MECS 4510 Evolutionary Computation & Design Automation

Assignment 2: Symbolic Regression

Zhengyang Du

UNI: zd2219

Instructor: Hod Lipson

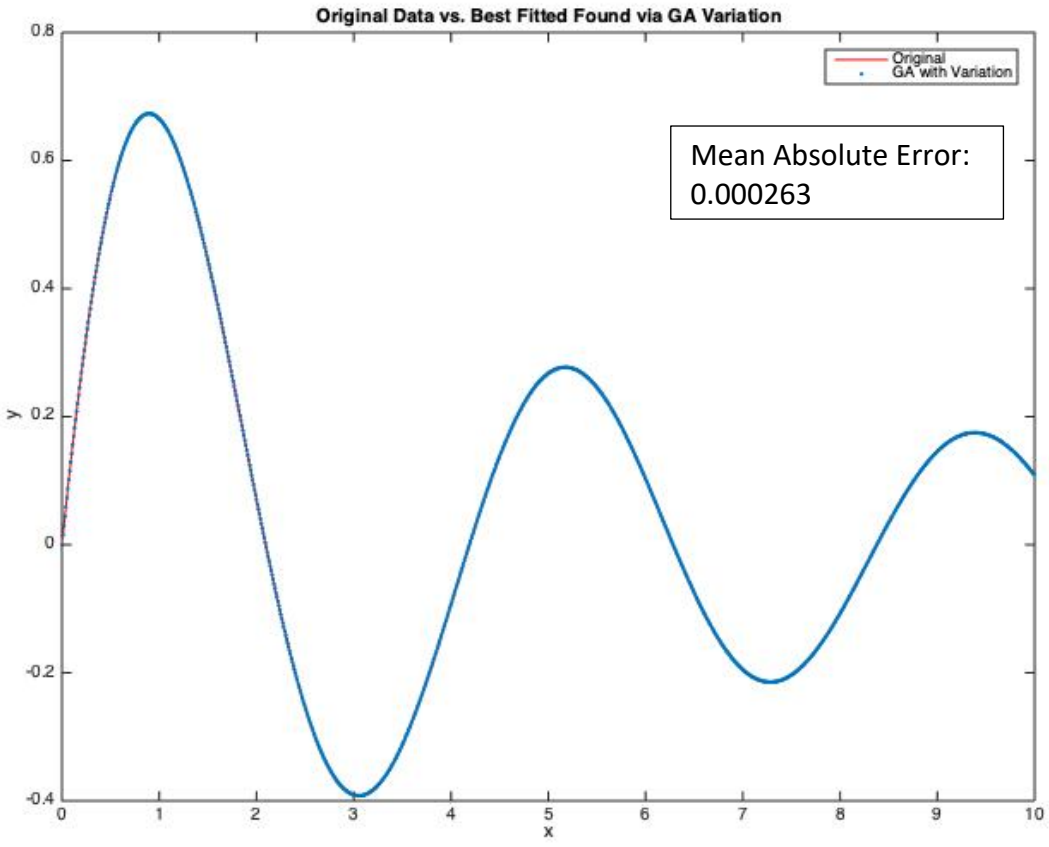
Submission Date: 10/19/2018

Grace Hours Used: 72

Grace Hours Remaining: 24 + 4

Result Summary

	Best Fitness (Mean Absolute Error)	# of Evaluations
Random Search	0.1172	1000000
Hill Climber	0.0022	
Genetic Algorithm	0.0018	
Genetic Algorithm with Variation	0.000263	
Best Formula Found	Y = Sin[(Sin[1.4601] X) (6.6802/(-4.4264))]/(((X - Cos[-5.5362]) + 2.7237)/-1.994) $\frac{\sin\left((\sin(1.4601)\,X)\left(-\frac{6.6802}{4.4264}\right)\right)}{-\frac{(X-\cos(-5.5362))+2.7237}{1.994}}$	



