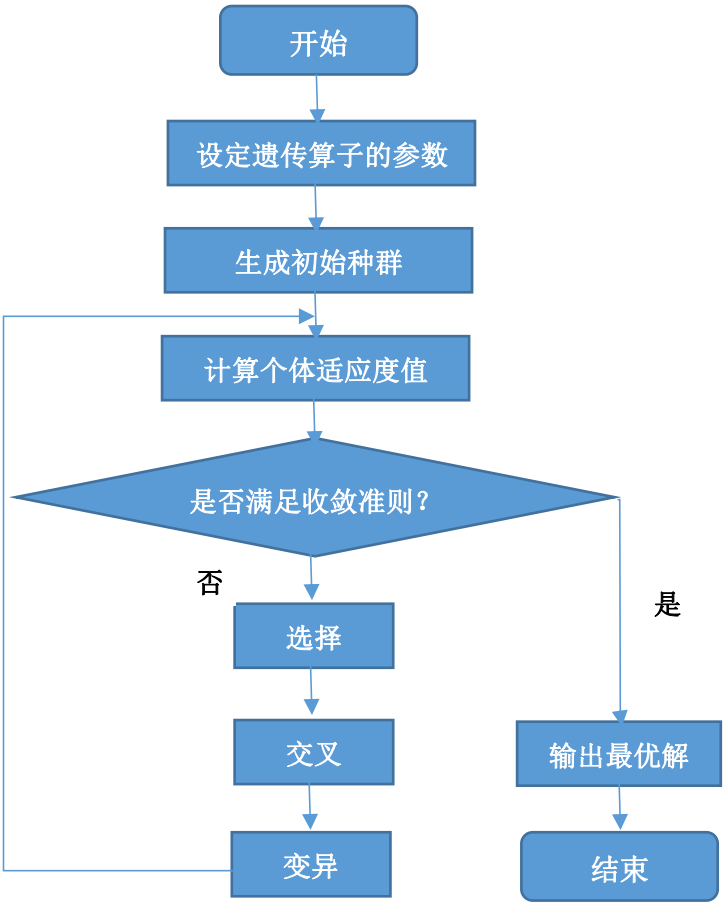
步骤六：按由个体适应度值决定的某个规则选择将进入下一代的个体。

步骤七：按交叉概率进行交叉操作。

步骤八：按变异概率进行变异操作。

步骤九：返回步骤五。

经典遗传算法的算法流程图如图所示：



五、代码实现及分析

(i)编码及遗传策略的选择

首先，我们定义编码函数，将所有 01 基因型序列映射到自变量的取值范围内。在这里，我们选取映射函数为：$x = B + \frac{B-A}{2^l}(\sum_1^l b_i 2^{i-1})$，式中，A 为自变量的左边界，B 为自变量的右边界，l 为编码长度。

之后，对适应度函数进行构造，定义适应度 $fitness = fitness - fitness_{min} + 1e - 3,$

这样可以使得所有的 $fitness$ 都为正值，便于之后利用转轮法对种群进行选择。其中，$+1e-3$ 的目的是防止 float 相除时小数点后的数被省略。

接着构造出交叉和变异函数 crossover_rate 的概率进行交叉，以为 mutation_rate 的概率进行变异。

对于选择函数，我们选择轮盘赌选择的方法，其基本原理是:首先计算种群中个体的适应度值，然后计算该个体的适应度在该种群中所占的比例，该比例九尾该个体的选择概率。种群中个体x_i选择概率如下：$p_{x_i}=\frac{f(x_i)}{\sum_{j=1}^{N}f(x_i)}$

(ii)超参数的选择

超参数对算法性能有着较大的影响，经过多次试验后，得出以下结论：

dna 长度 dna_size： dna 长度直接影响最终求解结果的精度，dna 过短，会使得算法精度较差，但是 dna 过长会造成计算代价较大，计算缓慢。

种群数目 pop_size：群体规模太小，会出现近亲交配，产生病态基因。而且会造成有效等位基因先天缺失，即使采用较大概率的变异算子，生成优良基因的概率依然很小，且大概率变异算子对现有基因的破坏作用较大。群体规模太大，结果难以收敛且浪费资源，稳健性下降。

