CO528 Assessment 1: Genetic Algorithms

S.O.Bowie so253@kent.ac.uk
P.Ghassemi pg272@kent.ac.uk
S.J.Pengelly sp611@kent.ac.uk

March 3, 2016

Introduction

The task was to produce a curve-fitting genetic algorithm (GA) that would find, given a limited set of data points, the six coefficients a to f of the function that produced the points, which is of the form: $f(x) = a + bx + cx^2 + dx^3 + ex^4 + fx^5$

The input data comprised 577 double-precision coordinate pairs, with x-values ranging from -25.0 to $+900.0$ and y-values ranging from approximately -1.55×10^6 to $+5.30 \times 10^{10}$. A plot of the input data can be seen in Figure 1.

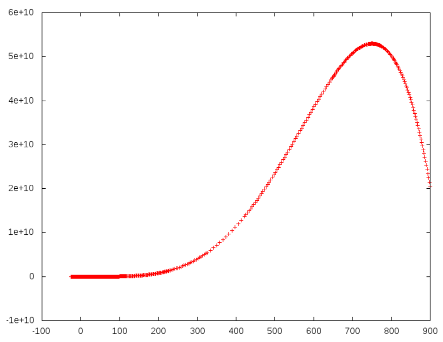


Figure 1: Plot of input data

Implementation

Encoding

The population comprised a number of individuals, each of which encoded a candidate solution in its genome. The genome for each individual consisted of six genes, one for each of the six coefficients a to f in the provided function (see **Introduction**).

Each gene stored its value as a double-precision floating-point number, which could be mutated in two different ways: 90% of the time, a gene mutation will increase or decrease the value of the coefficient by 5%, but the remaining 10% of the time, the gene will be assigned a completely random value. The value of the coefficient is constrained to a chosen range (see **Configuration**), and if a mutation causes the value to go out of range, the value will instead wrap around.

Configuration

Each gene is assigned a minimum and maximum value. The initial random number generation and subsequent mutations will not fall outside of this range. The coefficient ranges and GA parameters are shown in Figure 2.

Parameter	Value
Population size	50
Number of generations	200000
Mutation rate	0.2
Survivor ratio	0.5
Elite ratio	0.2
Number of crossover points	3

Coeff.	Valid range
a	-2.0×10^6 to $+2.0 \times 10^6$
b	-1.0×10^6 to $+1.0 \times 10^6$
c	-5.0×10^5 to $+5.0 \times 10^5$
d	-100 to $+100$
e	-10 to $+10$
f	-10 to $+10$

Figure 2: Values used in the configuration of the genetic algorithm

Execution

After the algorithm has been configured, each individual in the population is initialised with a random set of genes and the algorithm begins. In each iteration, mutations are performed with the configured probability on each individual, and then the population competes to produce offspring for the next generation (see **Selection**). This is repeated for the desired number of iterations. Upon completion, the current fittest individual is selected and a set of data points are produced using this individual's coefficients.

Selection

Individuals were selected using a hybrid of *Tournament Selection* and *Roulette Wheel Selection*. Firstly, the population is sorted according to individual fitness (see **Fitness Evaluation**) and then some percentage of the fittest individuals are guaranteed survival (elitism), while the remaining individuals compete via *Roulette Wheel Selection* for the remaining places. Once the final pool of survivors has been determined, these individuals are partnered at random and a number of offspring are produced until the population is back to its original size.

Crossover

The algorithm produces 'offspring' by performing a multi-point crossover operation between two 'parent' individuals. This will insert a number of crossover points at random between the genes (up to a defined maximum) and select which genes are to be used in the offspring's genome based on the location of these crossover points. An example is given in Figure 3 below:

Parent A	3123032.83	954.92	254.120	75.0	1.68	0.0
Parent B	3493482.115	293.0231	39.8554	72.5	6.02	2.1
Child	3493482.115	954.92	39.8554	75.0	1.68	0.0

Figure 3: Example of multi-point crossover operation

Fitness Evaluation

Individual fitness was evaluated by comparing the output of the function encoded by the individual with the known data points given as input. The fitness value is the square of the difference between the expected and actual y-values, for each x-value in the input:

$$\sum_{i=1}^{n} \frac{(y_i - f(x_i))^2}{n}$$

where $f(x)$ is the candidate solution encoded by the individual, and (x_1, y_1) to (x_n, y_n) refer to points in the input data set. The square of the difference is divided by n (the total number of data points) in an attempt to reduce the unnecessary size of 'bad' fitness values.

Conclusion

The initial version of the algorithm mutated genes by flipping a single random bit in the binary representation of the value that the gene encoded. This proved to be ineffective, and by the final version of the code, the algorithm used the hybrid mutation technique menioned in the **Encoding** section. Restricting mutations to a range of values greatly helped to guide the algorithm towards the solution. In the end, the best solution found by the genetic algorithm, plotted in Figure 4, was:

$$\begin{aligned} f(x) = 3999717.984336 + 999806.423681x \\ -43797.490059x^2 + 79.002418x^3 \\ +0.849185x^4 - 0.000948x^5 \end{aligned}$$

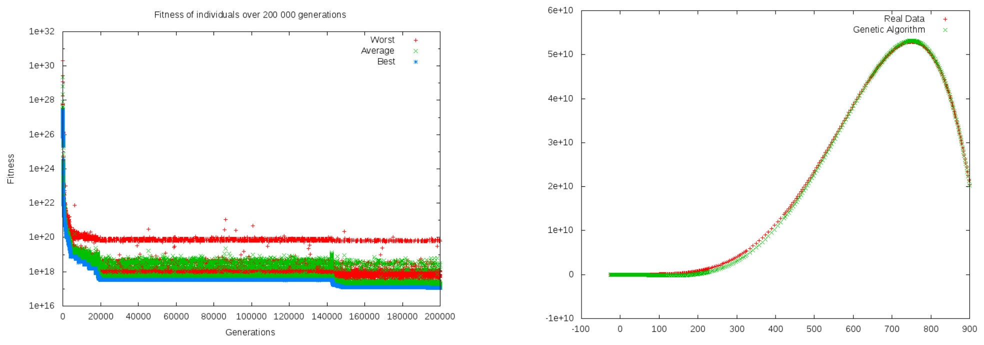
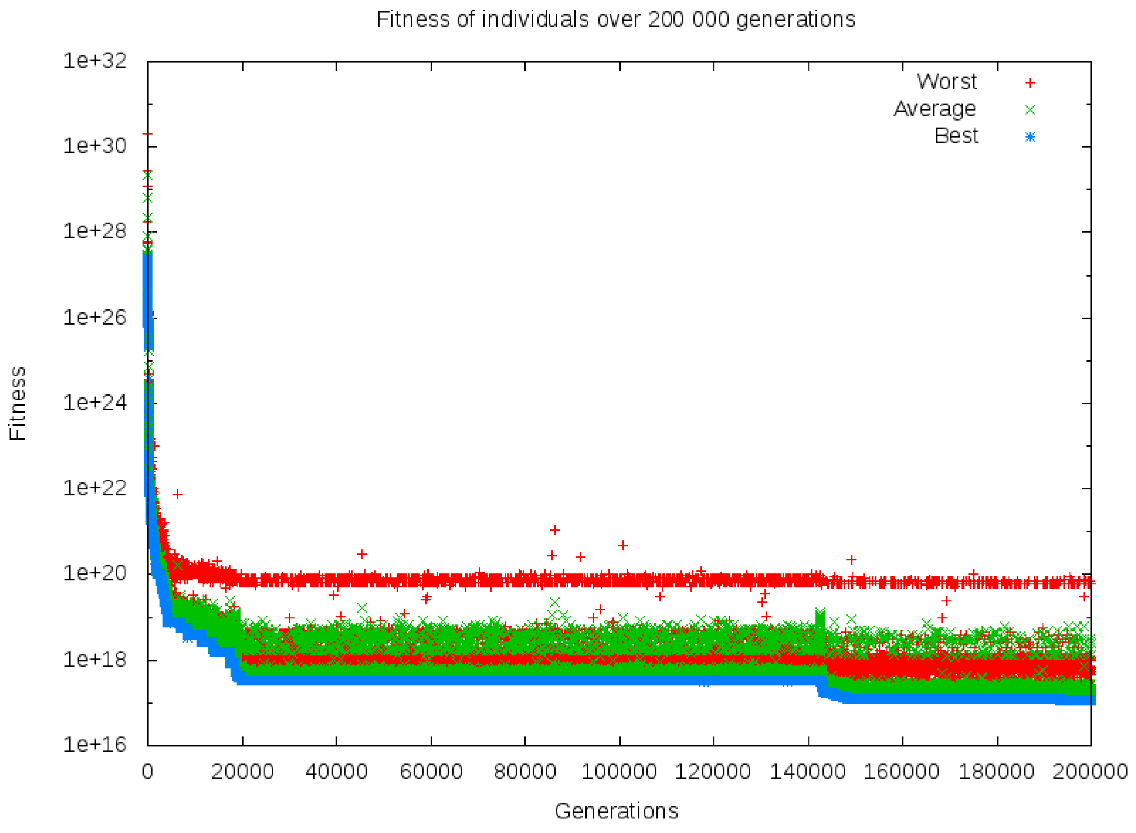


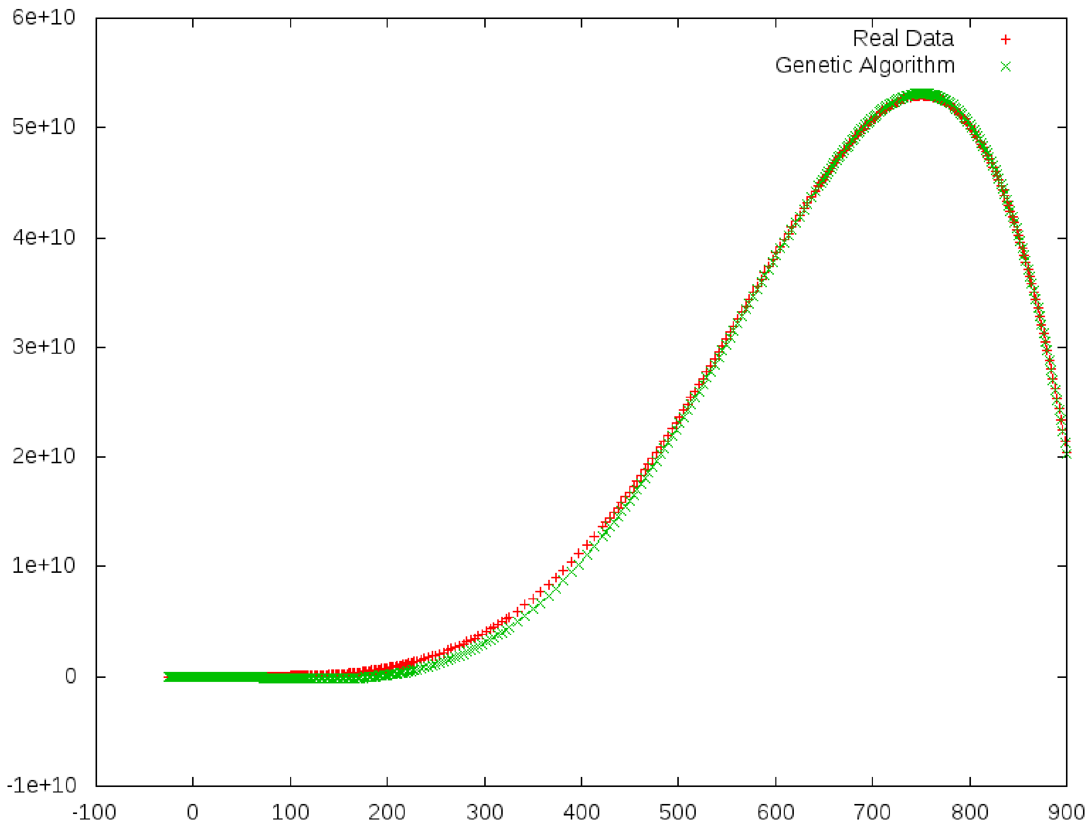
Figure 4: Plots of fitness change over time (left), and best solution (right)

Larger versions of the plots, and the program source code have been included as appendices 1 and 2 respectively.

Appendix 1



Fitness of individuals over 200 000 generations
Worst
Average
Best
Fitness
Generations
1e+32
1e+30
1e+28
1e+26
1e+24
1e+22
1e+20
1e+18
1e+16
0
20000
40000
60000
80000
100000
120000
140000
160000
180000
200000



Real Data
Genetic Algorithm
6e+10
5e+10
4e+10
3e+10
2e+10
1e+10
0
-1e+10
-100
0
100
200
300
400
500
600
700
800
900