# 计算智能实验报告

|  |  |
| --- | --- |
| 班级: | 软工2203 |
| 学号: | 221310332 |
| 姓名: | 周立成 |
| 老师: | 肖黎丽 |

[计算智能实验报告 1](#_Toc151400546)

[问题1 3](#_Toc151400547)

[问题描述: 3](#_Toc151400548)

[结构设计: 3](#_Toc151400549)

[整体代码: 4](#_Toc151400550)

[运行结果与分析 6](#_Toc151400551)

[问题2 8](#_Toc151400552)

[问题描述: 8](#_Toc151400553)

[结构设计: 8](#_Toc151400554)

[整体代码： 11](#_Toc151400555)

[运行结果和分析 11](#_Toc151400556)

[问题3 12](#_Toc151400557)

[问题描述: 12](#_Toc151400558)

[结构设计: 12](#_Toc151400559)

[整体代码: 13](#_Toc151400560)

[运行结果和分析 14](#_Toc151400561)

## 问题1

### 问题描述:

* 建立单层感知器,实现对坐标点的二分类模式实现,感知器主要由两个输入节点和一个输出节点组成:

|  |  |  |  |
| --- | --- | --- | --- |
| 点序号 |  |  | 所属类型标签 |
| 1 | -9 | 15 | 0 |
| 2 | 1 | 8 | 1 |
| 3 | -12 | 4 | 0 |
| 4 | -4 | 5 | 0 |
| 5 | 0 | 11 | 0 |
| 6 | 5 | 9 | 1 |

要求:利用训练好的感知器,识别下列点:(2,-5),(-10,10),(0,5),(-6,6)分别属于哪一类?

### 结构设计:

* 类比书上的结构,构建网络,其中,这里,(当然也可以设计成任何有意义的实数表示截距)故涉及到向量的加减法,需要先对pair类型重载运算符：

1. //重载向量加法:
2. pair<**double**, **double**> operator+(pair<**double**, **double**> d1, pair<**double**, **double**> d2) {
3. **return** make\_pair(d1.first + d2.first, d1.second + d2.second);
4. }
5. //重载数乘:
6. pair<**double**, **double**> operator\*(**double** k, pair<**double**, **double**> d1) {
7. **return** {k \* d1.first, k \* d1.second};
8. }
9. pair<**double**, **double**> operator\*(**int** k, pair<**double**, **double**> d1) {
10. **return** {k \* d1.first, k \* d1.second};
11. }
12. **double** operator\*(pair<**double**, **double**> d1, pair<**double**, **double**> d2) {
13. **return** d1.first \* d2.first + d1.second \* d2.second;
14. }
15. //减法
16. pair<**double**, **double**> operator-(pair<**double**, **double**> d1, pair<**double**, **double**> d2) {
17. **return** {d1.first - d2.first, d1.second - d2.second};
18. }

* 定义训练集和测试集,其中训练集需要有分类标签属性,测试集则不需要:

1. **struct** train\_set {
2. pair<**double**, **double**> X\_1\_X\_2;
3. **int** y;
4. train\_set(**const** pair<**double**, **double**> &x1X2, **int** y) : X\_1\_X\_2(x1X2), y(y) {}
5. };
6. **struct** test\_set{
7. pair<**double**, **double**> X\_1\_X\_2;
8. test\_set(**const** pair<**double**, **double**> &x1X2) : X\_1\_X\_2(x1X2) {}
9. };

* 定义集合是正向标签,定义集合是负向标签,首先需要初始化:

1. **for** (train\_set ts: W1W2) {
2. **if** (ts.y == 0) {
3. //反例
4. x1\_N += ts.X\_1\_X\_2.first;
5. x2\_N = ts.X\_1\_X\_2.second;
6. } **else** {
7. x1\_P += ts.X\_1\_X\_2.first;
8. x2\_P = ts.X\_1\_X\_2.second;
9. }
10. }
11. w1\_w2 = make\_pair(x1\_P - x2\_P, x1\_N - x2\_N);

* 利用check函数判断是否所有训练集都有正确的分类标签:

1. **bool** check\_ok(vector<train\_set> trainSet) {
2. **for** (train\_set ts: trainSet) {
3. **if** (get\_result(ts.X\_1\_X\_2) != ts.y) {
4. **return** **false**;
5. }
6. }
7. **return** **true**;
8. }

* 直到所有标签都能被正确判断,则退出循环,否则继续，这里令学习常数是1，它也可以是其他值。

1. **void** fit() {
2. **int** i = 0;
3. **while** (!check\_ok(trainset)) {
4. cout << "Now it is the " << i << "th training!" << endl;
5. **for** (train\_set ts: trainset) {
6. w1\_w2 = w1\_w2 + eta \* (ts.y - get\_result(ts.X\_1\_X\_2)) \* ts.X\_1\_X\_2;
7. }
8. cout << "now: w1 =" << w1\_w2.first << ",w2=" << w1\_w2.second << ",b=" << b << endl;
9. }
10. i++;
11. }