Methods

For this symbolic regression problem, random search, hill climber and genetic algorithm and with variation were applied to find the closest fitness and formula for the given data set. 4 times of running were applied to each of the algorithm for their learning curves. The programming language used was MATLAB for both running the algorithms and plotting. For the start of the search for the formula of the original data set, 6 layers of tree has to be constructed into a size of 63 array that included different constants, "x" variables, and different operators. The operators were stored as numbers from 1001 to 1006 such that 1001 represents "+", 1002 represents "-" and so on. By following the rule of the store the elements in the tree, the first two layers (index 1 to 3) of the tree were set to be any operators to avoid the formula to be too simple. After the first two layers, the assignment of the operators, constants, variable "x" were based on certain randomness and probability. Then the last layer (index of 32 to 63) would be all assigned by constants because the last layer of the tree and no operators allowed to proceed in the layer. The fitness would be represented by the sum of the difference between each of y value of the original data set and the formula (tree) constructed. Getting the y value of the constructed formula was achieved by plugging the x values from the original data set into the constructed formula using the concept of recursion. After having the functions for constructing the formula and finding the fitness (difference), the random search, hill climber, and the genetic programming could be applied.

Random Search:

Random search was done by first generating a formula (tree) randomly and set a dummy large fitness. After having the fitness of the first generated formula, then compare it to the dummy fitness. Once it was smaller than the dummy fitness, then the dummy fitness would be replaced by the fitness of the first generated formula. Next time, the same evaluation would be made between a new randomly generated formula and the current one, and the current one would be replaced if the fitness of the new one was better than the current one. By continuing this cycle to one million times, the final formula would be stored at the end of the cycle.

Because at each time a completely new formula was generated, so the results would be too random so that its efficiency was limited.

Hill Climber:

For hill climbing, the method used was to mutate the formula that we started with and then continuing to mutate its offspring for a million times and each time the fitness would be compared following the similar selection method used in random search. The mutation mechanism was to randomly select a node on the tree and randomly mutate all of its sub-trees. This mechanism was achieved by defining two functions that one was to assign random node elements at different selected layers of the tree and one was to call the first function and use recursion to finish assigning the elements into the rest of the tree. After having a sub-tree mutation, another mutation was introduced by randomly swapping the positions of any operators, swapping the position of a constant and a variable "x", and reassigning new values

to two random selected constants in the tree to increase the diversity. The hill climber method has a significant improvement than the random search.

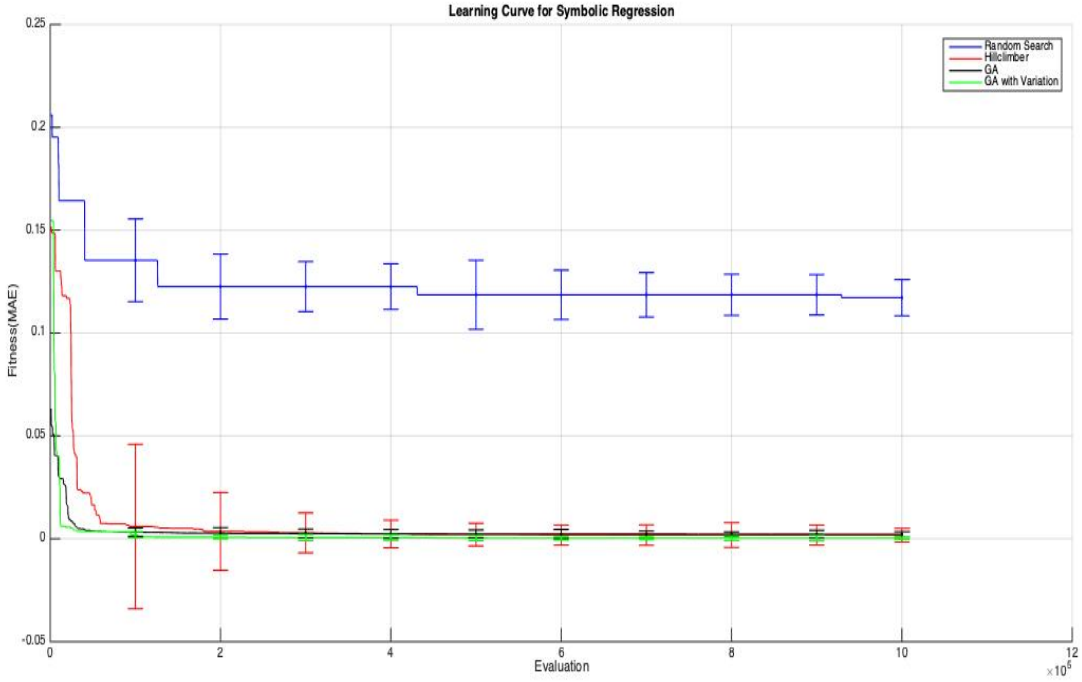
Genetic Algorithm:

The population size was kept at 20 for each generation. The selection method was to first randomly generate 20 parents and crossing over (swap sub-trees) between each two parents to generate two children. Then the two children would mutate using the mutation used in hill climber. The parents and children then would be put together to rank their fitness. The best 20 of them would be selected to become the new 20 parents for the next generation. The evaluation would be a million time to find the best fitted formula. The fitness found for the GA was not as good as the one with variation as can be seen in the results. The performance for both of the methods are similar (Variation is slightly better) but the GA with variation had better performance in finding the closest fitted solution.

