MECS 4510 HW2

Name: Jiong Lin

UNI: jl6017

Instructor: Hod Lipson

Grace hour used: 0

Grace hour remained: 96

Columbia Engineering

10/26/2021

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.

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

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

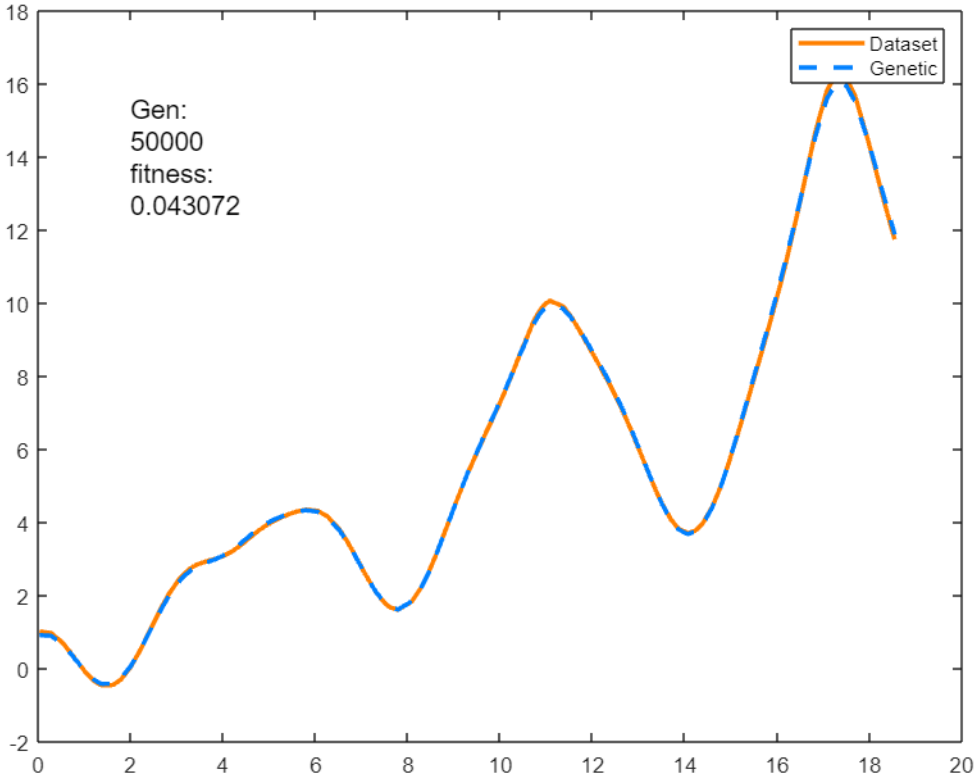

Figure 1: Dataset and best fit curve

1.2 Movie link:

Geneticprogramforsymbolicregression

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:

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

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

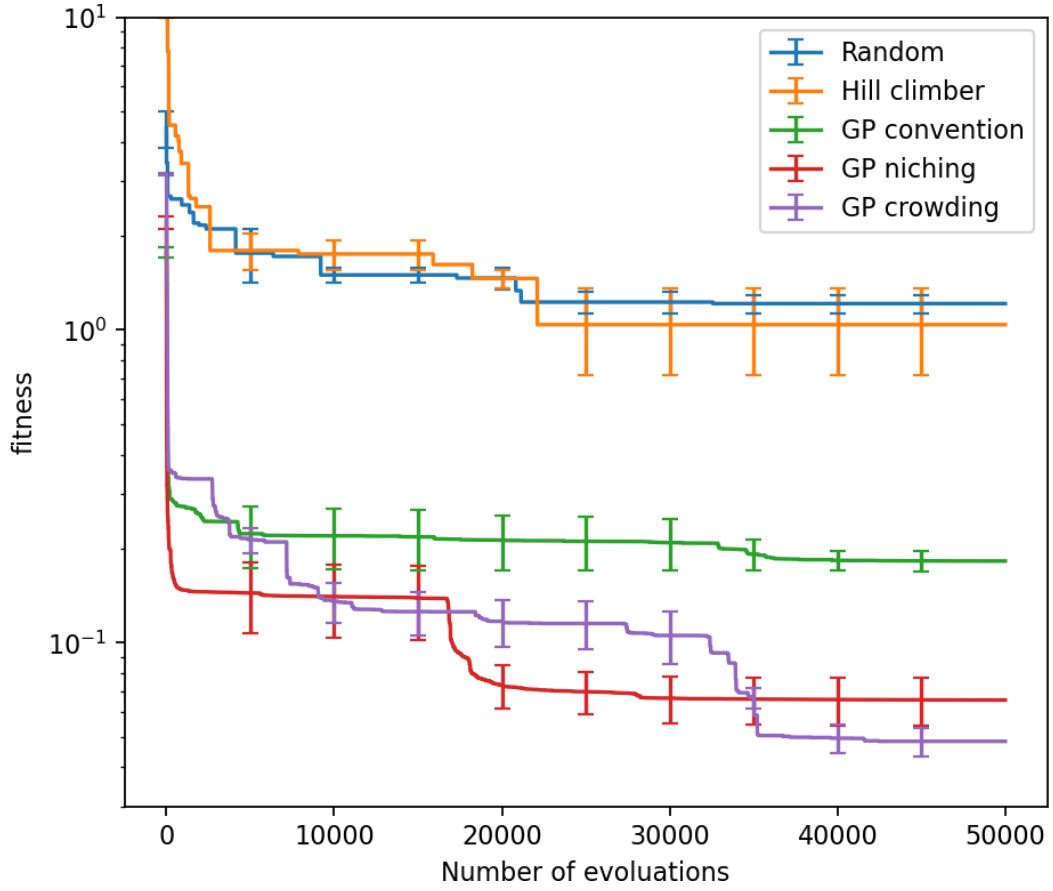

Figure 2: Learning curves with error bars

3.2 Dot plot

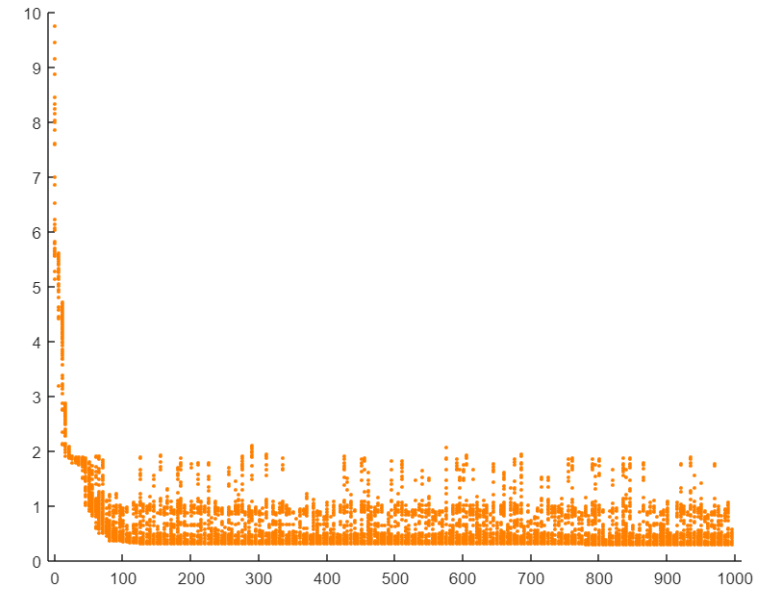

Figure 3: Dot plot for conventional genetic program

3.3 Diversity plot

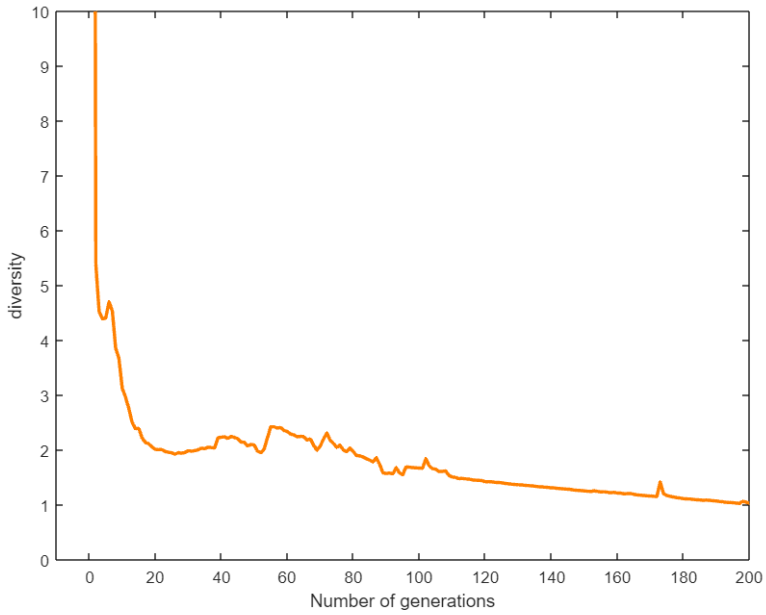

Figure 4: Diversity plot for conventional genetic program

3.4 Convergence plot

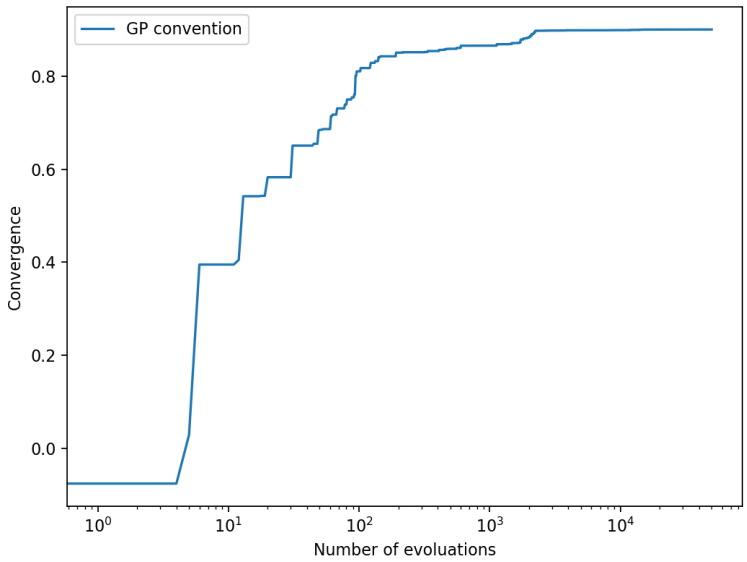

Figure 5: Convergence plot for conventional genetic program (Threshold=0.1)

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.

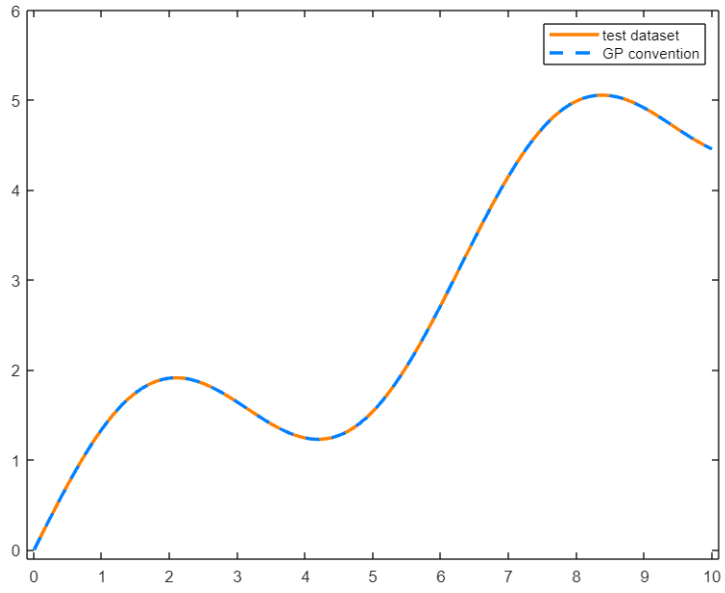

Figure 6: Debug test dataset plot

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

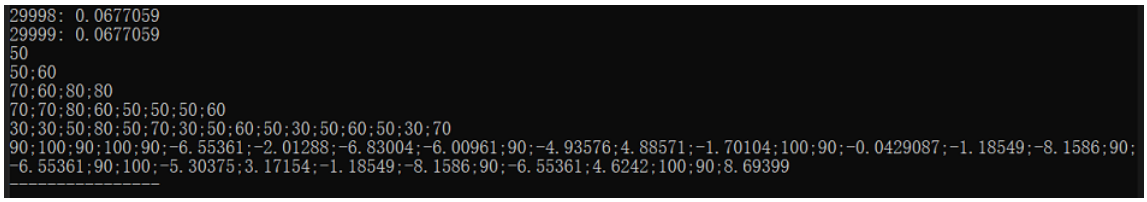

Figure 7: Automatically generated tree representing one of the solution

4 Appendix

4.1 Random search and hill climber c++

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <sstream>
#include <string>
#include <math.h>
#include <vector>
#include <random>
#include <ctime>
#include <stdlib.h>
#define LOG(x) std::cout<<x<<std::endl
using namespace std;

const int startHeight = 6;
const int pointNum = 1000;
const int fitNum = 200;
const int gen = 1000;
const int groupSize = 100;
const int halfSize = groupSize / 2;
const int chance = 30;
const int mysize = pow(2, startHeight) - 1;

void readFile(float x[], float y[]) {
        ifstream inFile("data.txt", ios::in);
        if (!inFile)
        {
                cout << "fail to open " << endl;
                exit(1);
        }
        int i = 0;
        string line;
        string field;
        while (getline(inFile, line))
        {
                string field;
                istringstream stin(line);
                getline(stin, field, ',');
                x[i] = atof(field.c_str());
                getline(stin, field, ',');
                y[i] = atof(field.c_str());
                i++;
        }
        inFile.close();
}

void create(vector<float>& myExp, int height) {
        //srand(time(NULL));
        //write the bottom nodes
        for (int i = pow(2, height - 1) - 1; i < pow(2, height) - 1; i = i + 2) {
                int r1 = rand() % 6;
                float r2 = rand() / float(RAND_MAX) + rand() % 19 - 10;
                float r3 = rand() / float(RAND_MAX) + rand() % 19 - 10;
                switch (rand() % 6)
                {
                case 0:myExp[i] = 100; myExp[i + 1] = 100; break;
                case 1:myExp[i] = 90; myExp[i + 1] = r2; break;
                case 2:myExp[i] = r2; myExp[i + 1] = 90; break;
                case 3:myExp[i] = 90; myExp[i + 1] = 100; break;
                case 4:myExp[i] = r2; myExp[i + 1] = r3; break;
                case 5:myExp[i] = r2; myExp[i + 1] = 100; break;
                }
        }
        for (int h = height - 1; h > 1; h--) {
                float r4 = rand() / float(RAND_MAX) + rand() % 19 - 10;
                for (int i = pow(2, h - 1) - 1; i < pow(2, h) - 1; i++) {
                        if (myExp[2 * i + 1] == 100 && myExp[2 * i + 2] == 100) {
                                switch (rand() % 3)
                                {
                                case 0:myExp[i] = 100; break;
```

```
                                case 1:myExp[i] = 90; break;
                                case 2:myExp[i] = r4; break;
                                }
                        }
                        else if (myExp[2 * i + 1] == 100 || myExp[2 * i + 2] == 100) {
                                switch (rand() % 2)
                                {
                                case 0:myExp[i] = 30; break;
                                case 1:myExp[i] = 40; break;
                                }
                        }
                        else {
                                int r = rand() % 4;
                                switch (r)
                                {
                                case 0:myExp[i] = 50; break;
                                case 1:myExp[i] = 60; break;
                                case 2:myExp[i] = 70; break;
                                case 3:myExp[i] = 80; break;
                                }
                        }
                }
        }

        // write the top nod
        if (myExp[1] == 100 && myExp[2] == 100) {
                myExp[0] = rand() / float(RAND_MAX) + rand() % 19 - 10;
        }
        else if (myExp[1] == 100 || myExp[2] == 100) {
                switch (rand() % 2)
                {
                case 0:myExp[0] = 30; break;
                case 1:myExp[0] = 40; break;
                }
        }
        else {
                int r = rand() % 4;
                switch (r)
                {
                case 0:myExp[0] = 50; break;
                case 1:myExp[0] = 60; break;
                case 2:myExp[0] = 70; break;
                case 3:myExp[0] = 80; break;
                }
        }

}

float calculate(vector<float>& myRes, float x) {
        for (int i = pow(2, startHeight) - 2; i > 0; i = i - 2) {
                if (myRes[i] == 100 && myRes[i - 1] == 100) {
                        continue;
                }
                else if (myRes[i] == 100) {
                        if (myRes[(i - 2) / 2] == 30) {
                                if (myRes[i - 1] == 90) {
                                        myRes[i - 1] = x;
                                        myRes[(i - 2) / 2] = sin(x);
                                }
                                else {
                                        myRes[(i - 2) / 2] = sin(myRes[i - 1]);
                                }
                        }
                        else {
                                if (myRes[i - 1] == 90) {
                                        myRes[i - 1] = x;
                                        myRes[(i - 2) / 2] = cos(x);
                                }
                                else {
                                        myRes[(i - 2) / 2] = cos(myRes[i - 1]);
                                }
                        }
                }
```

```
                else if (myRes[i - 1] == 100) {
                        if (myRes[(i - 2) / 2] == 30) {
                                if (myRes[i] == 90) {
                                        myRes[i] = x;
                                        myRes[(i - 2) / 2] = sin(x);
                                }
                                else {
                                        myRes[(i - 2) / 2] = sin(myRes[i]);
                                }
                        }
                        else {
                                if (myRes[i] == 90) {
                                        myRes[i] = x;
                                        myRes[(i - 2) / 2] = cos(x);
                                }
                                else {
                                        myRes[(i - 2) / 2] = cos(myRes[i]);
                                }
                        }
                }
                else {
                        if (myRes[(i - 2) / 2] == 50) {
                                if (myRes[i] == 90) { myRes[i] = x; }
                                if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
                                float res = myRes[i - 1] + myRes[i];
                                myRes[(i - 2) / 2] = res;
                        }
                        else if (myRes[(i - 2) / 2] == 60) {
                                if (myRes[i] == 90) { myRes[i] = x; }
                                if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
                                float res = myRes[i - 1] - myRes[i];
                                myRes[(i - 2) / 2] = res;
                        }
                        else if (myRes[(i - 2) / 2] == 70) {
                                if (myRes[i] == 90) { myRes[i] = x; }
                                if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
                                float res = myRes[i - 1] * myRes[i];
                                myRes[(i - 2) / 2] = res;
                        }
                        if (myRes[(i - 2) / 2] == 80) {
                                if (myRes[i] == 90) { myRes[i] = x; }
                                if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
                                float res = myRes[i - 1] / myRes[i];
                                myRes[(i - 2) / 2] = res;
                        }
                }
        }
        return myRes[0];
}

float fitness(vector<float>& exp01, float xf[], float yf[]) {
        float n = .0;
        float yf1[fitNum];
        vector<float> mycopy;
        for (int j = 0; j < fitNum; j++) {
                mycopy = exp01;
                yf1[j] = calculate(mycopy, xf[j]);
        }
        for (int i = 0; i < fitNum; i++) {
                if (yf1[i] > yf[i]) { n += (yf1[i] - yf[i]); }
                else { n += (yf[i] - yf1[i]); }
        }
        return n / fitNum;
}

void mutation(vector<float>& myExp) {
        int size = myExp.size();
        int height = log(size + 1) / log(2);
        int r0;
        int r2 = rand() % 2;
        while (true) {
                r0 = rand() % (size - 1) + 1;
                if (myExp[r0] != 100) { break; }
```

```
        }
        if (r0 > (size - 1) / 2 - 1) {
                myExp[r0] = rand() / float(RAND_MAX) + rand() % 19 - 10;
        }
        else {
                int h0 = log(r0 + 1) / log(2);
                int h1 = height - h0;
                vector<float>smallTree(pow(2, h1) - 1);
                create(smallTree, h1);
                myExp[r0] = smallTree[0];
                int j = 2 * r0 + 1;
                int k = 2 * r0 + 2;
                int n = 1;
                while (true) {
                        if (k > (size - 1)) { break; }
                        else {
                                for (int i = j; i <= k; i++) { myExp[i] = smallTree[n]; n = n
                                    + 1; }
                                j = 2 * j + 1;
                                k = 2 * k + 2;
                        }
                }
        }
}

void expLog(vector<float>& myExp) {
        for (int i = 0; i < myExp.size(); i++) {
                cout << myExp[i];
                if (i == 0 || i == 2 || i == 6 || i == 14 || i == 30 || i == 62 || i == 126) {
                        cout << endl;
                }
                else {
                        cout << ";";
                }
        }
        cout << "_________________" << endl;
}

int main() {
        float* x0 = new float[pointNum];
        float* y0 = new float[pointNum];
        float* xf = new float[fitNum];
        float* yf = new float[fitNum];
        float* y1 = new float[fitNum];
        vector<float>myExpression(mysize, 0);

        readFile(x0, y0);

        for (int i = 0; i < fitNum; i++) {
                xf[i] = x0[i * 5];
                yf[i] = y0[i * 5];
        }

        srand(time(NULL));

        ofstream outFile("random03.csv", ios::out);
        // random loop
        float myfit = 50;
        for (int n = 0; n < 50000; n++) {
                outFile << n << ",";
                cout << n << ": ";
                create(myExpression, startHeight);
                float newfit = fitness(myExpression, xf, yf);
                if (newfit < myfit) {
                        myfit = newfit;
                }
                LOG(myfit);
                outFile << myfit << endl;
        }
        // hill climber loop
        /*create(myExpression, startHeight);
        float myfit = 10;
        for (int n = 0; n < 50000; n++) {
```

```
                outFile << n << ",";
                cout << n << ": ";
                mutation(myExpression);
                float newfit = fitness(myExpression, xf, yf);
                if (newfit < myfit) {
                        myfit = newfit;
                }
                LOG(myfit);
                outFile << myfit << endl;
        }*/

        expLog(myExpression);
        return 0;
}
```

4.2 conventional genetic program

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <sstream>
#include <string>
#include <math.h>
#include <vector>
#include <random>
#include <ctime>
#include <stdlib.h>
#define LOG(x) std::cout<<x<<std::endl
using namespace std;

const int startHeight = 6;
const int pointNum = 1000;
const int fitNum = 200;
const int gen = 1000;
const int groupSize = 100;
const int halfSize = groupSize / 2;
const int chance = 30;
const int mysize = pow(2, startHeight) - 1;

void readFile(float x[], float y[]) {
        ifstream inFile("data.txt", ios::in);
        if (!inFile)
        {
                cout << "fail to open " << endl;
                exit(1);
        }
        int i = 0;
        string line;
        string field;
        while (getline(inFile, line))
        {
                string field;
                istringstream stin(line);
                getline(stin, field, ',');
                x[i] = atof(field.c_str());
                getline(stin, field, ',');
                y[i] = atof(field.c_str());
                i++;
        }
        inFile.close();
}

void create(vector<float>& myExp, int height) {
        //srand(time(NULL));
        //write the bottom nodes
        for (int i = pow(2, height - 1) - 1; i < pow(2, height) - 1; i = i + 2) {
                int r1 = rand() % 6;
                float r2 = rand() / float(RAND_MAX) + rand() % 19 - 10;
                float r3 = rand() / float(RAND_MAX) + rand() % 19 - 10;
                switch (rand() % 6)
                {
                case 0:myExp[i] = 100; myExp[i + 1] = 100; break;
```

```
                case 1:myExp[i] = 90; myExp[i + 1] = r2; break;
                case 2:myExp[i] = r2; myExp[i + 1] = 90; break;
                case 3:myExp[i] = 90; myExp[i + 1] = 100; break;
                case 4:myExp[i] = r2; myExp[i + 1] = r3; break;
                case 5:myExp[i] = r2; myExp[i + 1] = 100; break;
                }
        }
        for (int h = height - 1; h > 1; h--) {
                float r4 = rand() / float(RAND_MAX) + rand() % 19 - 10;
                for (int i = pow(2, h - 1) - 1; i < pow(2, h) - 1; i++) {
                        if (myExp[2 * i + 1] == 100 && myExp[2 * i + 2] == 100) {
                                switch (rand() % 3)
                                {
                                case 0:myExp[i] = 100; break;
                                case 1:myExp[i] = 90; break;
                                case 2:myExp[i] = r4; break;
                                }
                        }
                        else if (myExp[2 * i + 1] == 100 || myExp[2 * i + 2] == 100) {
                                switch (rand() % 2)
                                {
                                case 0:myExp[i] = 30; break;
                                case 1:myExp[i] = 40; break;
                                }
                        }
                        else {
                                int r = rand() % 3;
                                switch (r)
                                {
                                case 0:myExp[i] = 50; break;
                                case 1:myExp[i] = 60; break;
                                case 2:myExp[i] = 70; break;
                                }
                        }
                }
        }

        // write the top nod
        if (myExp[1] == 100 && myExp[2] == 100) {
                myExp[0] = rand() / float(RAND_MAX) + rand() % 19 - 10;
        }
        else if (myExp[1] == 100 || myExp[2] == 100) {
                switch (rand() % 2)
                {
                case 0:myExp[0] = 30; break;
                case 1:myExp[0] = 40; break;
                }
        }
        else { myExp[0] = 80; }
}

void create2(vector<float>& myExp, int height) {
        //srand(time(NULL));
        //write the bottom nodes
        for (int i = pow(2, height - 1) - 1; i < pow(2, height) - 1; i = i + 2) {
                int r1 = rand() % 6;
                float r2 = rand() / float(RAND_MAX) + rand() % 19 - 10;
                float r3 = rand() / float(RAND_MAX) + rand() % 19 - 10;
                switch (rand() % 6)
                {
                case 0:myExp[i] = 100; myExp[i + 1] = 100; break;
                case 1:myExp[i] = 90; myExp[i + 1] = r2; break;
                case 2:myExp[i] = r2; myExp[i + 1] = 90; break;
                case 3:myExp[i] = 90; myExp[i + 1] = 100; break;
                case 4:myExp[i] = r2; myExp[i + 1] = r3; break;
                case 5:myExp[i] = r2; myExp[i + 1] = 100; break;
                }
        }
        for (int h = height - 1; h > 1; h--) {
                float r4 = rand() / float(RAND_MAX) + rand() % 19 - 10;
                for (int i = pow(2, h - 1) - 1; i < pow(2, h) - 1; i++) {
                        if (myExp[2 * i + 1] == 100 && myExp[2 * i + 2] == 100) {
                                switch (rand() % 3)
```

```
                                {
                                case 0:myExp[i] = 100; break;
                                case 1:myExp[i] = 90; break;
                                case 2:myExp[i] = r4; break;
                                }
                        }
                        else if (myExp[2 * i + 1] == 100 || myExp[2 * i + 2] == 100) {
                                switch (rand() % 2)
                                {
                                case 0:myExp[i] = 30; break;
                                case 1:myExp[i] = 40; break;
                                }
                        }
                        else {
                                int r = rand() % 4;
                                switch (r)
                                {
                                case 0:myExp[i] = 50; break;
                                case 1:myExp[i] = 60; break;
                                case 2:myExp[i] = 70; break;
                                case 3:myExp[i] = 80; break;
                                }
                        }
                }
        }

        // write the top nod
        if (myExp[1] == 100 && myExp[2] == 100) {
                myExp[0] = rand() / float(RAND_MAX) + rand() % 19 - 10;
        }
        else if (myExp[1] == 100 || myExp[2] == 100) {
                switch (rand() % 2)
                {
                case 0:myExp[0] = 30; break;
                case 1:myExp[0] = 40; break;
                }
        }
        else {
                int r = rand() % 4;
                switch (r)
                {
                case 0:myExp[0] = 50; break;
                case 1:myExp[0] = 60; break;
                case 2:myExp[0] = 70; break;
                case 3:myExp[0] = 80; break;
                }
        }

}

float calculate(vector<float>& myRes, float x) {
        for (int i = pow(2, startHeight) - 2; i > 0; i = i - 2) {
                if (myRes[i] == 100 && myRes[i - 1] == 100) {
                        continue;
                }
                else if (myRes[i] == 100) {
                        if (myRes[(i - 2) / 2] == 30) {
                                if (myRes[i - 1] == 90) {
                                        myRes[i - 1] = x;
                                        myRes[(i - 2) / 2] = sin(x);
                                }
                                else {
                                        myRes[(i - 2) / 2] = sin(myRes[i - 1]);
                                }
                        }
                        else {
                                if (myRes[i - 1] == 90) {
                                        myRes[i - 1] = x;
                                        myRes[(i - 2) / 2] = cos(x);
                                }
                                else {
                                        myRes[(i - 2) / 2] = cos(myRes[i - 1]);
                                }
```

```
                        }
                }
                else if (myRes[i - 1] == 100) {
                        if (myRes[(i - 2) / 2] == 30) {
                                if (myRes[i] == 90) {
                                        myRes[i] = x;
                                        myRes[(i - 2) / 2] = sin(x);
                                }
                                else {
                                        myRes[(i - 2) / 2] = sin(myRes[i]);
                                }
                        }
                        else {
                                if (myRes[i] == 90) {
                                        myRes[i] = x;
                                        myRes[(i - 2) / 2] = cos(x);
                                }
                                else {
                                        myRes[(i - 2) / 2] = cos(myRes[i]);
                                }
                        }
                }
                else {
                        if (myRes[(i - 2) / 2] == 50) {
                                if (myRes[i] == 90) { myRes[i] = x; }
                                if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
                                float res = myRes[i - 1] + myRes[i];
                                myRes[(i - 2) / 2] = res;
                        }
                        else if (myRes[(i - 2) / 2] == 60) {
                                if (myRes[i] == 90) { myRes[i] = x; }
                                if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
                                float res = myRes[i - 1] - myRes[i];
                                myRes[(i - 2) / 2] = res;
                        }
                        else if (myRes[(i - 2) / 2] == 70) {
                                if (myRes[i] == 90) { myRes[i] = x; }
                                if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
                                float res = myRes[i - 1] * myRes[i];
                                myRes[(i - 2) / 2] = res;
                        }
                        if (myRes[(i - 2) / 2] == 80) {
                                if (myRes[i] == 90) { myRes[i] = x; }
                                if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
                                float res = myRes[i - 1] / myRes[i];
                                myRes[(i - 2) / 2] = res;
                        }
                }
        }
        return myRes[0];
}

float fitness(float yf1[], float yf2[]) {
        float n = .0;
        for (int i = 0; i < fitNum; i++) {
                if (yf1[i] > yf2[i]) { n += (yf1[i] - yf2[i]); }
                else { n += (yf2[i] - yf1[i]); }
        }
        return n / fitNum;
}

void mutation(vector<float>& myExp) {
        int size = myExp.size();
        int height = log(size + 1) / log(2);
        int r0;
        int r2 = rand() % 2;
        while (true) {
                r0 = rand() % (size - 1) + 1;
                if (myExp[r0] != 100) { break; }
        }
        if (r0 > (size - 1) / 2 - 1) {
                myExp[r0] = rand() / float(RAND_MAX) + rand() % 19 - 10;
        }
```

```
        else {
                int h0 = log(r0 + 1) / log(2);
                int h1 = height - h0;
                vector<float>smallTree(pow(2, h1) - 1);
                create2(smallTree, h1);
                myExp[r0] = smallTree[0];
                int j = 2 * r0 + 1;
                int k = 2 * r0 + 2;
                int n = 1;
                while (true) {
                        if (k > (size - 1)) { break; }
                        else {
                                for (int i = j; i <= k; i++) { myExp[i] = smallTree[n]; n = n + 1; }
                                j = 2 * j + 1;
                                k = 2 * k + 2;
                        }
                }
        }
}

void crossover(vector<float>& treeA, vector<float>& treeB) {
        int a; int b;
        while (true) {
                a = rand() % ((mysize - 1) / 2 - 1) + 1;
                if (treeA[a] != 100) { break; };
        }
        int h0 = log(a + 1) / log(2);
        int b1 = pow(2, h0);
        while (true) {
                b = rand() % b1 + b1 - 1;
                if (treeA[b] != 100) { break; };
        }
        int j1 = 2 * a + 1; int k1 = 2 * a + 2;
        int j2 = 2 * b + 1; int k2 = 2 * b + 2;
        float t = 0;
        t = treeA[a];
        treeA[a] = treeB[b];
        treeB[b] = t;
        while (true) {
                if (k2 > (mysize - 1)) { break; }
                else {
                        int i2 = j2;
                        for (int i1 = j1; i1 <= k1; i1++) {
                                float temp = 0;
                                temp = treeA[i1];
                                treeA[i1] = treeB[i2];
                                treeB[i2] = temp;
                                i2++;
                        }
                        j1 = 2 * j1 + 1; k1 = 2 * k1 + 2;
                        j2 = 2 * j2 + 1; k2 = 2 * k2 + 2;
                }
        }
}

float evo(vector<vector<float> >& group, float xf[], float yf[], float y1[]) {
        vector<float> temp01, temp02;
        vector<vector<float> > bigGroup;
        vector<vector<float> > yMatrix;
        vector<float> myResult;
        vector<float> myfit;
        int idx[2 * groupSize] = { 0 };
        // get big group, copy of group
        bigGroup = group;
        for (int i = 0; i < groupSize; i++) {
                temp01 = group[i];
                mutation(temp01);
                bigGroup.push_back(temp01); // double the big group with mutations
        }

        // get fitness
        for (int i = 0; i < 2 * groupSize; i++) {
```

```
                for (int j = 0; j < fitNum; j++) {
                        myResult = bigGroup[i];
                        y1[j] = calculate(myResult, xf[j]);
                }
                myfit.push_back(fitness(y1, yf)); // myfit has the same order as the big group
        }
        temp02 = myfit;
        sort(temp02.begin(), temp02.end()); // sort the copy of fitness list
        // write top half into group
        for (int i = 0; i < 2 * groupSize; i++) {
                for (int j = 0; j < 2 * groupSize; j++) {
                        if (temp02[i] == myfit[j]) { idx[i] = j; } // fitness from small to
                            large, write into new group
                }
        }
        for (int i = 0; i < groupSize; i++) {
                for (int j = 0; j < mysize; j++) {
                        group[i][j] = bigGroup[idx[i]][j];
                }
        }

        LOG(temp02[0]);
        return temp02[0];

}

float evo2(vector<vector<float> >& group, float xf[], float yf[]) {
        vector<float> temp;
        vector<vector<float> > bigGroup;
        vector<float> myResult;
        vector<float> myfit;
        float* y1 = new float[fitNum];
        int idx[2 * groupSize] = { 0 };
        bigGroup = group;
        for (int i = 0; i < groupSize; i = i + 2) {
                vector<float> temp01, temp02;
                temp01 = group[i];
                temp02 = group[i + 1];
                crossover(temp01, temp02);
                bigGroup.push_back(temp01);
                bigGroup.push_back(temp02);
        }
        //mutation
        for (int i = halfSize; i < groupSize; i++) {
                if (rand() % 100 < 50) {
                        vector<float> temp03;
                        temp03 = bigGroup[i];
                        mutation(temp03); //check here
                        bigGroup[i] = temp03;
                }
        }
        // get fitness
        for (int i = 0; i < 2 * groupSize; i++) {
                for (int j = 0; j < fitNum; j++) {
                        myResult = bigGroup[i];
                        y1[j] = calculate(myResult, xf[j]);
                }
                myfit.push_back(fitness(y1, yf)); // myfit has the same order as the big group
        }
        temp = myfit;
        sort(temp.begin(), temp.end()); // sort the copy of fitness list
        // write top half into group
        for (int i = 0; i < 2 * groupSize; i++) {
                for (int j = 0; j < 2 * groupSize; j++) {
                        if (temp[i] == myfit[j]) { idx[i] = j; }
                }
        }
        // fitness from small to large, write into new group
        for (int i = 0; i < groupSize; i++) {
                for (int j = 0; j < mysize; j++) {
                        group[i][j] = bigGroup[idx[i]][j];
                }
        }
```

```
        LOG(temp[0]);
        return temp[0];
}

void expLog(vector<float>& myExp) {
        for (int i = 0; i < myExp.size(); i++) {
                cout << myExp[i];
                if (i == 0 || i == 2 || i == 6 || i == 14 || i == 30 || i == 62 || i == 126) {
                        cout << endl;
                }
                else {
                        cout << ";";
                }
        }
        cout << "________________" << endl;
}

int main() {
        float* x0 = new float[pointNum];
        float* y0 = new float[pointNum];
        float* xf = new float[fitNum];
        float* yf = new float[fitNum];
        float* y1 = new float[fitNum];
        vector<vector<float> > group;
        vector<float>myExpression(mysize, 0);

        readFile(x0, y0);

        for (int i = 0; i < fitNum; i++) {
                xf[i] = x0[i * 5];
                yf[i] = y0[i * 5];
        }

        srand(time(NULL));
        // creat a group of expressions
        for (int i = 0; i < groupSize; i++) {
                create2(myExpression, startHeight);
                group.push_back(myExpression);
        }

        ofstream outFile("gasimple03.csv", ios::out);
        ofstream outFile02("Ygasimple03.csv", ios::out);
        // evo loop
        for (int n = 0; n < 50000; n++) {
                outFile << n << ",";
                cout << n << ": ";
                float tempfit = evo2(group, xf, yf);
                outFile << tempfit << endl;
        }

        expLog(group[0]);
        vector<float> result;
        for (int j = 0; j < fitNum; j++) {
                result = group[0];
                outFile02 << calculate(result, xf[j]) << ",";
        }
        return 0;
}
```

4.3 Niching genetic program

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <sstream>
#include <string>
#include <math.h>
#include <vector>
#include <random>
#include <ctime>
#include<stdlib.h>
#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                                  {
```

```
                                case 0:myExp[i] = 50; break;
                                case 1:myExp[i] = 60; break;
                                case 2:myExp[i] = 70; break;
                                case 3:myExp[i] = 80; break;
                                }
                        }
                }
        }

        // write the top nod
        if (myExp[1] == 100 && myExp[2] == 100) {
                myExp[0] = rand() / float(RAND_MAX) + rand() % 19 - 10;
        }
        else if (myExp[1] == 100 || myExp[2] == 100) {
                switch (rand() % 2)
                {
                case 0:myExp[0] = 30; break;
                case 1:myExp[0] = 40; break;
                }
        }
        else {
                int r = rand() % 4;
                switch (r)
                {
                case 0:myExp[0] = 50; break;
                case 1:myExp[0] = 60; break;
                case 2:myExp[0] = 70; break;
                case 3:myExp[0] = 80; break;
                }
        }

}

float calculate(vector<float>& myRes, float x) {
        for (int i = pow(2, startHeight) - 2; i > 0; i = i - 2) {
                if (myRes[i] == 100 && myRes[i - 1] == 100) {
                        continue;
                }
                else if (myRes[i] == 100) {
                        if (myRes[(i - 2) / 2] == 30) {
                                if (myRes[i - 1] == 90) {
                                        myRes[i - 1] = x;
                                        myRes[(i - 2) / 2] = sin(x);
                                }
                                else {
                                        myRes[(i - 2) / 2] = sin(myRes[i - 1]);
                                }
                        }
                        else {
                                if (myRes[i - 1] == 90) {
                                        myRes[i - 1] = x;
                                        myRes[(i - 2) / 2] = cos(x);
                                }
                                else {
                                        myRes[(i - 2) / 2] = cos(myRes[i - 1]);
                                }
                        }
                }
                else if (myRes[i - 1] == 100) {
                        if (myRes[(i - 2) / 2] == 30) {
                                if (myRes[i] == 90) {
                                        myRes[i] = x;
                                        myRes[(i - 2) / 2] = sin(x);
                                }
                                else {
                                        myRes[(i - 2) / 2] = sin(myRes[i]);
                                }
                        }
                        else {
                                if (myRes[i] == 90) {
                                        myRes[i] = x;
                                        myRes[(i - 2) / 2] = cos(x);
                                }
```

```
                                else {
                                        myRes[(i - 2) / 2] = cos(myRes[i]);
                                }
                        }
                }
                else {
                        if (myRes[(i - 2) / 2] == 50) {
                                if (myRes[i] == 90) { myRes[i] = x; }
                                if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
                                float res = myRes[i - 1] + myRes[i];
                                myRes[(i - 2) / 2] = res;
                        }
                        else if (myRes[(i - 2) / 2] == 60) {
                                if (myRes[i] == 90) { myRes[i] = x; }
                                if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
                                float res = myRes[i - 1] - myRes[i];
                                myRes[(i - 2) / 2] = res;
                        }
                        else if (myRes[(i - 2) / 2] == 70) {
                                if (myRes[i] == 90) { myRes[i] = x; }
                                if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
                                float res = myRes[i - 1] * myRes[i];
                                myRes[(i - 2) / 2] = res;
                        }
                        if (myRes[(i - 2) / 2] == 80) {
                                if (myRes[i] == 90) { myRes[i] = x; }
                                if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
                                float res = myRes[i - 1] / myRes[i];
                                myRes[(i - 2) / 2] = res;
                        }
                }
        }
        return myRes[0];
}

float fitness(float yf1[], float yf2[]) {
        float n = .0;
        for (int i = 0; i < fitNum; i++) {
                if (yf1[i] > yf2[i]) { n += (yf1[i] - yf2[i]); }
                else { n += (yf2[i] - yf1[i]); }
        }
        return n / fitNum;
}

void mutation(vector<float>& myExp) {
        int size = myExp.size();
        int height = log(size + 1) / log(2);
        int r0;
        int r2 = rand() % 2;
        while (true) {
                r0 = rand() % (size - 1) + 1;
                if (myExp[r0] != 100) { break; }
        }
        if (r0 > (size - 1) / 2 - 1) {
                myExp[r0] = rand() / float(RAND_MAX) + rand() % 19 - 10;
        }
        else {
                int h0 = log(r0 + 1) / log(2);
                int h1 = height - h0;
                vector<float>smallTree(pow(2, h1) - 1);
                create(smallTree, h1);
                myExp[r0] = smallTree[0];
                int j = 2 * r0 + 1;
                int k = 2 * r0 + 2;
                int n = 1;
                while (true) {
                        if (k > (size - 1)) { break; }
                        else {
                                for (int i = j; i <= k; i++) { myExp[i] = smallTree[n]; n = n
                                    + 1; }
                                j = 2 * j + 1;
                                k = 2 * k + 2;
                        }
```

```
                }
        }
}

void crossover(vector<float>& treeA, vector<float>& treeB) {
        int a; int b;
        while (true) {
                a = rand() % ((mysize - 1) / 2 - 1) + 1;
                if (treeA[a] != 100) { break; };
        }
        int h0 = log(a + 1) / log(2);
        int b1 = pow(2, h0);
        while (true) {
                b = rand() % b1 + b1 - 1;
                if (treeA[b] != 100) { break; };
        }
        int j1 = 2 * a + 1; int k1 = 2 * a + 2;
        int j2 = 2 * b + 1; int k2 = 2 * b + 2;
        float t = 0;
        t = treeA[a];
        treeA[a] = treeB[b];
        treeB[b] = t;
        while (true) {
                if (k2 > (mysize - 1)) { break; }
                else {
                        int i2 = j2;
                        for (int i1 = j1; i1 <= k1; i1++) {
                                float temp = 0;
                                temp = treeA[i1];
                                treeA[i1] = treeB[i2];
                                treeB[i2] = temp;
                                i2++;
                        }
                        j1 = 2 * j1 + 1; k1 = 2 * k1 + 2;
                        j2 = 2 * j2 + 1; k2 = 2 * k2 + 2;
                }
        }
}

void migration(vector<vector<float> >& group1, vector<vector<float> >& group2, vector<vector<
    float> >& group3,
        vector<vector<float> >& group4, vector<vector<float> >& group5) {
        int r1, r2, r3, r4, r5;
        r1 = rand() % 5 + 1;
        r2 = rand() % 5 + 1;
        r3 = rand() % 5 + 1;
        r4 = rand() % 5 + 1;
        r5 = rand() % 5 + 1;

        for (int i = 0; i < 4; i = i + 2) {
                vector<float> temp;
                temp = group1[r1 + i];
                group1[r1 + i] = group2[r2 + i];
                group2[r2 + i] = group3[r3 + i];
                group3[r3 + i] = group4[r4 + i];
                group4[r4 + i] = group5[r5 + i];
                group5[r5 + i] = temp;
        }
}

float evo2(vector<vector<float> >& group, float xf[], float yf[]) {
        vector<float> temp;
        vector<vector<float> > bigGroup;
        vector<float> myResult;
        vector<float> myfit;
        float* y1 = new float[pointNum];
        int idx[2 * groupSize] = { 0 };
        bigGroup = group;
        for (int i = 0; i < groupSize; i = i + 2) {
                vector<float> temp01, temp02;
                temp01 = group[i];
                temp02 = group[i + 1];
                crossover(temp01, temp02);
```

```
                bigGroup.push_back(temp01);
                bigGroup.push_back(temp02);
        }
        for (int i = halfSize; i < groupSize; i++) {
                if (rand() % 100 < chance) {
                        vector<float> temp03;
                        temp03 = bigGroup[i];
                        mutation(temp03); //check here
                }
        }
        // get fitness
        for (int i = 0; i < 2 * groupSize; i++) {
                for (int j = 0; j < pointNum; j++) {
                        myResult = bigGroup[i];
                        y1[j] = calculate(myResult, xf[j]);
                }
                myfit.push_back(fitness(y1, yf)); // myfit has the same order as the big group
        }
        temp = myfit;
        sort(temp.begin(), temp.end()); // sort the copy of fitness list
        // write top half into group
        for (int i = 0; i < 2 * groupSize; i++) {
                for (int j = 0; j < 2 * groupSize; j++) {
                        if (temp[i] == myfit[j]) { idx[i] = j; }
                }
        }
        // fitness from small to large, write into new group
        for (int i = 0; i < groupSize; i++) {
                for (int j = 0; j < mysize; j++) {
                        group[i][j] = bigGroup[idx[i]][j];
                }
        }
        LOG(temp[0]);
        return temp[0];
}

void expLog(vector<float>& myExp) {
        for (int i = 0; i < myExp.size(); i++) {
                cout << myExp[i];
                if (i == 0 || i == 2 || i == 6 || i == 14 || i == 30 || i == 62 || i == 126) {
                        cout << endl;
                }
                else {
                        cout << ";";
                }
        }
        cout << "------------------" << endl;
}

int main() {
        float* x0 = new float[pointNum];
        float* y0 = new float[pointNum];
        float* xf = new float[fitNum];
        float* yf = new float[fitNum];
        float* y1 = new float[fitNum];
        vector<vector<float> > group01, group02, group03, group04, group05;
        vector<float>myExpression(mysize, 0);

        readFile(x0, y0);
        float myFit = 1000;

        for (int i = 0; i < fitNum; i++) {
                xf[i] = x0[i * 10];
                yf[i] = y0[i * 10];
        }

        srand(time(NULL));
        // creat groups of expressions
        for (int i = 0; i < groupSize; i++) {
                create(myExpression, startHeight);
                group01.push_back(myExpression);
        }
        for (int i = 0; i < groupSize; i++) {
```

```
                create(myExpression, startHeight);
                group02.push_back(myExpression);
        }
        for (int i = 0; i < groupSize; i++) {
                create(myExpression, startHeight);
                group03.push_back(myExpression);
        }
        for (int i = 0; i < groupSize; i++) {
                create(myExpression, startHeight);
                group04.push_back(myExpression);
        }
        for (int i = 0; i < groupSize; i++) {
                create(myExpression, startHeight);
                group05.push_back(myExpression);
        }

        ofstream outFile("gene102201.csv", ios::out);
        ofstream outFile02("Y_gene01.csv", ios::out);
        // evo loop
        for (int n = 0; n < 10000; n++) {
                outFile << n << ",";
                cout << n << ": ";
                float tempfit01 = evo2(group01, xf, yf);
                outFile << tempfit01 << endl;

                cout << n << ": "; evo2(group02, xf, yf);
                cout << n << ": "; evo2(group03, xf, yf);
                cout << n << ": "; evo2(group04, xf, yf);
                cout << n << ": "; evo2(group05, xf, yf);
                if (n == 200) {
                        migration(group01, group02, group03, group04, group05);
                }
        }

        // print group
        expLog(group01[0]);
        vector<float> result;
        for (int j = 0; j < fitNum; j++) {
                result = group01[0];
                outFile02 << calculate(result, xf[j]) << ",";
        }
        /*
        for (int i = 0; i < groupSize; i++) {
                expLog(group[i]);
        }*/
        return 0;
}
```

4.4 Deterministic crowding genetic program

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <sstream>
#include <string>
#include <math.h>
#include <vector>
#include <random>
#include <ctime>
#include <stdlib.h>
#define LOG(x) std::cout<<x<<std::endl
using namespace std;

const int startHeight = 6;
const int pointNum = 1000;
const int fitNum = 100;
const int gen = 1000;
const int groupSize = 100;
const int halfSize = groupSize / 2;
const int chance = 50;
const int mysize = pow(2, startHeight) - 1;

```

```
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) {
```

```
                myExp[0] = rand() / float(RAND_MAX) + rand() % 19 - 10;
        }
        else if (myExp[1] == 100 || myExp[2] == 100) {
                switch (rand() % 2)
                {
                case 0:myExp[0] = 30; break;
                case 1:myExp[0] = 40; break;
                }
        }
        else {
                int r = rand() % 4;
                switch (r)
                {
                case 0:myExp[0] = 50; break;
                case 1:myExp[0] = 60; break;
                case 2:myExp[0] = 70; break;
                case 3:myExp[0] = 80; break;
                }
        }

}

float calculate(vector<float>& myRes, float x) {
        for (int i = pow(2, startHeight) - 2; i > 0; i = i - 2) {
                if (myRes[i] == 100 && myRes[i - 1] == 100) {
                        continue;
                }
                else if (myRes[i] == 100) {
                        if (myRes[(i - 2) / 2] == 30) {
                                if (myRes[i - 1] == 90) {
                                        myRes[i - 1] = x;
                                        myRes[(i - 2) / 2] = sin(x);
                                }
                                else {
                                        myRes[(i - 2) / 2] = sin(myRes[i - 1]);
                                }
                        }
                        else {
                                if (myRes[i - 1] == 90) {
                                        myRes[i - 1] = x;
                                        myRes[(i - 2) / 2] = cos(x);
                                }
                                else {
                                        myRes[(i - 2) / 2] = cos(myRes[i - 1]);
                                }
                        }
                }
                else if (myRes[i - 1] == 100) {
                        if (myRes[(i - 2) / 2] == 30) {
                                if (myRes[i] == 90) {
                                        myRes[i] = x;
                                        myRes[(i - 2) / 2] = sin(x);
                                }
                                else {
                                        myRes[(i - 2) / 2] = sin(myRes[i]);
                                }
                        }
                        else {
                                if (myRes[i] == 90) {
                                        myRes[i] = x;
                                        myRes[(i - 2) / 2] = cos(x);
                                }
                                else {
                                        myRes[(i - 2) / 2] = cos(myRes[i]);
                                }
                        }
                }
                else {
                        if (myRes[(i - 2) / 2] == 50) {
                                if (myRes[i] == 90) { myRes[i] = x; }
                                if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
                                float res = myRes[i - 1] + myRes[i];
                                myRes[(i - 2) / 2] = res;
```

```
                        }
                        else if (myRes[(i - 2) / 2] == 60) {
                                if (myRes[i] == 90) { myRes[i] = x; }
                                if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
                                float res = myRes[i - 1] - myRes[i];
                                myRes[(i - 2) / 2] = res;
                        }
                        else if (myRes[(i - 2) / 2] == 70) {
                                if (myRes[i] == 90) { myRes[i] = x; }
                                if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
                                float res = myRes[i - 1] * myRes[i];
                                myRes[(i - 2) / 2] = res;
                        }
                        if (myRes[(i - 2) / 2] == 80) {
                                if (myRes[i] == 90) { myRes[i] = x; }
                                if (myRes[i - 1] == 90) { myRes[i - 1] = x; }
                                float res = myRes[i - 1] / myRes[i];
                                myRes[(i - 2) / 2] = res;
                        }
                }
        }
        return myRes[0];
}

float fitness(vector<float>& exp01, float xf[], float yf[]) {
        float n = .0;
        float yf1[fitNum];
        vector<float> mycopy;
        for (int j = 0; j < fitNum; j++) {
                mycopy = exp01;
                yf1[j] = calculate(mycopy, xf[j]);
        }
        for (int i = 0; i < fitNum; i++) {
                if (yf1[i] > yf[i]) { n += (yf1[i] - yf[i]); }
                else { n += (yf[i] - yf1[i]); }
        }
        return n / fitNum;
}

void mutation(vector<float>& myExp) {
        int size = myExp.size();
        int height = log(size + 1) / log(2);
        int r0;
        int r2 = rand() % 2;
        while (true) {
                r0 = rand() % (size - 1) + 1;
                if (myExp[r0] != 100) { break; }
        }
        if (r0 > (size - 1) / 2 - 1) {
                myExp[r0] = rand() / float(RAND_MAX) + rand() % 19 - 10;
        }
        else {
                int h0 = log(r0 + 1) / log(2);
                int h1 = height - h0;
                vector<float>smallTree(pow(2, h1) - 1);
                create(smallTree, h1);
                myExp[r0] = smallTree[0];
                int j = 2 * r0 + 1;
                int k = 2 * r0 + 2;
                int n = 1;
                while (true) {
                        if (k > (size - 1)) { break; }
                        else {
                                for (int i = j; i <= k; i++) { myExp[i] = smallTree[n]; n = n
                                    + 1; }
                                j = 2 * j + 1;
                                k = 2 * k + 2;
                        }
                }
        }
}

void crossover(vector<float>& treeA, vector<float>& treeB) {
```

```
        int a; int b;
        while (true) {
                a = rand() % ((mysize - 1) / 2 - 1) + 1;
                if (treeA[a] != 100) { break; };
        }
        int h0 = log(a + 1) / log(2);
        int b1 = pow(2, h0);
        while (true) {
                b = rand() % b1 + b1 - 1;
                if (treeA[b] != 100) { break; };
        }
        int j1 = 2 * a + 1; int k1 = 2 * a + 2;
        int j2 = 2 * b + 1; int k2 = 2 * b + 2;
        float t = 0;
        t = treeA[a];
        treeA[a] = treeB[b];
        treeB[b] = t;
        while (true) {
                if (k2 > (mysize - 1)) { break; }
                else {
                        int i2 = j2;
                        for (int i1 = j1; i1 <= k1; i1++) {
                                float temp = 0;
                                temp = treeA[i1];
                                treeA[i1] = treeB[i2];
                                treeB[i2] = temp;
                                i2++;
                        }
                        j1 = 2 * j1 + 1; k1 = 2 * k1 + 2;
                        j2 = 2 * j2 + 1; k2 = 2 * k2 + 2;
                }
        }
}

float getDistance(vector<float>& exp01, vector<float>& exp02, float xf[]) {
        float y1[fitNum];
        float y2[fitNum];
        vector<float> mycopy;
        for (int j = 0; j < fitNum; j++) {
                mycopy = exp01;
                y1[j] = calculate(mycopy, xf[j]);
        }
        for (int j = 0; j < fitNum; j++) {
                mycopy = exp01;
                y2[j] = calculate(mycopy, xf[j]);
        }

        float n = .0;
        for (int i = 0; i < fitNum; i++) {
                if (y1[i] > y2[i]) { n += (y1[i] - y2[i]); }
                else { n += (y2[i] - y1[i]); }
        }
        return n / fitNum;
        delete[] y1;
        delete[] y2;
        mycopy.clear();
}

float evo3(vector<vector<float> >& group, float xf[], float yf[]) {
        shuffle(group.begin(), group.end(), std::default_random_engine(rand()));
        vector<float> myfit(groupSize, 0);
        vector<float> copyfit(groupSize, 0);
        vector<float> temp01, temp02, temp;
        vector<vector<float> > newgroup;
        float* dis = new float[8];
        newgroup = group;
        for (int i = 0; i < groupSize; i = i + 2) {
                temp01 = newgroup[i];
                temp02 = newgroup[i + 1];
                crossover(temp01, temp02);
                dis[0] = getDistance(temp01, newgroup[i], xf);
                dis[1] = getDistance(temp02, newgroup[i + 1], xf);
                dis[2] = getDistance(temp01, newgroup[i + 1], xf);
```

```
                dis[3] = getDistance(temp02, newgroup[i], xf);
                dis[4] = fitness(temp01, xf, yf);
                dis[5] = fitness(temp02, xf, yf);
                dis[6] = fitness(newgroup[i], xf, yf);
                dis[7] = fitness(newgroup[i + 1], xf, yf);
                if (dis[0] + dis[1] < dis[2] + dis[3]) {
                        if (dis[4] < dis[6]) {
                                newgroup[i] = temp01;
                                myfit[i] = dis[4];
                        }
                        else { myfit[i] = dis[6]; }
                        if (dis[5] < dis[7]) {
                                newgroup[i + 1] = temp02;
                                myfit[i + 1] = dis[5];
                        }
                        else { myfit[i + 1] = dis[7]; }
                }
                else {
                        if (dis[4] < dis[7]) {
                                newgroup[i + 1] = temp01;
                                myfit[i + 1] = dis[4];
                        }
                        else { myfit[i + 1] = dis[7]; }
                        if (dis[5] < dis[6]) {
                                newgroup[i] = temp02;
                                myfit[i] = dis[5];
                        }
                        else { myfit[i] = dis[6]; }
                }
        }

        copyfit = myfit;
        sort(myfit.begin(), myfit.end());

        int r0 = rand() % 50 + 20;
        for (int i = r0; i < r0 + 20; i = i + 2) {
                float t0 = myfit[i];
                for (int j = 0; j < groupSize; j++) {
                        if (copyfit[j] == t0) {
                                temp = newgroup[j];
                                mutation(temp);
                                newgroup[j] = temp;
                        }
                }
        }
        group = newgroup;


        LOG(myfit[0]);
        return myfit[0];
        newgroup.clear(); myfit.clear();
        temp.clear(); temp01.clear(); temp02.clear();

}

void expLog(vector<float>& myExp) {
        for (int i = 0; i < myExp.size(); i++) {
                cout << myExp[i];
                if (i == 0 || i == 2 || i == 6 || i == 14 || i == 30 || i == 62 || i == 126) {
                        cout << endl;
                }
                else {
                        cout << ";";
                }
        }
        cout << "_____________" << endl;
}

int main() {
        float* x0 = new float[pointNum];
        float* y0 = new float[pointNum];
        float* xf = new float[fitNum];
        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.