变异概率 mutation_rate：

变异概率太小，种群的多样性下降太快，容易导致有效基因的迅速丢失且不容易修补。

变异概率太大，尽管种群的多样性得到保证，但是优良基本被破坏的概率也随之增大。

交叉概率 crossover_rate：

与变异概率类似，交叉概率太大容易破坏已有的有利模式，随机性增大，容易错失最优个体；交叉概率太小不能有效更新种群。

进化代数 N_epoch：

进化代数太小，算法不容易收敛，种群还没有成熟；

进化代数太大，算法已经熟练或者种群过于早熟不可能再收敛，继续进化没有意义，只会增加时间开支和资源浪费。

经过多次试验和调参后，最终超参数选择如下：

pop_size = 400

dna_size = 24

crossover_rate = 0.8

mutation_rate = 0.005

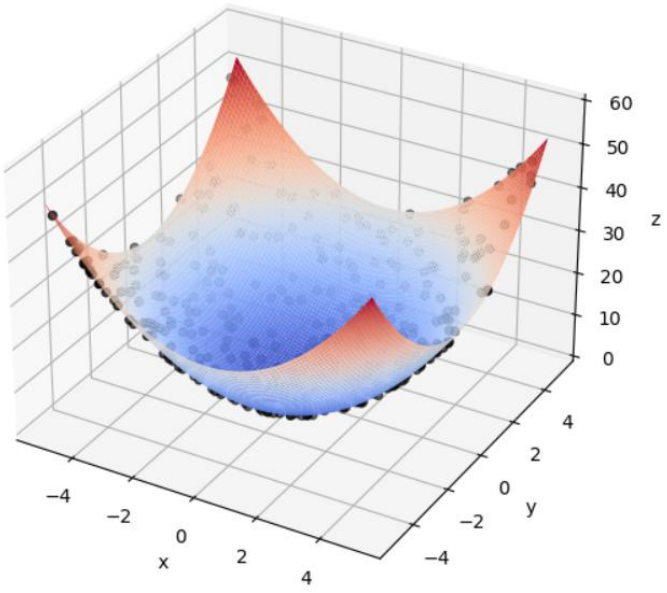
N_epoch = 50

(iii)函数优化结果

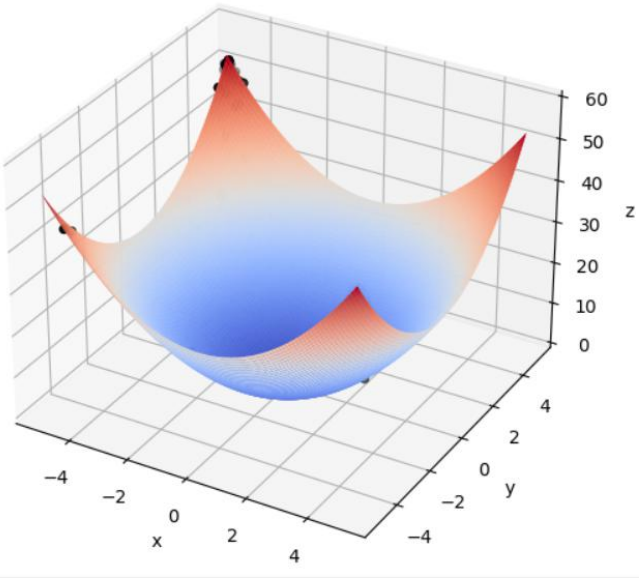
在该参数选择下，对三个函数进行优化：

1. 二维球形函数：$f_1(x,y)=x^2+y^2 \quad x,y\in[-5.12,\ 5.12]$

初始时种群分布：



迭代结束时种群分布：



