Symbolic Regression

Arman Paydarfar

AJP2246

MECS 4510 – Evolutionary Computation and Design Automation

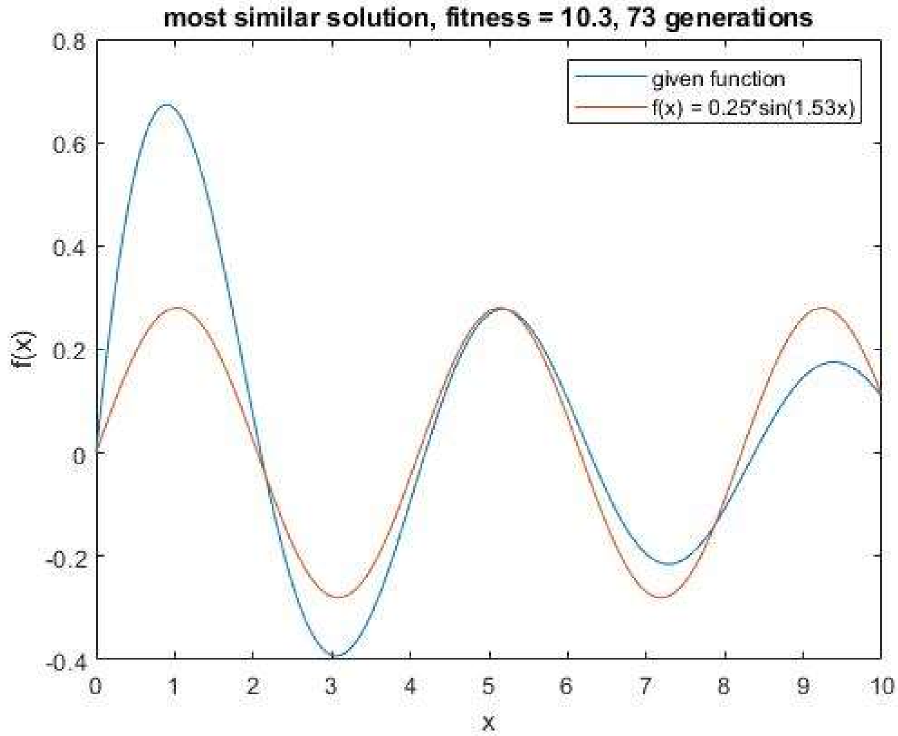
Hod Lipson

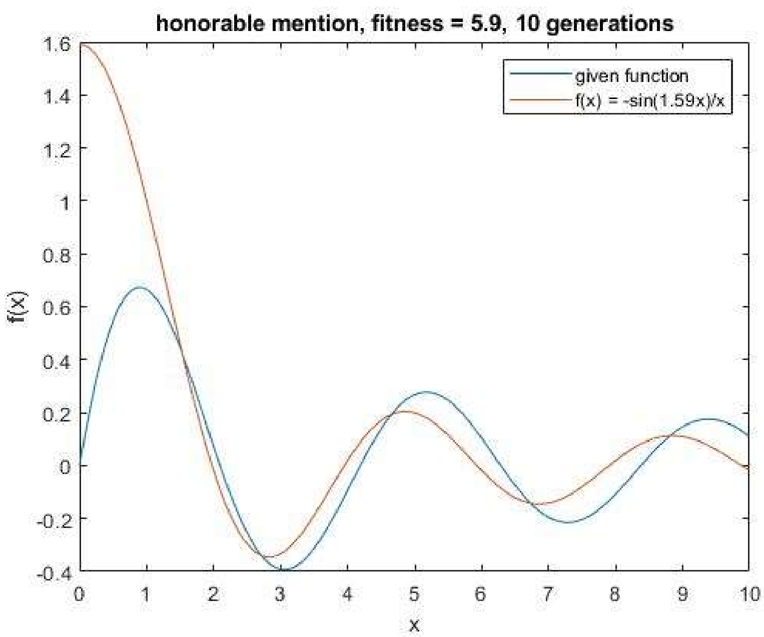
Submitted: October 19st, 2018

Grace Hours used: none (18 hours early)

Grace Hours remaining: 42 Hours

Results





Figs. 1A (top), 1B (bottom): Figure 1A is the best solution recorded by the genetic program written. The mean absolute error is equal to the inverse of the fitness, and therefore is equal to 0.097. Figure 1B was mentioned since, although the fitness was lower, and many better solutions were found, qualitatively seemed to have the most similar shape factor out of the solutions found. The best analytical solution was found to be f(x) = 0.25*sin(1.53*x).

The symbolic regression algorithm was attempted using multiple strategies. A random search, hill climber, and genetic program were implemented to find optimal solutions. All aspects of the problem were solved using MATLAB. The algorithms all have a few functions in common – one function forms a random cell array, where the first five elements are symbols denoting operations or constants, another creates an executable equation from that list, and finally, a short function that takes any dataset, and computes the mean absolute error, and its inverse, fitness. The hillclimber and random search use these three functions, whereas the genetic program uses these three, plus a few more that will be described in further detail.

The random search generates random lists using the function notated as “formlist”. Since this function forms a random list, it is called over and over again for a set number of evaluations. When the next list generated performs at a higher fitness level than the previous list, it is saved as the new “list”. This iterative process continues for an arbitrary number of evaluations. This algorithm was the quickest to run per evaluation, and performed surprisingly well.

The hill climber was created using a more deterministic approach. The algorithm forms a randomly generated list, and determines the fitness of that list. For the set number of iterations, the algorithm was designed to step through each element, and find a better performing replacement element. The list was effectively split into two, the first 5 elements were replaced with either symbols or constants, and the latter 6 elements were replaced with numbers. For each element of the first part of the list, the algorithm would cycle through the available replacement elements. If a replacement existed that created a greater fitness, the list was updated accordingly. For the latter 6 elements (all numbers), a random number was generated, and if the replacement number caused the list to perform better, it was kept. This process was conducted for all elements, and then repeated. This, in theory, should be a hill climber, as the algorithm will search the solution space and incrementally update the list with better solutions.

