(GA) 計畫書

資工三甲 1102924 李名智

一、計畫目標：產生初始群集並利用 Ackley function 將此群集基因有較好的品質，Fitness 越小則越佳。

二、計畫內容：利用隨機變數來產生基因，並將基因(Chromosome)存至群集(Poplution)中，利用 Ackley function 當作 fitness，其中值越小代表基因具有較好的品質，選擇方式利用菁英挑選法(預設 3 菁英)，挑選前幾名 fitness 較好的菁英基因來做交配及突變，交配方式利用單點交配(設定交配率為 0.9)，利用隨機變數來決定是否交配，突變利用單點突變(突變率為 0.2)，亦是利用隨機變數來決定是否突變。另外全部數量設為 30，維度設為 5，菁英及 bit 數個別設為 3 及 4，希望選擇方法 1000 次迭代後，最佳 fitness 值為 0，且該最佳基因為[0,0,0,0,0]。

三、計畫執行之方法與步驟： 利用 python 來實行，步驟設置個流程之函式，生成基因並放置族群，設置 fitness function(本計畫使用 Ackley function)，選方式用挑選前幾名 fitness 較好菁英挑選法，交配利用單點交配(交配率 0.9)，突變利用單點突變(突變率 0.2)，主程式利用以上函式找出最佳值。

四、執行程式碼與結果

Code

```
        import numpy as np
        import random
        import math
        import matplotlib.pyplot as plt
        from tqdm.notebook import tqdm
        import warnings
        np.set_printoptions(suppress=True)
        %matplotlib inline
        warnings.filterwarnings("ignore")
        class GA:
            def __init__(self,Num=30,Dimension=5,Bitnum=4,Elite_num=3,Cr
ossoverR=0.9,MutationR=0.2,Max_Iteration=1000):
```

```
12.                 self.N=Num  #數量
13.                 self.D=Dimension
14.                 self.B=Bitnum
15.                 self.n=Elite_num #菁英
16.                 self.cr=CrossoverR #交配率
17.                 self.mr=MutationR #突變率
18.                 self.max_iter=Max_Iteration
19.             def generatePopulation(self):
20.                 population=[] #族群設定空的
21.                 for number in range(self.N):
22.                     chrom_list=[] #基因設為空的
23.                     for run in range(self.D):
24.                         elemt=(np.zeros((1,self.B))).astype(int)
25.                         for i in range(1):
26.                             for j in range(self.B):
27.                                 elemt[i,j]=np.random.randint(0,2) #0,1 隨機
    編排
28.                         Chromosome=list(elemt[0])
29.                         chrom_list.append(Chromosome) #放入列表中
30.                     population.append(chrom_list) #再將基因放入族群
31.                 return population #並回傳族群
32.             def BitToDec(self,pop):
33.                 dec=str(pop[0])+str(pop[1])+str(pop[2])+str(pop[3]) #因 b
    itnum 設為 4
34.                 return int(str(dec),2)
```

```
35.             def DecToBit(self,num):
36.                 return [int(i) for i in (bin(10)[2:])]
37.             # Ackley function
38.             def fun(self,pop):
39.                 X=np.array(pop)
40.                 fun_sum=0
41.                 sum1=0
42.                 sum2=0
43.                 term1=0
44.                 term2=0
45.                 for i in range(self.D):
46.                     x=X[:,i]
47.                     sum1+=x**2
48.                     sum2+=np.cos(2*np.pi*x)
49.                 term1=-20*np.exp(-0.2*np.sqrt(sum1/self.D))
50.                 term2=-np.exp(sum2/self.D)
51.                 fun_sum=term1+term2+20+np.exp(1)
52.                 return list(fun_sum)
53.             # 選擇 菁英挑選法
54.             def Selection(self,n,pop_bin,fitness):
55.                 select_bin=pop_bin.copy() #選擇的 bin
56.                 fitness1=fitness.copy()
57.                 Parents=[]
58.                 #最佳的情形 直接選入 parent
59.                 if sum(fitness1)==0:
60.                     for i in range(self.n):
61.                         parent=select_bin[random.randint(0,(self.N)-1)]
62.                         Parents.append(parent)
63.                 else:
64.                     for i in range(4):
65.                         #挑選之中較佳的
66.                         arr = fitness1.index(min(fitness1))
67.                         Parents.append(select_bin[arr])
68.                         del select_bin[arr]
```