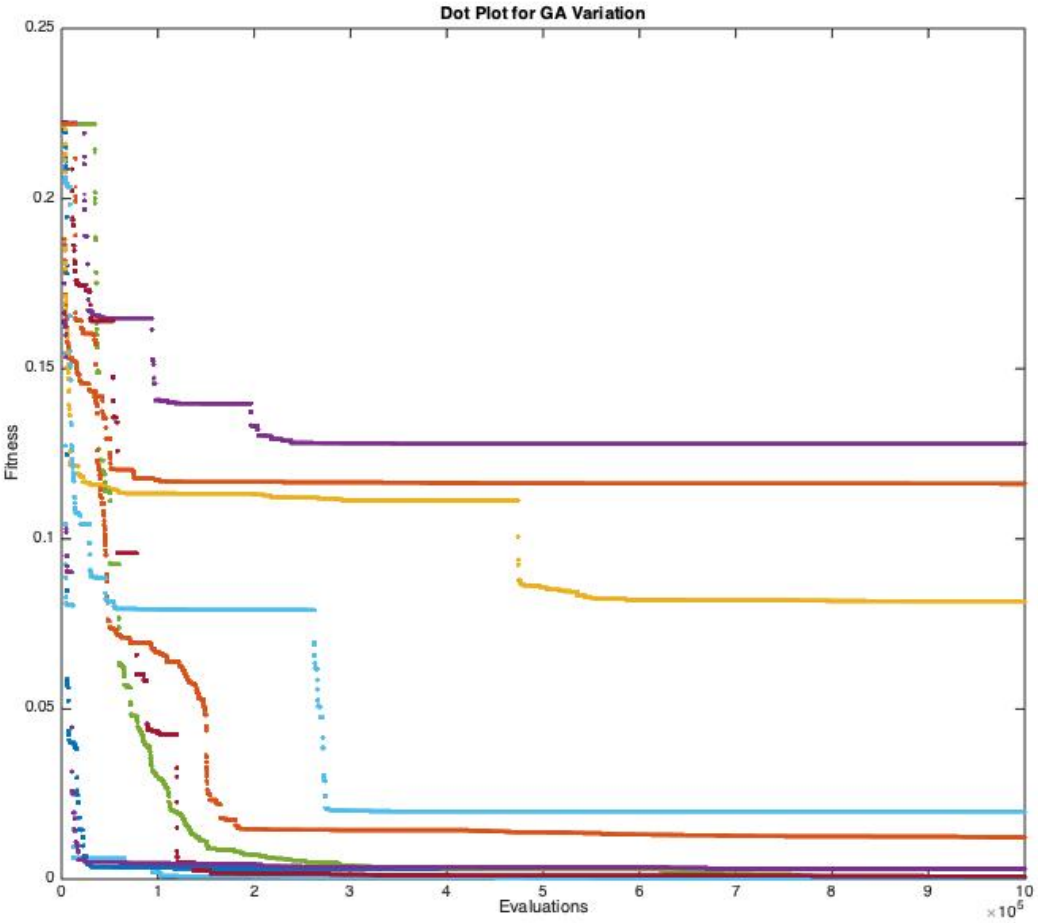
Genetic Algorithm With Variation:

For the genetic algorithm variation, the deterministic crowding mechanism was utilized. The population size of was kept at 20 so at the beginning there would be randomly generated parents. The crossover mechanism was to randomly swap the sub-trees of the two parents to generate the two children. Each time, two parents would be selected and crossover to generate two children and the two children would be mutated. Then both parents and children would be compared for their fitness and two of them with better fitness would be selected to become the parents for the next generation. This would keep running until new 20 parents were selected so that the same mechanism could be applied again to generate new parents. This deterministic crowding selection method found the better fitness faster than the previous GA.

