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:

$$vi + 1 = w * vi + c1 * rand1 * (pbesti - xi) + c2 * rand2 * (gbesti - xi)$$

Position formula:

$$x_i = x_i + v_{i+1}$$

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

1.3 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

1.4 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

1.5 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

1.6 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

2. Result

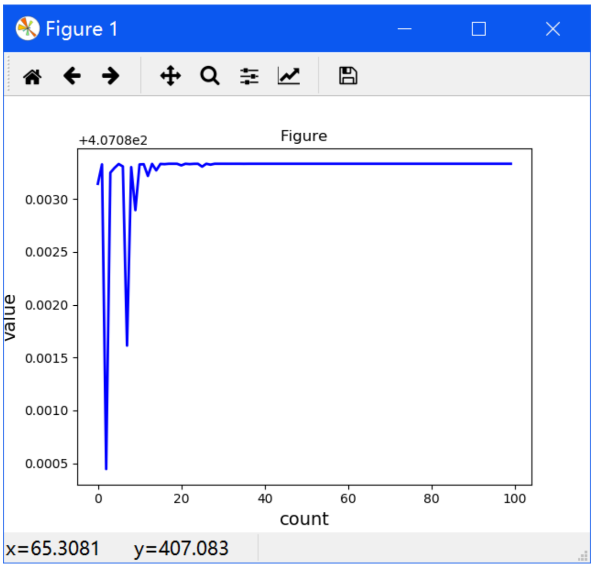

Figure 2.1 Graph

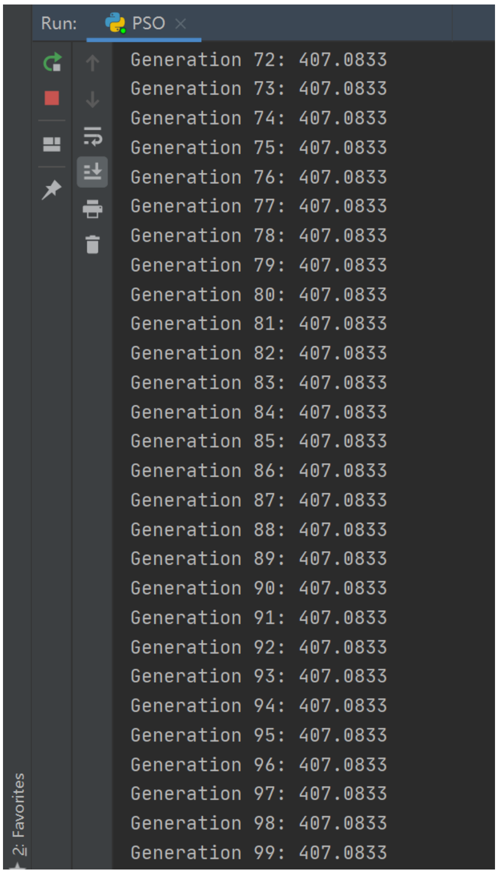

Figure 2.2 Max