The genetic program was written to include mutation and crossover, while utilizing the same functions to create initial lists, construct equations, and determine fitness. The algorithm generates a population of random lists, and each list is evaluated to determine its fitness. The lists that perform the best are assigned as the initial parents, and thus the parents of the first generation. For each generation, the parents are input into a crossover function. The crossover function chooses a number of random pairs of parents (the same as the initial population size), and mates the parents to produce a population of children. For each parent in the pair, a tree is selected at random. These trees are then swapped to produce two children. If the first child performs better than either parent, then it joins the population. Otherwise, either the second child, or the best parent will join the population of children. After repeating this process, and creating a population of children, the children are all mutated. The mutation was designed to be tournament style. A random chance was assigned for each of the following events:

1. Swapping of two symbols
2. Swapping two numbers
3. Switching of a symbol to a different symbol or to a constant (snipping)
4. Switching the first element to a random algebraic operator
5. Switching a number or “x” to another number or “x”

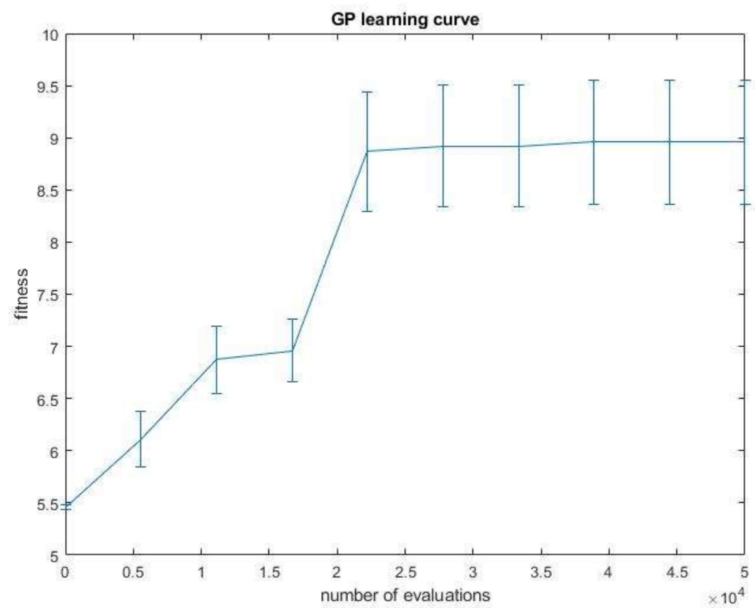
6. Switching a constant to another constant

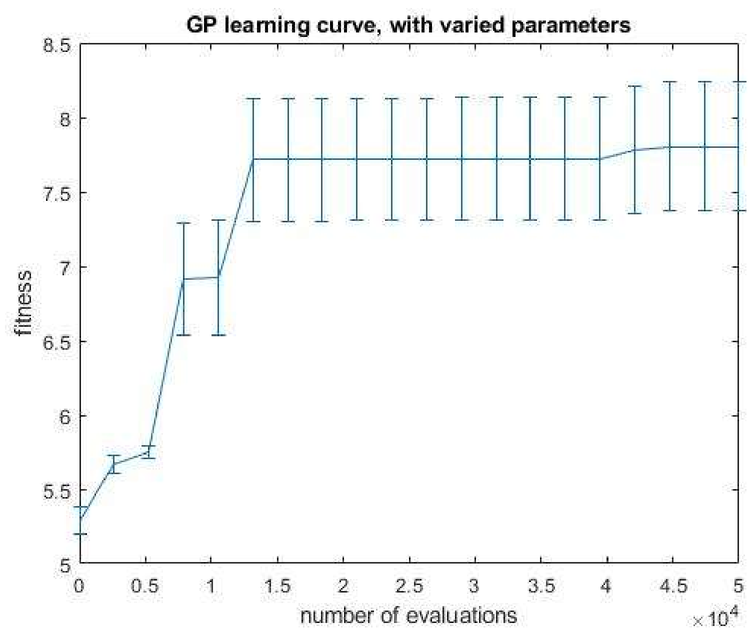
The chances are compiled and the changes are made, then the fitness of the mutated child is computed and compared with the fitness of the child. If the mutations fitness is less then that of the child, the process is repeated until a mutation is found that has a greater fitness. If no such mutated version is found within a set number of iterations, the algorithm is designed to select the original child. All the variable operators, such as the chance of each mutation, the size of the population, and number of parents, were experimented with to find the best result. The genetic program was run for 10 generations, which is approximately 50000 evaluations, the same as the random search and the hill climber. However, the best solution found was run for 73 generations, which is approximately 365000 evaluations.

The algorithm did not find the optimal solution overall, but it did outperform the random search and the hillclimber over 10 generations (50000 evaluations). There were a few possible reasons why the algorithm did not find the correct solution. Testing was conducted on simpler functions to reveal that the algorithm did indeed converge fairly quickly on the optimal result for simple equations. A more complicated equation was also tested that required a longer tree, and the algorithm got very close, but did not converge on the correct result, suggesting that the tree depth was not high enough. It can be deduced that the correct answer requires a tree for which the list is longer than the list that has been programmed for this assignment. The longest possible tree in the algorithm created for this assignment has a depth of 3. However, it is likely that a larger depth was required to find the optimal solution. In fact, it is possible that the solution found by the algorithm was the optimal solution with that constraint. Refer to the appendix to view the simpler functions that were used for testing. Additionally, referring to the dot plot and diversity plot shows an interesting behavior. It seems that genetic diversity is nearly zero once the algorithm reaches a certain generation, but only for the crossover function. Every time crossover is called, there is no diversity in the children that are populating the population matrix. Indeed, this was confirmed by viewing the "children" matrix after several generations and seeing that almost every child was in fact the same, or very similar. The reduction in diversity to this extent makes the only function that is useful the mutation function, but it is unclear why the lack of diversity is occurring in the first place. If the child found is truly the optimal child (with the constraints introduced by design), then it makes sense that after several generations, all of the children would be the same since the algorithm would have falsely converged on a local optimum. The alternative is that the crossover was not properly designed. Training was conducted on test cases, and the algorithms final tests were done on the function given by the assignment.