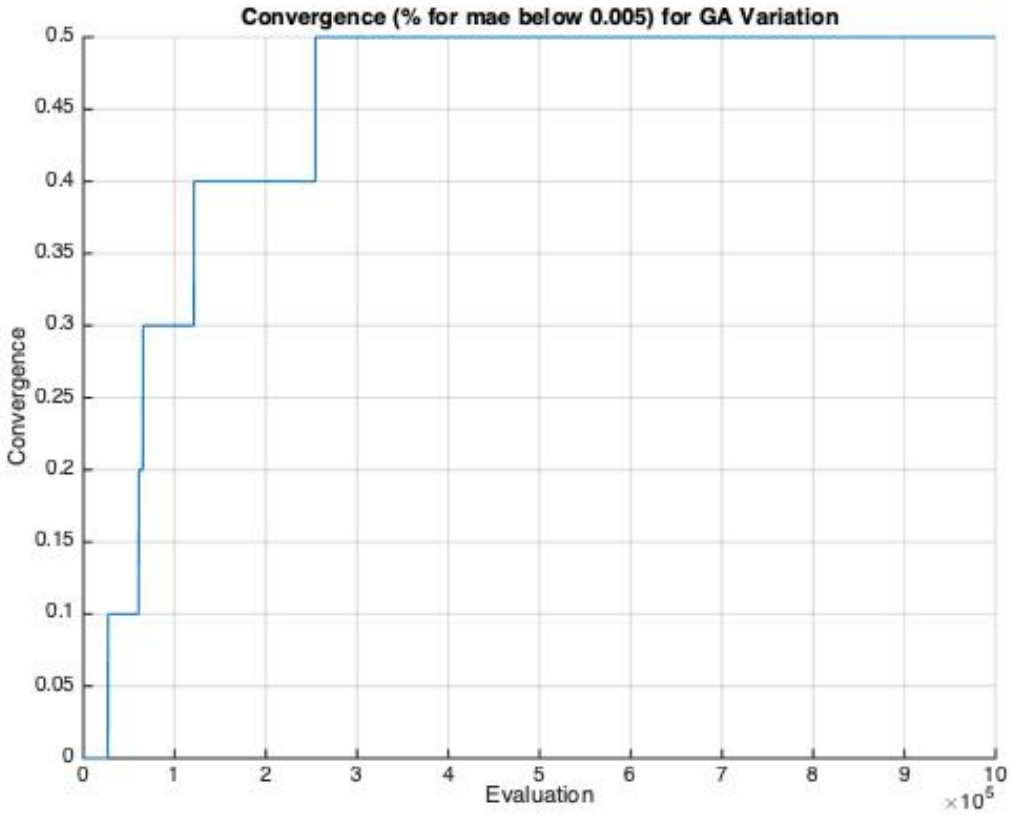
Performance Plot



Learning Curve for Symbolic Regression



Dot Plot for GA with Variation



Convergence Plot For GA with Variation

Appendix

Random Search

```
% MECS 4510 Assignment 2
% Symbolic Regression
% Random Search
close all
clear
clc

dataset = csvread('function1.csv');
x = dataset(:,1)';
y = dataset(:,2)';
% tree1 = store_tree();
% % func = find_func(1, tree1)
% s1 = find_solution(1, tree1, x);
% difference1 = find_diff(s1, y);

% random search
new_tree = zeros(1, 63);
evals = zeros(1, 1000000);
d = zeros(1, 1000000);
difference1 = 1000000;
for i = 1:1000000
    evals(i) = i;
    tree = store_tree();
    s = find_solution(1, tree, x);
    difference = find_diff(s, y);
    if difference < difference1
        new_tree = tree;
        difference1 = difference;
        sf = s;
    end
    d(i) = difference1;
end
d_mean = d./1000;
% [M, I] = min(best_diff);
% final_tree = best_trees(I);
% sf = find_solution(1, final_tree, x)
% finalfunc = find_func(1, final_tree)
e = evals(200:1000000);
di = d_mean(200:1000000);
figure()
plot(e, di)
title('fitness vs evaluations')
xlabel('evaluations')
ylabel('fitness')
figure()
plot(x, sf,'b')
hold on
plot(x, y, 'r')
title('data vs rsbest')
xlabel('x')
ylabel('y')
legend('randomsearch', 'original')
```

