

# MECS 4510 HW2

Name: Jiong Lin

UNI: jl6017

Instructor: Hod Lipson

Grace hour used: 0

Grace hour remained: 96

# 1 Result summary

## 1.1 Result table and graph

|                             | Evaluations | MAE    |
|-----------------------------|-------------|--------|
| Random search               | 50000       | 0.6113 |
| Hill climber                | 50000       | 0.7187 |
| GP (conventional)           | 50000       | 0.1696 |
| GP (niching)                | 50000       | 0.0677 |
| GP (deterministic crowding) | 50000       | 0.0430 |

Table 1: Result summary

The solution listed here is simplified from a six level tree, and might have rounding error.

$$y = (0.05x - 1.96\sin x - 1.09) * \frac{2.1 + x}{2.3 + x} * \sin x + \frac{2x + 3.6}{\sin x + 3.3}$$

The graph is plotted from a list of points (share the same x positions with the given dataset)

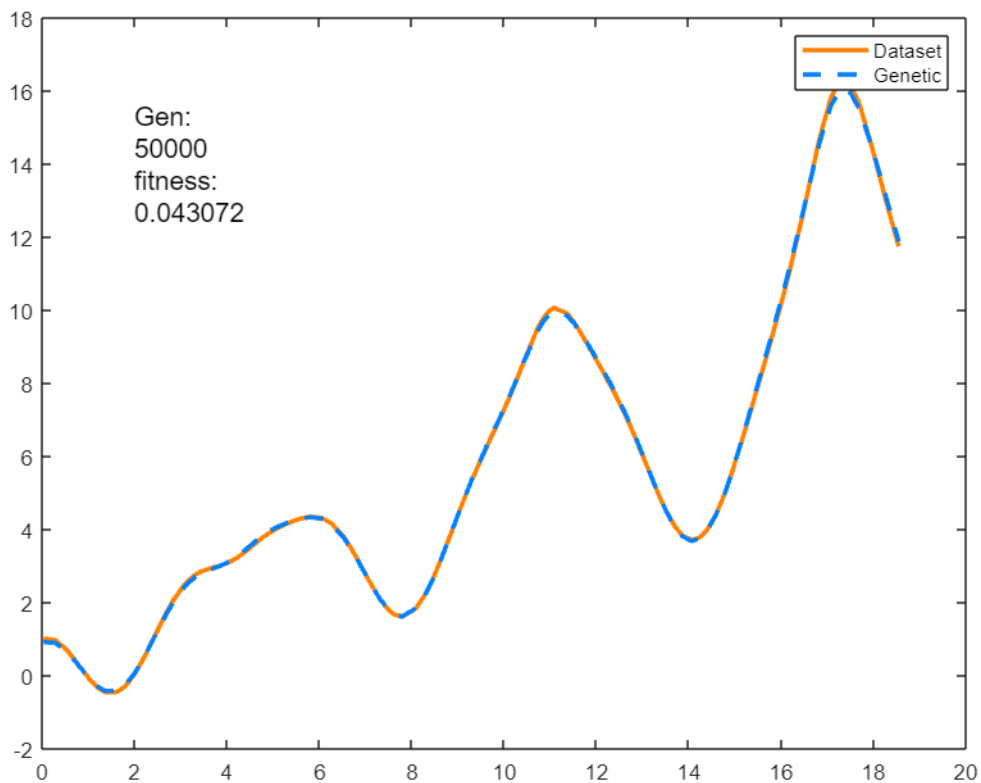


Figure 1: Dataset and best fit curve

## 1.2 Movie link:

[Genetic program for symbolic regression](#)

## 2 Methods

In this homework, we are required to use genetic program to do the symbolic regression. Basically, I used c++ to calculate the evaluations and write the results into CSV files, and then used python or matlab to read files and plot them into graphics.

Although some code from last homework we can still use for this assignment. The key point to have good performance is the representation and the methods to maintain the diversity.

### 2.1 Representation

Binary tree is an applicable data structure to represent the function. It is easier to do the crossover process in a tree structure than a list, as it naturally avoid the repetition problem. When calculating the value of a function, we need to use a recursive manner. Reading the tree in a reverse order also works. In this assignment, there are six possible operators (sin, cos, +, -, \*, /), and three possible end nodes (x, float number, and NULL).

Instead of using char strings to represent the nodes, which I tried at the beginning, I used the big integers, such as 50, 60, to represent the operators, such as + and -. It is almost zero chance for a random float to evolve into a big integer. In this way, the tree can become a float tree, and the fitness-calculating process can be much faster.

### 2.2 Random search

1. Generate a random function represented by a binary tree.
2. Calculate the fitness between the function and the given dataset.
3. If it is smaller than the last fitness, then keep it as the best function.
4. Back to the top and form the loop.

### 2.3 Hill climber

1. Generate a function represented by a binary tree
2. Calculate the fitness between the function and the given dataset.
3. If it is smaller than the last fitness, then keep it as the best function.
4. Use mutation method to get the new generation of function. Form the loop.

The difference between hill climber and random search is that hill climber uses mutation to get the new function. They are both hopeless for this task. RMHC has a group of functions and uses the selection method. It has better performance than simple hill climber, but not as good as GP.

### 2.4 Genetic program

1. parent genes: get 100 random functions represented by binary trees.
2. Offspring genes: crossovers and mutations and get a new generation of functions.
3. Calculate the fitness of each genes and sort them.
4. Select top 50%(or use roulette method) as the next generation. Form the loop.

#### 2.4.1 variation operators

**Crossover:** randomly select two valid nodes from two function trees, which are the parent trees. Swap the two subtrees under this two nodes. Then we get two new function trees, which are the offspring trees.

**Mutation:** There are many situations need to be covered in the mutation process. Also, the desired mutation needs to have similar chance to increase the subtree height and to decrease the height. The method I used in this assignment is to select a node in the original function tree, and then replace the subtree with a new small tree (here we can reuse the tree-creating function). The small tree can be a float number (one level tree), and can also be a small function (two or three level tree).

## 2.4.2 selection methods

### Conventional genetic program:

I used the basic truncation method this time. First I sort the group of genes by their fitness. Then I select the top 50% of the genes as the next generation.

### Genetic program with niching:

Both niching and deterministic crowding are methods used to maintain the diversity during the evolution. The basic idea of niching is to have several groups evolve separately and have migration occasionally. In this assignment, I have five groups evolve separately and have migration every 200 generations. The migration process is to choose 10 individuals from one group and exchange with another group. The selection method I used for niching is the same as conventional genetic program.

### Genetic program with deterministic crowding:

The basic idea is:

```
1 if (d(p1,c1)+d(p2,c2) < d(p1,c2)+d(p2,c1)) {
2     compare c1 to p1 and c2 to p2 and replace parents if offspring better
3 }else{
4     compare c1 to p2 and c2 to p1 and replace parents if offspring better
5 }
```

The selection method of deterministic crowding is different with conventional genetic program. Because the selection has been done during the evolution process. And the selection pressure is 50%.

## 2.5 Result analysis

We can see from the learning curves that:

1. Random search and simple hill climber don't get a good fitness.
2. Genetic programs with diversity methods run better than conventional genetic program.
3. Genetic program with deterministic crowding evolves slowly at the beginning and gets the best fitness at last.
4. What worked: variation methods such as mutation and crossover; diversity maintaining methods such as niching and deterministic crowding.

Reasons:

1. Random search and hill climber don't have good connection between parent and offspring.
2. Genetic programs have better performance than random search and hill climber because of the recombination. Crossover is a good variation operator that has a strong link between parent and offspring.
3. In this assignment, the diversity of conventional genetic program drops very fast. And therefore the methods to keep the diversity, such as niching and deterministic crowding, can improve the performance of genetic program.

### 3 Performance plot

#### 3.1 Learning curves

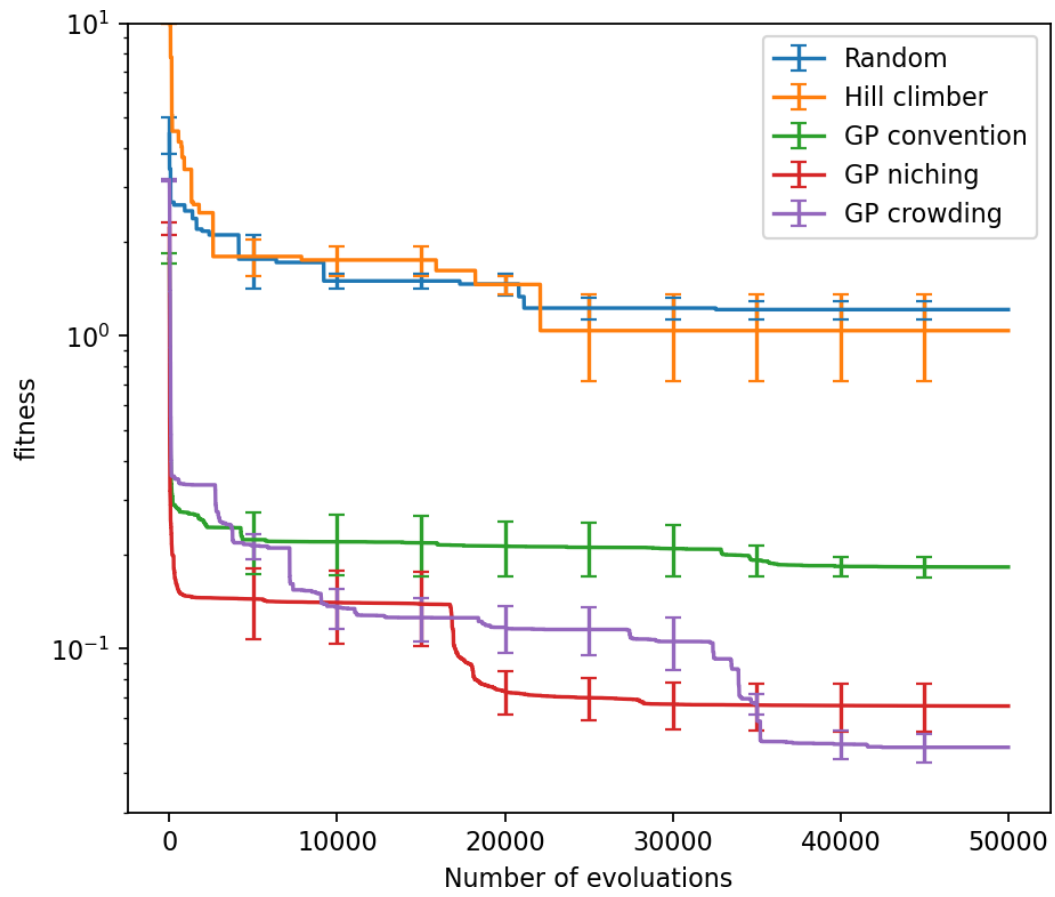


Figure 2: Learning curves with error bars

### 3.2 Dot plot

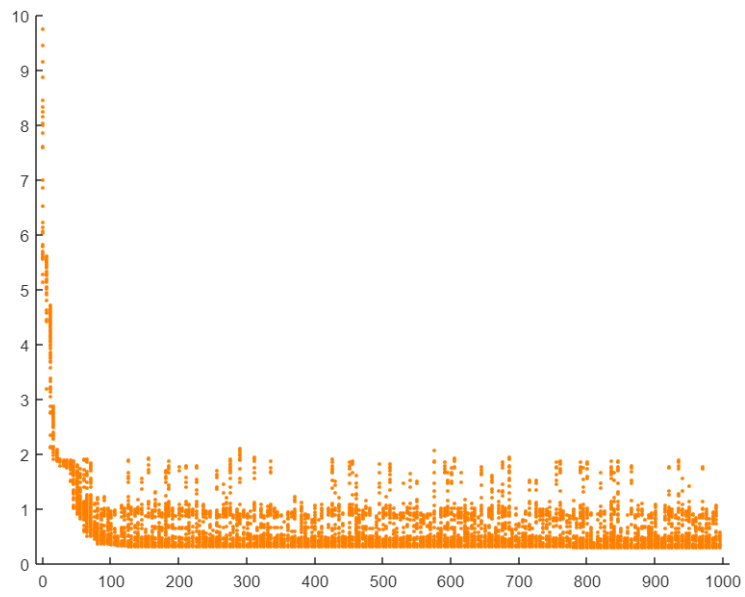


Figure 3: Dot plot for conventional genetic program

### 3.3 Diversity plot

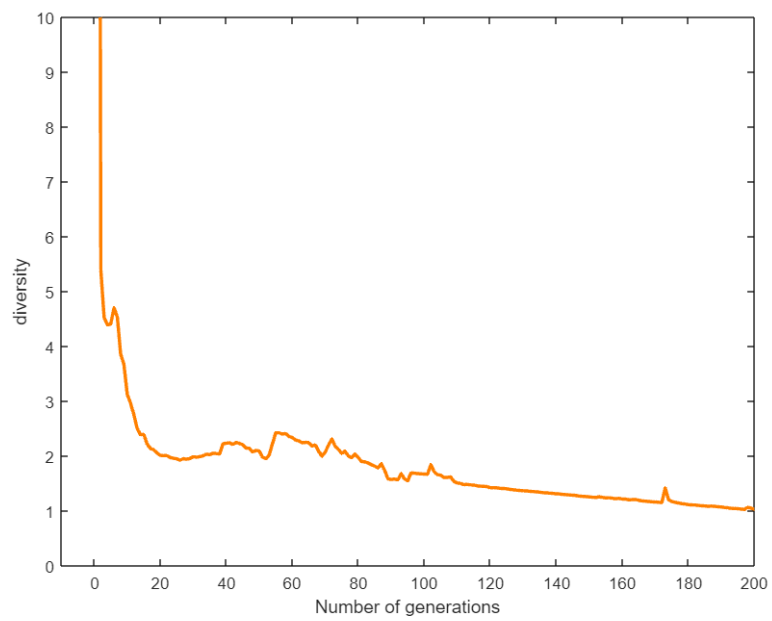


Figure 4: Diversity plot for conventional genetic program

### 3.4 Convergence plot

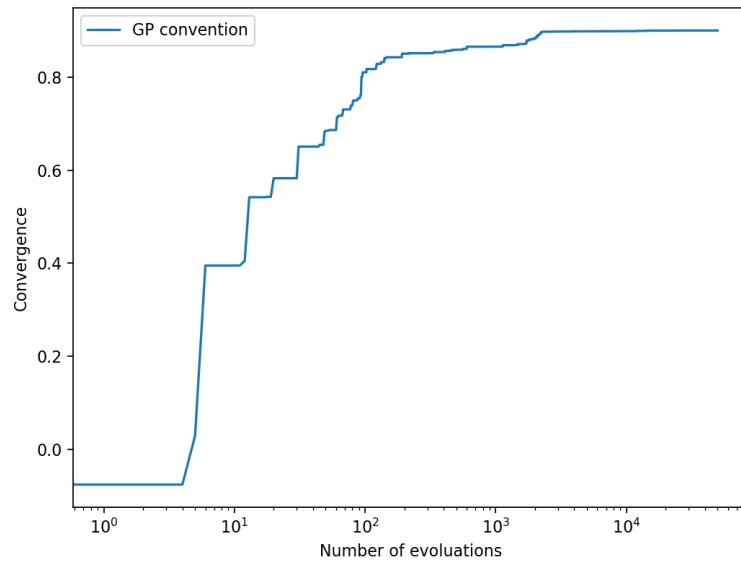


Figure 5: Convergence plot for conventional genetic program (Threshold=0.1)

### 3.5 Simple task test for debugging

I generate 100 points based on the function  $y = \sin(x) + 0.5x$ . Conventional genetic program can find the result in 200 generations. Also, we can set the largest height as 3 when doing the test, so that the program can run fast.

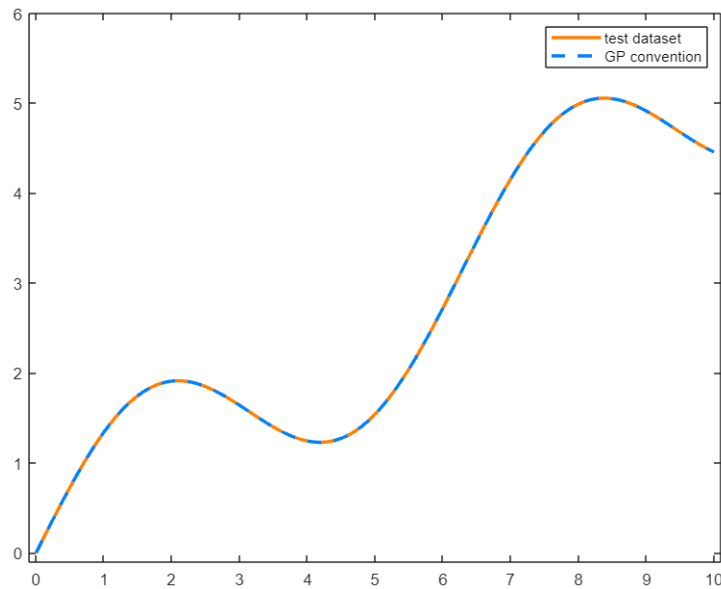


Figure 6: Debug test dataset plot

### 3.6 Automatically generated tree

As mentioned before, I used the big integers to represent the operators and values. It is almost zero chance for a random float to evolve into a big integer. The tree can become a float tree, and the fitness-calculating process can be much faster.

| Numbers                             | operators and values |
|-------------------------------------|----------------------|
| 30                                  | sin                  |
| 40                                  | cos                  |
| 50                                  | +                    |
| 60                                  | -                    |
| 70                                  | *                    |
| 80                                  | /                    |
| 90                                  | x                    |
| 100                                 | NULL                 |
| others (usually between -20 and 20) | random float numbers |

Table 2: Set the matchups

```

29998: 0.0677059
29999: 0.0677059
50
50:60
70:60:80:80
70:70:80:60:50:50:50:60
30:30:50:80:50:70:30:50:60:50:30:50:60:50:30:70
90:100:90:100:90;-6.55361;-2.01288;-6.83004;-6.00961;90;-4.93576;4.88571;-1.70104;100:90;-0.0429087;-1.18549;-8.1586;90;
-6.55361;90;100;-5.30375;3.17154;-1.18549;-8.1586;90;-6.55361;4.6242;100;90;8.69399
-----

```

Figure 7: Automatically generated tree representing one of the solution



## 4 Appendix

### 4.1 Random search and hill climber c++

```
1 #include <iostream>
2 #include <fstream>
3 #include <iomanip>
4 #include <sstream>
5 #include <string>
6 #include <math.h>
7 #include <vector>
8 #include <random>
9 #include <ctime>
10 #include <stdlib.h>
11 #define LOG(x) std::cout<<x<<std::endl
12 using namespace std;
13
14 const int startHeight = 6;
15 const int pointNum = 1000;
16 const int fitNum = 200;
17 const int gen = 1000;
18 const int groupSize = 100;
19 const int halfSize = groupSize / 2;
20 const int chance = 30;
21 const int mysize = pow(2, startHeight) - 1;
22
23 void readFile(float x[], float y[]) {
24     ifstream inFile("data.txt", ios::in);
25     if (!inFile)
26     {
27         cout << "fail to open " << endl;
28         exit(1);
29     }
30     int i = 0;
31     string line;
32     string field;
33     while (getline(inFile, line))
34     {
35         string field;
36         istringstream stin(line);
37         getline(stin, field, ',');
38         x[i] = atof(field.c_str());
39         getline(stin, field, ',');
40         y[i] = atof(field.c_str());
41         i++;
42     }
43     inFile.close();
44 }
45
46 void create(vector<float>& myExp, int height) {
47     //srand(time(NULL));
48     //write the bottom nodes
49     for (int i = pow(2, height - 1) - 1; i < pow(2, height) - 1; i = i + 2) {
50         int r1 = rand() % 6;
51         float r2 = rand() / float(RAND_MAX) + rand() % 19 - 10;
52         float r3 = rand() / float(RAND_MAX) + rand() % 19 - 10;
53         switch (rand() % 6)
54         {
55             case 0:myExp[i] = 100; myExp[i + 1] = 100; break;
56             case 1:myExp[i] = 90; myExp[i + 1] = r2; break;
57             case 2:myExp[i] = r2; myExp[i + 1] = 90; break;
58             case 3:myExp[i] = 90; myExp[i + 1] = 100; break;
59             case 4:myExp[i] = r2; myExp[i + 1] = r3; break;
60             case 5:myExp[i] = r2; myExp[i + 1] = 100; break;
61         }
62     }
63     for (int h = height - 1; h > 1; h--) {
64         float r4 = rand() / float(RAND_MAX) + rand() % 19 - 10;
65         for (int i = pow(2, h - 1) - 1; i < pow(2, h) - 1; i++) {
66             if (myExp[2 * i + 1] == 100 && myExp[2 * i + 2] == 100) {
67                 switch (rand() % 3)
68                 {
69                     case 0:myExp[i] = 100; break;
```

```

70         case 1:myExp[i] = 90; break;
71         case 2:myExp[i] = r4; break;
72     }
73     }
74     else if (myExp[2 * i + 1] == 100 || myExp[2 * i + 2] == 100) {
75         switch (rand() % 2)
76         {
77             case 0:myExp[i] = 30; break;
78             case 1:myExp[i] = 40; break;
79         }
80     }
81     else {
82         int r = rand() % 4;
83         switch (r)
84         {
85             case 0:myExp[i] = 50; break;
86             case 1:myExp[i] = 60; break;
87             case 2:myExp[i] = 70; break;
88             case 3:myExp[i] = 80; break;
89         }
90     }
91 }
92 }
93
94 // write the top nod
95 if (myExp[1] == 100 && myExp[2] == 100) {
96     myExp[0] = rand() / float(RAND_MAX) + rand() % 19 - 10;
97 }
98 else if (myExp[1] == 100 || myExp[2] == 100) {
99     switch (rand() % 2)
100    {
101        case 0:myExp[0] = 30; break;
102        case 1:myExp[0] = 40; break;
103    }
104 }
105 else {
106     int r = rand() % 4;
107     switch (r)
108     {
109         case 0:myExp[0] = 50; break;
110         case 1:myExp[0] = 60; break;
111         case 2:myExp[0] = 70; break;
112         case 3:myExp[0] = 80; break;
113     }
114 }
115 }
116 }
117
118 float calculate(vector<float>& myRes, float x) {
119     for (int i = pow(2, startHeight) - 2; i > 0; i = i - 2) {
120         if (myRes[i] == 100 && myRes[i - 1] == 100) {
121             continue;
122         }
123         else if (myRes[i] == 100) {
124             if (myRes[(i - 2) / 2] == 30) {
125                 if (myRes[i - 1] == 90) {
126                     myRes[i - 1] = x;
127                     myRes[(i - 2) / 2] = sin(x);
128                 }
129                 else {
130                     myRes[(i - 2) / 2] = sin(myRes[i - 1]);
131                 }
132             }
133             else {
134                 if (myRes[i - 1] == 90) {
135                     myRes[i - 1] = x;
136                     myRes[(i - 2) / 2] = cos(x);
137                 }
138                 else {
139                     myRes[(i - 2) / 2] = cos(myRes[i - 1]);
140                 }
141             }
142         }

```

```

143         else if (myRes[i - 1] == 100) {
144             if (myRes[(i - 2) / 2] == 30) {
145                 if (myRes[i] == 90) {
146                     myRes[i] = x;
147                     myRes[(i - 2) / 2] = sin(x);
148                 }
149                 else {
150                     myRes[(i - 2) / 2] = sin(myRes[i]);
151                 }
152             }
153             else {
154                 if (myRes[i] == 90) {
155                     myRes[i] = x;
156                     myRes[(i - 2) / 2] = cos(x);
157                 }
158                 else {
159                     myRes[(i - 2) / 2] = cos(myRes[i]);
160                 }
161             }
162         }
163     else {
164         if (myRes[(i - 2) / 2] == 50) {
165             if (myRes[i] == 90) { myRes[i] = x; }
166             if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
167             float res = myRes[i - 1] + myRes[i];
168             myRes[(i - 2) / 2] = res;
169         }
170         else if (myRes[(i - 2) / 2] == 60) {
171             if (myRes[i] == 90) { myRes[i] = x; }
172             if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
173             float res = myRes[i - 1] - myRes[i];
174             myRes[(i - 2) / 2] = res;
175         }
176         else if (myRes[(i - 2) / 2] == 70) {
177             if (myRes[i] == 90) { myRes[i] = x; }
178             if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
179             float res = myRes[i - 1] * myRes[i];
180             myRes[(i - 2) / 2] = res;
181         }
182         if (myRes[(i - 2) / 2] == 80) {
183             if (myRes[i] == 90) { myRes[i] = x; }
184             if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
185             float res = myRes[i - 1] / myRes[i];
186             myRes[(i - 2) / 2] = res;
187         }
188     }
189 }
190 return myRes[0];
191 }
192
193 float fitness(vector<float>& exp01, float xf[], float yf[]) {
194     float n = .0;
195     float yf1[fitNum];
196     vector<float> mycopy;
197     for (int j = 0; j < fitNum; j++) {
198         mycopy = exp01;
199         yf1[j] = calculate(mycopy, xf[j]);
200     }
201     for (int i = 0; i < fitNum; i++) {
202         if (yf1[i] > yf[i]) { n += (yf1[i] - yf[i]); }
203         else { n += (yf[i] - yf1[i]); }
204     }
205     return n / fitNum;
206 }
207
208 void mutation(vector<float>& myExp) {
209     int size = myExp.size();
210     int height = log(size + 1) / log(2);
211     int r0;
212     int r2 = rand() % 2;
213     while (true) {
214         r0 = rand() % (size - 1) + 1;
215         if (myExp[r0] != 100) { break; }

```

```

216     }
217     if (r0 > (size - 1) / 2 - 1) {
218         myExp[r0] = rand() / float(RAND_MAX) + rand() % 19 - 10;
219     }
220     else {
221         int h0 = log(r0 + 1) / log(2);
222         int h1 = height - h0;
223         vector<float>smallTree(pow(2, h1) - 1);
224         create(smallTree, h1);
225         myExp[r0] = smallTree[0];
226         int j = 2 * r0 + 1;
227         int k = 2 * r0 + 2;
228         int n = 1;
229         while (true) {
230             if (k > (size - 1)) { break; }
231             else {
232                 for (int i = j; i <= k; i++) { myExp[i] = smallTree[n]; n = n
233                     + 1; }
234                 j = 2 * j + 1;
235                 k = 2 * k + 2;
236             }
237         }
238     }
239 }
240 void expLog(vector<float>& myExp) {
241     for (int i = 0; i < myExp.size(); i++) {
242         cout << myExp[i];
243         if (i == 0 || i == 2 || i == 6 || i == 14 || i == 30 || i == 62 || i == 126) {
244             cout << endl;
245         }
246         else {
247             cout << ",";
248         }
249     }
250     cout << "-----" << endl;
251 }
252
253 int main() {
254     float* x0 = new float[pointNum];
255     float* y0 = new float[pointNum];
256     float* xf = new float[fitNum];
257     float* yf = new float[fitNum];
258     float* y1 = new float[fitNum];
259     vector<float>myExpression(mysize, 0);
260
261     readFile(x0, y0);
262
263     for (int i = 0; i < fitNum; i++) {
264         xf[i] = x0[i * 5];
265         yf[i] = y0[i * 5];
266     }
267
268     srand(time(NULL));
269
270     ofstream outFile("random03.csv", ios::out);
271     // random loop
272     float myfit = 50;
273     for (int n = 0; n < 50000; n++) {
274         outFile << n << ",";
275         cout << n << ": ";
276         create(myExpression, startHeight);
277         float newfit = fitness(myExpression, xf, yf);
278         if (newfit < myfit) {
279             myfit = newfit;
280         }
281         LOG(myfit);
282         outFile << myfit << endl;
283     }
284     // hill climber loop
285     /*create(myExpression, startHeight);
286     float myfit = 10;
287     for (int n = 0; n < 50000; n++) {

```

```

288         outFile << n << ", ";
289         cout << n << ": ";
290         mutation(myExpression);
291         float newfit = fitness(myExpression, xf, yf);
292         if (newfit < myfit) {
293             myfit = newfit;
294         }
295         LOG(myfit);
296         outFile << myfit << endl;
297     }*/
298
299     expLog(myExpression);
300     return 0;
301 }

```

## 4.2 conventional genetic program

```

1  #include <iostream>
2  #include <fstream>
3  #include <iomanip>
4  #include <sstream>
5  #include <string>
6  #include <math.h>
7  #include <vector>
8  #include <random>
9  #include <ctime>
10 #include <stdlib.h>
11 #define LOG(x) std::cout<<x<<std::endl
12 using namespace std;
13
14 const int startHeight = 6;
15 const int pointNum = 1000;
16 const int fitNum = 200;
17 const int gen = 1000;
18 const int groupSize = 100;
19 const int halfSize = groupSize / 2;
20 const int chance = 30;
21 const int mysize = pow(2, startHeight) - 1;
22
23 void readFile(float x[], float y[]) {
24     ifstream inFile("data.txt", ios::in);
25     if (!inFile)
26     {
27         cout << "fail to open " << endl;
28         exit(1);
29     }
30     int i = 0;
31     string line;
32     string field;
33     while (getline(inFile, line))
34     {
35         string field;
36         istringstream stin(line);
37         getline(stin, field, ',');
38         x[i] = atof(field.c_str());
39         getline(stin, field, ',');
40         y[i] = atof(field.c_str());
41         i++;
42     }
43     inFile.close();
44 }
45
46 void create(vector<float>& myExp, int height) {
47     //srand(time(NULL));
48     //write the bottom nodes
49     for (int i = pow(2, height - 1) - 1; i < pow(2, height) - 1; i = i + 2) {
50         int r1 = rand() % 6;
51         float r2 = rand() / float(RAND_MAX) + rand() % 19 - 10;
52         float r3 = rand() / float(RAND_MAX) + rand() % 19 - 10;
53         switch (rand() % 6)
54         {
55             case 0:myExp[i] = 100; myExp[i + 1] = 100; break;

```

```

56         case 1:myExp[i] = 90; myExp[i + 1] = r2; break;
57         case 2:myExp[i] = r2; myExp[i + 1] = 90; break;
58         case 3:myExp[i] = 90; myExp[i + 1] = 100; break;
59         case 4:myExp[i] = r2; myExp[i + 1] = r3; break;
60         case 5:myExp[i] = r2; myExp[i + 1] = 100; break;
61     }
62 }
63 for (int h = height - 1; h > 1; h--) {
64     float r4 = rand() / float(RAND_MAX) + rand() % 19 - 10;
65     for (int i = pow(2, h - 1) - 1; i < pow(2, h) - 1; i++) {
66         if (myExp[2 * i + 1] == 100 && myExp[2 * i + 2] == 100) {
67             switch (rand() % 3)
68             {
69                 case 0:myExp[i] = 100; break;
70                 case 1:myExp[i] = 90; break;
71                 case 2:myExp[i] = r4; break;
72             }
73         }
74         else if (myExp[2 * i + 1] == 100 || myExp[2 * i + 2] == 100) {
75             switch (rand() % 2)
76             {
77                 case 0:myExp[i] = 30; break;
78                 case 1:myExp[i] = 40; break;
79             }
80         }
81         else {
82             int r = rand() % 3;
83             switch (r)
84             {
85                 case 0:myExp[i] = 50; break;
86                 case 1:myExp[i] = 60; break;
87                 case 2:myExp[i] = 70; break;
88             }
89         }
90     }
91 }
92
93 // write the top nod
94 if (myExp[1] == 100 && myExp[2] == 100) {
95     myExp[0] = rand() / float(RAND_MAX) + rand() % 19 - 10;
96 }
97 else if (myExp[1] == 100 || myExp[2] == 100) {
98     switch (rand() % 2)
99     {
100         case 0:myExp[0] = 30; break;
101         case 1:myExp[0] = 40; break;
102     }
103 }
104 else { myExp[0] = 80; }
105 }
106
107 void create2(vector<float>& myExp, int height) {
108     //srand(time(NULL));
109     //write the bottom nodes
110     for (int i = pow(2, height - 1) - 1; i < pow(2, height) - 1; i = i + 2) {
111         int r1 = rand() % 6;
112         float r2 = rand() / float(RAND_MAX) + rand() % 19 - 10;
113         float r3 = rand() / float(RAND_MAX) + rand() % 19 - 10;
114         switch (rand() % 6)
115         {
116             case 0:myExp[i] = 100; myExp[i + 1] = 100; break;
117             case 1:myExp[i] = 90; myExp[i + 1] = r2; break;
118             case 2:myExp[i] = r2; myExp[i + 1] = 90; break;
119             case 3:myExp[i] = 90; myExp[i + 1] = 100; break;
120             case 4:myExp[i] = r2; myExp[i + 1] = r3; break;
121             case 5:myExp[i] = r2; myExp[i + 1] = 100; break;
122         }
123     }
124     for (int h = height - 1; h > 1; h--) {
125         float r4 = rand() / float(RAND_MAX) + rand() % 19 - 10;
126         for (int i = pow(2, h - 1) - 1; i < pow(2, h) - 1; i++) {
127             if (myExp[2 * i + 1] == 100 && myExp[2 * i + 2] == 100) {
128                 switch (rand() % 3)

```

```

129         {
130             case 0:myExp[i] = 100; break;
131             case 1:myExp[i] = 90; break;
132             case 2:myExp[i] = r4; break;
133         }
134     }
135     else if (myExp[2 * i + 1] == 100 || myExp[2 * i + 2] == 100) {
136         switch (rand() % 2)
137         {
138             case 0:myExp[i] = 30; break;
139             case 1:myExp[i] = 40; break;
140         }
141     }
142     else {
143         int r = rand() % 4;
144         switch (r)
145         {
146             case 0:myExp[i] = 50; break;
147             case 1:myExp[i] = 60; break;
148             case 2:myExp[i] = 70; break;
149             case 3:myExp[i] = 80; break;
150         }
151     }
152 }
153 }
154
155 // write the top nod
156 if (myExp[1] == 100 && myExp[2] == 100) {
157     myExp[0] = rand() / float(RAND_MAX) + rand() % 19 - 10;
158 }
159 else if (myExp[1] == 100 || myExp[2] == 100) {
160     switch (rand() % 2)
161     {
162         case 0:myExp[0] = 30; break;
163         case 1:myExp[0] = 40; break;
164     }
165 }
166 else {
167     int r = rand() % 4;
168     switch (r)
169     {
170         case 0:myExp[0] = 50; break;
171         case 1:myExp[0] = 60; break;
172         case 2:myExp[0] = 70; break;
173         case 3:myExp[0] = 80; break;
174     }
175 }
176 }
177 }
178
179 float calculate(vector<float>& myRes, float x) {
180     for (int i = pow(2, startHeight) - 2; i > 0; i = i - 2) {
181         if (myRes[i] == 100 && myRes[i - 1] == 100) {
182             continue;
183         }
184         else if (myRes[i] == 100) {
185             if (myRes[(i - 2) / 2] == 30) {
186                 if (myRes[i - 1] == 90) {
187                     myRes[i - 1] = x;
188                     myRes[(i - 2) / 2] = sin(x);
189                 }
190                 else {
191                     myRes[(i - 2) / 2] = sin(myRes[i - 1]);
192                 }
193             }
194             else {
195                 if (myRes[i - 1] == 90) {
196                     myRes[i - 1] = x;
197                     myRes[(i - 2) / 2] = cos(x);
198                 }
199                 else {
200                     myRes[(i - 2) / 2] = cos(myRes[i - 1]);
201                 }

```

```

202     }
203     }
204     else if (myRes[i - 1] == 100) {
205         if (myRes[(i - 2) / 2] == 30) {
206             if (myRes[i] == 90) {
207                 myRes[i] = x;
208                 myRes[(i - 2) / 2] = sin(x);
209             }
210             else {
211                 myRes[(i - 2) / 2] = sin(myRes[i]);
212             }
213         }
214         else {
215             if (myRes[i] == 90) {
216                 myRes[i] = x;
217                 myRes[(i - 2) / 2] = cos(x);
218             }
219             else {
220                 myRes[(i - 2) / 2] = cos(myRes[i]);
221             }
222         }
223     }
224     else {
225         if (myRes[(i - 2) / 2] == 50) {
226             if (myRes[i] == 90) { myRes[i] = x; }
227             if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
228             float res = myRes[i - 1] + myRes[i];
229             myRes[(i - 2) / 2] = res;
230         }
231         else if (myRes[(i - 2) / 2] == 60) {
232             if (myRes[i] == 90) { myRes[i] = x; }
233             if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
234             float res = myRes[i - 1] - myRes[i];
235             myRes[(i - 2) / 2] = res;
236         }
237         else if (myRes[(i - 2) / 2] == 70) {
238             if (myRes[i] == 90) { myRes[i] = x; }
239             if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
240             float res = myRes[i - 1] * myRes[i];
241             myRes[(i - 2) / 2] = res;
242         }
243         if (myRes[(i - 2) / 2] == 80) {
244             if (myRes[i] == 90) { myRes[i] = x; }
245             if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
246             float res = myRes[i - 1] / myRes[i];
247             myRes[(i - 2) / 2] = res;
248         }
249     }
250 }
251 return myRes[0];
252 }
253
254 float fitness(float yf1[], float yf2[]) {
255     float n = .0;
256     for (int i = 0; i < fitNum; i++) {
257         if (yf1[i] > yf2[i]) { n += (yf1[i] - yf2[i]); }
258         else { n += (yf2[i] - yf1[i]); }
259     }
260     return n / fitNum;
261 }
262
263 void mutation(vector<float>& myExp) {
264     int size = myExp.size();
265     int height = log(size + 1) / log(2);
266     int r0;
267     int r2 = rand() % 2;
268     while (true) {
269         r0 = rand() % (size - 1) + 1;
270         if (myExp[r0] != 100) { break; }
271     }
272     if (r0 > (size - 1) / 2 - 1) {
273         myExp[r0] = rand() / float(RAND_MAX) + rand() % 19 - 10;
274     }

```



```

275     else {
276         int h0 = log(r0 + 1) / log(2);
277         int h1 = height - h0;
278         vector<float> smallTree(pow(2, h1) - 1);
279         create2(smallTree, h1);
280         myExp[r0] = smallTree[0];
281         int j = 2 * r0 + 1;
282         int k = 2 * r0 + 2;
283         int n = 1;
284         while (true) {
285             if (k > (size - 1)) { break; }
286             else {
287                 for (int i = j; i <= k; i++) { myExp[i] = smallTree[n]; n = n
                + 1; }
288                 j = 2 * j + 1;
289                 k = 2 * k + 2;
290             }
291         }
292     }
293 }
294
295 void crossover(vector<float>& treeA, vector<float>& treeB) {
296     int a; int b;
297     while (true) {
298         a = rand() % ((mysize - 1) / 2 - 1) + 1;
299         if (treeA[a] != 100) { break; };
300     }
301     int h0 = log(a + 1) / log(2);
302     int b1 = pow(2, h0);
303     while (true) {
304         b = rand() % b1 + b1 - 1;
305         if (treeA[b] != 100) { break; };
306     }
307     int j1 = 2 * a + 1; int k1 = 2 * a + 2;
308     int j2 = 2 * b + 1; int k2 = 2 * b + 2;
309     float t = 0;
310     t = treeA[a];
311     treeA[a] = treeB[b];
312     treeB[b] = t;
313     while (true) {
314         if (k2 > (mysize - 1)) { break; }
315         else {
316             int i2 = j2;
317             for (int i1 = j1; i1 <= k1; i1++) {
318                 float temp = 0;
319                 temp = treeA[i1];
320                 treeA[i1] = treeB[i2];
321                 treeB[i2] = temp;
322                 i2++;
323             }
324             j1 = 2 * j1 + 1; k1 = 2 * k1 + 2;
325             j2 = 2 * j2 + 1; k2 = 2 * k2 + 2;
326         }
327     }
328 }
329
330 float evo(vector<vector<float>> & group, float xf[], float yf[], float y1[]) {
331     vector<float> temp01, temp02;
332     vector<vector<float>> bigGroup;
333     vector<vector<float>> yMatrix;
334     vector<float> myResult;
335     vector<float> myfit;
336     int idx[2 * groupSize] = { 0 };
337     // get big group, copy of group
338     bigGroup = group;
339     for (int i = 0; i < groupSize; i++) {
340         temp01 = group[i];
341         mutation(temp01);
342         bigGroup.push_back(temp01); // double the big group with mutations
343     }
344
345     // get fitness
346     for (int i = 0; i < 2 * groupSize; i++) {

```

```

347         for (int j = 0; j < fitNum; j++) {
348             myResult = bigGroup[i];
349             y1[j] = calculate(myResult, xf[j]);
350         }
351         myfit.push_back(fitness(y1, yf)); // myfit has the same order as the big group
352     }
353     temp02 = myfit;
354     sort(temp02.begin(), temp02.end()); // sort the copy of fitness list
355     // write top half into group
356     for (int i = 0; i < 2 * groupSize; i++) {
357         for (int j = 0; j < 2 * groupSize; j++) {
358             if (temp02[i] == myfit[j]) { idx[i] = j; } // fitness from small to
359                 large, write into new group
360         }
361     }
362     for (int i = 0; i < groupSize; i++) {
363         for (int j = 0; j < mysize; j++) {
364             group[i][j] = bigGroup[idx[i]][j];
365         }
366     }
367     LOG(temp02[0]);
368     return temp02[0];
369 }
370 }
371
372 float evo2(vector<vector<float>> & group, float xf[], float yf[]) {
373     vector<float> temp;
374     vector<vector<float>> bigGroup;
375     vector<float> myResult;
376     vector<float> myfit;
377     float* y1 = new float[fitNum];
378     int idx[2 * groupSize] = { 0 };
379     bigGroup = group;
380     for (int i = 0; i < groupSize; i = i + 2) {
381         vector<float> temp01, temp02;
382         temp01 = group[i];
383         temp02 = group[i + 1];
384         crossover(temp01, temp02);
385         bigGroup.push_back(temp01);
386         bigGroup.push_back(temp02);
387     }
388     //mutation
389     for (int i = halfSize; i < groupSize; i++) {
390         if (rand() % 100 < 50) {
391             vector<float> temp03;
392             temp03 = bigGroup[i];
393             mutation(temp03); //check here
394             bigGroup[i] = temp03;
395         }
396     }
397     // get fitness
398     for (int i = 0; i < 2 * groupSize; i++) {
399         for (int j = 0; j < fitNum; j++) {
400             myResult = bigGroup[i];
401             y1[j] = calculate(myResult, xf[j]);
402         }
403         myfit.push_back(fitness(y1, yf)); // myfit has the same order as the big group
404     }
405     temp = myfit;
406     sort(temp.begin(), temp.end()); // sort the copy of fitness list
407     // write top half into group
408     for (int i = 0; i < 2 * groupSize; i++) {
409         for (int j = 0; j < 2 * groupSize; j++) {
410             if (temp[i] == myfit[j]) { idx[i] = j; }
411         }
412     }
413     // fitness from small to large, write into new group
414     for (int i = 0; i < groupSize; i++) {
415         for (int j = 0; j < mysize; j++) {
416             group[i][j] = bigGroup[idx[i]][j];
417         }
418     }

```

```

419     LOG(temp[0]);
420     return temp[0];
421 }
422
423 void expLog(vector<float>& myExp) {
424     for (int i = 0; i < myExp.size(); i++) {
425         cout << myExp[i];
426         if (i == 0 || i == 2 || i == 6 || i == 14 || i == 30 || i == 62 || i == 126) {
427             cout << endl;
428         }
429         else {
430             cout << ",";
431         }
432     }
433     cout << "—————" << endl;
434 }
435
436 int main() {
437     float* x0 = new float[pointNum];
438     float* y0 = new float[pointNum];
439     float* xf = new float[fitNum];
440     float* yf = new float[fitNum];
441     float* y1 = new float[fitNum];
442     vector<vector<float>> > group;
443     vector<float>myExpression(mysize, 0);
444
445     readFile(x0, y0);
446
447     for (int i = 0; i < fitNum; i++) {
448         xf[i] = x0[i * 5];
449         yf[i] = y0[i * 5];
450     }
451
452     srand(time(NULL));
453     // creat a group of expressions
454     for (int i = 0; i < groupSize; i++) {
455         create2(myExpression, startHeight);
456         group.push_back(myExpression);
457     }
458
459     ofstream outFile("gasimple03.csv", ios::out);
460     ofstream outFile02("Ygasimple03.csv", ios::out);
461     // evo loop
462     for (int n = 0; n < 50000; n++) {
463         outFile << n << ",";
464         cout << n << ": ";
465         float tempfit = evo2(group, xf, yf);
466         outFile << tempfit << endl;
467     }
468
469     expLog(group[0]);
470     vector<float> result;
471     for (int j = 0; j < fitNum; j++) {
472         result = group[0];
473         outFile02 << calculate(result, xf[j]) << ",";
474     }
475     return 0;
476 }

```

### 4.3 Niching genetic program

```

1 #include <iostream>
2 #include <fstream>
3 #include <iomanip>
4 #include <sstream>
5 #include <string>
6 #include <math.h>
7 #include <vector>
8 #include <random>
9 #include <ctime>
10 #include<stdlib.h>
11 #define LOG(x) std::cout<<x<<std::endl

```

```

12 using namespace std;
13
14 const int startHeight = 6;
15 const int pointNum = 1000;
16 const int fitNum = 100;
17 const int gen = 1000;
18 const int groupSize = 100;
19 const int halfSize = groupSize / 2;
20 const int chance = 30;
21 const int mysize = pow(2, startHeight) - 1;
22
23 void readFile(float x[], float y[]) {
24     ifstream inFile("data.txt", ios::in);
25     if (!inFile)
26     {
27         cout << "fail to open " << endl;
28         exit(1);
29     }
30     int i = 0;
31     string line;
32     string field;
33     while (getline(inFile, line))
34     {
35         string field;
36         istringstream stin(line);
37         getline(stin, field, ',');
38         x[i] = atof(field.c_str());
39         getline(stin, field, ',');
40         y[i] = atof(field.c_str());
41         i++;
42     }
43     inFile.close();
44 }
45
46 void create(vector<float>& myExp, int height) {
47     //srand(time(NULL));
48     //write the bottom nodes
49     for (int i = pow(2, height - 1) - 1; i < pow(2, height) - 1; i = i + 2) {
50         int r1 = rand() % 6;
51         float r2 = rand() / float(RAND_MAX) + rand() % 19 - 10;
52         float r3 = rand() / float(RAND_MAX) + rand() % 19 - 10;
53         switch (rand() % 6)
54         {
55             case 0:myExp[i] = 100; myExp[i + 1] = 100; break;
56             case 1:myExp[i] = 90; myExp[i + 1] = r2; break;
57             case 2:myExp[i] = r2; myExp[i + 1] = 90; break;
58             case 3:myExp[i] = 90; myExp[i + 1] = 100; break;
59             case 4:myExp[i] = r2; myExp[i + 1] = r3; break;
60             case 5:myExp[i] = r2; myExp[i + 1] = 100; break;
61         }
62     }
63     for (int h = height - 1; h > 1; h--) {
64         float r4 = rand() / float(RAND_MAX) + rand() % 19 - 10;
65         for (int i = pow(2, h - 1) - 1; i < pow(2, h) - 1; i++) {
66             if (myExp[2 * i + 1] == 100 && myExp[2 * i + 2] == 100) {
67                 switch (rand() % 3)
68                 {
69                     case 0:myExp[i] = 100; break;
70                     case 1:myExp[i] = 90; break;
71                     case 2:myExp[i] = r4; break;
72                 }
73             }
74             else if (myExp[2 * i + 1] == 100 || myExp[2 * i + 2] == 100) {
75                 switch (rand() % 2)
76                 {
77                     case 0:myExp[i] = 30; break;
78                     case 1:myExp[i] = 40; break;
79                 }
80             }
81             else {
82                 int r = rand() % 4;
83                 switch (r)
84                 {

```

```

85         case 0:myExp[i] = 50; break;
86         case 1:myExp[i] = 60; break;
87         case 2:myExp[i] = 70; break;
88         case 3:myExp[i] = 80; break;
89     }
90 }
91 }
92 }
93
94 // write the top nod
95 if (myExp[1] == 100 && myExp[2] == 100) {
96     myExp[0] = rand() / float(RAND_MAX) + rand() % 19 - 10;
97 }
98 else if (myExp[1] == 100 || myExp[2] == 100) {
99     switch (rand() % 2)
100     {
101     case 0:myExp[0] = 30; break;
102     case 1:myExp[0] = 40; break;
103     }
104 }
105 else {
106     int r = rand() % 4;
107     switch (r)
108     {
109     case 0:myExp[0] = 50; break;
110     case 1:myExp[0] = 60; break;
111     case 2:myExp[0] = 70; break;
112     case 3:myExp[0] = 80; break;
113     }
114 }
115 }
116 }
117
118 float calculate(vector<float>& myRes, float x) {
119     for (int i = pow(2, startHeight) - 2; i > 0; i = i - 2) {
120         if (myRes[i] == 100 && myRes[i - 1] == 100) {
121             continue;
122         }
123         else if (myRes[i] == 100) {
124             if (myRes[(i - 2) / 2] == 30) {
125                 if (myRes[i - 1] == 90) {
126                     myRes[i - 1] = x;
127                     myRes[(i - 2) / 2] = sin(x);
128                 }
129                 else {
130                     myRes[(i - 2) / 2] = sin(myRes[i - 1]);
131                 }
132             }
133             else {
134                 if (myRes[i - 1] == 90) {
135                     myRes[i - 1] = x;
136                     myRes[(i - 2) / 2] = cos(x);
137                 }
138                 else {
139                     myRes[(i - 2) / 2] = cos(myRes[i - 1]);
140                 }
141             }
142         }
143         else if (myRes[i - 1] == 100) {
144             if (myRes[(i - 2) / 2] == 30) {
145                 if (myRes[i] == 90) {
146                     myRes[i] = x;
147                     myRes[(i - 2) / 2] = sin(x);
148                 }
149                 else {
150                     myRes[(i - 2) / 2] = sin(myRes[i]);
151                 }
152             }
153             else {
154                 if (myRes[i] == 90) {
155                     myRes[i] = x;
156                     myRes[(i - 2) / 2] = cos(x);
157                 }

```

```

158         else {
159             myRes[(i - 2) / 2] = cos(myRes[i]);
160         }
161     }
162 }
163 else {
164     if (myRes[(i - 2) / 2] == 50) {
165         if (myRes[i] == 90) { myRes[i] = x; }
166         if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
167         float res = myRes[i - 1] + myRes[i];
168         myRes[(i - 2) / 2] = res;
169     }
170     else if (myRes[(i - 2) / 2] == 60) {
171         if (myRes[i] == 90) { myRes[i] = x; }
172         if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
173         float res = myRes[i - 1] - myRes[i];
174         myRes[(i - 2) / 2] = res;
175     }
176     else if (myRes[(i - 2) / 2] == 70) {
177         if (myRes[i] == 90) { myRes[i] = x; }
178         if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
179         float res = myRes[i - 1] * myRes[i];
180         myRes[(i - 2) / 2] = res;
181     }
182     if (myRes[(i - 2) / 2] == 80) {
183         if (myRes[i] == 90) { myRes[i] = x; }
184         if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
185         float res = myRes[i - 1] / myRes[i];
186         myRes[(i - 2) / 2] = res;
187     }
188 }
189 }
190 return myRes[0];
191 }
192
193 float fitness(float yf1[], float yf2[]) {
194     float n = .0;
195     for (int i = 0; i < fitNum; i++) {
196         if (yf1[i] > yf2[i]) { n += (yf1[i] - yf2[i]); }
197         else { n += (yf2[i] - yf1[i]); }
198     }
199     return n / fitNum;
200 }
201
202 void mutation(vector<float>& myExp) {
203     int size = myExp.size();
204     int height = log(size + 1) / log(2);
205     int r0;
206     int r2 = rand() % 2;
207     while (true) {
208         r0 = rand() % (size - 1) + 1;
209         if (myExp[r0] != 100) { break; }
210     }
211     if (r0 > (size - 1) / 2 - 1) {
212         myExp[r0] = rand() / float(RAND_MAX) + rand() % 19 - 10;
213     }
214     else {
215         int h0 = log(r0 + 1) / log(2);
216         int h1 = height - h0;
217         vector<float> smallTree(pow(2, h1) - 1);
218         create(smallTree, h1);
219         myExp[r0] = smallTree[0];
220         int j = 2 * r0 + 1;
221         int k = 2 * r0 + 2;
222         int n = 1;
223         while (true) {
224             if (k > (size - 1)) { break; }
225             else {
226                 for (int i = j; i <= k; i++) { myExp[i] = smallTree[n]; n = n
+ 1; }
227                 j = 2 * j + 1;
228                 k = 2 * k + 2;
229             }

```

```

230     }
231     }
232 }
233
234 void crossover(vector<float>& treeA, vector<float>& treeB) {
235     int a; int b;
236     while (true) {
237         a = rand() % ((mysize - 1) / 2 - 1) + 1;
238         if (treeA[a] != 100) { break; };
239     }
240     int h0 = log(a + 1) / log(2);
241     int b1 = pow(2, h0);
242     while (true) {
243         b = rand() % b1 + b1 - 1;
244         if (treeA[b] != 100) { break; };
245     }
246     int j1 = 2 * a + 1; int k1 = 2 * a + 2;
247     int j2 = 2 * b + 1; int k2 = 2 * b + 2;
248     float t = 0;
249     t = treeA[a];
250     treeA[a] = treeB[b];
251     treeB[b] = t;
252     while (true) {
253         if (k2 > (mysize - 1)) { break; }
254         else {
255             int i2 = j2;
256             for (int i1 = j1; i1 <= k1; i1++) {
257                 float temp = 0;
258                 temp = treeA[i1];
259                 treeA[i1] = treeB[i2];
260                 treeB[i2] = temp;
261                 i2++;
262             }
263             j1 = 2 * j1 + 1; k1 = 2 * k1 + 2;
264             j2 = 2 * j2 + 1; k2 = 2 * k2 + 2;
265         }
266     }
267 }
268
269 void migration(vector<vector<float> >& group1, vector<vector<float> >& group2, vector<vector<
float> >& group3,
270 vector<vector<float> >& group4, vector<vector<float> >& group5) {
271     int r1, r2, r3, r4, r5;
272     r1 = rand() % 5 + 1;
273     r2 = rand() % 5 + 1;
274     r3 = rand() % 5 + 1;
275     r4 = rand() % 5 + 1;
276     r5 = rand() % 5 + 1;
277
278     for (int i = 0; i < 4; i = i + 2) {
279         vector<float> temp;
280         temp = group1[r1 + i];
281         group1[r1 + i] = group2[r2 + i];
282         group2[r2 + i] = group3[r3 + i];
283         group3[r3 + i] = group4[r4 + i];
284         group4[r4 + i] = group5[r5 + i];
285         group5[r5 + i] = temp;
286     }
287 }
288
289 float evo2(vector<vector<float> >& group, float xf[], float yf[]) {
290     vector<float> temp;
291     vector<vector<float> > bigGroup;
292     vector<float> myResult;
293     vector<float> myfit;
294     float* y1 = new float[pointNum];
295     int idx[2 * groupSize] = { 0 };
296     bigGroup = group;
297     for (int i = 0; i < groupSize; i = i + 2) {
298         vector<float> temp01, temp02;
299         temp01 = group[i];
300         temp02 = group[i + 1];
301         crossover(temp01, temp02);

```

```

302         bigGroup.push_back(temp01);
303         bigGroup.push_back(temp02);
304     }
305     for (int i = halfSize; i < groupSize; i++) {
306         if (rand() % 100 < chance) {
307             vector<float> temp03;
308             temp03 = bigGroup[i];
309             mutation(temp03); //check here
310         }
311     }
312     // get fitness
313     for (int i = 0; i < 2 * groupSize; i++) {
314         for (int j = 0; j < pointNum; j++) {
315             myResult = bigGroup[i];
316             y1[j] = calculate(myResult, xf[j]);
317         }
318         myfit.push_back(fitness(y1, yf)); // myfit has the same order as the big group
319     }
320     temp = myfit;
321     sort(temp.begin(), temp.end()); // sort the copy of fitness list
322     // write top half into group
323     for (int i = 0; i < 2 * groupSize; i++) {
324         for (int j = 0; j < 2 * groupSize; j++) {
325             if (temp[i] == myfit[j]) { idx[i] = j; }
326         }
327     }
328     // fitness from small to large, write into new group
329     for (int i = 0; i < groupSize; i++) {
330         for (int j = 0; j < mysize; j++) {
331             group[i][j] = bigGroup[idx[i]][j];
332         }
333     }
334     LOG(temp[0]);
335     return temp[0];
336 }
337
338 void expLog(vector<float>& myExp) {
339     for (int i = 0; i < myExp.size(); i++) {
340         cout << myExp[i];
341         if (i == 0 || i == 2 || i == 6 || i == 14 || i == 30 || i == 62 || i == 126) {
342             cout << endl;
343         }
344         else {
345             cout << ",";
346         }
347     }
348     cout << "—————" << endl;
349 }
350
351 int main() {
352     float* x0 = new float[pointNum];
353     float* y0 = new float[pointNum];
354     float* xf = new float[fitNum];
355     float* yf = new float[fitNum];
356     float* y1 = new float[fitNum];
357     vector<vector<float>> group01, group02, group03, group04, group05;
358     vector<float>myExpression(mysize, 0);
359
360     readFile(x0, y0);
361     float myFit = 1000;
362
363     for (int i = 0; i < fitNum; i++) {
364         xf[i] = x0[i * 10];
365         yf[i] = y0[i * 10];
366     }
367
368     srand(time(NULL));
369     // creat groups of expressions
370     for (int i = 0; i < groupSize; i++) {
371         create(myExpression, startHeight);
372         group01.push_back(myExpression);
373     }
374     for (int i = 0; i < groupSize; i++) {

```



```

375         create(myExpression, startHeight);
376         group02.push_back(myExpression);
377     }
378     for (int i = 0; i < groupSize; i++) {
379         create(myExpression, startHeight);
380         group03.push_back(myExpression);
381     }
382     for (int i = 0; i < groupSize; i++) {
383         create(myExpression, startHeight);
384         group04.push_back(myExpression);
385     }
386     for (int i = 0; i < groupSize; i++) {
387         create(myExpression, startHeight);
388         group05.push_back(myExpression);
389     }
390
391     ofstream outFile("gene102201.csv", ios::out);
392     ofstream outFile02("Y_gene01.csv", ios::out);
393     // evo loop
394     for (int n = 0; n < 10000; n++) {
395         outFile << n << ",";
396         cout << n << ": ";
397         float tempfit01 = evo2(group01, xf, yf);
398         outFile << tempfit01 << endl;
399
400         cout << n << ": "; evo2(group02, xf, yf);
401         cout << n << ": "; evo2(group03, xf, yf);
402         cout << n << ": "; evo2(group04, xf, yf);
403         cout << n << ": "; evo2(group05, xf, yf);
404         if (n == 200) {
405             migration(group01, group02, group03, group04, group05);
406         }
407     }
408
409     // print group
410     expLog(group01[0]);
411     vector<float> result;
412     for (int j = 0; j < fitNum; j++) {
413         result = group01[0];
414         outFile02 << calculate(result, xf[j]) << ",";
415     }
416     /*
417     for (int i = 0; i < groupSize; i++) {
418         expLog(group[i]);
419     }*/
420     return 0;
421 }

```

#### 4.4 Deterministic crowding genetic program

```

1  #include <iostream>
2  #include <fstream>
3  #include <iomanip>
4  #include <sstream>
5  #include <string>
6  #include <math.h>
7  #include <vector>
8  #include <random>
9  #include <ctime>
10 #include <stdlib.h>
11 #define LOG(x) std::cout<<x<<std::endl
12 using namespace std;
13
14 const int startHeight = 6;
15 const int pointNum = 1000;
16 const int fitNum = 100;
17 const int gen = 1000;
18 const int groupSize = 100;
19 const int halfSize = groupSize / 2;
20 const int chance = 50;
21 const int mysize = pow(2, startHeight) - 1;
22

```

```

23 void readFile(float x[], float y[]) {
24     ifstream inFile("data.txt", ios::in);
25     if (!inFile)
26     {
27         cout << "fail to open " << endl;
28         exit(1);
29     }
30     int i = 0;
31     string line;
32     string field;
33     while (getline(inFile, line))
34     {
35         string field;
36         istringstream stin(line);
37         getline(stin, field, ',');
38         x[i] = atof(field.c_str());
39         getline(stin, field, ',');
40         y[i] = atof(field.c_str());
41         i++;
42     }
43     inFile.close();
44 }
45
46 void create(vector<float>& myExp, int height) {
47     //srand(time(NULL));
48     //write the bottom nodes
49     for (int i = pow(2, height - 1) - 1; i < pow(2, height) - 1; i = i + 2) {
50         int r1 = rand() % 6;
51         float r2 = rand() / float(RAND_MAX) + rand() % 19 - 10;
52         float r3 = rand() / float(RAND_MAX) + rand() % 19 - 10;
53         switch (rand() % 6)
54         {
55             case 0:myExp[i] = 100; myExp[i + 1] = 100; break;
56             case 1:myExp[i] = 90; myExp[i + 1] = r2; break;
57             case 2:myExp[i] = r2; myExp[i + 1] = 90; break;
58             case 3:myExp[i] = 90; myExp[i + 1] = 100; break;
59             case 4:myExp[i] = r2; myExp[i + 1] = r3; break;
60             case 5:myExp[i] = r2; myExp[i + 1] = 100; break;
61         }
62     }
63     for (int h = height - 1; h > 1; h--) {
64         float r4 = rand() / float(RAND_MAX) + rand() % 19 - 10;
65         for (int i = pow(2, h - 1) - 1; i < pow(2, h) - 1; i++) {
66             if (myExp[2 * i + 1] == 100 && myExp[2 * i + 2] == 100) {
67                 switch (rand() % 3)
68                 {
69                     case 0:myExp[i] = 100; break;
70                     case 1:myExp[i] = 90; break;
71                     case 2:myExp[i] = r4; break;
72                 }
73             }
74             else if (myExp[2 * i + 1] == 100 || myExp[2 * i + 2] == 100) {
75                 switch (rand() % 2)
76                 {
77                     case 0:myExp[i] = 30; break;
78                     case 1:myExp[i] = 40; break;
79                 }
80             }
81             else {
82                 int r = rand() % 4;
83                 switch (r)
84                 {
85                     case 0:myExp[i] = 50; break;
86                     case 1:myExp[i] = 60; break;
87                     case 2:myExp[i] = 70; break;
88                     case 3:myExp[i] = 80; break;
89                 }
90             }
91         }
92     }
93
94     // write the top nod
95     if (myExp[1] == 100 && myExp[2] == 100) {

```

```

96         myExp[0] = rand() / float(RAND_MAX) + rand() % 19 - 10;
97     }
98     else if (myExp[1] == 100 || myExp[2] == 100) {
99         switch (rand() % 2)
100         {
101             case 0:myExp[0] = 30; break;
102             case 1:myExp[0] = 40; break;
103         }
104     }
105     else {
106         int r = rand() % 4;
107         switch (r)
108         {
109             case 0:myExp[0] = 50; break;
110             case 1:myExp[0] = 60; break;
111             case 2:myExp[0] = 70; break;
112             case 3:myExp[0] = 80; break;
113         }
114     }
115 }
116 }
117
118 float calculate(vector<float>& myRes, float x) {
119     for (int i = pow(2, startHeight) - 2; i > 0; i = i - 2) {
120         if (myRes[i] == 100 && myRes[i - 1] == 100) {
121             continue;
122         }
123         else if (myRes[i] == 100) {
124             if (myRes[(i - 2) / 2] == 30) {
125                 if (myRes[i - 1] == 90) {
126                     myRes[i - 1] = x;
127                     myRes[(i - 2) / 2] = sin(x);
128                 }
129                 else {
130                     myRes[(i - 2) / 2] = sin(myRes[i - 1]);
131                 }
132             }
133             else {
134                 if (myRes[i - 1] == 90) {
135                     myRes[i - 1] = x;
136                     myRes[(i - 2) / 2] = cos(x);
137                 }
138                 else {
139                     myRes[(i - 2) / 2] = cos(myRes[i - 1]);
140                 }
141             }
142         }
143         else if (myRes[i - 1] == 100) {
144             if (myRes[(i - 2) / 2] == 30) {
145                 if (myRes[i] == 90) {
146                     myRes[i] = x;
147                     myRes[(i - 2) / 2] = sin(x);
148                 }
149                 else {
150                     myRes[(i - 2) / 2] = sin(myRes[i]);
151                 }
152             }
153             else {
154                 if (myRes[i] == 90) {
155                     myRes[i] = x;
156                     myRes[(i - 2) / 2] = cos(x);
157                 }
158                 else {
159                     myRes[(i - 2) / 2] = cos(myRes[i]);
160                 }
161             }
162         }
163         else {
164             if (myRes[(i - 2) / 2] == 50) {
165                 if (myRes[i] == 90) { myRes[i] = x; }
166                 if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
167                 float res = myRes[i - 1] + myRes[i];
168                 myRes[(i - 2) / 2] = res;

```

```

169         }
170         else if (myRes[(i - 2) / 2] == 60) {
171             if (myRes[i] == 90) { myRes[i] = x; }
172             if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
173             float res = myRes[i - 1] - myRes[i];
174             myRes[(i - 2) / 2] = res;
175         }
176         else if (myRes[(i - 2) / 2] == 70) {
177             if (myRes[i] == 90) { myRes[i] = x; }
178             if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
179             float res = myRes[i - 1] * myRes[i];
180             myRes[(i - 2) / 2] = res;
181         }
182         if (myRes[(i - 2) / 2] == 80) {
183             if (myRes[i] == 90) { myRes[i] = x; }
184             if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
185             float res = myRes[i - 1] / myRes[i];
186             myRes[(i - 2) / 2] = res;
187         }
188     }
189 }
190 return myRes[0];
191 }
192
193 float fitness(vector<float>& exp01, float xf[], float yf[]) {
194     float n = .0;
195     float yf1[fitNum];
196     vector<float> mycopy;
197     for (int j = 0; j < fitNum; j++) {
198         mycopy = exp01;
199         yf1[j] = calculate(mycopy, xf[j]);
200     }
201     for (int i = 0; i < fitNum; i++) {
202         if (yf1[i] > yf[i]) { n += (yf1[i] - yf[i]); }
203         else { n += (yf[i] - yf1[i]); }
204     }
205     return n / fitNum;
206 }
207
208 void mutation(vector<float>& myExp) {
209     int size = myExp.size();
210     int height = log(size + 1) / log(2);
211     int r0;
212     int r2 = rand() % 2;
213     while (true) {
214         r0 = rand() % (size - 1) + 1;
215         if (myExp[r0] != 100) { break; }
216     }
217     if (r0 > (size - 1) / 2 - 1) {
218         myExp[r0] = rand() / float(RAND_MAX) + rand() % 19 - 10;
219     }
220     else {
221         int h0 = log(r0 + 1) / log(2);
222         int h1 = height - h0;
223         vector<float> smallTree(pow(2, h1) - 1);
224         create(smallTree, h1);
225         myExp[r0] = smallTree[0];
226         int j = 2 * r0 + 1;
227         int k = 2 * r0 + 2;
228         int n = 1;
229         while (true) {
230             if (k > (size - 1)) { break; }
231             else {
232                 for (int i = j; i <= k; i++) { myExp[i] = smallTree[n]; n = n
233                     + 1; }
234                 j = 2 * j + 1;
235                 k = 2 * k + 2;
236             }
237         }
238     }
239 }
240 void crossover(vector<float>& treeA, vector<float>& treeB) {

```

```

241     int a; int b;
242     while (true) {
243         a = rand() % ((mysize - 1) / 2 - 1) + 1;
244         if (treeA[a] != 100) { break; };
245     }
246     int h0 = log(a + 1) / log(2);
247     int b1 = pow(2, h0);
248     while (true) {
249         b = rand() % b1 + b1 - 1;
250         if (treeA[b] != 100) { break; };
251     }
252     int j1 = 2 * a + 1; int k1 = 2 * a + 2;
253     int j2 = 2 * b + 1; int k2 = 2 * b + 2;
254     float t = 0;
255     t = treeA[a];
256     treeA[a] = treeB[b];
257     treeB[b] = t;
258     while (true) {
259         if (k2 > (mysize - 1)) { break; }
260         else {
261             int i2 = j2;
262             for (int i1 = j1; i1 <= k1; i1++) {
263                 float temp = 0;
264                 temp = treeA[i1];
265                 treeA[i1] = treeB[i2];
266                 treeB[i2] = temp;
267                 i2++;
268             }
269             j1 = 2 * j1 + 1; k1 = 2 * k1 + 2;
270             j2 = 2 * j2 + 1; k2 = 2 * k2 + 2;
271         }
272     }
273 }
274
275 float getDistance(vector<float>& exp01, vector<float>& exp02, float xf[]) {
276     float y1[fitNum];
277     float y2[fitNum];
278     vector<float> mycopy;
279     for (int j = 0; j < fitNum; j++) {
280         mycopy = exp01;
281         y1[j] = calculate(mycopy, xf[j]);
282     }
283     for (int j = 0; j < fitNum; j++) {
284         mycopy = exp02;
285         y2[j] = calculate(mycopy, xf[j]);
286     }
287
288     float n = .0;
289     for (int i = 0; i < fitNum; i++) {
290         if (y1[i] > y2[i]) { n += (y1[i] - y2[i]); }
291         else { n += (y2[i] - y1[i]); }
292     }
293     return n / fitNum;
294     delete[] y1;
295     delete[] y2;
296     mycopy.clear();
297 }
298
299 float evo3(vector<vector<float>> & group, float xf[], float yf[]) {
300     shuffle(group.begin(), group.end(), std::default_random_engine(rand()));
301     vector<float> myfit(groupSize, 0);
302     vector<float> copyfit(groupSize, 0);
303     vector<float> temp01, temp02, temp;
304     vector<vector<float>> newgroup;
305     float* dis = new float[8];
306     newgroup = group;
307     for (int i = 0; i < groupSize; i = i + 2) {
308         temp01 = newgroup[i];
309         temp02 = newgroup[i + 1];
310         crossover(temp01, temp02);
311         dis[0] = getDistance(temp01, newgroup[i], xf);
312         dis[1] = getDistance(temp02, newgroup[i + 1], xf);
313         dis[2] = getDistance(temp01, newgroup[i + 1], xf);

```

```

314         dis[3] = getDistance(temp02, newgroup[i], xf);
315         dis[4] = fitness(temp01, xf, yf);
316         dis[5] = fitness(temp02, xf, yf);
317         dis[6] = fitness(newgroup[i], xf, yf);
318         dis[7] = fitness(newgroup[i + 1], xf, yf);
319         if (dis[0] + dis[1] < dis[2] + dis[3]) {
320             if (dis[4] < dis[6]) {
321                 newgroup[i] = temp01;
322                 myfit[i] = dis[4];
323             }
324             else { myfit[i] = dis[6]; }
325             if (dis[5] < dis[7]) {
326                 newgroup[i + 1] = temp02;
327                 myfit[i + 1] = dis[5];
328             }
329             else { myfit[i + 1] = dis[7]; }
330         }
331         else {
332             if (dis[4] < dis[7]) {
333                 newgroup[i + 1] = temp01;
334                 myfit[i + 1] = dis[4];
335             }
336             else { myfit[i + 1] = dis[7]; }
337             if (dis[5] < dis[6]) {
338                 newgroup[i] = temp02;
339                 myfit[i] = dis[5];
340             }
341             else { myfit[i] = dis[6]; }
342         }
343     }
344
345     copyfit = myfit;
346     sort(myfit.begin(), myfit.end());
347
348     int r0 = rand() % 50 + 20;
349     for (int i = r0; i < r0 + 20; i = i + 2) {
350         float t0 = myfit[i];
351         for (int j = 0; j < groupSize; j++) {
352             if (copyfit[j] == t0) {
353                 temp = newgroup[j];
354                 mutation(temp);
355                 newgroup[j] = temp;
356             }
357         }
358     }
359     group = newgroup;
360
361     LOG(myfit[0]);
362     return myfit[0];
363     newgroup.clear(); myfit.clear();
364     temp.clear(); temp01.clear(); temp02.clear();
365 }
366
367
368
369 void expLog(vector<float>& myExp) {
370     for (int i = 0; i < myExp.size(); i++) {
371         cout << myExp[i];
372         if (i == 0 || i == 2 || i == 6 || i == 14 || i == 30 || i == 62 || i == 126) {
373             cout << endl;
374         }
375         else {
376             cout << ",";
377         }
378     }
379     cout << "-----" << endl;
380 }
381
382 int main() {
383     float* x0 = new float[pointNum];
384     float* y0 = new float[pointNum];
385     float* xf = new float[fitNum];
386     float* yf = new float[fitNum];

```

```

387     vector<vector<float> > group;
388     vector<float>myExpression(mysize, 0);
389
390     readFile(x0, y0);
391     float myFit = 1000;
392
393     for (int i = 0; i < fitNum; i++) {
394         xf[i] = x0[i * 10];
395         yf[i] = y0[i * 10];
396     }
397
398     srand(time(NULL));
399     // creat a group of expressions
400     for (int i = 0; i < groupSize; i++) {
401         create(myExpression, startHeight);
402         group.push_back(myExpression);
403     }
404
405     ofstream outFile("gpcrowding01.csv", ios::out);
406     ofstream outFile02("Ycrowding01.csv", ios::out);
407     // evo loop
408     for (int n = 0; n < 30000; n++) {
409         outFile << n << ", ";
410         cout << n << ": ";
411         float tempfit = evo3(group, xf, yf);
412         outFile << tempfit << endl;
413     }
414
415     // print group
416     expLog(group[0]);
417     vector<float> result;
418     for (int j = 0; j < fitNum; j++) {
419         result = group[0];
420         outFile02 << calculate(result, xf[j]) << ", ";
421     }
422
423     return 0;
424 }

```

## 4.5 Plot codes

The plot codes similar to HW01, so I didn't list here.