Appendix

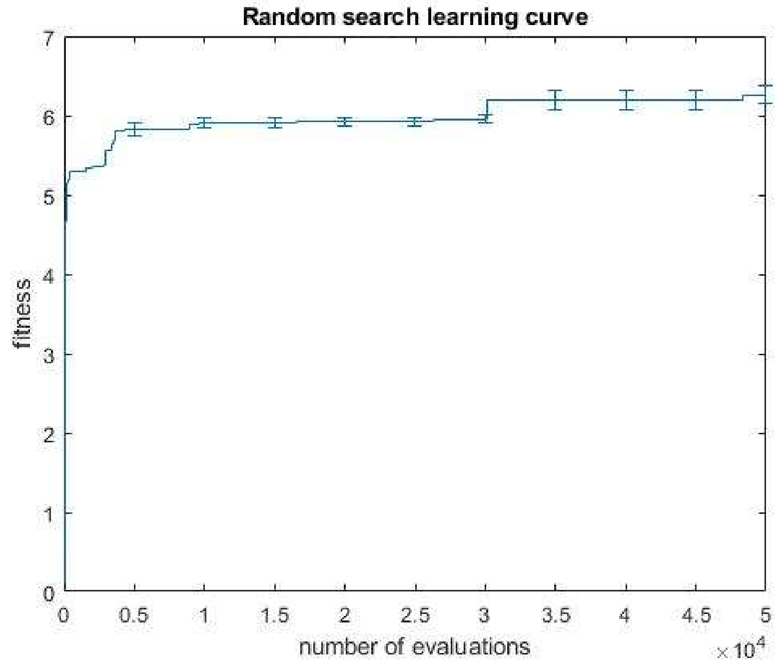
Learning curves

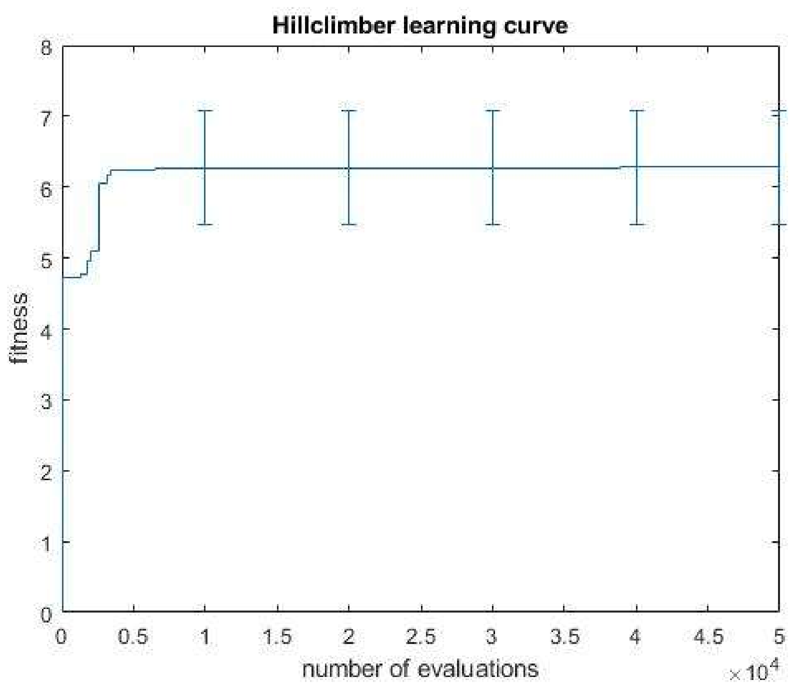




Figs. 2A (top), 2B (bottom): The learning curves for the genetic program with different sets of parameters. For case 2A, the chances of each of the 6 mutations (refer to the previous description) are as follows: (0.2,0.2,0.1,0.2,0.2,0.2). The population size is set to 100, and the number of selected parents are set to 20. For the bottom curve, the mutation rates are doubled, and both the population size and number of selected parents are halved. Note that while the number of evaluations are the same, the second case is run for twice as many generations. These plots are averaged over 4 trials.

Learning curves (continued)





Figs. 3A (top), 3B (bottom): The figures above show the learning curves for the random search, and hillclimber. The plots are averaged over 4 trials.