### 整体代码:

1. #include <vector>
2. #include <iostream>
4. **using** **namespace** std;
6. //重载向量加法:
7. pair<**double**, **double**> operator+(pair<**double**, **double**> d1, pair<**double**, **double**> d2) {
8. **return** make\_pair(d1.first + d2.first, d1.second + d2.second);
9. }
11. //重载数乘:
12. pair<**double**, **double**> operator\*(**double** k, pair<**double**, **double**> d1) {
13. **return** {k \* d1.first, k \* d1.second};
14. }
16. pair<**double**, **double**> operator\*(**int** k, pair<**double**, **double**> d1) {
17. **return** {k \* d1.first, k \* d1.second};
18. }
20. **double** operator\*(pair<**double**, **double**> d1, pair<**double**, **double**> d2) {
21. **return** d1.first \* d2.first + d1.second \* d2.second;
22. }
24. //减法
25. pair<**double**, **double**> operator-(pair<**double**, **double**> d1, pair<**double**, **double**> d2) {
26. **return** {d1.first - d2.first, d1.second - d2.second};
27. }
29. **struct** train\_set {
30. pair<**double**, **double**> X\_1\_X\_2;
31. **int** y;
33. train\_set(**const** pair<**double**, **double**> &x1X2, **int** y) : X\_1\_X\_2(x1X2), y(y) {}
34. };
36. **struct** test\_set {
37. pair<**double**, **double**> X\_1\_X\_2;
39. test\_set(**const** pair<**double**, **double**> &x1X2) : X\_1\_X\_2(x1X2) {}
40. };
42. **class** Single\_Layer\_Perceptron {
43. **private**:
44. pair<**double**, **double**> w1\_w2;
45. **double** b = 0;
46. **double** eta = 0;//学习常数
47. vector<train\_set> trainset;
48. **public**:
49. Single\_Layer\_Perceptron(vector<train\_set> W1W2, **double** eta = 1, **double** b = 0) {
50. **this**->b = b;
51. **this**->eta = eta;
52. **double** x1\_P = 0;//正例中的x1
53. **double** x2\_P = 0;//正例中的x2
54. **double** x1\_N = 0;//反例中的x1
55. **double** x2\_N = 0;//反例中的x2
56. //1是正,0是反
57. **for** (train\_set ts: W1W2) {
58. **if** (ts.y == 0) {
59. //反例
60. x1\_N += ts.X\_1\_X\_2.first;
61. x2\_N = ts.X\_1\_X\_2.second;
62. } **else** {
63. x1\_P += ts.X\_1\_X\_2.first;
64. x2\_P = ts.X\_1\_X\_2.second;
65. }
66. }
67. w1\_w2 = make\_pair(x1\_P - x2\_P, x1\_N - x2\_N);
68. **this**->trainset = W1W2;
69. }
71. **int** get\_result(pair<**double**, **double**> d1) {
72. **double** result = w1\_w2 \* d1 + b;
73. **if** (result < 0) {
74. **return** 0;
75. } **else** {
76. **return** 1;
77. }
78. }
80. **void** print\_result(vector<test\_set> testSet) {
81. **for** (test\_set ts: testSet) {
82. cout << "The test case:(" << ts.X\_1\_X\_2.first << "," << ts.X\_1\_X\_2.second << ")" << "'s result is "
83. << get\_result(ts.X\_1\_X\_2) << endl;
84. }
85. }
87. **bool** check\_ok(vector<train\_set> trainSet) {
88. **for** (train\_set ts: trainSet) {
89. **if** (get\_result(ts.X\_1\_X\_2) != ts.y) {
90. **return** **false**;
91. }
92. }
93. **return** **true**;
94. }
96. **void** fit() {
97. **int** i = 0;
98. **while** (!check\_ok(trainset)) {
99. cout << "Now it is the " << i << "th training!" << endl;
100. **for** (train\_set ts: trainset) {
101. w1\_w2 = w1\_w2 + eta \* (ts.y - get\_result(ts.X\_1\_X\_2)) \* ts.X\_1\_X\_2;
102. }
103. cout << "now: w1 =" << w1\_w2.first << ",w2=" << w1\_w2.second << ",b=" << b << endl;
104. }
105. i++;
106. }
108. };
110. **int** main() {
111. vector<train\_set> trainset;
112. trainset.push\_back({{-9, 15}, 0});
113. trainset.push\_back({{1, 8}, 1});
114. trainset.push\_back({{-12, 4}, 0});
115. trainset.push\_back({{-4, 5}, 0});
116. trainset.push\_back({{0, 11}, 0});
117. trainset.push\_back({{5, 9}, 1});
118. Single\_Layer\_Perceptron slp = Single\_Layer\_Perceptron(trainset);
119. slp.fit();
120. cout << "fitting end! ..................................." << endl;
121. vector<test\_set> ts;
122. ts.push\_back({{2, -5}});
123. ts.push\_back({{-10, 10}});
124. ts.push\_back({{0, 5}});
125. ts.push\_back({{-6, 6}});
126. slp.print\_result(ts);
127. **return** 0;
128. }