Hill Climber

```
% MECS 4510 Assignment 2
% Symbolic Regression
% Hill Climbing

clear all
close all
clc

dataset = csvread('function1.csv');
x = dataset(:,1)';
y = dataset(:,2)';

tree = store_tree();
```

```
solution = find_solution(1, tree, x);
difference1 = find_diff(solution, y);
evals = zeros(1, 1000000);
d = zeros(1, 1000000);

for i = 1:1000000
    evals(i) = i;
    random = randi([1, 63]);
    tree_mut = mutation(random, tree);
    treemutate = mutate(tree_mut);
    s = find_solution(1, treemutate, x);
    difference = find_diff(s, y);
    if difference < difference1
        tree = treemutate;
        difference1 = difference;
        sf = s;
    end
    d(i) = difference1;
end
d_mean = d./1000;
e = evals(200:1000000);
di = d_mean(200:1000000);
figure()
plot(e, di)
xlabel('evaluations')
ylabel('fitness')
title('fitness vs evaluations')

figure()
plot(x,y)
hold on
plot(x,sf)
xlabel('x')
ylabel('y')
title('muatation')
legend('original', 'mutation')
```

Genetic Algorithm

```
% Symbloc Regression
% Genetic Algorithm

clear all
close all
clc
tic
dataset = csvread('function1.csv');
x = dataset(:,1)';
y = dataset(:,2)';

% tree = store_tree();
% a = randi([1, 31]);
% j = 0;
% j = crossover(a,tree,j);
% z = zeros(1,j);
% k = 0;
% [k take_index] = take(a, k, z, tree);
% crosssection = tree(take_index);

% using deterministic crowding method to increase diversity
population = zeros(20, 63);
for i = 1:20
    population(i,:) = store_tree();
end
best = zeros(1, 1000000);
all_fitness = zeros(20, 1000000);
for l = 1: 1000000
    total_fitness = zeros(1, 20);
    total_y = zeros(20,1000);
    for j = 1:10
        parent1 = population(2*j-1, :);
        parent2 = population(2*j, :);
```

```
        child1 = parent1;
        child2 = parent2;
        a = randi([1, 31]);
        r = randi([1, 63]);
        h = crossover(a, parent1, 0);
        z = zeros(1, h);
        [k take_index] = take(a, 0, z, parent1);
        child1(take_index) = parent2(take_index);
        child2(take_index) = parent1(take_index);
        child1 = mutation(r, child1);
        child2 = mutation(r, child2);
        pc = zeros(4, 63);
        pc(1,:) = parent1;
        pc(2,:) = parent2;
        pc(3,:) = child1;
        pc(4,:) = child2;
        fitness = zeros(4, 1);
        yplots = zeros(4, 1000);
        yp1 = find_solution(1, parent1, x);
        dp1 = find_diff(yp1, y);
        yp2 = find_solution(1, parent2, x);
        dp2 = find_diff(yp2, y);
        yc1 = find_solution(1, child1, x);
        dc1 = find_diff(yc1, y);
        yc2 = find_solution(1, child2, x);
        dc2 = find_diff(yc2, y);
        fitness(1,:) = dp1;
        fitness(2,:) = dp2;
        fitness(3,:) = dc1;
        fitness(4,:) = dc2;
        yplots(1,:) = yp1;
        yplots(2,:) = yp2;
        yplots(3,:) = yc1;
        yplots(4,:) = yc2;
        [fitness, I] = sort(fitness);
        pc = pc(I',:);
        yplots = yplots(I',:);
        new_parent1 = pc(1, :);
        new_parent2 = pc(2, :);
        total_fitness(1, 2*j-1) = fitness(1,1);
        total_fitness(1, 2*j) = fitness(2,1);
        total_y(2*j-1, :) = yplots(1, :);
        total_y(2*j, :) = yplots(2, :);
        population(2*j-1, :) = new_parent1;
        population(2*j, :) = new_parent2;
    end
    all_fitness(:,l) = total_fitness';
    [total_fitness, I] = sort(total_fitness);
    total_y = total_y(I', :);
    best(1, l) = total_fitness(1,1);
end
population = population(I', :);
final_function = population(1, :);
final_y = total_y(1, :);
best_mean = best/1000;
final_x = [1:1000000];

figure()
plot(final_x, -best_mean)
xlabel('evaluations')
ylabel('fitness')
title('GA Fitness vs. Evaluations')

figure()
plot(x,y,'r')
hold on
plot(x,final_y,'b')
xlabel('x')
ylabel('y')
title('Data Set vs. GA')
legend('Original', 'GA')

toc
```