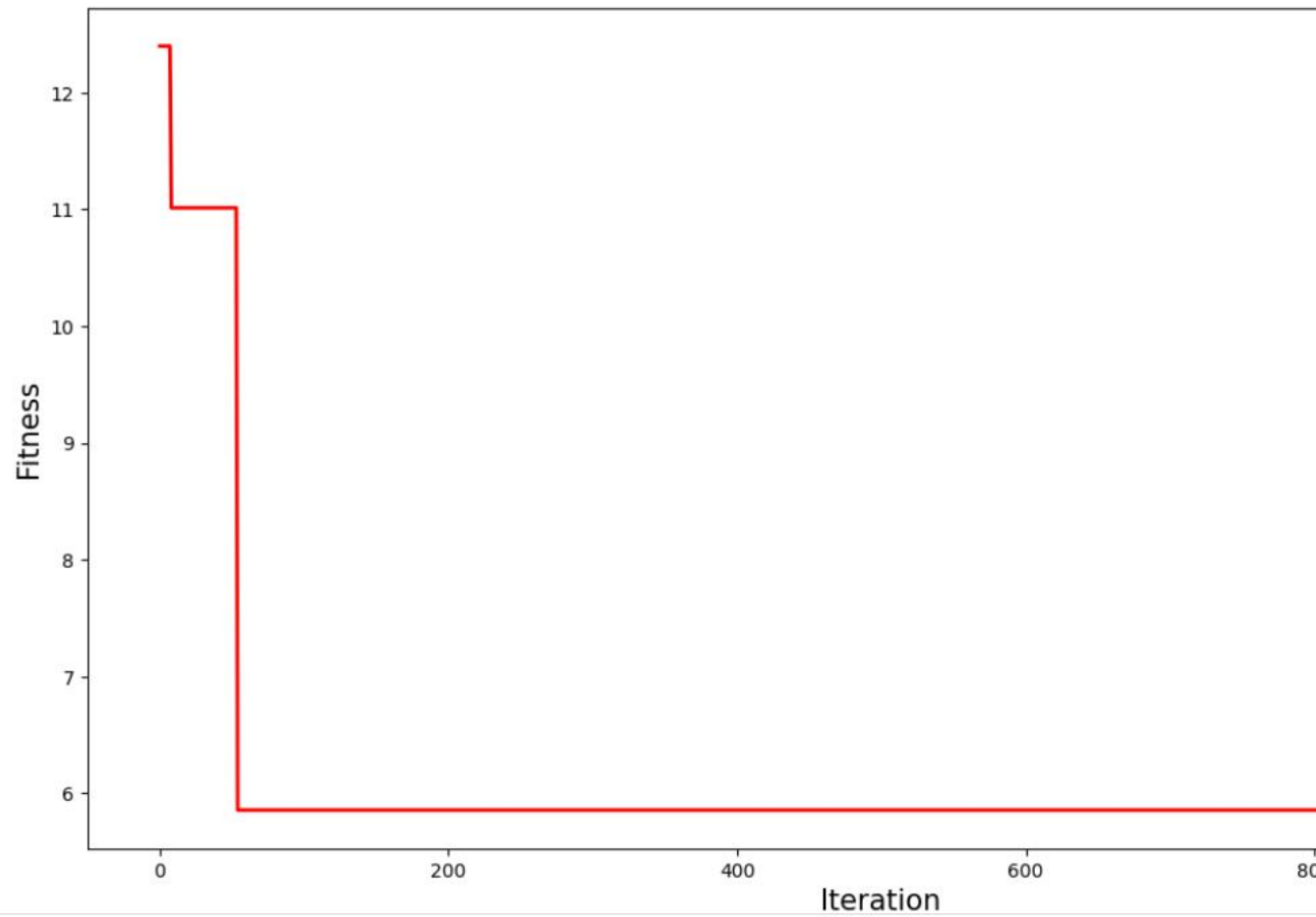
```
                        del fitness1[arr]
                return Parents
            # 交配突變
            def Crossover_Mutation(self,parent1,parent2):
                #將兩元素交換位置(單點交配)
                def swap_machine(elemt1,elemt2):
                    temp=elemt1
                    elemt1=elemt2
                    elemt2=temp
                    return elemt1,elemt2
                child1=[]
                child2=[]
                for i in range(len(parent1)):
                    #0~1 之間利用 random 隨機生成數字，決定是否交配(跟交配率比)
                    z1=random.uniform(0,1)
                    if z1<self.cr:
                        z2 =random.uniform(0,1)
                        #決定要交換的位置點
                        cross_location=math.ceil(z2*(len(parent1[i])-1))
                        #進行交配
                        parent1[i][:cross_location],parent2[i][:cross_loc
ation]=swap_machine(parent1[i][:cross_location],parent2[i][:cross_locat
ion])
                        p_list=[parent1[i],parent2[i]]
                        #隨機生成一數字，用以決定是否進行 mutation
                        for i in range(len(p_list)):
                            z3=random.uniform(0,1)
                            if z3<self.mr:
                                #決定要 mutate 的數字
                                z4=random.uniform(0,1)
                                temp_location=z4*(len(p_list[i])-1)
```

```
                            mutation_location=0 if temp_location < 0.5 else math.ceil(temp_location)
                            p_list[i][mutation_location]=0 if p_list[i][mutation_location] == 1 else 1
                        child1.append(p_list[0])
                        child2.append(p_list[1])
                    else:
                        child1.append(parent1[i])
                        child2.append(parent2[i])
                return child1,child2
        def main():
            ga=GA()
            print("數量:",ga.N)
            print("維度:",ga.D)
            print("bit 數:",ga.B)
            pop_bin=ga.generatePopulation()
            pop_dec=[]
            for i in range(ga.N):
                chrom_rv=[]
                for j in range(ga.D):
                    chrom_rv.append(ga.BitToDec(pop_bin[i][j])) #轉十進位
                pop_dec.append(chrom_rv) #放入群集中
            fitness=ga.fun(pop_dec) #計算 fitness 值
            best_fitness=min(fitness)
            arr=fitness.index(best_fitness)
            best_dec=pop_dec[arr]
            best_rvlist=[]
            best_valuelist=[]
            it=0
            while it<ga.max_iter:
                Parents_list=ga.Selection(ga.n,pop_bin,fitness)#菁英挑選