### 运行结果与分析

**Now it is the 0th training!**

**now: w1 =3,w2=-19,b=0**

**Now it is the 1th training!**

**now: w1 =9,w2=-2,b=0**

**Now it is the 2th training!**

**now: w1 =10,w2=-5,b=0**

**Now it is the 3th training!**

**now: w1 =16,w2=1,b=0**

**Now it is the 4th training!**

**now: w1 =21,w2=-1,b=0**

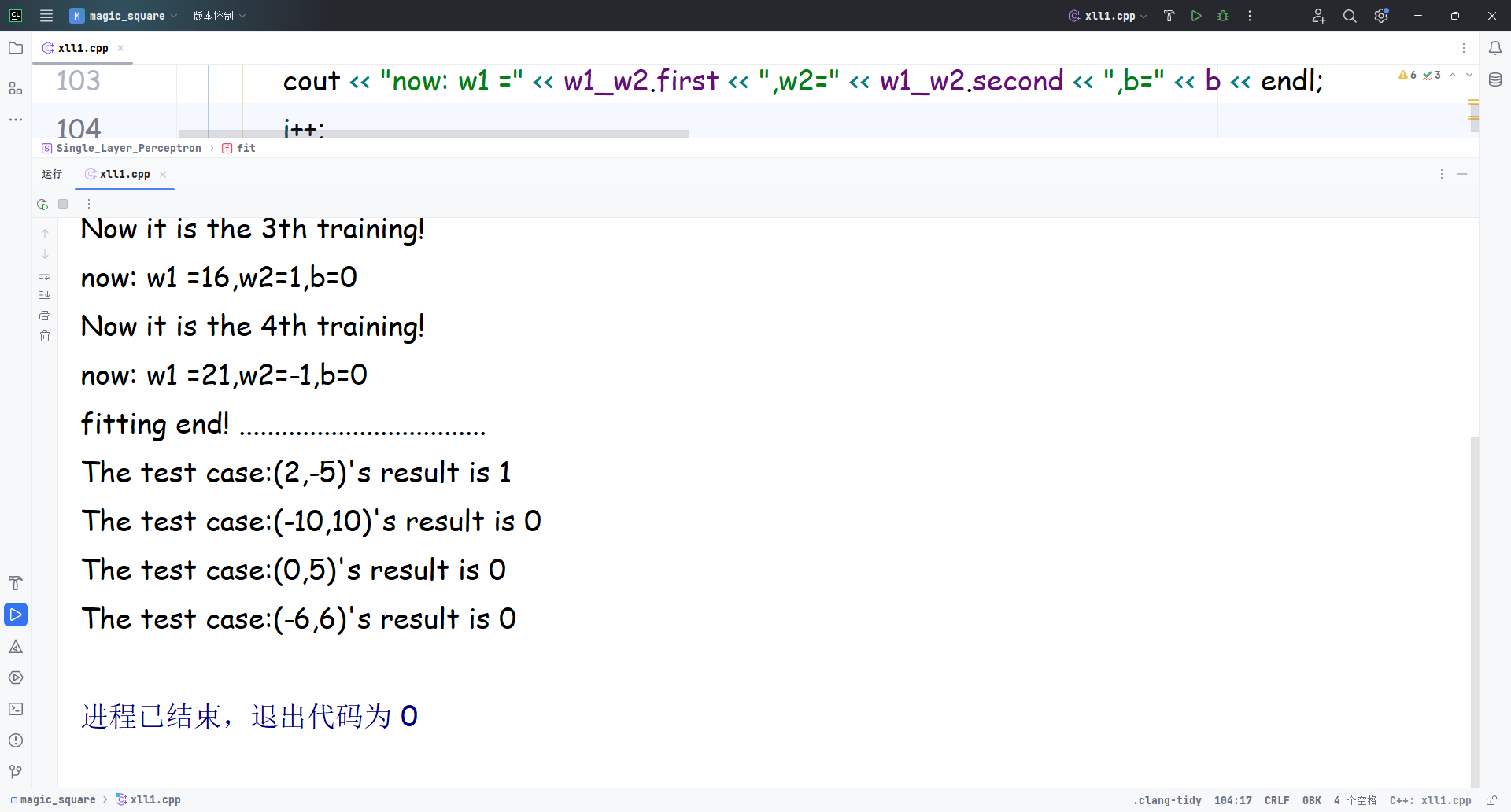
**fitting end! ...................................**

