1. Basic variables

Num of one generation=1000
Max iteration times=5000
Crossover rate=0.5
Mutation rate=0.1

2. Decode

I map a 16-bit binary code to a real number at the range of [-1,15], by using the two formulas below:

$$tmp = \sum_{i=0}^{15} 2^i * code[i]$$

$$decode = \frac{16}{2^{16}} * tmp - 1$$

the first one maps the whole code to a integer at the range of [0,65535], and the second maps this integer to a real number at the range of [-1,15].

3. Fitness function

First, I use the function (1), which is obviously a direct function to measure the fitness of the minimum.

$$f(x) = \begin{cases} -x * sin(x), if \ \ x * sin(x) < 0 \\ 0, \ \ otherwise \end{cases}$$

However, after I ran programs for several times, I saw the whole algorithm is not easy to reach the convergence in 5000 iterations, and another problem is that this program has a big probability to terminate at a local minimum instead of the correct result.

Therefore, I design the fitness function below, the effect of this function will be analyzed at the Experiment part.

$$f(x) = \frac{1}{(x * sin(x) + 15)^2}$$

4. End criterion

We think that we get an answer that fits this question precisely enough, if difference between the sum of one generation's all binary codes' fitness score and the best score is less than 0.001.

5. Crossover operator

In the class Generation, we have two private vectors, one records all the binary codes in this generation, and another one records all the decode results. When we need to do a crossover operation, we choose some binary codes by the given crossover rate. And once we choose a pair of binary code, we randomly decide a start position and an end position, then we swap all the bits between the start and the end.

6. Mutation operator

All the bits have a given mutation rate. We choose some bits randomly by the mutation rate. And for the chosen bit, we change them randomly in set {0,1}.

7. Selection operator

I use Roulette-Wheel Selection as the method of selection. Specifically, I choose codes by probability which can be calculate in formula below.

$$P(x_k) = \frac{f(x_k)}{\sum_{i=1}^{1000} f(x_i)}$$

8. Experiment

To test this program, I wrote a python file to test it automatically. This file will output two files. This script will automatically run the program 20000 times, and I uploaded to my Tencent cloud server to test it. The script will output two files, which are added to this folder.

“result.out”: shows all the results in 20000 times.
“exception.out”: shows all the results that not satisfy the requirement.

As we can see from these two files, at most of the time it will get the correct result. However, it occurred 58 times that did not satisfy the requirement. I think this error rate (0.0029) is acceptable, therefore, I think this fitness function works well.