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

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	x	y	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$net = b + \omega_1 x + \omega_2 y$,其中 label $= f(net) = \begin{cases} 0 & net < 0 \\ 1 & net \geq 0 \end{cases}$,这里$\boxed{b = 0}$,(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) {}
```

```
};
```

- 定义集合 P 是正向标签,定义集合 N 是负向标签,首先需要初始化 $\vec{\omega} = \sum_{(x,y)\in P}(x,y) - \sum_{(x,y)\in N}(x,y)$:

```
for (train_set ts: W1W2) {
            if (ts.y == 0) {
                //反例
                x1_N += ts.X_1_X_2.first;
                x2_N = ts.X_1_X_2.second;
            } else {
                x1_P += ts.X_1_X_2.first;
                x2_P = ts.X_1_X_2.second;
            }
        }
        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:

```
bool check_ok(vector<train_set> trainSet) {
    for (train_set ts: trainSet) {
        if (get_result(ts.X_1_X_2) != ts.y) {
            return false;
        }
    }
    return true;
}
```

- 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 η 是 1，它也可以是其他值。

```
void fit() {
        int i = 0;
        while (!check_ok(trainset)) {
            cout << "Now it is the " << i << "th training!" << endl;
            for (train_set ts: trainset) {
                w1_w2 = w1_w2 + eta * (ts.y - get_result(ts.X_1_X_2)) * ts.X_1_X
_2;
            }
            cout << "now: w1 =" << w1_w2.first << ",w2=" << w1_w2.second << ",b=
" << b << endl;
        }
        i++;
    }
```

整体代码:

```
#include <vector>
#include <iostream>

using namespace std;

//重载向量加法:
pair<double, double> operator+(pair<double, double> d1, pair<double, double> d2) {

    return make_pair(d1.first + d2.first, d1.second + d2.second);
}

//重载数乘:
pair<double, double> operator*(double k, pair<double, double> d1) {
    return {k * d1.first, k * d1.second};
}

pair<double, double> operator*(int k, pair<double, double> d1) {
```