**The test case:(2,-5)'s result is 1**

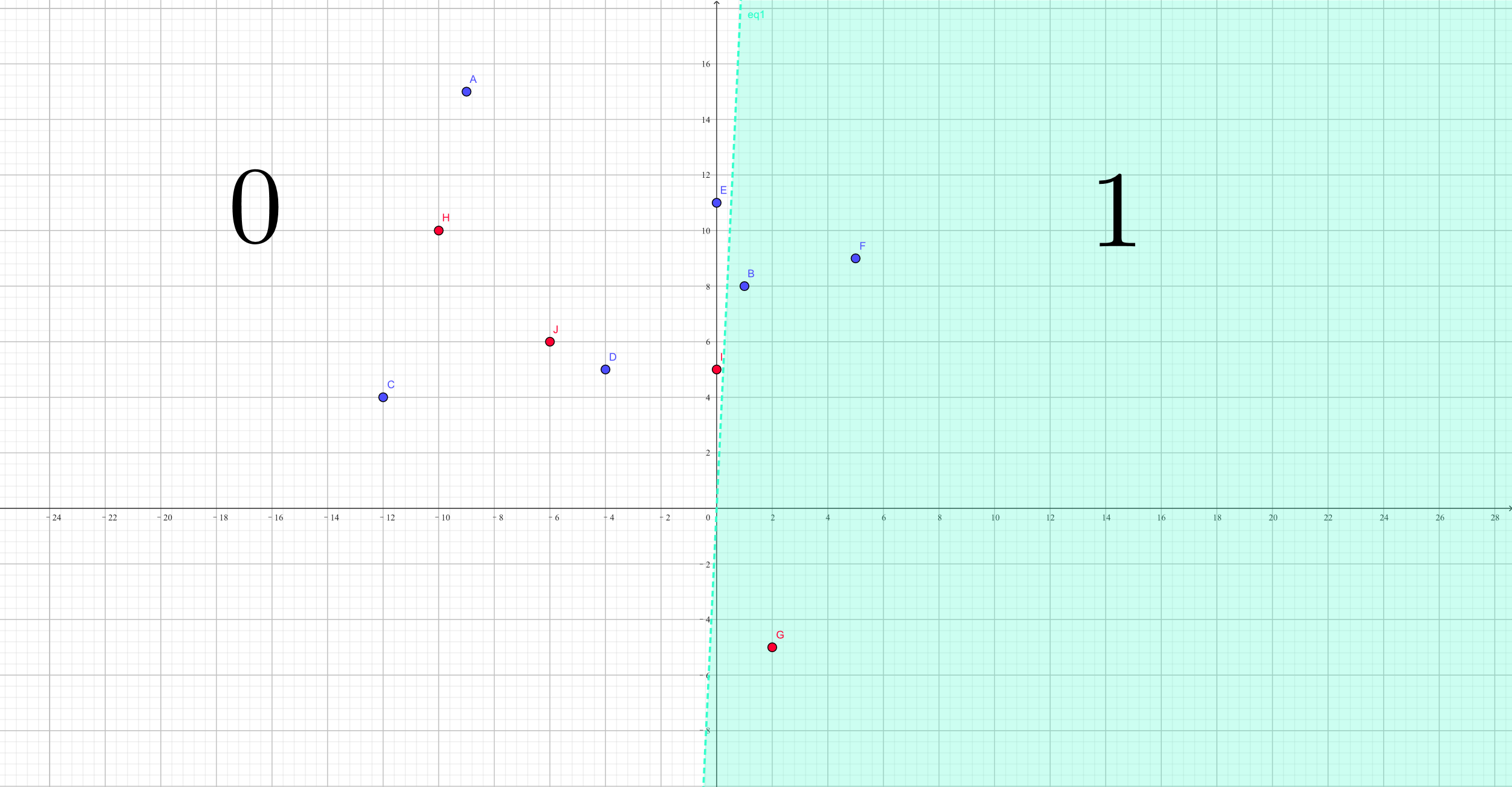
**The test case:(-10,10)'s result is 0**