```

```
127.                Offspring_list=[] #子代設定
128.                for i in range(int((ga.N-ga.n)/2)):
129.                    candidate=[Parents_list[random.randint(0,len(Parents_
    list)-1)] for i in range(2)]
130.                    after_cr_mu=ga.Crossover_Mutation(candidate[0], candi
    date[1])
131.                    offspring1,offspring2=after_cr_mu[0],after_cr_mu[1]
132.                    Offspring_list.append(offspring1)
133.                    Offspring_list.append(offspring2)
134.                final_bin=Parents_list+Offspring_list
135.                final_dec=[]
136.                for i in range(ga.N):
137.                    rv=[]
138.                    for j in range(ga.D):
139.                        rv.append(ga.BitToDec(final_bin[i][j]))
140.                    final_dec.append(rv)
141.                #fitness 值
142.                final_fitness=ga.fun(final_dec)
143.                #拿取迭代中最佳的值(越小則越佳，故使用 min)
144.                smallest_fitness=min(final_fitness)
145.                index=final_fitness.index(smallest_fitness)
146.                smallest_dec=final_dec[index]
147.                #儲存最佳 fitness 至列表
148.                best_rvlist.append(smallest_dec)
149.                best_valuelist.append(smallest_fitness)
150.                #參數回到初始值
151.                pop_bin=final_bin
152.                pop_dec=final_dec
153.                fitness=final_fitness
154.                it += 1
155.            #儲存最佳解
156.            every_best_value=[]
```

```
157.            every_best_value.append(best_valuelist[0])
158.            for i in range(ga.max_iter-1):
159.                if every_best_value[i]>=best_valuelist[i+1]:
160.                    every_best_value.append(best_valuelist[i+1])
161.                elif every_best_value[i]<=best_valuelist[i+1]:
162.                    every_best_value.append(every_best_value[i])
163.            print('The best fitness: ',min(best_valuelist))
164.            best_index=best_valuelist.index(min(best_valuelist))
165.            print('Setup list is: ')
166.            print(best_rvlist[best_index])
167.            #圖形
168.            plt.figure(figsize=(15,8))
169.            plt.xlabel("Iteration",fontsize=15)
170.            plt.ylabel("Fitness",fontsize=15)
171.            plt.plot(every_best_value,linewidth=2,label="Best fitness co
    nvergence",color='r')
172.            plt.legend()
173.            plt.show()
174.        if __name__ == '__main__':
175.            main()
```

```
數量: 30
維度: 5
bit數: 4
The best fitness:  5.855552955621501
Setup list is:
[0, 2, 3, 1, 1]
```