Diagnostic plots

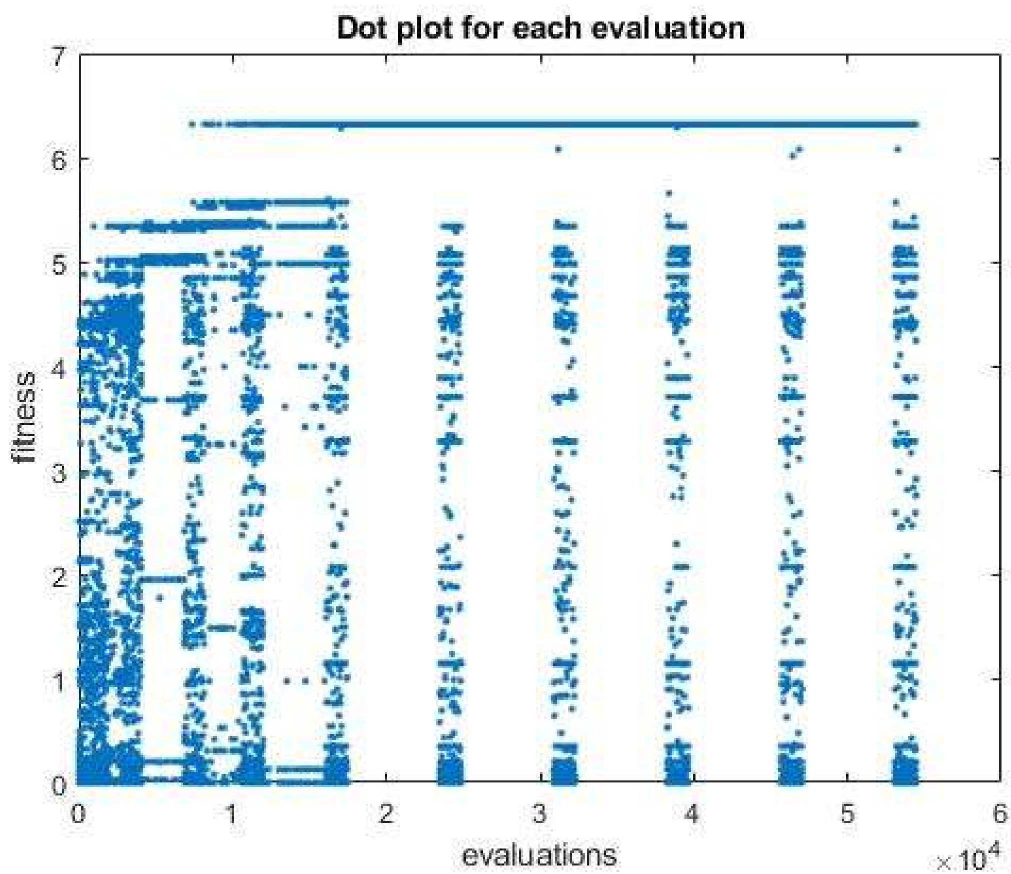


Fig. 4: As discussed in the report, and shown by the dot plot and the diversity plot (***Fig.*** 5), there is a lack of diversity that occurs during crossover for later generations. This effect becomes very apparent after a few generations. Fitness (inverse of mean absolute error vs. evaluation number) is plotted.

Diagnostic plots (continued)

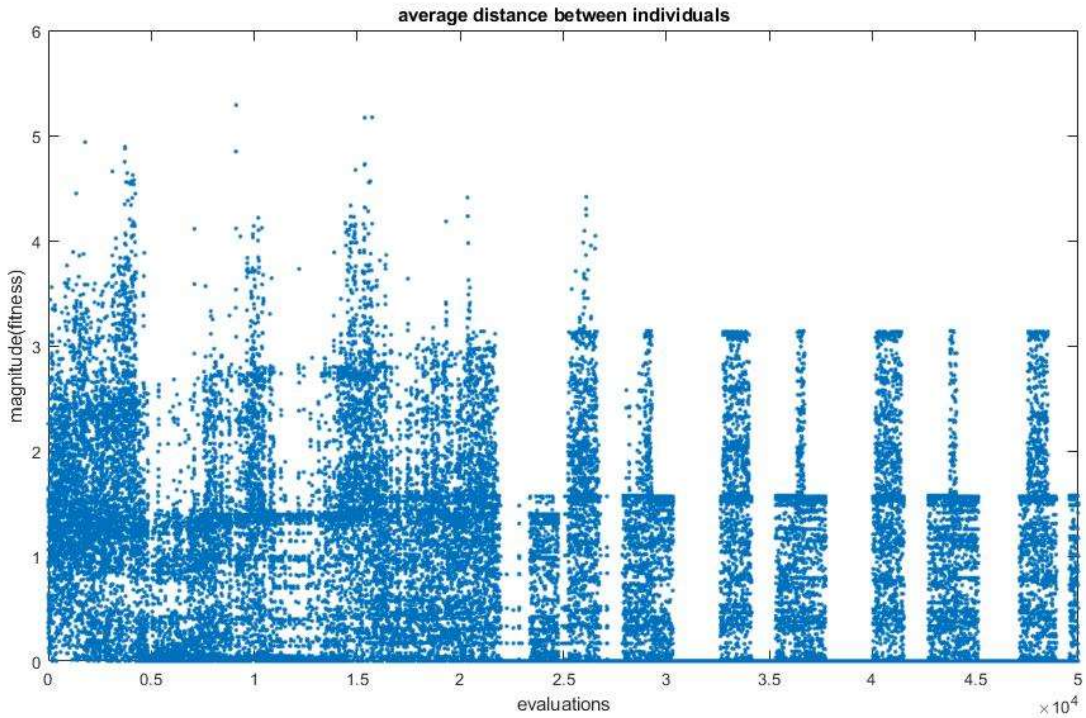


Fig. 5: Above shows the diversity plot, or the average distance between individuals as a function of the evaluation number. Distance was considered to be an absolute value. This plot is averaged over 4 separate trials.

Diagnostic plots (continued)