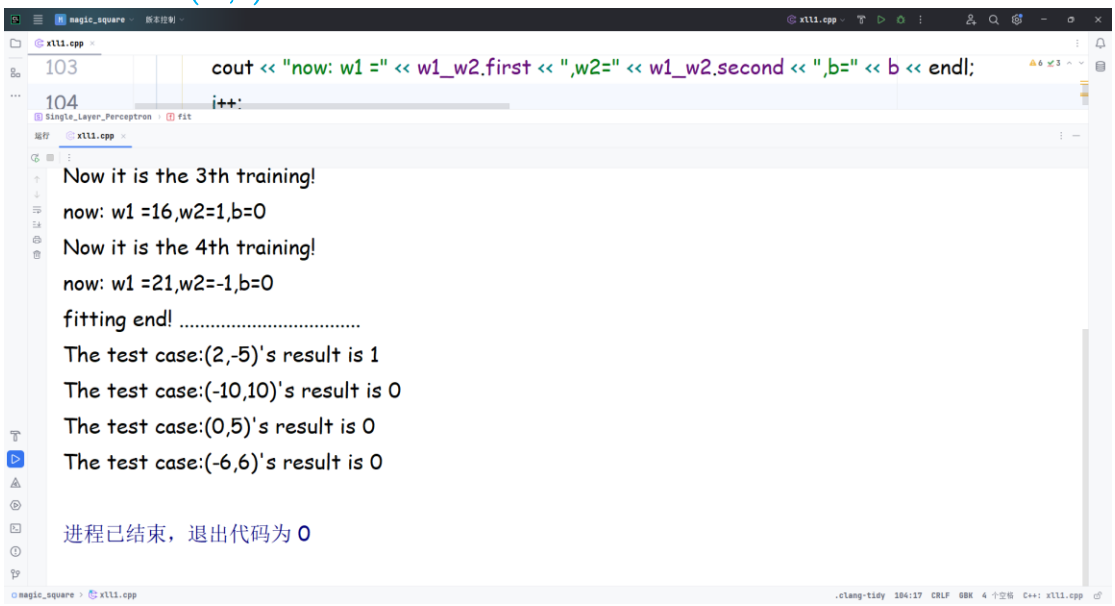
```
    return {k * d1.first, k * d1.second};
}

double operator*(pair<double, double> d1, pair<double, double> d2) {
    return d1.first * d2.first + d1.second * d2.second;
}

//减法
pair<double, double> operator-
(pair<double, double> d1, pair<double, double> d2) {
    return {d1.first - d2.first, d1.second - d2.second};
}

struct train_set {
    pair<double, double> X_1_X_2;
    int y;

    train_set(const pair<double, double> &x1X2, int y) : X_1_X_2(x1X2), y(y) {}
};

struct test_set {
    pair<double, double> X_1_X_2;

    test_set(const pair<double, double> &x1X2) : X_1_X_2(x1X2) {}
};

class Single_Layer_Perceptron {
private:
    pair<double, double> w1_w2;
    double b = 0;
    double eta = 0;//学习常数
    vector<train_set> trainset;
public:
    Single_Layer_Perceptron(vector<train_set> W1W2, double eta = 1, double b = 0) {

        this->b = b;
        this->eta = eta;
        double x1_P = 0;//正例中的 x1
        double x2_P = 0;//正例中的 x2
        double x1_N = 0;//反例中的 x1
        double x2_N = 0;//反例中的 x2
        //1 是正,0 是反
        for (train_set ts: W1W2) {
            if (ts.y == 0) {
                //反例
                x1_N += ts.X_1_X_2.first;
                x2_N = ts.X_1_X_2.second;
            } else {
                x1_P += ts.X_1_X_2.first;
                x2_P = ts.X_1_X_2.second;
            }
        }
        w1_w2 = make_pair(x1_P - x2_P, x1_N - x2_N);
        this->trainset = W1W2;
    }

    int get_result(pair<double, double> d1) {
        double result = w1_w2 * d1 + b;
        if (result < 0) {
            return 0;
        } else {
            return 1;
        }
    }
```

```
79.
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.seco
    nd << ")" << "'s result is "
83.                  << get_result(ts.X_1_X_2) << endl;
84.         }
85.     }
86.
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.     }
95.
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.        }
107.
108.    };
109.
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.    }
```

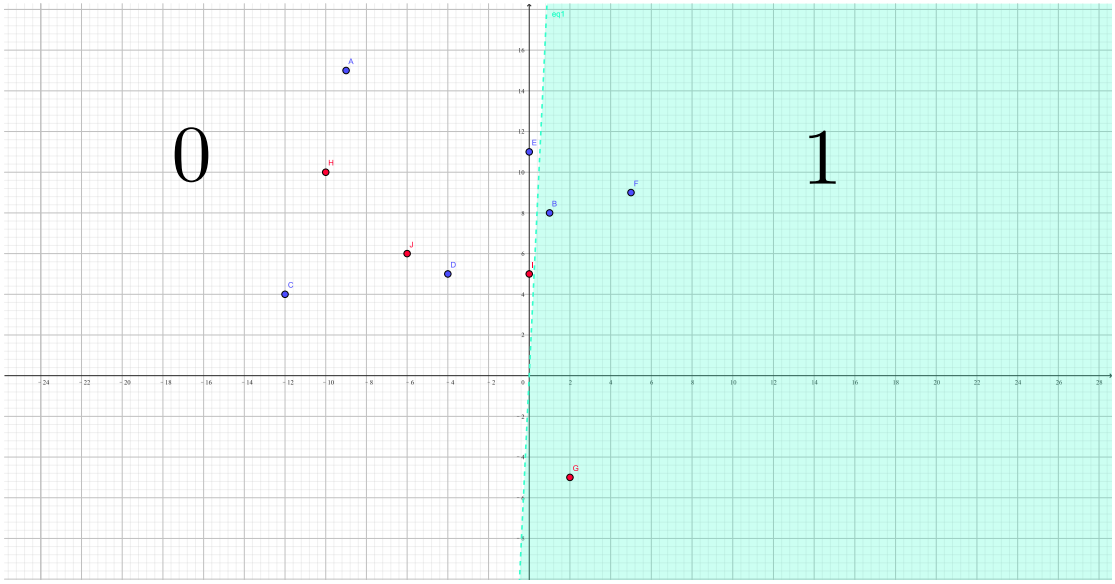
运行结果与分析

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$f(x)=x^2$的最大值的位置,其中$x\in[0,31]$

