Report on Computational Intelligence course of

Genetic Algorithm (GA)

Class: 1804 Student ID: 201894090 Name: Changrui Zuo

1. Overview of genetic algorithms

Genetic algorithm is a computational model that simulates natural selection and genetic mechanism of Darwinian evolution. It seeks the optimal solution by simulating the natural evolution process. The main feature of the algorithm is that it can directly calculate the objective function, and is not limited by the derivative and continuity of the function. It has good global optimization ability and usually uses probability model to find the optimal solution. It can automatically obtain and guide the optimized search space without defining rules. The search direction can be adjusted adaptively.

In this experiment, we need to use genetic algorithm to find the maximum and minimum of a given objective function:

$$y = 2x_1^2 - 3x_2^2 - 4x_1 + 5x_2 + x_3$$

I implemented the whole genetic algorithm based on Python and visualized the results. In the second part, I'll show the source code, and in the third part, I'll explain the functions in the source code.

2. Presentation of source code

About the source code, see file `GA.py`.

3. Explanation of source code

In this section, I will describe the implementation of my algorithm and the purpose of each function briefly.

In my algorithm implementation, I used a string of binary numbers to represent each person's gene. Each gene has 12 binary numbers. The first four digits represent the integral part and the last eight digits represent the decimal part. The purpose of the above code is to increase the accuracy of the calculation. Therefore, genotype and phenotype are one-to-one correspondence. We can easily calculate the phenotype of the gene, that is, the binary number corresponding to the decimal value. In the iterative process, phenotypes are calculated, and then fitness is calculated according to the selection function. Then, we eliminate some of the poor fitness values. According to the biological process, the genes of parents are exchanged and mutated. Parents are randomly selected, and children's genes are inherited from their parents. Once the offspring of the gene are obtained, the loop is repeated. It's an iterative process. In order to improve the execution speed, we need to control the number of individuals within a certain range. The size of the range is determined by a parameter called Population retention. By recording the extremum of each iteration, we can draw a curve. The maximum (minimum) of the curve is the maximum (minimum) of the objective function.

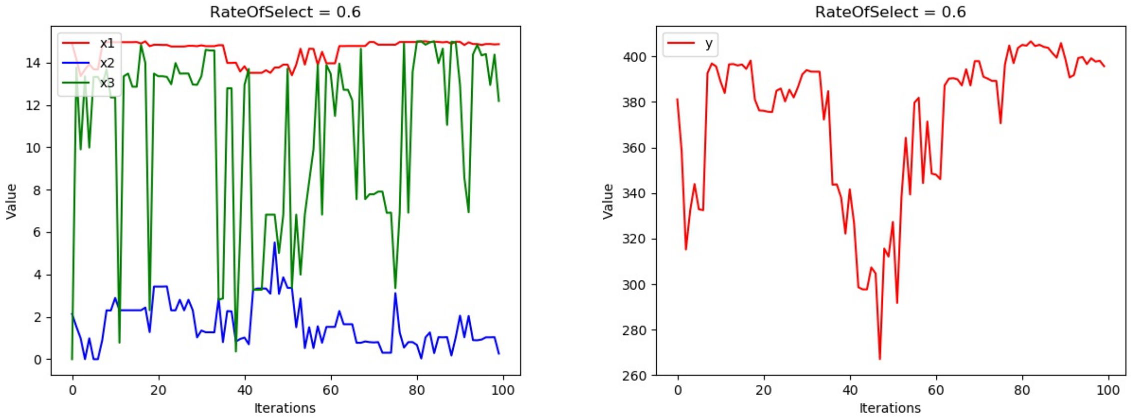
In the main function, the genotype of the population is initialized firstly. The initialization process is a completely random process and then it goes into the loop. The function `EncodeGenes` is to Initialize the genes of the population. And then we calculate the phenotype from the genotype. This is equivalent to a decoding process. The function `DecodeGenes` is to get the phenotype. The function `Fitness` is to calculate the fitness of each individual. For the selection function, I used the roulette algorithm. The function `CreateRoulette` is to create the roulette for selecting the individuals. In this function, I've improved a bit on some details. Traditional algorithms calculate the proportion of fitness to the population. After multiple iterations, the fitness of individuals will be more evenly distributed. So, I'm going to use the smallest fitness in the population as a baseline. Accordingly, function `SelectFromRoulette` extracts the evolved individual from the roulette. Then, function `CrossGene` is for chromosomal crossing while function `MutateGene` is for chromosomal variation. At last, the function `Inherit` merges the offspring into the population. In this way, the genetic algorithm is implemented.

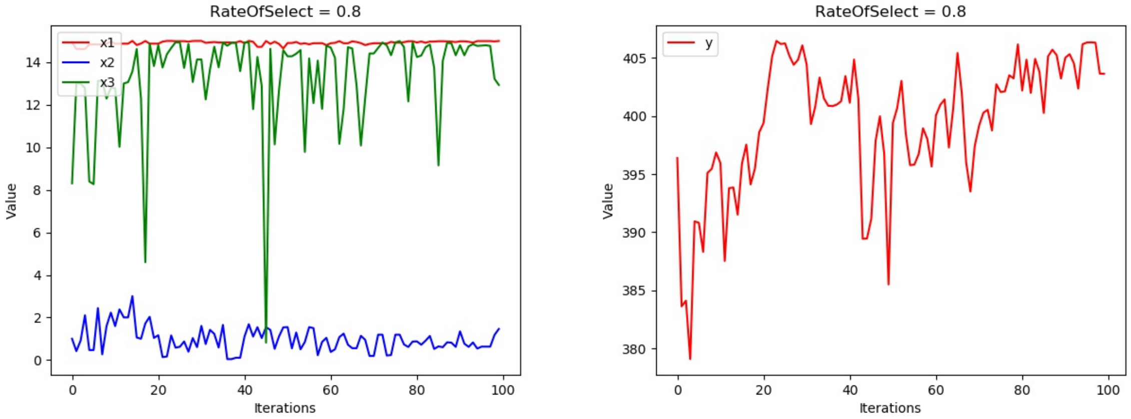
4. The results of my algorithm

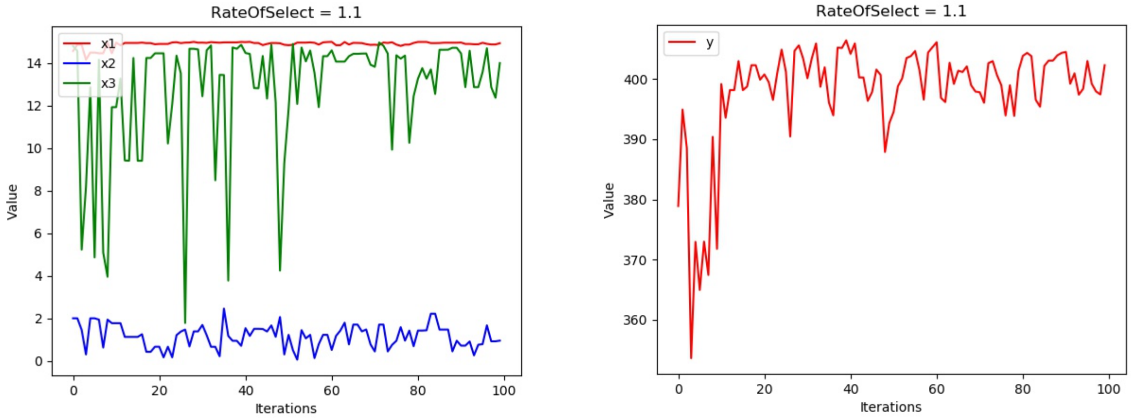
In my program, 4 parameters are set. It includes the number of iterations (`Iteration`), the ratio of population retention (`RateOfStay`), the ratio of selection times using the roulette (`RateOfSelect`) and the proportion of model selection in chromosome crossing (`RateOfModel`). In these parameters, `Iteration` is a fixed value of 100. I use the control variables method to discuss the effect of the other three parameters on the result.

By default, `RateOfSelect` was set as 1.1, `RateOfSelect` was set as 0.7 and `RateOfStay` was set as 8.

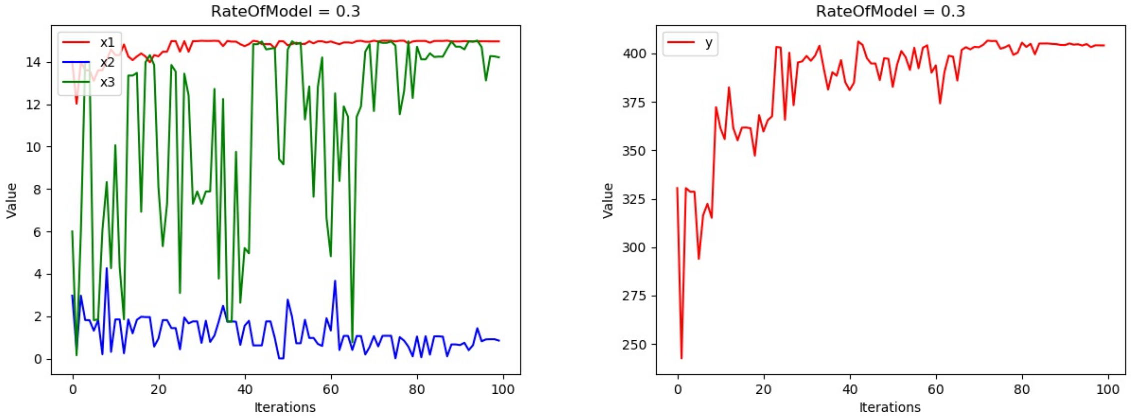
First, `RateOfSelect` is set as 0.6, 0.8 and 1.1. I'll display the results. We can draw the following conclusion from the figure: the higher the `RateOfSelect` is, the more stable the result will be. It should be noted that due to the fixed iterations, we can get the correct results in the end. The result is roughly as follows: when x_1 and x_3 are close to 15 and x_2 is close to 1, the function value is the largest which is about 406.

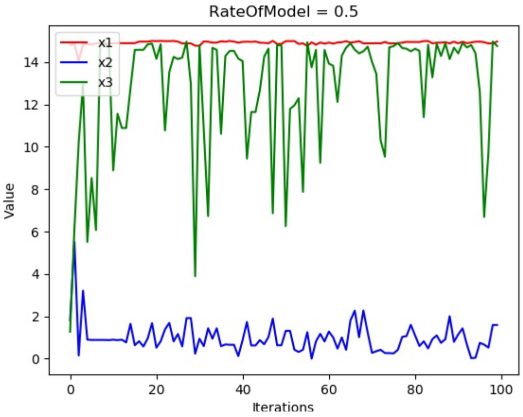


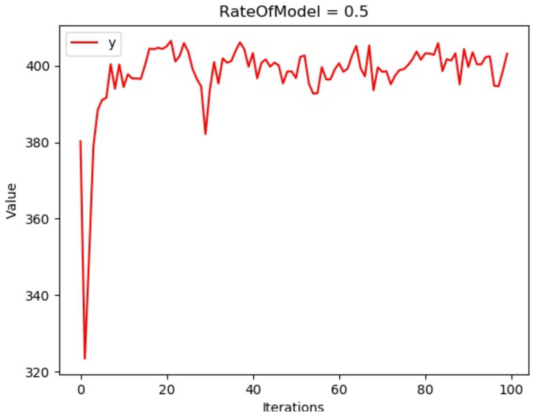


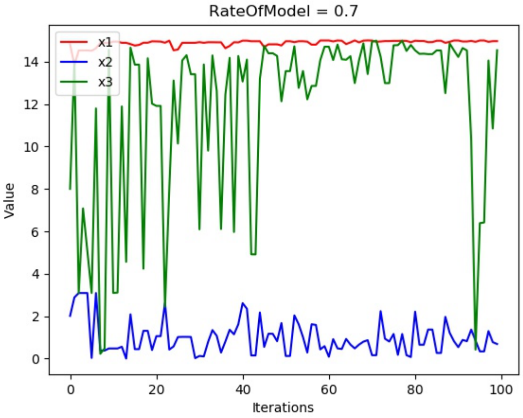


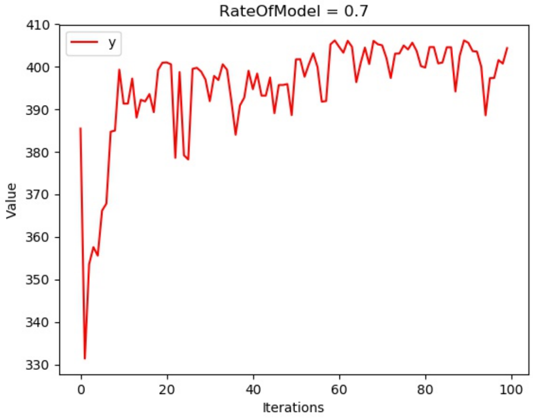
Next, `RateOfModel` is set as 0.3, 0.5 and 0.7. In general, the change of this parameter does not have a great influence on the results.



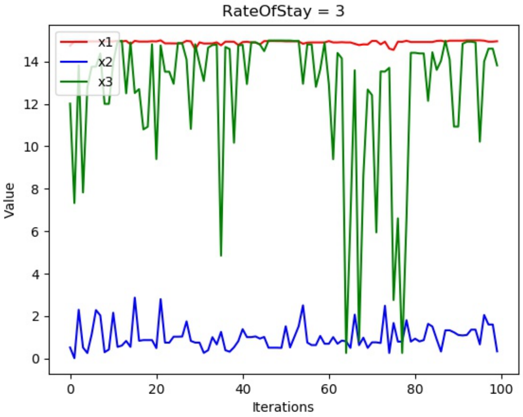


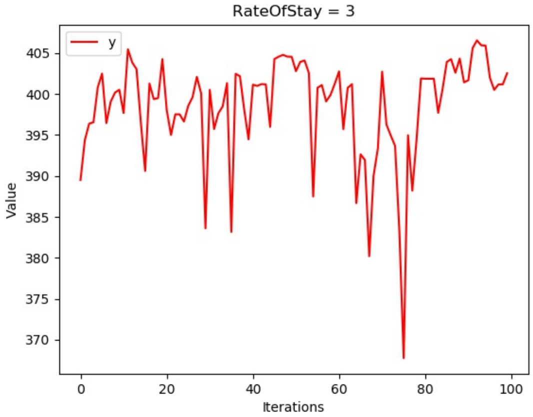


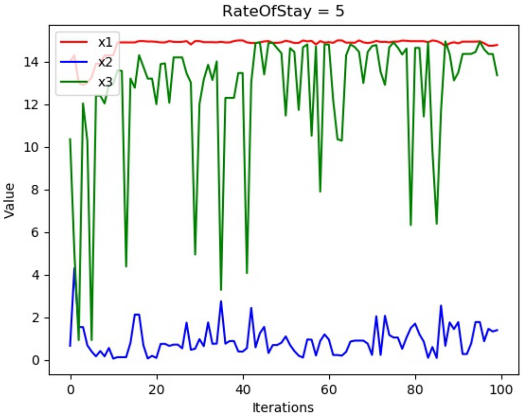


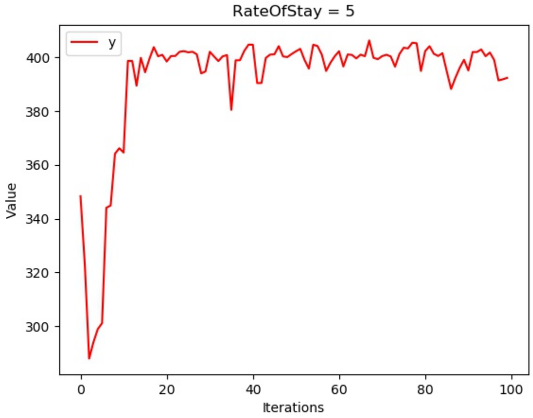


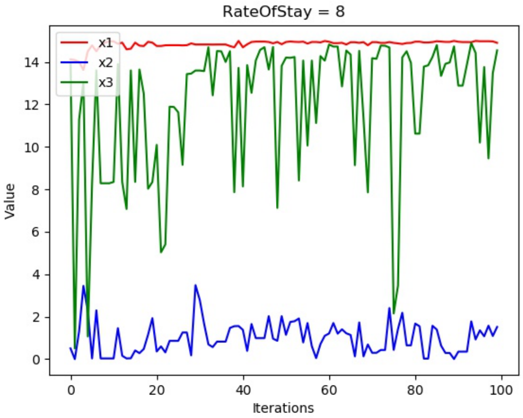
Last, `RateOfStay` is set as 3, 5 and 8. We can draw the following conclusion that the higher the `RateOfStay` is, the more stable the result will be. However, if this value is too large, it will affect the speed.

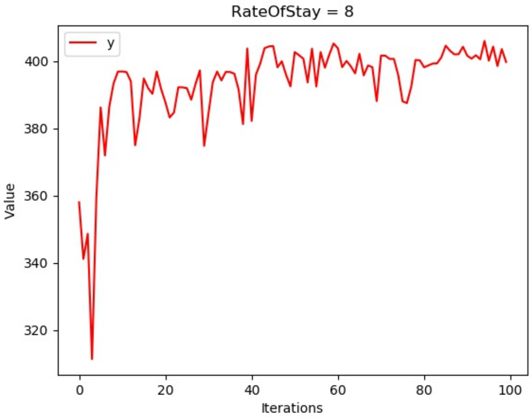












Finally, one of the maximum results we got is as following:

$$\begin{cases} x_1 = 14.9921875 \\ x_2 = 0.92578125 \\ x_3 = 14.4296875 \\ y = 406.0500030517578 \end{cases}$$

In the same way, one of the minimum results we got is as following:

$$\begin{cases} x_1 = 1.19921875 \\ x_2 = 14.9921875 \\ x_3 = 2.33203125 \\ y = -598.9247131347656 \end{cases}$$