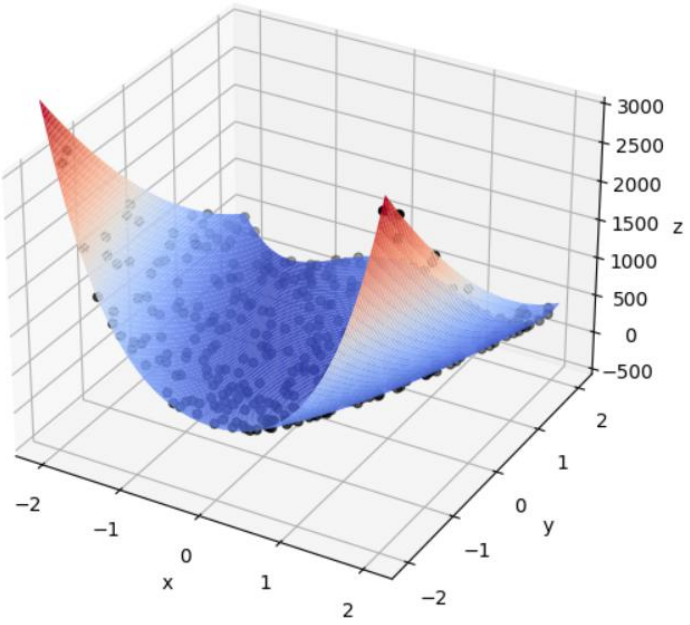
所得最优的基因型： [1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1 1 0 1 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 1 0 1 1]

(x, y): (5.116293945091601, -5.079279172377537)

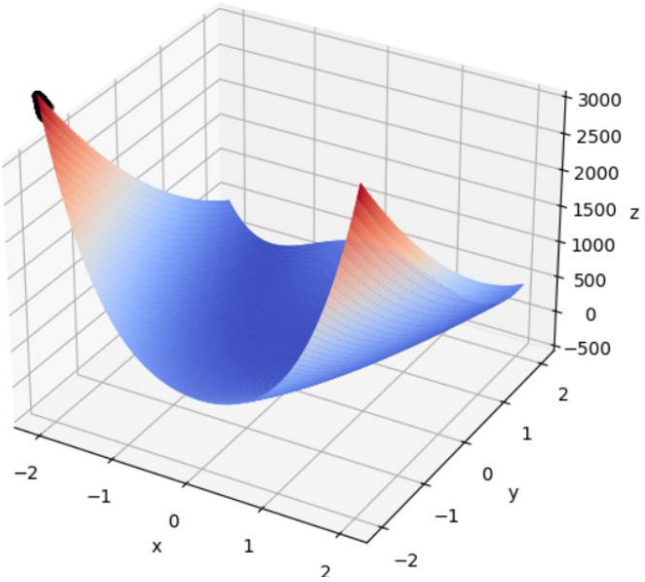
(x, y)对应的函数值: 51.975540643529214

2. De.Jong 函数：$f_2(x,y) = 100(y - x^2)^2 + (x - 1)^2 \quad x,y \in [-2.048,\ 2.048]$

初始时种群分布：



迭代结束时种群分布：



所得最优的基因型： [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 1 1 1 0 1 1 0 1
0 0 0 1 1 1 0 1 1 1 1]

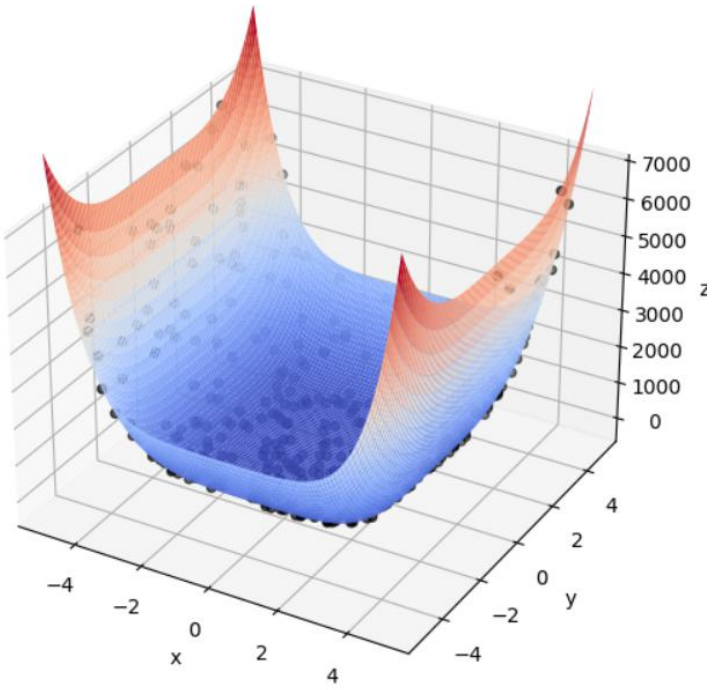
(x, y): (-2.0469104003256797, -2.03777856384388)