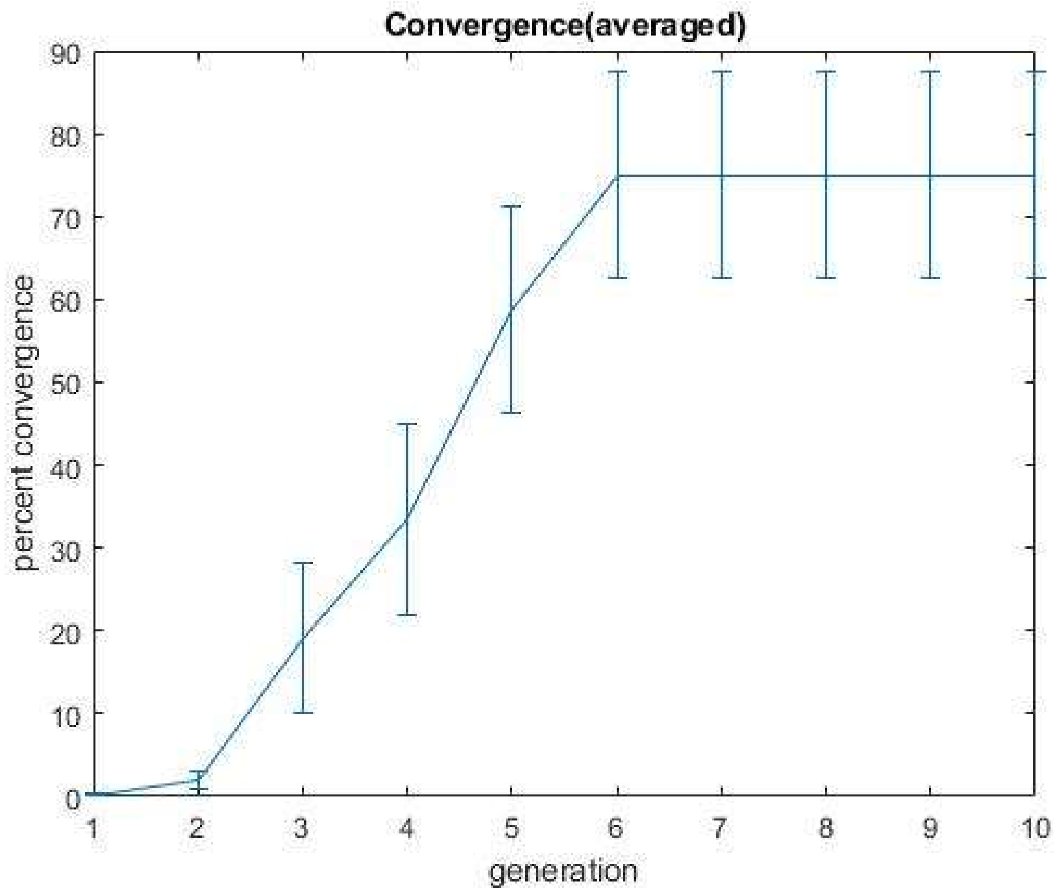
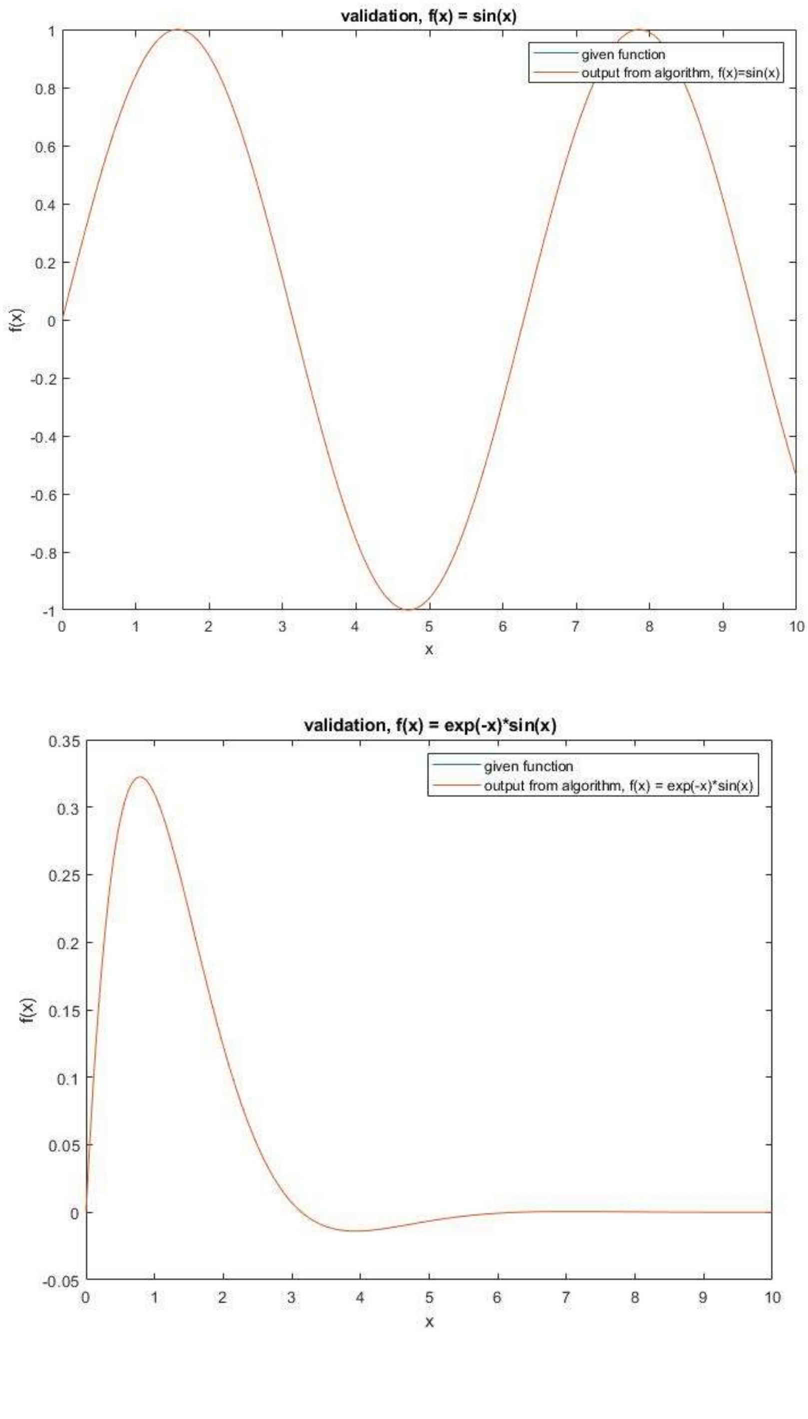


Fig. 6: Figure 6 shows the convergence plots, where the definition of convergence is when the genetic program has less than 18% error. The convergence plot was averaged over 4 trials.

Simpler problems tested



Figs. 7A (top), 7B (bottom): The figures show that the genetic program converges on the correct result for simple equations.

