**Report on Computational Intelligence course of**

**Particle Swarm Optimization（PSO）**

Class: 1804 Student ID: 201894090 Name: Changrui Zuo

**Abstract：Another problem of this problem is to find the best solution through particle swarm optimization. Due to the limited time, only the maximum value was found this time. I think this method is simpler and more effective than GA.**

1. Code description

1.1 Overview

Librandom, numpy and matplotlib of Python3.7 are used. Due to time constraints, this code is just a basic example without classes in the PSO algorithm.

PSO is initialized as a set of random particles (random solutions). Then iteratively find the optimal solution. In each iteration, the particle updates itself by tracking two “extreme values”. The first is the optimal solution p\_best found by the particle itself. The other extreme value is the optimal solution currently found by the entire population, namely the global extreme value g\_best.

Here is the formula:

Speed formula:

Position formula:

In this code, w = 0.6, c1 = 2, c2 = 2.

1.2 Initialize

Population scale is set as pop\_size, the number of target equation is set as variable\_num, the parameter dec\_min\_val and dec\_max\_val are for minimum variable value and maximum variable value. The variable pop\_x stands for position. And the variable pop\_v stands for speed. The variable p\_best is for the best solution so far.

def init\_population(pop\_size, variable\_num, dec\_min\_val, dec\_max\_val, pop\_x, pop\_v, p\_best):  
 for i in range(pop\_size):  
 for j in range(variable\_num):  
 pop\_x[i][j] = random.uniform(dec\_min\_val[j], dec\_max\_val[j])  
 pop\_v[i][j] = random.uniform(0, 1)  
 p\_best[i] = pop\_x[i]

**Code 1.1 Initialize particles**

* 1. Fitness

Because only the maximum solution is required at this time, the objective function can be regarded as a fitness function.

def fitness(s):  
 x1 = s[0]  
 x2 = s[1]  
 x3 = s[2]  
 y = 2 \* x1 \*\* 2 - 3 \* x2 \*\* 2 - 4 \* x1 + 5 \* x2 + x3  
 return y

**Code 1.2 Fitness function**

* 1. Save the best value of each term

Find the best value in the group and save it as g\_best.

for i in range(pop\_size):  
 fit = fitness(p\_best[i])  
 if fit > temp:  
 g\_best = p\_best[i]  
 temp = fit

**Code 1.3 Judgement and save**

* 1. Process in term

Update the speed and position of each individual, then judge them if they are out of range. In the end, renew the value of p\_best and g\_best.

for i in range(max\_gen):  
 for j in range(pop\_size):  
 # update individual  
 pop\_v[j] = w \* pop\_v[j] + c1 \* random.uniform(0, 1) \* (p\_best[j] - pop\_x[j]) + \  
 c2 \* random.uniform(0, 1) \* (g\_best - pop\_x[j])  
 pop\_x[j] = pop\_x[j] + pop\_v[j]  
 for k in range(variable\_num): # limit range  
 if pop\_x[j][k] < dec\_min\_val[k]:  
 pop\_x[j][k] = dec\_min\_val[k]  
 if pop\_x[j][k] > dec\_max\_val[k]:  
 pop\_x[j][k] = dec\_max\_val[k]  
 # Update p\_best and g\_best  
 if fitness(pop\_x[j]) > fitness(p\_best[j]):  
 p\_best[j] = pop\_x[j]  
 if fitness(pop\_x[j]) > fitness(g\_best):  
 g\_best = pop\_x[j]  
 result.append(fitness(g\_best))  
 print("Generation %d: %.4f" % (i, fitness(g\_best)))

**Code 1.4 Term process**

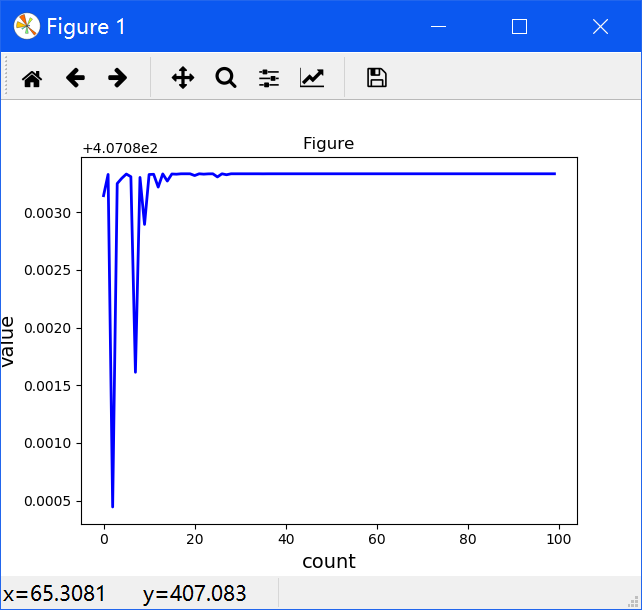
* 1. Visualization

Draw the graph of all process and the best value found.

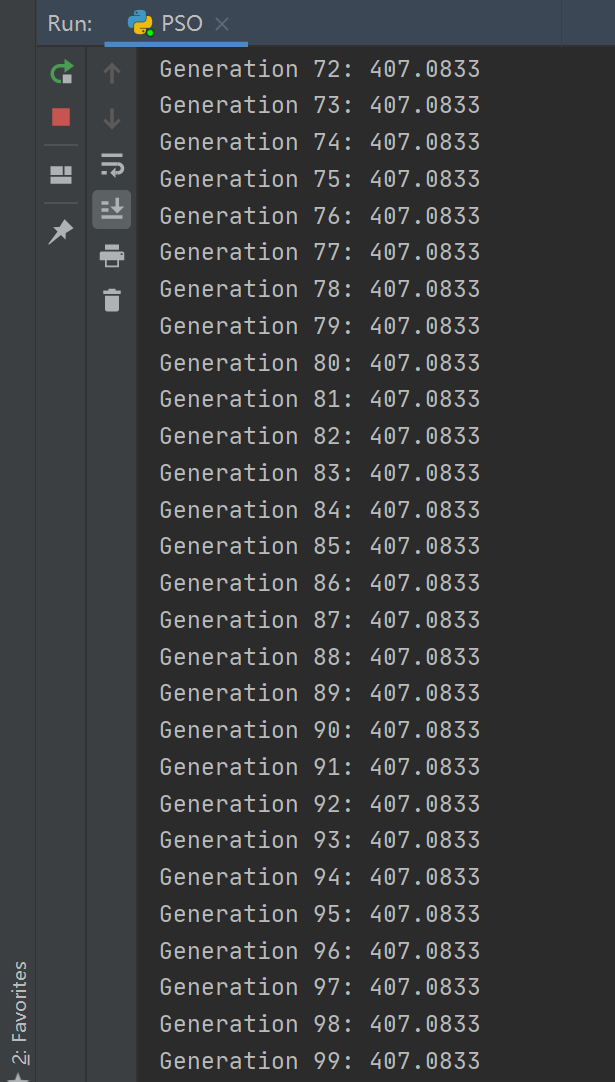
# Visualization  
plt.figure(1)  
plt.title("Figure")  
plt.xlabel("count", size=14)  
plt.ylabel("value", size=14)  
plt.plot(result, color='b', linewidth=2)  
plt.show()

**Code 1.4 Term process**

1. Result



**Figure 2.1 Graph**



**Figure 2.2 Max**