**The test case:(0,5)'s result is 0**

**The test case:(-6,6)'s result is 0**



* **通过图例,可以看成被正确分类**

****

## 问题2

### 问题描述:

复现遗传算法,求解的最大值的位置,其中

### 结构设计:

* 根据遗传算法的执行流程,需要预先定义迭代次数,适应度评价,搜索上界,搜索下界,搜索精度

1. MAX\_FIT = []
2. NUM = []
3. G = 0  # 迭代次数
4. FES = 0  # 适应度评价
5. U\_x = 31
6. L\_x = 0
7. Search\_Accuracy = 0.01  # 搜索精度0.01

* 利用公式:计算二进制串的长度,产生N个个体的群体,计算实际搜索精度

1. **def** generate\_unique\_binary\_strings(num\_strings, length):  # 生成长度固定的N个2进制串(不重复)
2. unique\_strings = set()
3. **while** len(unique\_strings) < num\_strings:
4. binary\_string = ''.join(random.choice('01') **for** \_ **in** range(length))
5. unique\_strings.add(binary\_string)
6. **return** list(unique\_strings)
8. l = math.ceil(math.log2((U\_x - L\_x) / Search\_Accuracy))  # 二进制字符串长度
9. N = 30  # 初始产生个数为30个群体
10. # 实际搜索精度:
11. delta = (U\_x - L\_x) / (np.exp2(l) - 1)
12. # N个个体的初始群体
13. P\_G = generate\_unique\_binary\_strings(N, l)

* 进行解码

1. **def** cal\_fit\_G(P\_G\_decoded: list):
2. fit\_G = []
3. **for** value **in** P\_G\_decoded:
4. fit\_G.append(f\_x(value))
5. **return** fit\_G
7. P\_G\_decoded = decode(P\_G, delta, L\_x)

* 得到解码后的适应度

1. #计算适应度
2. **def** cal\_fit\_G(P\_G\_decoded: list):
3. fit\_G = []
4. **for** value **in** P\_G\_decoded:
5. fit\_G.append(f\_x(value))
6. **return** fit\_G

9. # 计算解码后的适应度
10. fit\_g = cal\_fit\_G(P\_G\_decoded)

* 进行精英选择算法,找出群体中具有最高适应度和最低适应度的个体,对中剩余个个体,根据适应度进行选择操作,方式是先计算每个个体适应度所占群体适应度总和的比例,记为,从第一个个体开始,对适应度比例进行累加,记为,产生一个之间的随机数,找到第一个比大的对应的加入父代个体,总共需要产生个

1. #计算适应度比例
2. **def** cal\_fxi\_scale(fit\_g: list):
3. total = sum(fit\_g)
4. B = []
5. **for** value **in** fit\_g:
6. B.append(value / total)
7. **return** B
9. #比例和
10. **def** get\_sum\_scale\_C(B: list):
11. c = np.array(B)
12. **return** c.cumsum(axis=0)
14. #轮赌法
15. **def** Roulette\_Wheel\_Choice(P\_G: list, C: list, N):
16. S\_G = []
17. **for** i **in** range(0, N):
18. rand = np.random.rand()  # 生成0-1之间均匀分布的随机数
19. **if** (rand == 1):
20. S\_G.append(P\_G[N - 1])
21. **else**:
22. index = np.searchsorted(np.array(C), rand, side='right')  # 找到第一个比rand大的个体
23. S\_G.append(P\_G[index])
24. **return** S\_G

27. NUM.append(G)
28. MAX\_FIT.append(max(fit\_g))
29. # 找到pG中拥有最高适应度的个体
30. fit\_max = max(fit\_g)
31. max\_index = fit\_g.index(fit\_max)
32. P\_G\_max = P\_G[max\_index]
33. fit\_min = min(fit\_g)
34. min\_index = fit\_g.index(fit\_min)
35. P\_G\_min = P\_G[min\_index]
36. # 丢弃一个即可
37. fit\_g.remove(fit\_max)
38. fit\_g.remove(fit\_min)
39. P\_G.remove(P\_G\_min)
40. P\_G.remove(P\_G\_max)
41. # 选择操作
42. # 计算fit(x\_i)/sum(fit(x\_i))的比例B
43. B = cal\_fxi\_scale(fit\_g)
44. # 计算比例的累加值C
45. C = get\_sum\_scale\_C(B)
46. # print(C)
47. # 得到第一个父代个体:
48. S\_G = Roulette\_Wheel\_Choice(P\_G=P\_G, C=C, N=N - 2)