(x, y)对应的函数值: 3887.609684573708

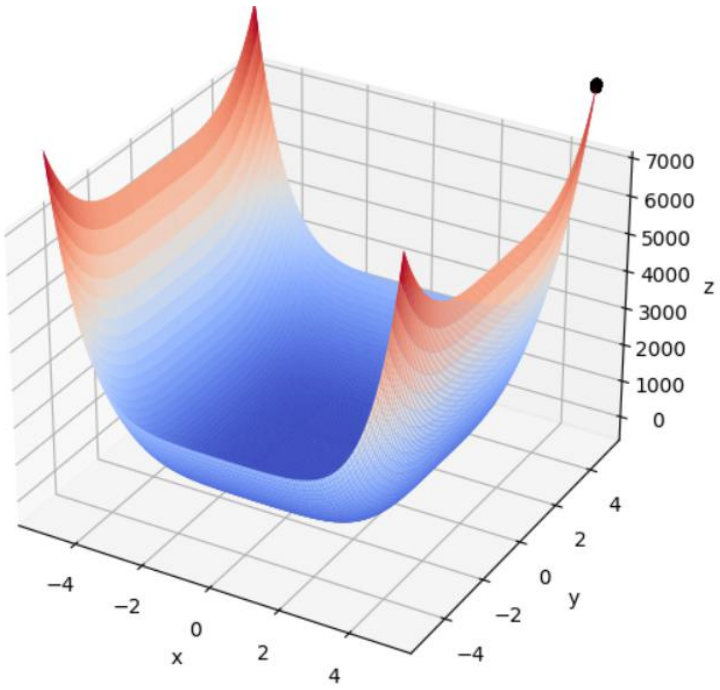
3. Six-hump cameback 函数：$f_3(x,y) = 4x^2 + 2.1x^4 + \frac{1}{3x}^6 + xy - 4y^2 + 4y^4$

$x,y \in [-5, 5]$

初始时种群分布：



迭代结束时种群分布：



所得最优的基因型： [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 0]

(x, y): (4.999230503990084, 4.998763203547192)

(x, y)对应的函数值: 9037.754031468361

六、实验收获

在本次实验中，我掌握了遗产算法的基本原理及其实现方法，并通过遗传算法完成了对三个低维单目标函数的优化。同时，通过查阅相关资料并进行多次试验，我了解了各个超参数对遗传算法的影响，对遗传算法有了更深一步的理解。