五、效能比較

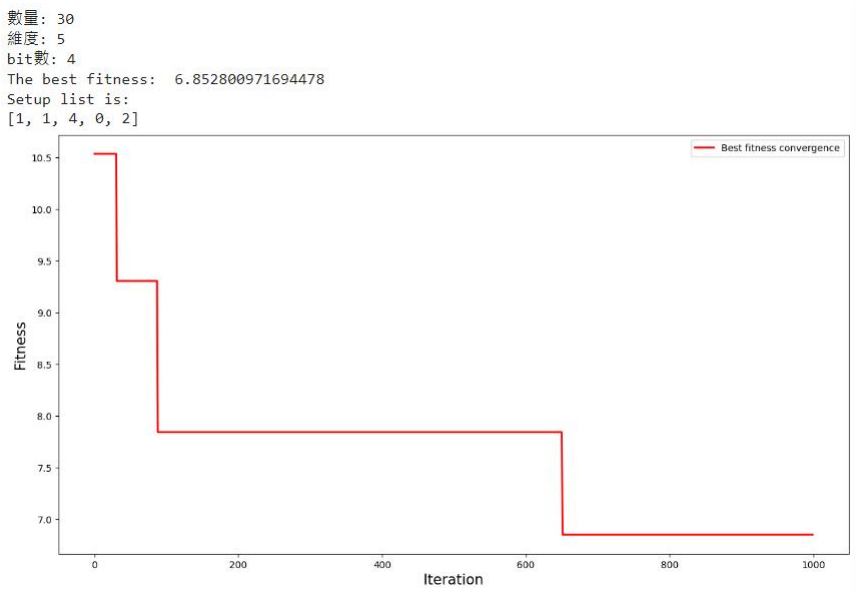
(1)若所有條件都相同情況下，挑選法更改為輪盤式的菁英挑選法，以下程式碼為輪盤式

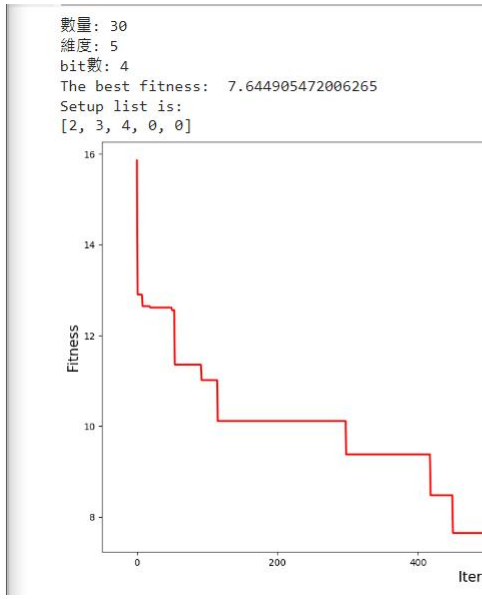
```
1.              # 選擇
2.              def  Selection(self,n,pop_bin,fitness):
3.                  select_bin=pop_bin.copy()
4.                  fitness1=fitness.copy()
5.                  Parents=[]
```

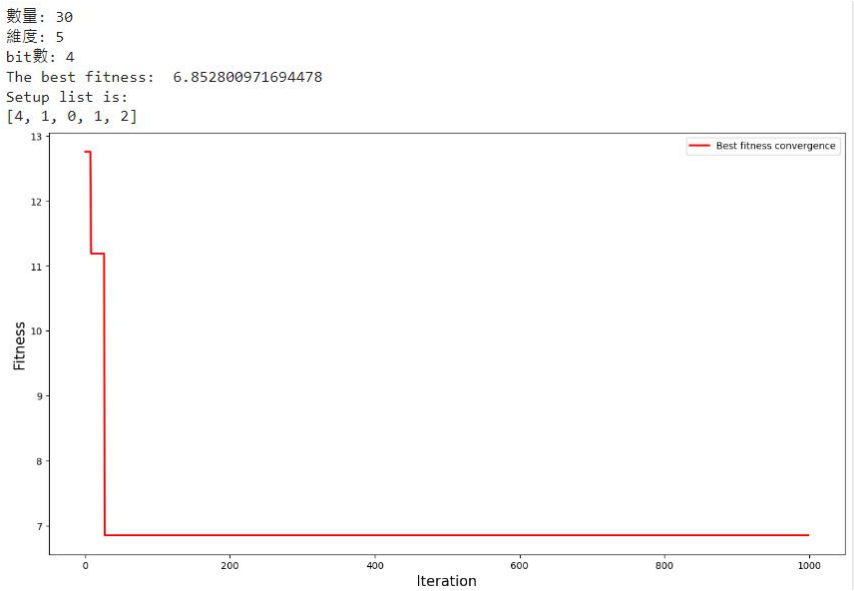
```
6.                  if sum(fitness1)==0:
7.                      for i in range(self.n):
8.                          parent=select_bin[random.randint(0,(self.N)-1)]
9.                          Parents.append(parent)
10.                 else:
11.                     NorParent=[(1-indivi/sum(fitness1))/((self.N-1)) for indivi in
    fitness1]
12.                     tep=0
13.                     Cumulist=[]
14.                     for i in range(len(NorParent)):
15.                         tep+=NorParent[i]
16.                         Cumulist.append(tep)
17.                     #找父親
18.                     for i in range(self.n):
19.                         z1=random.uniform(0,1)
20.                         for pick in range(len(Cumulist)):
21.                             if z1<=Cumulist[0]:
22.                                 parent=select_bin[NorParent.index(NorParent
    [0])]
```

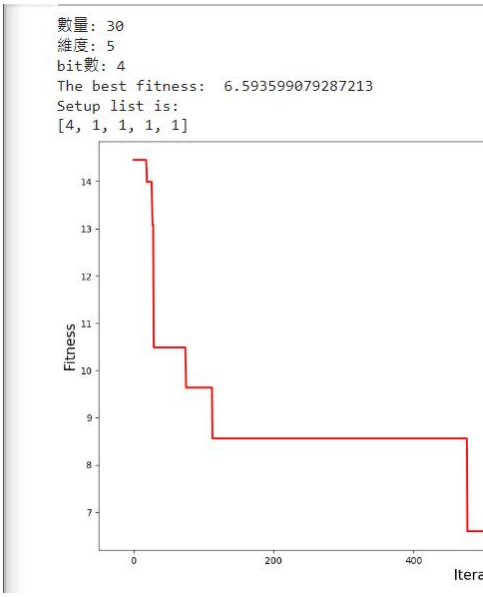
```
                elif  Cumulist[pick]<  z1  <=Cumulist[pick+1]:
                    parent=select_bin[NorParent.index(NorParent[pick+1])]
                Parents.append(parent)
        return  Parents
```