GA with Variation

```
% Symbloc Regression
% Genetic Algorithm with variation

clear all
close all
clc
tic
dataset = csvread('function1.csv');
x = dataset(:,1)';
y = dataset(:,2)';

% tree = store_tree();
% a = randi([1, 31]);
% j = 0;
% j = crossover(a,tree,j);
% z = zeros(1,j);
% k = 0;
% [k take_index] = take(a, k, z, tree);
% crosssection = tree(take_index);

% using deterministic crowding method to increase diversity
population = zeros(40, 63);
for i = 1:20
    population(i,:) = store_tree();
end
best = zeros(1, 1000000);
for l = 1: 1000000
    total_fitness = zeros(1, 40);
    total_y = zeros(40,1000);
    for j = 1:10
        parent1 = population(2*j-1, :);
        parent2 = population(2*j, :);
        child1 = parent1;
        child2 = parent2;
        a = randi([1, 31]);
        r = randi([1, 63]);
        h = crossover(a, parent1, 0);
        z = zeros(1, h);
        [k take_index] = take(a, 0, z, parent1);
        child1(take_index) = parent2(take_index);
        child2(take_index) = parent1(take_index);
        child1 = mutation(r, child1);
        child2 = mutation(r, child2);
        population(19+2*j, :) = child1;
        population(20+2*j, :) = child2;
        yp1 = find_solution(1, parent1, x);
        dp1 = find_diff(yp1, y);
        yp2 = find_solution(1, parent2, x);
        dp2 = find_diff(yp2, y);
        yc1 = find_solution(1, child1, x);
        dc1 = find_diff(yc1, y);
        yc2 = find_solution(1, child2, x);
        dc2 = find_diff(yc2, y);
        total_fitness(1, 2*j-1) = dp1;
        total_fitness(1, 2*j) = dp2;
        total_fitness(1, 19+2*j) = dc1;
        total_fitness(1, 20+2*j) = dc2;
        total_y(2*j-1, :) = yp1;
        total_y(2*j, :) = yp2;
        total_y(19+2*j, :) = yc1;
        total_y(20+2*j, :) = yc2;
    end
    [total_fitness, I] = sort(total_fitness);
    total_y = total_y(I', :);
    population = population(I', :);
    best(1, l) = total_fitness(1,1);
end
final_function = population(1, :);
final_y = total_y(1, :);
best_mean = best/1000;
```

```
final_x = [1:1000000];

figure()
plot(final_x, -best_mean)
xlabel('evaluations')
ylabel('fitness')
title('GA varation Fitness vs. Evaluations')

figure()
plot(x,y,'r')
hold on
plot(x,final_y,'b')
xlabel('x')
ylabel('y')
title('Data Set vs. GA varaition')
legend('Original', 'GA variation')

toc
```