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G,适应度评价FES,搜索上界U_x,搜索下界L_x,搜索精度Search_Accuracy

```
MAX_FIT = []
NUM = []
G = 0  # 迭代次数
FES = 0  # 适应度评价
U_x = 31
L_x = 0
Search_Accuracy = 0.01  # 搜索精度 0.01
```

- 利用公式:$l=\left\lceil \log_2\left(\frac{U_x-L_x}{\text{Search_Accuracy}}\right)\right\rceil$计算二进制串的长度,产生 N 个个体的群体$P_G$,计算实际搜索精度$\delta=\frac{U_x-L_x}{2^l-1}$

```
def generate_unique_binary_strings(num_strings, length):  # 生成长度固定的 N 个 2 进制串(不重复)
    unique_strings = set()
    while len(unique_strings) < num_strings:
        binary_string = ''.join(random.choice('01') for _ in range(length))
        unique_strings.add(binary_string)
    return list(unique_strings)

l = math.ceil(math.log2((U_x - L_x) / Search_Accuracy))  # 二进制字符串长度
N = 30  # 初始产生个数为 30 个群体
# 实际搜索精度:
delta = (U_x - L_x) / (np.exp2(l) - 1)
# N 个个体的初始群体
P_G = generate_unique_binary_strings(N, l)
```

- 进行解码

```
def cal_fit_G(P_G_decoded: list):
    fit_G = []
    for value in P_G_decoded:
        fit_G.append(f_x(value))
    return fit_G

P_G_decoded = decode(P_G, delta, L_x)
```

- 得到解码后的适应度fit_G

```
#计算适应度
def cal_fit_G(P_G_decoded: list):
    fit_G = []
    for value in P_G_decoded:
        fit_G.append(f_x(value))
    return fit_G


# 计算解码后的适应度
```

```
fit_g = cal_fit_G(P_G_decoded)
```

- 进行精英选择算法,找出群体P_G中具有最高适应度和最低适应度的个体,对P_G中剩余$N-2$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$fit(x_i)$所占群体适应度总和$\sum_{i=1}^{N} fit(x_i)$的比例,记为$B_1,B_2,\cdots,B_N$,从第一个个体开始,对适应度比例进行累加,记为$C_1,C_2,\cdots,C_N$,产生一个$[0,1]$之间的随机数rand,找到第一个比rand大的C_k对应的x_k加入父代个体S_G,总共需要产生$N-2$个

```
#计算适应度比例
def cal_fxi_scale(fit_g: list):
    total = sum(fit_g)
    B = []
    for value in fit_g:
        B.append(value / total)
    return B

#比例和
def get_sum_scale_C(B: list):
    c = np.array(B)
    return c.cumsum(axis=0)

#轮赌法
def Roulette_Wheel_Choice(P_G: list, C: list, N):
    S_G = []
    for i in range(0, N):
        rand = np.random.rand()  # 生成 0-1 之间均匀分布的随机数
        if (rand == 1):
            S_G.append(P_G[N - 1])
        else:
            index = np.searchsorted(np.array(C), rand, side='right')  # 找到第一个比 rand 大的个体
            S_G.append(P_G[index])
    return S_G


NUM.append(G)
MAX_FIT.append(max(fit_g))
# 找到 pG 中拥有最高适应度的个体
fit_max = max(fit_g)
max_index = fit_g.index(fit_max)
P_G_max = P_G[max_index]
fit_min = min(fit_g)
min_index = fit_g.index(fit_min)
P_G_min = P_G[min_index]
# 丢弃一个即可
fit_g.remove(fit_max)
fit_g.remove(fit_min)
P_G.remove(P_G_min)
P_G.remove(P_G_max)
# 选择操作
# 计算 fit(x_i)/sum(fit(x_i))的比例 B
B = cal_fxi_scale(fit_g)
# 计算比例的累加值 C
C = get_sum_scale_C(B)
# print(C)
# 得到第一个父代个体:
S_G = Roulette_Wheel_Choice(P_G=P_G, C=C, N=N - 2)
```