* 将中的个体随机分为组,对每组中的两个个体,以概率执行交叉算子,得到一个新的群体,这里定义

1. **def** split\_SG(SG: list):
2. # 打乱原始数组的顺序
3. np.random.shuffle(S\_G)
5. # 将原数组分为N-2/2个子数组，每个子数组包含两个值
6. n = len(S\_G) // 2
7. split\_arrays = [S\_G[i:i + 2] **for** i **in** range(0, len(S\_G), 2)]
8. # 返回
9. **return** split\_arrays
11. #交叉
12. **def** crossOver(S\_G\_splited: list, pc):
13. # 固定Cpoint
14. Cpoint = 3
15. C\_G = []
16. **for** value **in** S\_G\_splited:
17. rand = np.random.rand()
18. val1 = value[0]
19. val2 = value[1]
20. **if** rand < pc:
21. left1 = val1[:Cpoint]
22. right1 = val1[Cpoint:]
23. left2 = val2[:Cpoint]
24. right2 = val2[Cpoint:]
25. result1 = left1 + right2
26. result2 = left2 + right1
27. C\_G.append(result1)
28. C\_G.append(result2)
29. **else**:
30. C\_G.append(val1)
31. C\_G.append(val2)
32. **return** C\_G

35. # 随机分为N-2/2组
36. S\_G\_splited = split\_SG(S\_G)
37. # print(S\_G\_splited)
38. # 交叉操作
39. pc = 0.8
40. C\_G = crossOver(S\_G\_splited, 0.8)

* 对于中的每个二进制位,以概率执行变异操作,得到子代个体集,这里定义

1. **def** Mutation(C\_G: list, pm):
2. M\_G = []
3. C\_G\_ = copy.copy(C\_G)
4. **for** j **in** range(0, len(C\_G\_)):
5. # 遍历每个二进制数
6. **for** i **in** range(0, len(C\_G\_[j])):
7. rand = np.random.rand()
8. **if** rand < pm:
9. # 变异操作
10. # print("变异!"+str(j))
11. binary\_list = list(C\_G\_[j])
12. **if** binary\_list[i] == '0':
13. binary\_list[i] = '1'
14. **else**:
15. binary\_list[i] = '0'
16. mutated\_binary = ''.join(binary\_list)
17. C\_G\_[j] = mutated\_binary  # 更新原始列表
18. M\_G.append(C\_G\_[j])
19. **return** M\_G
20. pm = 0.01
21. M\_G = Mutation(C\_G, pm)

* 

1. FES = FES + N - 2

* 将中具有最高适应度的个体复制两份,将复制后的两个个体加入,并将其适应度加入,执行替换操作,令

1. fit\_g = cal\_fit\_G(M\_G\_decoded)
2. **if** max(fit\_g) > fit\_max:
3. fit\_g.append(max(fit\_g))
4. fit\_g.append(max(fit\_g))
5. # 复制2份
6. max\_index\_ = fit\_g.index(max(fit\_g))
7. M\_G.append(M\_G[max\_index\_])
8. M\_G.append(M\_G[max\_index\_])
9. **else**:
10. fit\_g.append(fit\_max)
11. fit\_g.append(fit\_max)
12. M\_G.append(P\_G\_max)
13. M\_G.append(P\_G\_max)
14. P\_G = M\_G
15. G = G + 1  # 执行100次