七、附录

```
#author:fkw
#creat time:2021//11/30
#coding:utf-8
#version: V1.0
'''''
可以求解任何纬度的函数，函数的维度=dna_num
x_bound 为自变量的取值矩阵
纬度不同时需要对 fitness 函数进行更改
'''
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt
#超参数
dna_size = 31 #dna 长度
pop_size = 400 #种群个体数
dna_num=2 #dna 条数
crossover_rate=0.8 #交叉互换强度
mutation_rate=0.005 #变异强度
N_epoch = 50 #迭代次数
x_bound = [[-5.12, 5.12],[-5.12, 5.12]]#自变量的取值范围，矩阵形式
size=(dna_size*2)
#定义函数
#二维球状函数
def F(x,y):
    return x**2+y**2
    #适应函数
#得到(pop_size,1)的适应度矩阵
#因为如果直接返回 pred 可能是负值，而我们在计算概率的时候不能为负值。
#要进行处理，np.min 表示取最小，为最大的负数，可以使全部只变成正的
# 1e-3 为了让 float 进行相除防止小数点后的数被省略
def get_fitness(pop):
    pop=transDNA(pop)
    args=[]
    for i in range(dna_num):
        args.append(pop[:,i])
    fitness = F(*args)
    fitness=(fitness - np.min(fitness)) + 1e-3
    return fitness
#DNA 编码转换
#DNA 用 pop 矩阵存储，数值为布尔值，pop.shape=(pop_size,dna_size)
#在实际问题中 pop.shape=(pop_size,dna_size*dna_num)
```

```
#将 pop 矩阵映射到变量的限定范围
#pop = np.random.randint(2, size=(pop_size,dna_size*dna_num))
def transDNA(pop):
    ans=np.zeros((pop_size,dna_num))
    #计算 dna 对应的数值,(pop_size,dna_size*dna_num)--->(pop_size,dna_num)
    for i in range(dna_num):
        ans[:,i]=pop[:,i::dna_num].dot(2**np.arange(dna_size)[::-1])*(x_bound[i][1]-x_bound[i][0])/(2**dna_size-1)+x_bound[i][0]
        #print(ans)
    return ans  #ans.shape=(pop_size,dna_num)
    #基因突变
def mutation(child, mutation_rate=mutation_rate):
    #以 mutation_rate 的概率进行变异
    if np.random.rand() < mutation_rate:
        #随机产生一个实数，代表要变异基因的位置
        mutate_point = np.random.randint(0, dna_size*dna_num)
        #将变异点的二进制为反转
        child[mutate_point] = child[mutate_point]^1
    return child
    #交叉互换
def crossover_and_mutation(pop, crossover_rate= crossover_rate):
    new_pop = []
    #遍历种群中的每一个个体，将该个体作为父亲
    for father in pop:
        #孩子先得到父亲的全部基因
        child = father
        #一定概率发生交叉
        if np.random.rand() < crossover_rate:
            #随机选择选择另一个个体作为母亲
            mother = pop[np.random.randint(pop_size)]
            #随机产生交叉的点
            cross_points = np.random.randint(low=0, high=dna_size*2)
            #孩子得到位于交叉点后的母亲的基因
            child[cross_points:] = mother[cross_points:]
        #每个后代有一定的机率发生变异
        mutation(child)
        new_pop.append(child)
    return new_pop
    #自然选择，使用转轮法
def select(pop, fitness):    # nature selection wrt pop's fitness
    idx = np.random.choice(np.arange(pop_size), size=pop_size, replace=True, p=(fitness)/(fitness.sum()))
    return pop[idx]
    #输出最优值
```

```
def print_info(pop):
    fitness = get_fitness(pop)
    max_fitness_index = np.argmax(fitness)
    ans = transDNA(pop)
    x=ans[:,0]
    y=ans[:,1]
    print("最优的基因型：", pop[max_fitness_index])
    print("(x, y):", (x[max_fitness_index], y[max_fitness_index]))
    print("(x, y)对应的函数值:", F(x[max_fitness_index], y[max_fitness_index]))
    #绘图，只限于二维
def plot_3d(ax):

    X = np.linspace(*x_bound[0], 100)
    Y = np.linspace(*x_bound[1], 100)
    X,Y = np.meshgrid(X, Y)
    Z = F(X, Y)
     ax.plot_surface(X,Y,Z,rstride=1,cstride=1,cmap=cm.coolwarm)
     ax.set_zlim(0,60)
     ax.set_xlabel('x')
     ax.set_ylabel('y')
     ax.set_zlabel('z')
     plt.pause(1)
     plt.show()
if __name__ == "__main__":
    fig = plt.figure()
    ax = Axes3D(fig)
    #将画图模式改为交互模式，程序遇到 plt.show 不会暂停，而是继续执行
    plt.ion()
    plot_3d(ax)

    pop = np.random.randint(2, size=(pop_size, dna_size*dna_num))
    #迭代 N 代
    for _ in range(N_epoch):
        ans = transDNA(pop)
        x=ans[:,0]
        y=ans[:,1]
        if 'sca' in locals():
            sca.remove()
        sca = ax.scatter(x, y, F(x,y), c='black', marker='o');plt.show();plt.pause(0.1)
        pop = np.array(crossover_and_mutation(pop, crossover_rate))
        #F_values = F(translateDNA(pop)[0], translateDNA(pop)[1])#x, y --> Z matrix
```

```
125.         fitness = get_fitness(pop)
126.         pop = select(pop, fitness) #选择生成新的种群
127.
128.     print_info(pop)
129.     plt.ioff()
130.     plot_3d(ax)
```