Defined Function

```
function tree = store_tree() % define a function to assign random elements forming a tree
p1 = 0.5; % probability of assigning a operator
p2 = 0.5; % probability of assigning a variable x
operators = [1001 1002 1003 1004 1005 1006]; % all operators is assigned by a number
tree = zeros(1,63);  % 6 layers tree that has 63 elements
for i = 1:63
    if i == 1 || i == 2 || i == 3  % make sure first 3 nodes are operators
        random1 = randi([1, 6]);
        tree(i) = operators(random1);
    elseif rand(1) > p1 && i < 32
        random2 = randi([1, 6]);
        tree(i) = operators(random2);
    else
        if rand(1) > p2
            tree(i) = 1000; % set x is represented by 1000
        else
            tree(i) = rand(1)*20-10;  % assign random numbers from -10 to 10 except 0 into the
tree
        end
    end
end
end
```

```
function s = find_solution(z, tree, x)
n = tree(z) - 1000;
if n == 0
    s = x;
elseif n == 1
    s = find_solution(2*z, tree, x) + find_solution(2*z+1, tree, x);
elseif n == 2
    s = find_solution(2*z, tree, x) - find_solution(2*z+1, tree, x);
elseif n == 3
    s = find_solution(2*z, tree, x) .* find_solution(2*z+1, tree, x);
elseif n == 4
    s = find_solution(2*z, tree, x) ./ find_solution(2*z+1, tree, x);
elseif n == 5
    s = sin(find_solution(2*z, tree, x));
elseif n == 6
    s = cos(find_solution(2*z, tree, x));
else
    s = tree(z)*ones(1,1000);
end
end
```

```
function difference = find_diff(s, y)
difference = abs(s - y);
difference = sum(difference);
difference = min(difference, 1000000);
end


function func = find_func(h, tree)
g = tree(h) - 1000;
if g == 0
    func = string('x');
elseif g == 1
    func = join([string('(') find_func(2*h, tree) string(')+(') find_func(2*h+1, tree)
string(')')]);
elseif g == 2
    func = join([string('(') find_func(2*h, tree) string(')-(') find_func(2*h+1, tree)
string(')')]);
elseif g == 3
    func = join([string('(') find_func(2*h, tree) string(')*(') find_func(2*h+1, tree)
string(')')]);
elseif g == 4
    func = join([string('(') find_func(2*h, tree) string(')/(') find_func(2*h+1, tree)
string(')')]);
elseif g == 5
    func = join([string('sin(') find_func(2*h, tree) string(')')]);
elseif g == 6
    func = join([string('cos(') find_func(2*h, tree) string(')')]);
else
    func = string(tree(h));
end
end


function treemutate = mutate(tree)
flag1 = 0;
flag2 = 0;
flag3 = 0;
treemutate = tree;
while flag1 == 0 || flag2 == 0 || flag3 == 0
    r1 = randi([1, 31]);
    r2 = randi([1, 31]);
    r3 = randi([4, 63]);
    r4 = randi([4, 63]);
    r5 = randi([4, 63]);
    r6 = randi([4, 63]);
    if tree(r3) < 1000 && tree(r4) < 1000 && flag1 == 0
        treemutate(r3) = rand(1)*20-10;
        treemutate(r4) = rand(1)*20-10;
        flag1 = 1;
%         disp('r3 is ')
%         disp(r3)
%         disp('r4 is ')
%         disp(r4)
    end
    if tree(r5) < 1000 && tree(r6) == 1000 && flag3 == 0
        treemutate([r5 r6]) = treemutate([r6 r5]);
        flag3 = 1;
%         disp('r5 is ')
%         disp(r5)
%         disp('r6 is ')
%         disp(r6)
    end
    if (tree(r1)-1000 == 1 || tree(r1)-1000 == 2 || tree(r1) == 3 || tree(r1)-1000 == 4 ||
tree(r1)-1000 == 5 || tree(r1)-1000 == 6) && flag2 == 0
        if tree(r2)-1000 == 1 || tree(r2)-1000 == 2 || tree(r2) == 3 || tree(r2)-1000 == 4 ||
tree(r2)-1000 == 5 || tree(r2)-1000 == 6
            treemutate([r1 r2]) = treemutate([r2 r1]);
            flag2 = 1;
%             disp('r1 is ')
%             disp(r1)
%             disp('r2 is ')
```

```
%             disp(r2)
        end
    end
end
end


function branch = new_branch(i)
operators = [1001:1006];
p1 = 0.5;
p2 = 0.5;
if i == 1
    random1 = randi([1,6]);
    branch = operators(random1);
elseif rand(1) > p1 && i < 32
    random2 = randi([1,6]);
    branch = operators(random2);
else
    if rand(1) > p2
        branch = 1000;
    else
        branch = rand(1)*20-10;
    end
end
end



function tree = mutation(i, tree)
tree(i) = new_branch(i);
if i < 32
    tree = mutation(2*i, tree);
    tree = mutation(2*i+1, tree);
end
end



function [k z] = take(a, k, z, tree)
k = k + 1;
z(k) = a;
n = tree(a) - 1000;
if n > 0 && n < 5
    [k z] = take(2*a, k, z, tree);
    [k z] = take(2*a+1, k, z, tree);
elseif n > 4 && n < 7
    [k z] = take(2*a, k, z, tree);
end
end


function j = crossover(a,tree,j)
j = j + 1;
n = tree(a) - 1000;
if n>0 && n<5
    j = crossover(2*a,tree,j);
    j = crossover(2*a+1,tree,j);
elseif n>4 && n<7
    j = crossover(2*a,tree,j);
end
end
```