* 重复1000次

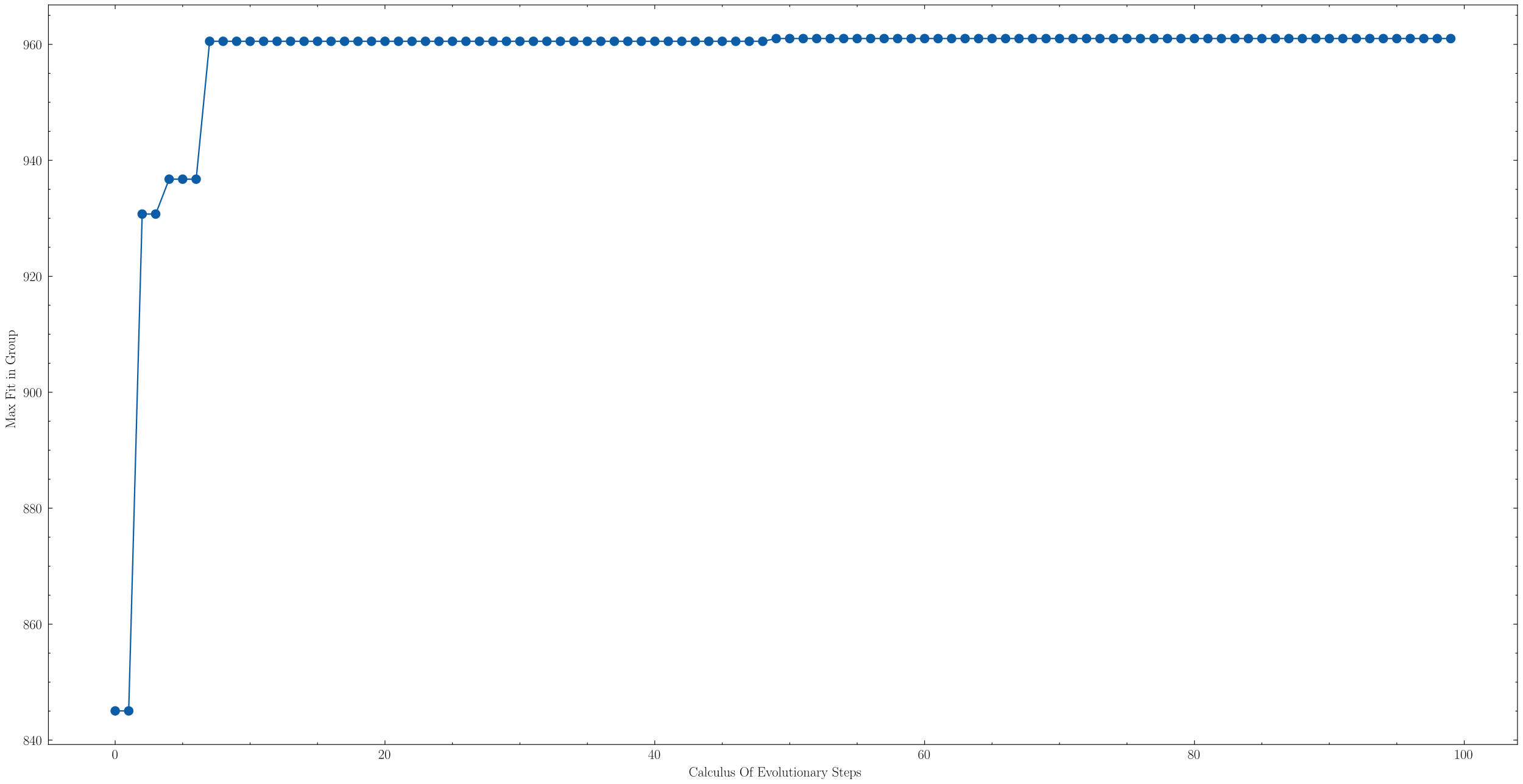
### 整体代码：

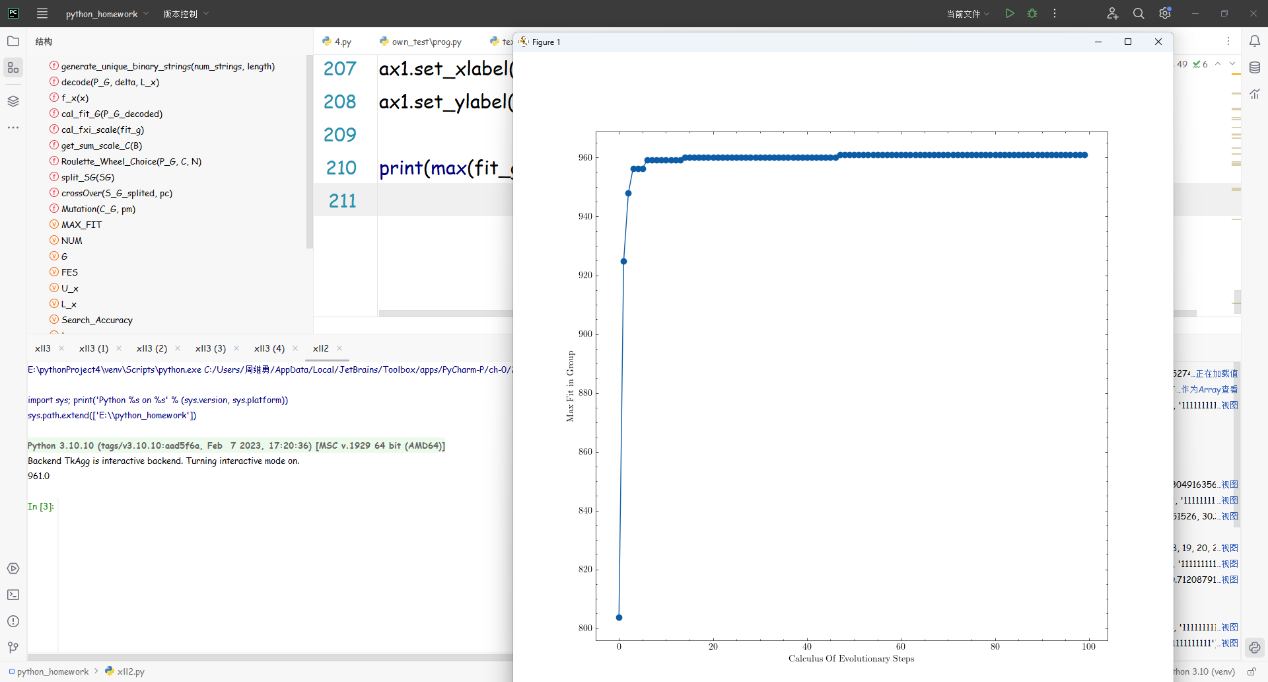
全部的代码已间接在上面给出,此处不再赘述,可见附件,其中绘图代码如下:

1. figure: plt.Figure = plt.figure(figsize=(10, 10))
2. ax1: plt.Axes = figure.add\_subplot(1, 1, 1)
3. ax1.plot(NUM, MAX\_FIT, linestyle='-',marker='o')
4. plt.show()
5. ax1.set\_xlabel("Calculus Of Evolutionary Steps")
6. ax1.set\_ylabel("Max Fit in Group")
8. **print**(max(fit\_g))

### 运行结果和分析

控制台成功输出**961**，其中通过图像,可以看出其快速收敛的趋势:





## 问题3

### 问题描述:

复现粒子群算法,求解的最大值的位置,其中

### 结构设计:

定义类class Particle:描述粒子群算法

* 对粒子群进行初始化,使得在时每个粒子,使得使每个粒子在超空间中的位置是随机的。

1. self.position = random.uniform(bounds[0], bounds[1])  #随机生成点的坐标

* 通过每个粒子的当前位置评价其性能:

1. **def** eval\_position(self):
2. **return** self.position \*\* 2  # f(x) = x^2

* 比较每个个体当前性能与它至今有过的最好性能,如果,那么:



1. current\_value = self.eval\_position()
2. **if** current\_value > self.best\_value:
3. self.best\_value = current\_value
4. self.best\_position = self.position

* 把每个粒子的性能和全局最佳粒子的性能进行比较,如果,那么:



1. **for** particle **in** particles:
2. **if** particle.best\_value > global\_best\_value:
3. global\_best\_value = particle.best\_value
4. global\_best\_position = particle.best\_position

* 改变粒子的速度矢量:



1. **def** update\_velocity(self, global\_best\_position):
2. c1 = 1
3. c2 = 2
4. r1 = random.random()
5. r2 = random.random()
6. self.velocity = (self.velocity +
7. c1 \* r1 \* (self.best\_position - self.position) +
8. c2 \* r2 \* (global\_best\_position - self.position))

* 把每个粒子移动到新的位置:



1. **def** update\_position(self, bounds):
2. self.position += self.velocity

* 重复递归直至收敛

### 整体代码:

1. **import** random

4. **class** Particle:
5. **def** \_\_init\_\_(self, bounds):
6. self.position = random.uniform(bounds[0], bounds[1])  #随机生成点的坐标
7. self.velocity = random.uniform(-1, 1)  # 速度
8. self.best\_position = self.position  # 最佳位置
9. self.best\_value = self.eval\_position()  # 最佳值
11. **def** eval\_position(self):
12. **return** self.position \*\* 2  # f(x) = x^2
14. **def** update\_velocity(self, global\_best\_position):
15. c1 = 1
16. c2 = 2
17. r1 = random.random()
18. r2 = random.random()
19. self.velocity = (self.velocity +
20. c1 \* r1 \* (self.best\_position - self.position) +
21. c2 \* r2 \* (global\_best\_position - self.position))
23. **def** update\_position(self, bounds):
24. self.position += self.velocity
25. **if** self.position < bounds[0]:
26. self.position = bounds[0]
27. **if** self.position > bounds[1]:
28. self.position = bounds[1]
30. **def** update(self, global\_best\_position, bounds):
31. self.update\_velocity(global\_best\_position)
32. self.update\_position(bounds)
33. current\_value = self.eval\_position()
34. **if** current\_value > self.best\_value:
35. self.best\_value = current\_value
36. self.best\_position = self.position

39. **def** pso(num\_particles, bounds, max\_iter):
40. particles = [Particle(bounds) **for** \_ **in** range(num\_particles)]
41. global\_best\_value = float('-inf')
42. global\_best\_position = None
44. **for** \_ **in** range(max\_iter):
45. **for** particle **in** particles:
46. **if** particle.best\_value > global\_best\_value:
47. global\_best\_value = particle.best\_value
48. global\_best\_position = particle.best\_position
50. **for** particle **in** particles:
51. particle.update(global\_best\_position, bounds)
53. **return** global\_best\_position, global\_best\_value

56. # PSO 参数
57. num\_particles = 30
58. bounds = (0, 31)  # Bounds for x
59. max\_iter = 100
61. #  PSO
62. best\_position, best\_value = pso(num\_particles, bounds, max\_iter)
63. **print**(best\_position, best\_value)

### 运行结果和分析

结果正确:

