INFO6205 Final Project Report

Find the maximum value of the multivariate function

Group 516
Xing Li 001873479
Haolin Xu 001821867

In daily life, there are many scenarios in which we need to find the best case to get the maximum or the minimum value of a function within a specific range. Such as minimization of transportation logistics costs, airline arrangement, workshop engineering process arrangement.
In our project, we try to use the genetic algorithm get the maximum value of a multivariate function.

To find a relatively optimized solution:

- Randomly generate an initial population, which is a collection of solutions of the mathematical problem.
- Use genetic algorithms to continuously evolve, which involves crossover and variation, the original inaccurate population.
- After a large number of evolutions, the chromosome, which is the solution we need, will gradually approaching the exact optimal solution.

Principle and Logic

- **Population**
  Population is a collection of individuals. The first generation is randomly generated and evolved toward better solutions during every generation. Set the size of population to 1000.

- **Chromosome (Individual)**
  1. Each individual is a solution to the function. It is a m x n matrix. The number of the rows m is equal to the number of variables, which is in binary form, in the mathematical function. Set the number of columns n to 31, which represents all positive integers.
  2. Each row is a 31-bit integer array. It is initialized by randomly replacing several bits 0 with 1. Convert to decimal before calculating the fitness.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- **Fitness**
  Fitness is the value of the objective function calculated from every individual. Since our goal is to get the local maximum value of the function, the larger the fitness is, the better the population is.

- **Crossover**
  Keep the first 500 individuals. Generate 500 new individuals from the first 500. Replace the second 500 ones with the new-generated 500 ones.

Generation a new individual:

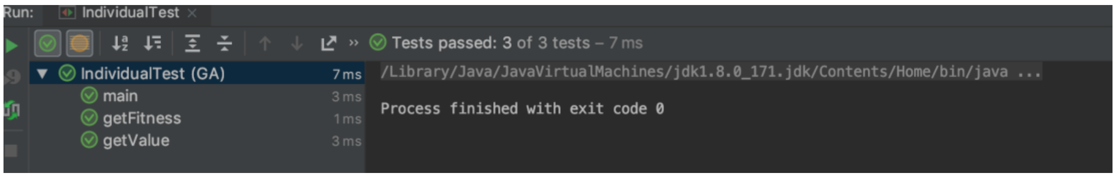
1. Randomly select two individuals from the first 500 ones to do crossover.
2. According to the preset crossover rate, exchange corresponding bits of every row of the two individuals to generate two new ones.
3. If these two integers exceed the preset valid range, invalid this operation and redo.

- **Mutation**
  Mutate every individual in one generation of population. For each bit in each row of an individual, according to the preset mutation rate, mutate it with 1 if it is 0 or with 0 if it is 1.

- **Evolve**
  1. Sort the population by the descending order according to every individual's fitness
  2. Implement a function that chooses a small number of individuals both from the first and the second 500 individuals to exchange to avoid local convergence
  3. Do crossover and mutation to get the next generation
  4. Stop the evolution after the change rate of fitness is less than the preset percentage for 500 generations or the current generation is larger than the maximum generation

Test cases:

- **IndividualTest:**

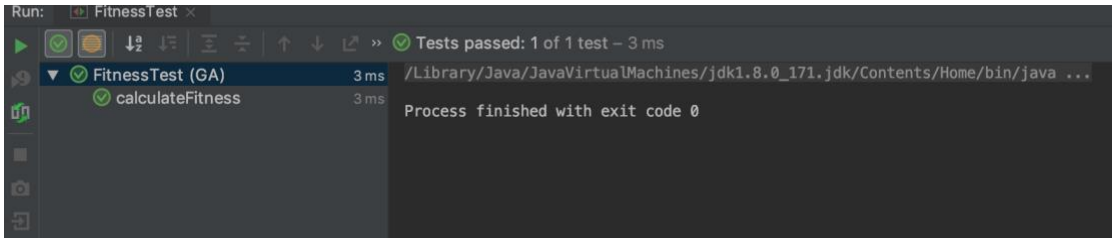


```
Run:  IndividualTest
Tests passed: 3 of 3 tests – 7 ms
IndividualTest (GA)   7 ms
  main                3 ms
  getFitness          1 ms
  getValue            3 ms

/Library/Java/JavaVirtualMachines/jdk1.8.0_171.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

- **FitnessTest:**

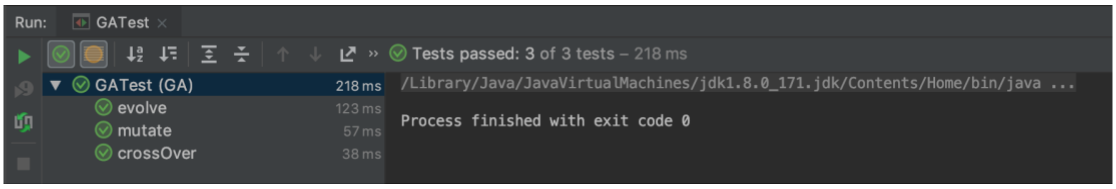


```
Run:  FitnessTest
Tests passed: 1 of 1 test – 3 ms
FitnessTest (GA)      3 ms
  calculateFitness    3 ms

/Library/Java/JavaVirtualMachines/jdk1.8.0_171.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

- **GATest:**



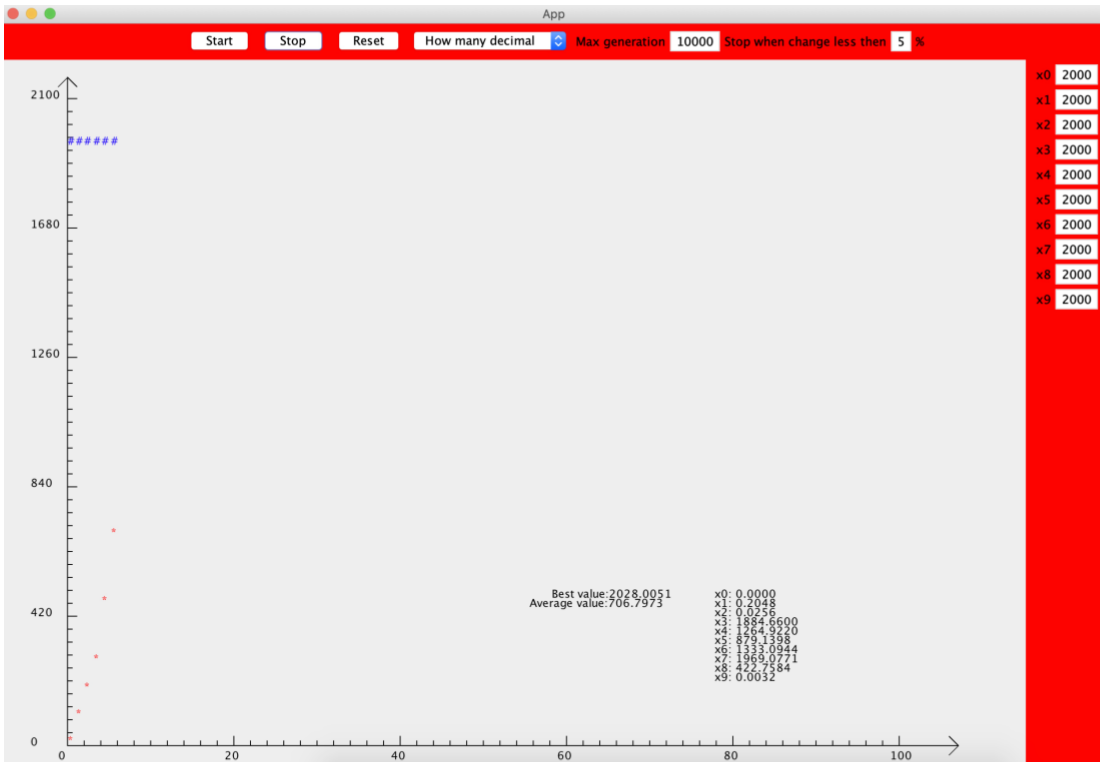
```
Run:  GATest
Tests passed: 3 of 3 tests – 218 ms
GATest (GA)         218 ms
  evolve            123 ms
  mutate             57 ms
  crossOver          38 ms

/Library/Java/JavaVirtualMachines/jdk1.8.0_171.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

Chart Interface:

Horizontal axis:	generation
Vertical axis:	fitness
Best value:	best fitness
x[i]:	the variables
Start Button:	start the evolution
Stop Button:	stop the evolution
How many decimals:	accuracy
Right side bar:	the range of x[i]



Max generation 10000 Stop when change less then 5 %

x0 2000
x1 2000
x2 2000
x3 2000
x4 2000
x5 2000
x6 2000
x7 2000
x8 2000
x9 2000

```
   Best value:2028.0051
Average value:706.7973
```

```
x0: 0.0000
x1: 0.2048
x2: 0.0256
x3: 1884.6600
x4: 1264.9220
x5: 879.1398
x6: 1333.0944
x7: 1969.0771
x8: 422.7584
x9: 0.0032
```

Test Functions:

Function 1:

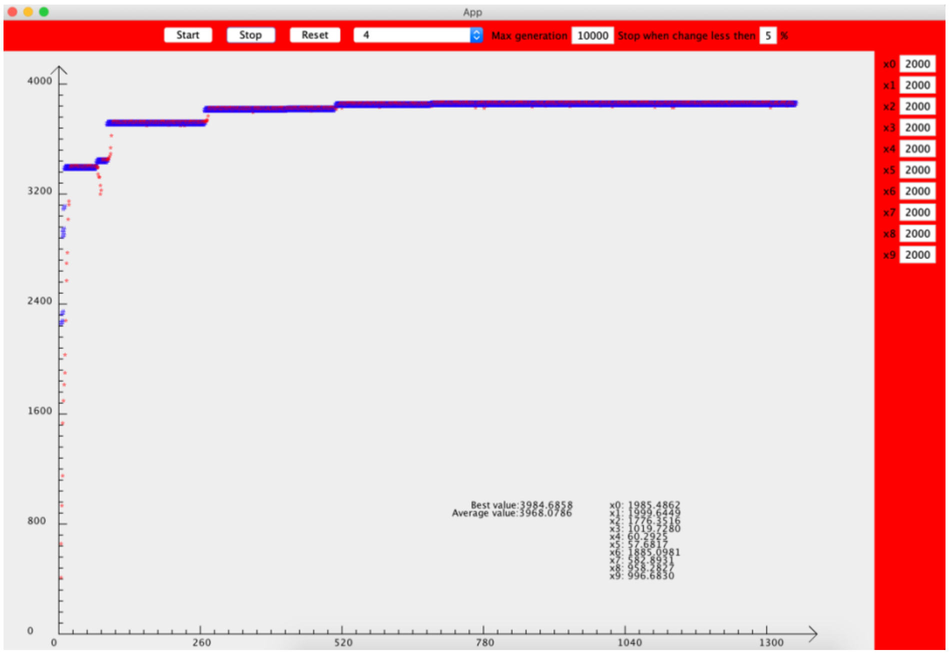
We choose a function:

$$z = x[0] * sin(x[1]) + x[1] * cos(x[0])$$

Result:

Maximum: 3984.6858

x[0] = 1985.4862, x[1] = 1999.6449



Max generation 10000 Stop when change less then 5 %

x0 2000
x1 2000
x2 2000
x3 2000
x4 2000
x5 2000
x6 2000
x7 2000
x8 2000
x9 2000

```
     Best value:3984.6858
Average value:3968.0786

x0: 1985.4862
x1: 1999.6449
x2: 1776.3516
x3: 1019.7280
x4: 60.2925
x5: 57.6817
x6: 1885.0981
x7: 582.8931
x8: 958.2827
x9: 996.6830
```

Function 2:

We choose a function:

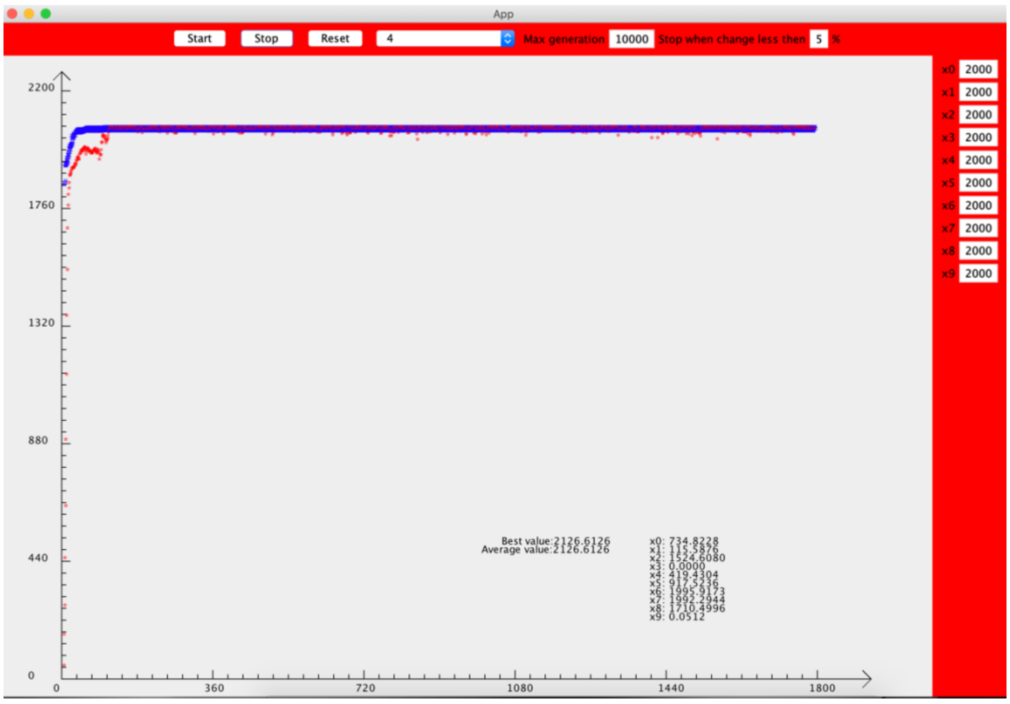
y = cos(x[5]) * sqrt(x[6]) – 100 * sin(x[0]^2) + pow(x[1]^ 2) - cos(x[3])) * sin(x[4]) + x[7] * sin(x[8]) - |(x[9])|;

Result:

Maximum: 2126.6126

x[0] = 734.8228, x[1] = 115.5876, x[3] = 0, x[4] = 419.4304, x[5] = 917.5236

x[6] = 1995.9173, x[7] = 1992.2944, x[8] = 1710.4996, x[9] = 0.0512



Max generation 10000 Stop when change less then 5 %

x0 2000
x1 2000
x2 2000
x3 2000
x4 2000
x5 2000
x6 2000
x7 2000
x8 2000
x9 2000

```
     Best value:2126.6126
Average value:2126.6126

x0: 734.8228
x1: 115.5876
x2: 1524.6080
x3: 0.0000
x4: 419.4304
x5: 917.5236
x6: 1995.9173
x7: 1992.2944
x8: 1710.4996
x9: 0.0512
```

Function 3:

We choose a function:

$$y = 100^{*} \sin(x[0] + x[1] + x[2] + x[3] + x[4] + x[5] + x[6] + x[7] + x[8] + x[9])$$

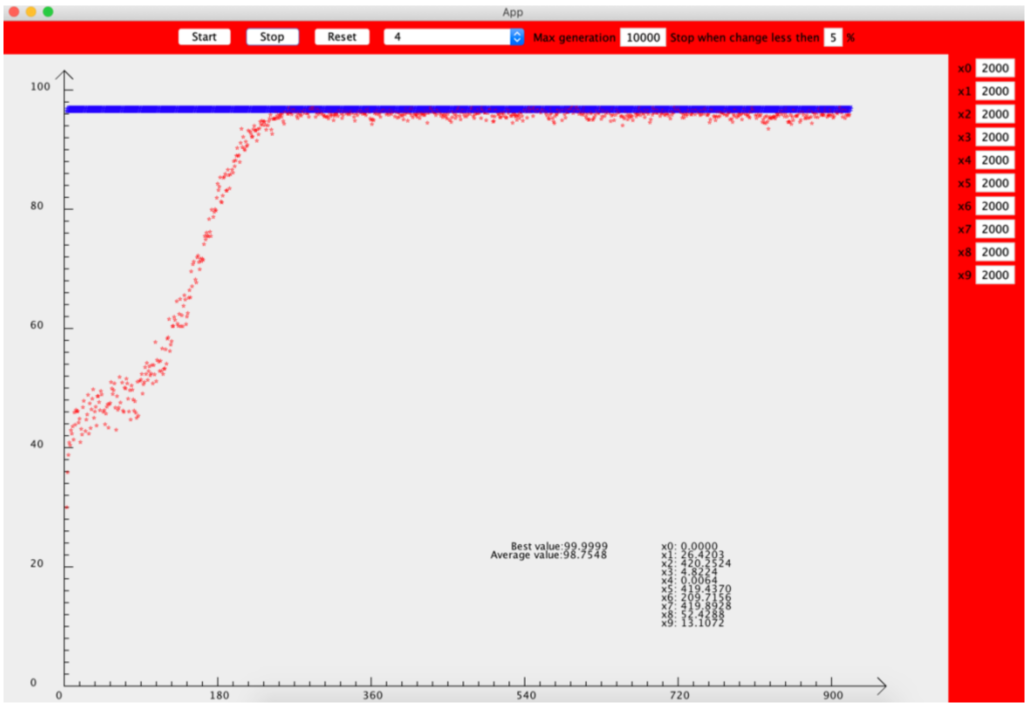
Result:

Maximum: 99.9999

x[0] = 0, x[1] = 26.4203, x[2] = 420.2524, x[3] =4.8224,

x[4] = 0.0064, x[5] = 419.4370, x[6] = 209.7156,

x[7] = 419.8928, x[8] = 52.4288, x[9] = 13.1072



4 Max generation 10000 Stop when change less then 5 %

x0 2000
x1 2000
x2 2000
x3 2000
x4 2000
x5 2000
x6 2000
x7 2000
x8 2000
x9 2000

```
     Best value:99.9999
Average value:98.7548
```

```
x0: 0.0000
x1: 26.4203
x2: 420.2524
x3: 4.8224
x4: 0.0064
x5: 419.4370
x6: 209.7156
x7: 419.8928
x8: 52.4288
x9: 13.1072
```

• Conclusion

1. After three function tests, we can get the relatively accurate maximum value of the function we choose.
2. If a function has a small number of peaks, the best fitness value in a generation of population will have a fast convergence, and the average fitness of the population increase smoothly.
3. If a function has a large number of peaks, the best fitness value in a generation of population may converge to a local best value. The population will jump to a better value due to the mutation function.