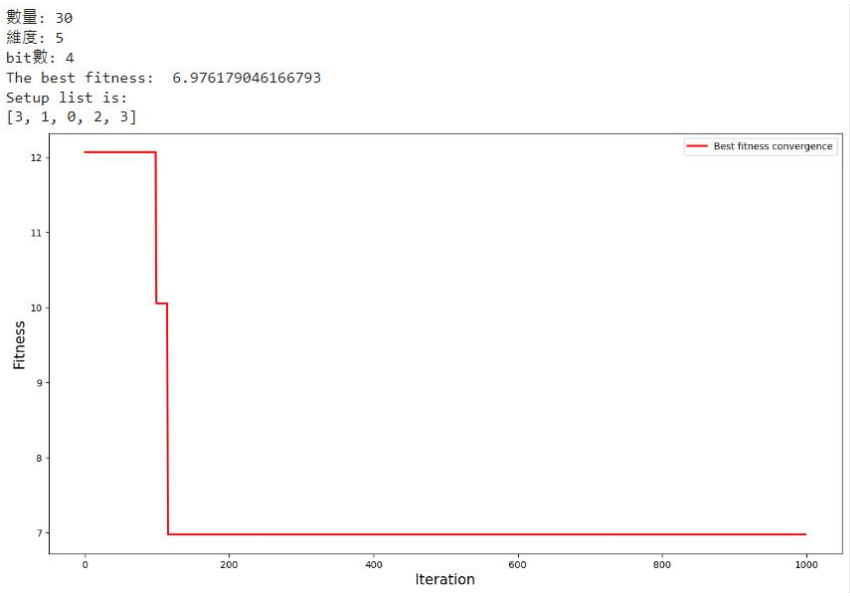
左為選前幾個 fitness 好的為精英，右為用輪盤法來挑選菁英

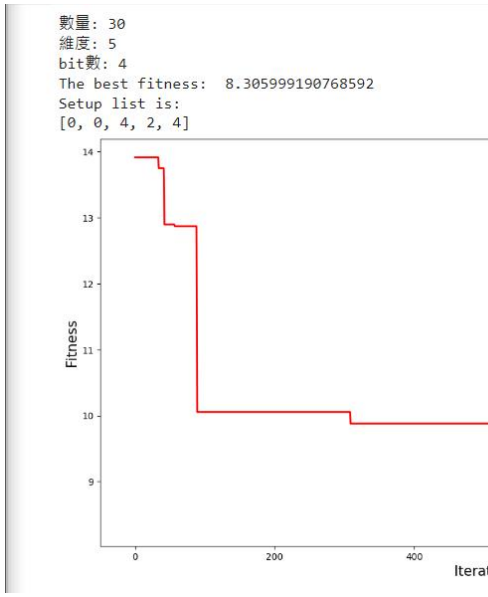










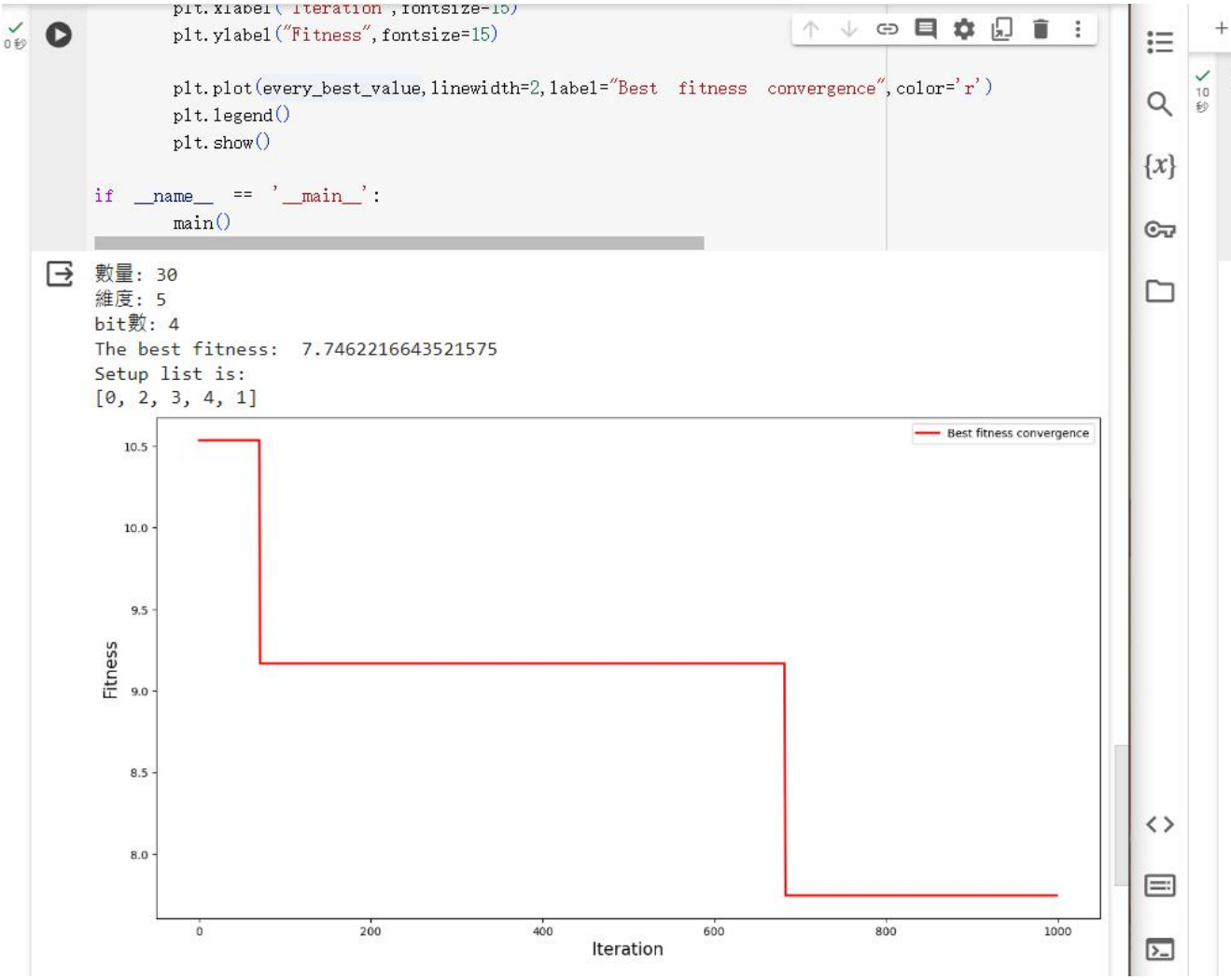


經過數次執行的結果而知，兩方法的挑選方式差不多。

(2)改變最大迭代次數(1000vs10000)

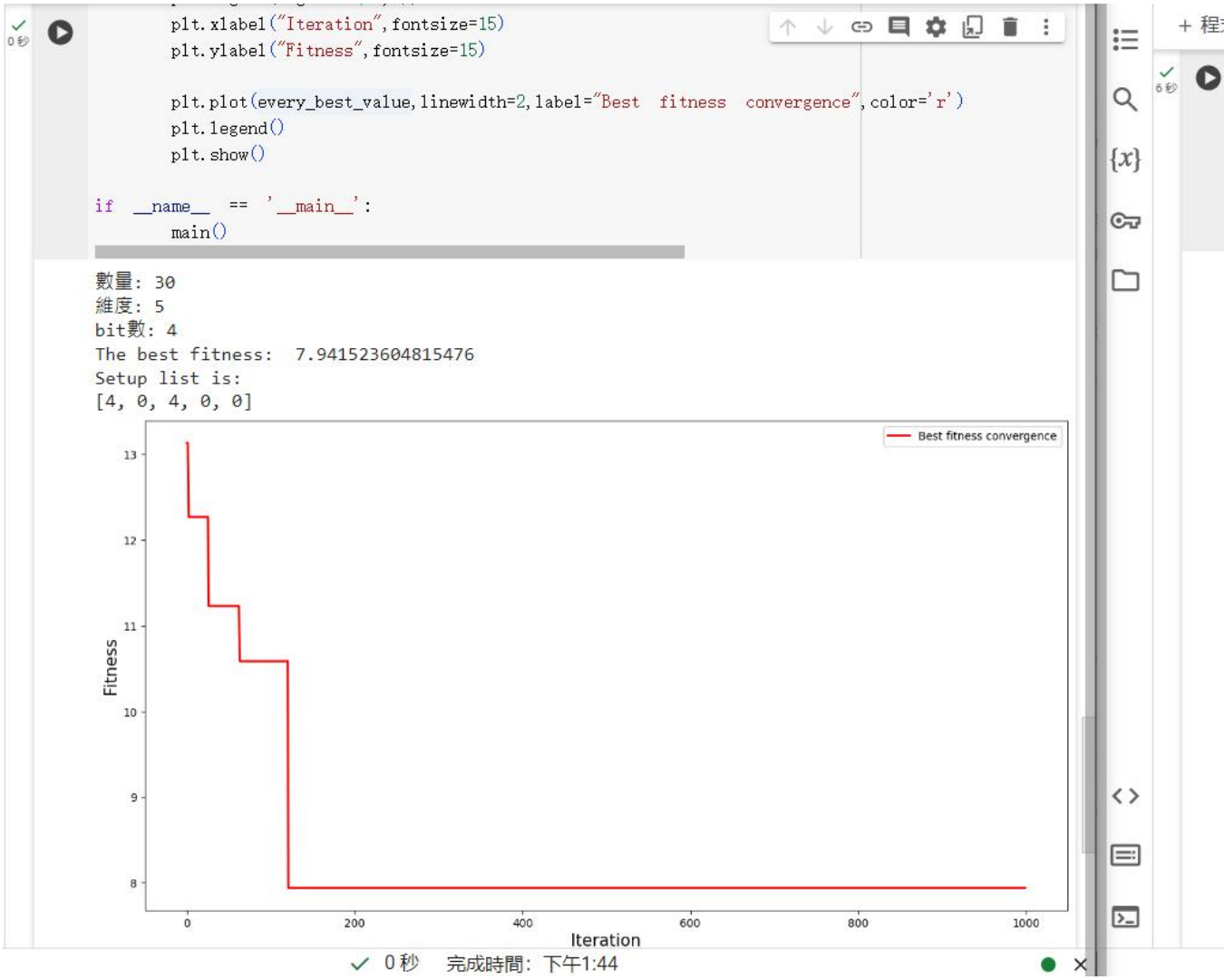
Test1:左為迭代 1000(共花 0 秒，最佳值約 7.7)，右為迭代 10000(共花 10 秒，最佳值約 2.4)

```
def __init__(self,Num=30,Dimension=5,Bitnum=4,Elite_num=3,CrossoverR=0.9,MutationR=0.2,Max_Iteration=1000):
```



Test2:左為迭代 1000(共花 1 秒，最佳值約 4.9)，右為迭代 10000(共花 7 秒，最佳值約 4.2)

Test3:左為迭代 1000(共花 0 秒，最佳值約 7.9)，右為迭代 10000(共花 6 秒，最佳值約 3.9)



結論，迭代多次花的時間比較久但可以找到更好的最佳值。

https://colab.research.google.com/drive/16Ckip0VgOJOJTlA2WKW8NmeVW2f99qFN?usp=sharing