- 将S_G中的个体随机分为$\frac{(N-2)}{2}$组,对每组中的两个个体,以概率pc执行交叉算子,得到一个新的群体C_G,这里定义$\mathbf{CPOINT}=3,\mathbf{pc}=0.8$

```python
def split_SG(SG: list):
    # 打乱原始数组的顺序
    np.random.shuffle(S_G)

    # 将原数组分为 N-2/2 个子数组，每个子数组包含两个值
    n = len(S_G) // 2
    split_arrays = [S_G[i:i + 2] for i in range(0, len(S_G), 2)]
    # 返回
    return split_arrays

#交叉
def crossOver(S_G_splited: list, pc):
    # 固定 Cpoint
    Cpoint = 3
    C_G = []
    for value in S_G_splited:
        rand = np.random.rand()
        val1 = value[0]
        val2 = value[1]
        if rand < pc:
            left1 = val1[:Cpoint]
            right1 = val1[Cpoint:]
            left2 = val2[:Cpoint]
            right2 = val2[Cpoint:]
            result1 = left1 + right2
            result2 = left2 + right1
            C_G.append(result1)
            C_G.append(result2)
        else:
            C_G.append(val1)
            C_G.append(val2)
    return C_G


# 随机分为 N-2/2 组
    S_G_splited = split_SG(S_G)
    # print(S_G_splited)
    # 交叉操作
    pc = 0.8
    C_G = crossOver(S_G_splited, 0.8)
```

- 对于C_G中的每个二进制位,以概率$\mathbf{pm}$执行变异操作,得到子代个体集M_G,这里定义$\mathbf{pm}=0.01$

```python
def Mutation(C_G: list, pm):
    M_G = []
    C_G_ = copy.copy(C_G)
    for j in range(0, len(C_G_)):
        # 遍历每个二进制数
        for i in range(0, len(C_G_[j])):
            rand = np.random.rand()
            if rand < pm:
                # 变异操作
                # print("变异!"+str(j))
                binary_list = list(C_G_[j])
                if binary_list[i] == '0':
                    binary_list[i] = '1'
                else:
                    binary_list[i] = '0'
                mutated_binary = ''.join(binary_list)
                C_G_[j] = mutated_binary  # 更新原始列表
        M_G.append(C_G_[j])
    return M_G
pm = 0.01
```

```
M_G = Mutation(C_G, pm)
```

- $FES = FES + N - 2$

```
FES = FES + N - 2
```

- 将P_G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M_G,并将其适应度加入fit_G',执行替换操作,令$P_G = M_G\, fit_G = fit_G'$

```
fit_g = cal_fit_G(M_G_decoded)
    if max(fit_g) > fit_max:
        fit_g.append(max(fit_g))
        fit_g.append(max(fit_g))
        # 复制 2 份
        max_index_ = fit_g.index(max(fit_g))
        M_G.append(M_G[max_index_])
        M_G.append(M_G[max_index_])
    else:
        fit_g.append(fit_max)
        fit_g.append(fit_max)
        M_G.append(P_G_max)
        M_G.append(P_G_max)
    P_G = M_G
    G = G + 1  # 执行 100 次
```

- 重复 1000 次

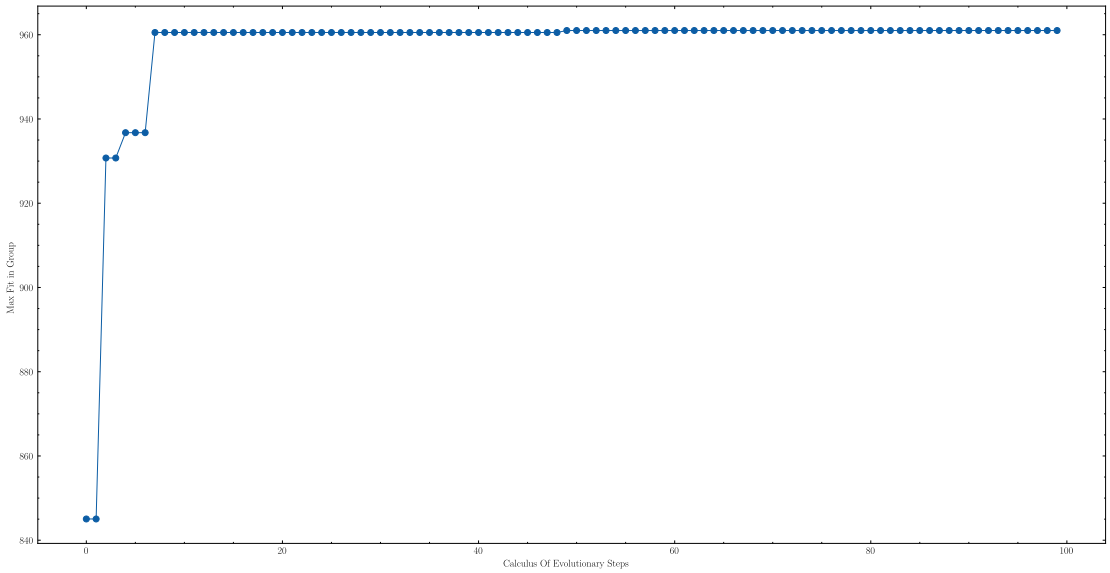
整体代码:

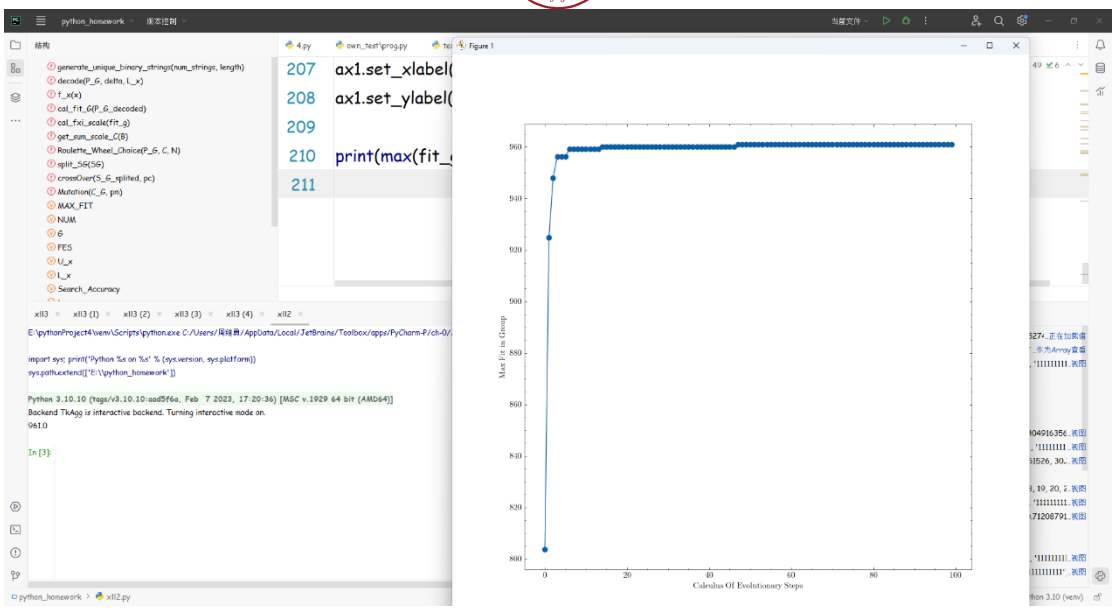
全部的代码已间接在上面给出,此处不再赘述,可见附件,其中绘图代码如下:

```
figure: plt.Figure = plt.figure(figsize=(10, 10))
ax1: plt.Axes = figure.add_subplot(1, 1, 1)
ax1.plot(NUM, MAX_FIT, linestyle='-',marker='o')
plt.show()
ax1.set_xlabel("Calculus Of Evolutionary Steps")
ax1.set_ylabel("Max Fit in Group")

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$f(x)=x^2$的最大值的位置,其中$x\in[0,31]$

结构设计:

定义类 class Particle:描述粒子群算法

- 对粒子群$P(t)$进行初始化,使得在$t=0$时每个粒子$P_i\in P(t)$,使得$t=0$使每个粒子$P_i\in P(t)$在超空间中的位置$x_i(t)$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$x_i(t)$评价其性能$\mathcal{F}$:

```
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$\mathcal{F}(\boldsymbol{x}_i(t))<\mathtt{pbest}_i$,那么:

$$\begin{cases}\mathtt{pbest}_i=\mathcal{F}(\boldsymbol{x}_i(t))\\ \boldsymbol{x}_{pbest_i}=\boldsymbol{x}_i(t)\end{cases}$$

```
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$\mathcal{F}(x_i(t))<\mathtt{gbest}_i$,那么:

$$\begin{cases}\mathtt{gbest}=\mathcal{F}(x_i(t))\\ \mathtt{x}_{\mathtt{gbest}_i}=x_i(t)\end{cases}$$

```
1.  for particle in particles:
2.              if particle.best_value > global_best_value:
3.                  global_best_value = particle.best_value
```

```
                global_best_position = particle.best_position
```

- 改变粒子的速度矢量:

$$v_i(t) = v_i(t-1) + r_1 c_1 (x_{\text{pbest}_i} - x_i(t)) + r_2 c_2 (x_{\text{gbest}} - x_i(t))$$

```
def update_velocity(self, global_best_position):
    c1 = 1
    c2 = 2
    r1 = random.random()
    r2 = random.random()
    self.velocity = (self.velocity +
                     c1 * r1 * (self.best_position - self.position) +
                     c2 * r2 * (global_best_position - self.position))
```

- 把每个粒子移动到新的位置:

$$\begin{cases} x_i(t) = x_i(t-1) + v_i(t) \\ t = t+1 \end{cases}$$

```
def update_position(self, bounds):
    self.position += self.velocity
```

- 重复递归直至收敛

整体代码:

```
import random


class Particle:
    def __init__(self, bounds):
        self.position = random.uniform(bounds[0], bounds[1])  #随机生成点的坐标
        self.velocity = random.uniform(-1, 1)  # 速度
        self.best_position = self.position  # 最佳位置
        self.best_value = self.eval_position()  # 最佳值

    def eval_position(self):
        return self.position ** 2  # f(x) = x^2

    def update_velocity(self, global_best_position):
        c1 = 1
        c2 = 2
        r1 = random.random()
        r2 = random.random()
        self.velocity = (self.velocity +
                         c1 * r1 * (self.best_position - self.position) +
                         c2 * r2 * (global_best_position - self.position))

    def update_position(self, bounds):
        self.position += self.velocity
        if self.position < bounds[0]:
            self.position = bounds[0]
        if self.position > bounds[1]:
            self.position = bounds[1]

    def update(self, global_best_position, bounds):
        self.update_velocity(global_best_position)
        self.update_position(bounds)
        current_value = self.eval_position()
        if current_value > self.best_value:
            self.best_value = current_value
            self.best_position = self.position


```

```python
def pso(num_particles, bounds, max_iter):
    particles = [Particle(bounds) for _ in range(num_particles)]
    global_best_value = float('-inf')
    global_best_position = None

    for _ in range(max_iter):
        for particle in particles:
            if particle.best_value > global_best_value:
                global_best_value = particle.best_value
                global_best_position = particle.best_position

        for particle in particles:
            particle.update(global_best_position, bounds)

    return global_best_position, global_best_value


# PSO 参数
num_particles = 30
bounds = (0, 31)  # Bounds for x
max_iter = 100

#  PSO
best_position, best_value = pso(num_particles, bounds, max_iter)
print(best_position, best_value)
```

运行结果和分析

结果正确: