Assignment 2 – Symbolic Regression

Hongbo Zhu UNI: hz2629

Evolutionary Computation & Design Automation

(MECS E4510)

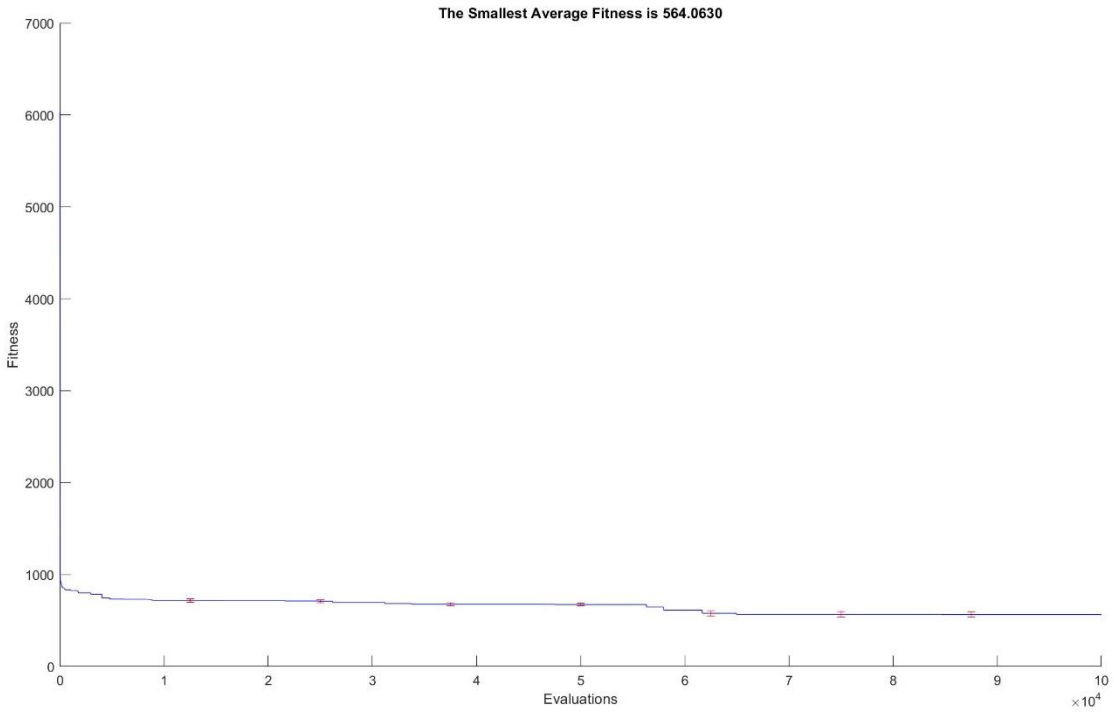
Instructor: Prof. Hod Lipson

Date Submitted: 10/12/2019

Grace Hours Used: 0 h

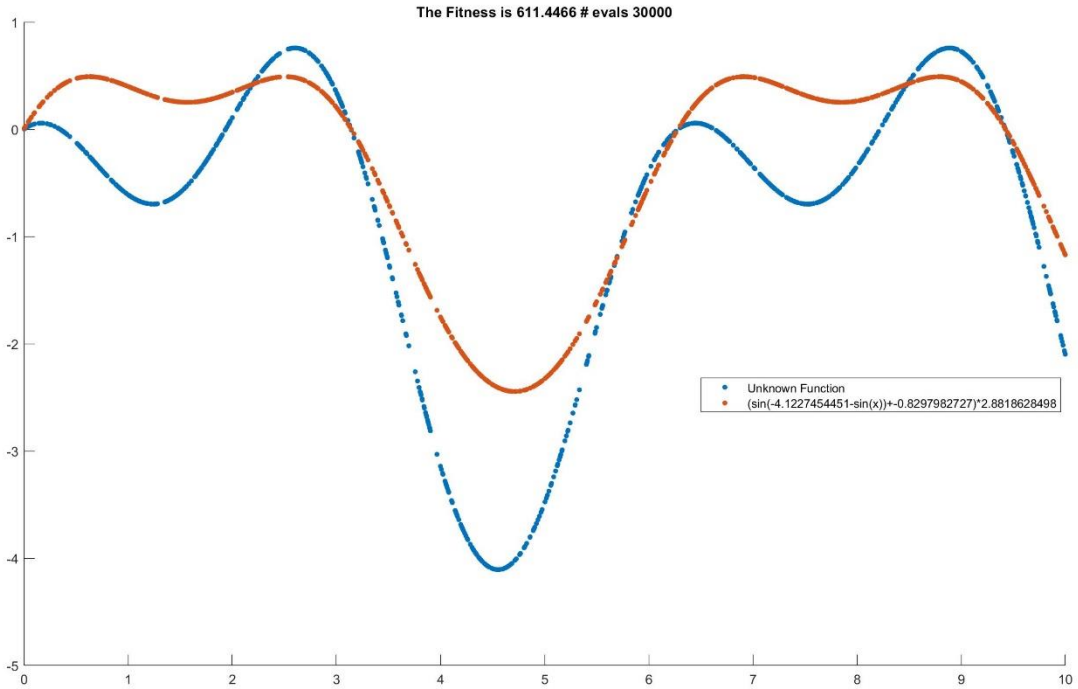
Grace Hours Remaining: 96 h

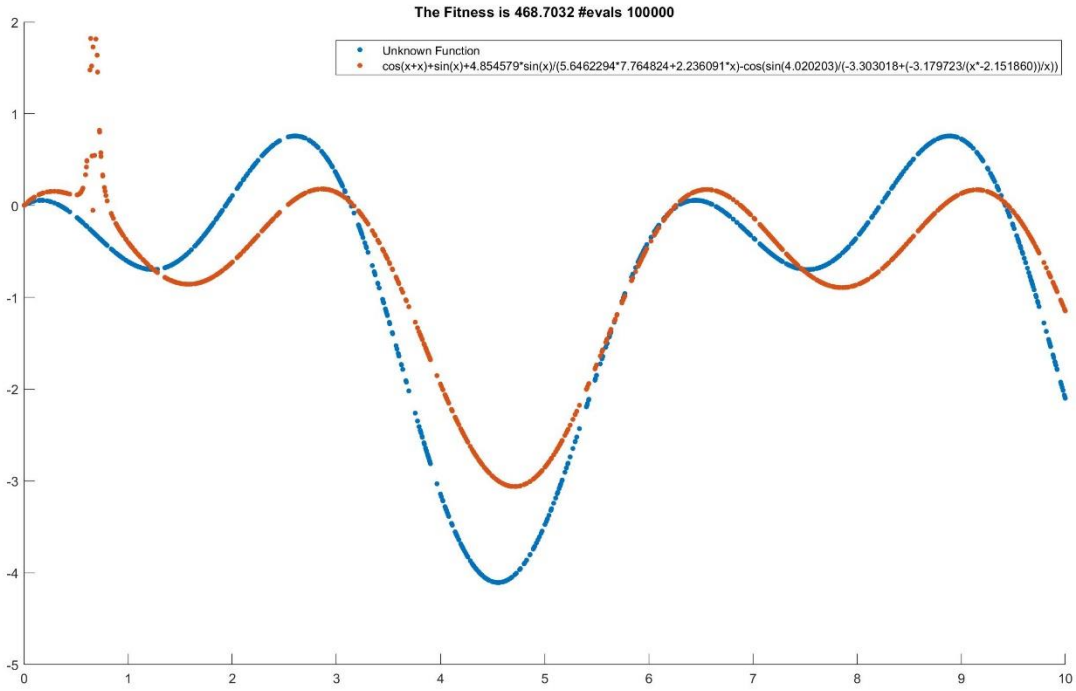
1. Learning Curve



After 100000 generations, the average fitness is 564.0630. Fitness is the sum of the difference value between each y coordinate value in data.txt and the outcome corresponding to each x coordinate of the function founded by random search method. The error bars in figure are calculated from 5 different fitness curves.

2. Function Figure




The Fitness is 468.7032 #evals 100000
Unknown Function
cos(x+x)+sin(x)+4.854579*sin(x)/(5.6462294*7.764824+2.236091*x)-cos(sin(4.020203)/(-3.303018+(-3.179723/(x*-2.151860))/x))
2
1
0
-1
-2
-3
-4
-5
0
1
2
3
4
5
6
7
8
9
10

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <chrono>
#include <random>
#include <ctime>
#include <iomanip>
#include <string>
#include <cmath>
#include <cstdio>

using namespace std;

#define POINT_NUM 1000
#define GENERATION 100000

void Read();
void Generate();
string To_string(string gene[],int x);
double Operate();

static string operator_dic[8] = { "+","-","*","/","sin","cos","x","a" };
static string calculator[4] = { "+","-","*","/" };
static string tri[2] = { "sin","cos" };
static string cons[2] = { "x","a" };

double xlist[POINT_NUM] = { 0 };
double ylist[POINT_NUM] = { 0 };
double BestYlist[POINT_NUM] = { 0 };
double fitnesslist[GENERATION] = { 0 };
double constant[100] = { 0 };

static string tree[256];

int main()
{
	double fitness = 10000.0;
	double bestfitness = fitness;
	srand(int(time(0)));

	Read();
	for (int loop = 0; loop < GENERATION; loop++)
	{
		for (int i = 0; i < 100; i++)
		{
			constant[i] = 0.0;
		}
		for (int i = 0; i < 256; i++)
		{
			tree[i] = (string)"0";
		}
		Generate();



		for (int i = 0; i < sizeof(tree); i++)
		{
			if (i < (256/2 - 1) && tree[i] == (string)"/" && tree[2 * i + 1] == (string)"0")
			{
				loop--;
				continue;
			}
			if (i < (256 / 2 - 1) && tree[i] == "*" && (tree[2 * i] == "0" || tree[2 * i + 1] ==
"0"))
			{
				loop--;
				continue;
			}
		}

		fitness = Operate();
		int j = 0;
		for (int i = 255; i > 0; i--)
		{
			if (tree[i] == (string)"a")
			{
				char buffer[20];
				sprintf_s(buffer, "%.10f", constant[j]);
				string str = buffer;
```

```
                                tree[i] = str;
                                j++;
                        }
                }

                string func = To_string(tree, 1);



                if (bestfitness >= fitness)
                {
                        bestfitness = fitness;
                        cout << "--------------------" << endl;
                        cout << loop << endl;
                        cout << func << endl;
                        cout << fitness << endl;
                }
                fitnesslist[loop] = bestfitness;
        }
        ofstream outputfitness("fitness.txt");  //import fitness into a txt file

        for (int i = 0; i < GENERATION; i++)
                outputfitness << setprecision(9) << fitnesslist[i] << endl;

        outputfitness.close();
}

void Read()
{
        int i = 0;
        ifstream input("data.txt");

        double a, b;
        while (input >> a >> b) {
                xlist[i] = a;
                ylist[i] = b;
                i++;
        }
        input.close();
}

void Generate()
{
        bool flag = false;
        tree[1] = operator_dic[rand() % 6];
        for (int i = 1; i < 7; i++)
        {
                for (int j = (int)pow(2, i); j < (int)pow(2, i+1); j++)
                {
                        for (int k = 0; k < sizeof(calculator); k++)
                        {
                                if (tree[int(j / 2.0)] == calculator[k])
                                {
                                        tree[j] = operator_dic[rand() % 8];
                                        flag = true;
                                        break;
                                }
                        }

                        if (tree[int(j / 2)] == tri[0] || tree[int(j / 2)] == tri[1])
                        {
                                if (j % 2 == 0)
                                        tree[j] = operator_dic[rand() % 8];
                        }
                }
        }

        for (int i = (int)pow(2, 7); i < (int)pow(2, 8); i++)
        {
                if (tree[int(i / 2)] != (string)"0" && tree[int(i / 2)] != (string)"x" && tree[int(i / 2)] !=
(string)"a")
                {
                        if (tree[int(i / 2)] == (string)"cos" || tree[int(i / 2)] == (string)"sin")
                        {
                                if (i % 2 == 0)
                                {
                                        tree[i] = operator_dic[6 + rand() % 2];
                                }
                        }
```

```
                        else
                        {
                                tree[i] = operator_dic[6 + rand() % 2];
                        }
                }
        }
}


string To_string(string gene[], int x)
{
        string function = "\0";
        if ((2 * x >= 255) || (gene[2 * x] == (string)"0"))
                function += gene[x];
        else if (gene[x] == tri[0] || gene[x] == tri[1])
                function = function + gene[x] + (string)"(" + To_string(gene, 2 * x) + (string)")";
        else
                function = function + (string)"(" + To_string(gene, 2 * x) + gene[x] + To_string(gene, 2 * x + 1)
+ (string)")";
        return function;
}

double Operate()
{
        int j = 0;
        for (int i = 0; i < 256; i++)
        {
                if (tree[i] == (string)"a")
                {
                        constant[j] = (double)rand() / (double)RAND_MAX * 20.0 - 10.0;
                        j++;
                }
        }

        double error = 0.0;
        double fitness[256];
        for (int k = 0; k < POINT_NUM; k++)
        {
                int j = 0;
                for (int i = 0; i < 256; i++)
                {
                        fitness[i] = 0.0;
                }

                for (int i = 255; i > 0; i--)
                {
                        if (tree[i] == (string)"a")
                        {
                                fitness[i] = constant[j];
                                j++;
                        }

                        else if (tree[i] == (string)"x")
                                fitness[i] = xlist[k];

                        else if (tree[i] == (string)"+")
                                fitness[i] = fitness[2 * i] + fitness[2 * i + 1];

                        else if (tree[i] == (string)"-")
                                fitness[i] = fitness[2 * i] - fitness[2 * i + 1];

                        else if (tree[i] == (string)"*")
                                fitness[i] = fitness[2 * i] * fitness[2 * i + 1];

                        else if (tree[i] == (string)"/")
                                fitness[i] = fitness[2 * i] / fitness[2 * i + 1];

                        else if (tree[i] == (string)"sin")
                                fitness[i] = sin(fitness[2 * i]);

                        else if (tree[i] == (string)"cos")
                                fitness[i] = cos(fitness[2 * i]);
                }
                error += fabs(fitness[1] - ylist[k]);
        }
        return error;
}
```