Simpler problems tested (continued)

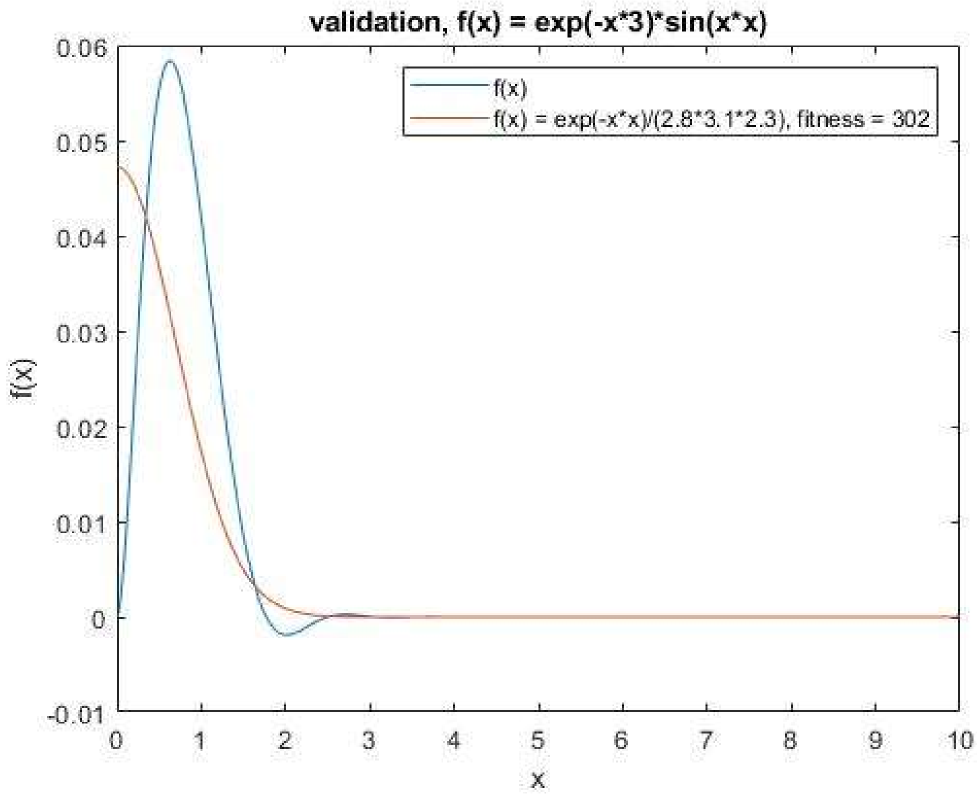


Fig. 8: A slightly more complicated equation reveals that the algorithm converges on a close result, however it is inaccurate with a mean absolute error of 0.003.

Code: genetic program

```
clc;close all;clear all;
checkFunc = csvread('function1.csv');

y = (0:0.01:9.99)';
fx = exp(-y).*sin(y);
checkFunc = [y, fx];
generations = 1;
popsize = 100;
number_of_parents = 20;
evaluations = 1;
tic;
%make 100 lists initially

%fix error when forming equations, some times a sign drops

for i = 1:popsize
   list(1:11,i) = formlist();


end



for j = 1:popsize

%conVal = optimize_constant(list(:,j),checkFunc);
equation = construct(list(:,j));


x = (0:0.01:9.99)';

data = [x, arrayfun(equation,x)];

if isnan(data(1,2)) || isinf(data(1,2)) %some functions are not well behaved around zero, deletes
zeros if this is the case
data = data(2:end,:);
end

[fitness(j) , mean_absolute_error(j)] = checkFitness(data,checkFunc);
evaluations = evaluations+1;
[value, index] = maxk(fitness,number_of_parents);
Best_Parents = list(:,index);
[maxv,maxi] = max(value);
Best_Parent = Best_Parents(:,maxi);


end

while generations <= 10





[children,evaluations] = crossover(Best_Parents,checkFunc,popsize,number_of_parents,evaluations);
[mutate_children,evaluations] = mutate(children,checkFunc,popsize,evaluations);

for k = 1:popsize

equation = construct(mutate_children(:,k));
x = (0:0.01:9.99)';
data = [x, arrayfun(equation,x)];
        if isnan(data(1,2)) || isinf(data(1,2)) %some functions are not well behaved around zero,
deletes zeros if this is the case
            data = data(2:end,:);
        end
```

```
[fitness_Mchildren(k) , mean_absolute_error_Mchildren(k)] = checkFitness(data,checkFunc);
end




[val, ind] = maxk(fitness_Mchildren,number_of_parents);
best_children = mutate_children(:,ind);






[minv,mini] = min(val);
bad_child = fitness_Mchildren(:,mini);
if maxv > minv
    best_children(:,mini) = Best_Parent;
    val(mini) = maxv;
end




Best_Parents = best_children;

%for k = 1:number_of_parents

%equation = construct(Best_Parents(:,k));
%x = (0:0.01:9.99)';
%dataP = [x, arrayfun(equation,x)];
 %       if isnan(dataP(1,2)) || isinf(dataP(1,2)) %some functions are not well behaved around
zero, deletes zeros if this is the case
  %          dataP = dataP(2:end,:);
 %       end
%[fitness_Best_Parents(k) , mean_absolute_error_Best_Parents(k)] = checkFitness(dataP,checkFunc);
%end

[maxv,maxi] = max(val);

Best_Parent = Best_Parents(:,maxi);


max_fitness(generations) = maxv;
generations = generations+1;
end


equation = construct(Best_Parent);
data = [x, arrayfun(equation,x)];
        if isnan(data(1,2)) || isinf(data(1,2)) %some functions are not well behaved around zero,
deletes zeros if this is the case
            data = data(2:end,:);
        end

plot(checkFunc(:,1),checkFunc(:,2))
hold on
plot(data(:,1),data(:,2))
xlabel('x')
ylabel('f(x)')
title('validation, f(x) = sin(x)')
legend('given function','output from algorithm, f(x)=sin(x)')


toc



function list = formlist()
symbolic_table = zeros(2,5);
vars_table = zeros(2,6);
master_list = ["*","/", "+", "-", "sin", "const", "exp"];
```

```
nums_table = cell(1,6);



symbolic_table = master_list(randi([1 7],5,1));
%nums_table = 20*rand(6,1)-10;
x_nums = randi([1 6],1);
choose_switch = randperm(length(nums_table));
choose_switch = choose_switch(1:x_nums);
%following loops ensures list is "well behaved"
checkvals = find(ismember(symbolic_table,{'sin','const','exp'}));
if ~isempty(checkvals) && checkvals(1) == 1 && length(checkvals) < 5
   replace_val = find(ismember(symbolic_table,{'+','-','/','*'}));
   storeconst = symbolic_table(replace_val(1));
   symbolic_table(2) = symbolic_table(1);
   symbolic_table(1) = storeconst;

elseif ~isempty(checkvals) && checkvals(1) == 1 && length(checkvals) == 5
    randreplace = master_list(randi([1 4],1));
    symbolic_table(1) = cellstr(randreplace);
end

for i = 1:length(choose_switch)
chance = rand(1);
    if chance<0.5
        nums_table{choose_switch(i)} = 'x';
    else
        nums_table{choose_switch(i)} = '-x';
    end
end
for j = 1:length(nums_table)
if strcmp(nums_table{j},'x')==false && strcmp(nums_table{j},'-x') == false
    nums_table{j} = num2str(20*rand(1,'double')-10);
end
end

for z = 1:11
    if z<=5
   list{z} = symbolic_table{z};
    else
        list{z} =nums_table{z-5};
    end


end

for s = 1:5
    if list{s} == "const"
        list{s} = string(-10+rand(1)*20);

    end

end

end

function equation = construct(list)
P = 1;
N = 1;
for s = 1:5
   if ~isnan(str2double(list{s}))

        conVal{N} = list{s};
        list{s} = "const";
        N = N+1;
   end
end


for k = 1:5
 if isempty(list{k})
```

```
     continue

 elseif   list{k} == "const"
                list{2*k} = [];
                list{2*k+1} = [];
 elseif list{k} == "exp" || list{k} == "sin"
            list{2*k+1} = [];

 end


 end

empty_list = find(cellfun(@isempty,list(1:5)));

if isempty(empty_list)
   i = 5;
else
    i = empty_list(1)-1;
end



Z = 1;




if i == 2 && list{2} == "const"
   S_eqn{Z} = list{1};
   S_eqn{Z+1} = list{2};
elseif i==2 && (list{2} =="sin" || list{2} == "exp")
   S_eqn{Z} = list{1};
   S_eqn{Z+1} = list{2};
   if list{4} == "const"
        S_eqn{Z+2} = list{4};

   else
       S_eqn{Z+2} = list{4};
       S_eqn{Z+3} = list{8};

   end

end





while i > 2
     if list{i} == "sin" || list{i} == "exp"
         if mod(i,2) == 0
         connecting_sign = list{i/2};
         checkConst = "noCheck";
         else
         connecting_sign = list{(i-1)/2};
         checkConst = list{i-1};
         end

         if strcmp(checkConst,"const")
             try

             connecting2 = list{(i-1)/4};
             S_eqn{Z} = connecting2;
             S_eqn{Z+1} = "const";
             S_eqn{Z+2} = connecting_sign;
             S_eqn{Z+3} = list{i};
             S_eqn{Z+4} = list{2*i};
             Z=Z+5;
             i = i-2;
             catch
```

```
            S_eqn{Z} = "const";
            S_eqn{Z+1} = connecting_sign;
            S_eqn{Z+2} = list{i};
            S_eqn{Z+3} = list{2*i};
            Z=Z+4;
            i = i-2;
            end
        else
        S_eqn{Z} = connecting_sign;
        S_eqn{Z+1} = list{i} ;
        S_eqn{Z+2} = list{2*i};
        Z = Z+3;
        i = i-1;
        end
    elseif list{i} == "const"
        if mod(i,2) == 0
           S_eqn{Z} = list{i/2};
        else
           S_eqn{Z} = list{(i-1)/2};
        end
       S_eqn{Z+1} = "const";
       Z = Z+2;
       i=i-1;
    elseif list{i} == '*' || list{i} == '-' || list{i} == '+' || list{i} == '/'
        if mod(i,2) == 0
            S_eqn{Z} = list{i/2};

        else
            S_eqn{Z} = list{(i-1)/2};
        end
           S_eqn{Z+1} = list{2*i};
           S_eqn{Z+2} = list{i};
           S_eqn{Z+3} = list{2*i+1};
           Z = Z+4;
           i=i-1;
    end


end


check = S_eqn{1};
%deletes algebraic opporations at the beginning of string
if strcmp(check,'*') || strcmp(check,'/') || strcmp(check,'+') || strcmp(check,'-')
   S_eqn(1) = [];
end

%change "const" to random number (arbitrary constant)
for F = 1:length(S_eqn)
   if strcmp(S_eqn{F},"const")
       %S_eqn{F} = string(-10+20*rand(1));
        S_eqn{F} = string(conVal(P));
        P = P+1;
   end
   if strcmp(S_eqn{F}, '*')
        S_eqn{F} = '.*';
   elseif strcmp(S_eqn{F},'/')
       S_eqn{F} = './';
   end
end
equation_0 = string(cellstr(S_eqn));

%"package" equation so it can be converted to function
X = 1;
Z = 1;

while X <= length(equation_0)
    if strcmp(equation_0{X},"sin") || strcmp(equation_0{X}, "exp")
        if (X+2)<=length(equation_0)  && (strcmp(equation_0{X+2},'+') ||
strcmp(equation_0{X+2},'./') || strcmp(equation_0{X+2},'.*') || strcmp(equation_0{X+2},'-'))
```

```
                if strcmp(equation_0{X+3},"sin") || strcmp(equation_0{X+3},"exp")
                    equation_1{Z} = equation_0{X}+"(" +
equation_0{X+1}+")"+equation_0{X+2}+equation_0{X+3}+"("+equation_0{X+4} + ")";
                    Z = Z+1;
                    X = X+5;

                else
                 equation_1{Z} = equation_0{X}+"(" +
equation_0{X+1}+equation_0{X+2}+equation_0{X+3} + ")";
                 Z = Z+1;
                 X = X+4;
                end

        elseif strcmp(equation_0{X+1},"sin") || strcmp(equation_0{X+1},"exp")
            if strcmp(equation_0{X+2},"sin") || strcmp(equation_0{X+2},"exp")
                equation_1{Z} =
equation_0{X}+"("+equation_0{X+1}+"("+equation_0{X+2}+"("+equation_0{X+3}+")"+")"+")";
                Z = Z+1;
                X = X+4;
            else

            equation_1{Z} = equation_0{X}+"("+equation_0{X+1}+"("+equation_0{X+2}+")"+")";
            X = X+3;
            Z = Z+1;
            end
        else


            equation_1{Z} = equation_0{X}+"(" +equation_0{X+1}+")";

            Z = Z+1;
            X = X+2;

        end

    else
          equation_1{Z} = equation_0{X};
          Z = Z+1;
          X = X+1;



    end


end




string_equation = join(string(cellstr(equation_1)));

var = "@(x)";

equation = str2func(var+string_equation);

end



function [mutated,evaluations] = mutate(children,checkFunc,popsize,evaluations)



for i = 1:popsize
    equation_child = construct(children(:,i));
    x = (0:0.01:9.99)';
    data_child = [x, arrayfun(equation_child,x)];

        if isnan(data_child(1,2)) || isinf(data_child(1,2)) %some functions are not well behaved
around zero, deletes zeros if this is the case
```

```
            data_child = data_child(2:end,:);
        end
    fitness_child = checkFitness(data_child,checkFunc);
    stop = 0;
    counter = 1;
    while counter < 15 && stop == 0


        chance1 = rand(1);
        chance2 = rand(1);
        chance3 = rand(1);
        chance4 = rand(1);
        chance5 = rand(1);
        chance6 = rand(1);
        mutated(:,i) = children(:,i);
            if chance1 < 0.2
                rand0 = randperm(4)+1;
             rand1 =rand0(1:2);
            temp1 = children(rand1(1),i);
             temp2 = children(rand1(2),i);
              mutated(rand1(1),i) = temp2;
               mutated(rand1(2),i) = temp1;
            end

             if chance2 <0.2
                 rand0_prime= randperm(6)+5;
                   rand2 = rand0_prime(1:2);
                  temp1 = children(rand2(1),i);
                 temp2 = children(rand2(2),i);
                 mutated(rand2(1),i) = temp2;
                 mutated(rand2(2),i) = temp1;

             end
             if chance3 < 0.1
               rand_num = randi(4);
              char_vector = {"+","*","-","/"};
              mutated(1,i) = char_vector(rand_num);

             end
             if chance4 < 0.2
                 master_list = {"*","/", "+", "-", "sin", "const", "exp"};
                 rand0 = randi([2 5]);
                 randswitch = randi([1 7]);
                 switch_select = master_list(randswitch);
                 mutated(rand0,i) = switch_select;
                 for L = 1:5
                     if mutated{L,i} == "const"
                         mutated{L,i} = string(-10+rand(1)*20);
                     end
                 end
             end
             if chance5<0.2
                 master_list = {"-x","x",string(-10+20*rand(1,'double')),string(-
10+20*rand(1,'double'))};
                 rand0 = randi([6 11]);
                 randswitch = randi([1 4]);
                 switch_select = master_list(randswitch);
                 mutated(rand0,i) =  switch_select;

             end
             if chance6 < 0.2
                 for K = 1:5
                    if ~isnan(str2double(mutated{K,i}))
                        mutated{K,i} = string(-10+20*rand(1));
                    end
                 end
             end

     equation_mutated = construct(mutated(:,i));
    data_mutated = [x, arrayfun(equation_mutated,x)];
```

```
     if isnan(data_mutated(1,2)) || isinf(data_mutated(1,2)) %some functions are not well behaved
around zero, deletes zeros if this is the case
            data_mutated = data_mutated(2:end,:);
     end
      fitness_mutated = checkFitness(data_mutated,checkFunc);
      evaluations = evaluations+1;
      if fitness_mutated > fitness_child
         stop = 1;

      else
          mutated(:,i) = children(:,i);

      end

    counter = counter+1;
    end

end

end




function [children,evaluations] =
crossover(Best_Parents,checkFunc,popsize,number_of_parents,evaluations)
%takes the opperations of first parent, and puts the order of numbers of
%the second parent, popsize times
x = (0:0.01:9.99)';


for i = 1:popsize
    randnums = randperm(number_of_parents);

    selection_vector(i,1) =randnums(1) ;
    selection_vector(i,2) = randnums(2);
end

for j = 1:popsize
 counter = 0;
 stop = 0;
 stopcounter = 15;
while counter < stopcounter && stop == 0

seed1 = randi([2 5]);
seed2 = randi([2 5]);
try
    tree1 = [seed1; 2*seed1;2*seed1+1;4*seed1;4*seed1+1;2*(seed1*2+1);2*(seed1*2+1)+1];
    for U = 1:length(tree1)
    FirstTree{U} = Best_Parents{tree1(U),selection_vector(j,1)};
    end

    catch
    tree1 = [seed1; 2*seed1;2*seed1+1];
    for U = 1:length(tree1)
    FirstTree{U} = Best_Parents{tree1(U),selection_vector(j,1)};
    end

end

try

   tree2 = [seed2; 2*seed2;2*seed2+1;4*seed2;4*seed2+1;2*(seed2*2+1);2*(seed2*2+1)+1];
    for G = 1:length(tree2)
    SecondTree{G} = Best_Parents{tree2(G),selection_vector(j,2)};
    end

catch
    tree2 = [seed2; 2*seed2;2*seed2+1];
    for G = 1:length(tree2)
    SecondTree{G} = Best_Parents{tree2(G),selection_vector(j,2)};
```

```
    end

end

child_1 = Best_Parents(:,selection_vector(j,2));
child_2 = Best_Parents(:,selection_vector(j,1));
Z = 1;
L = 1;
for k = 1:length(child_1)
    if Z <= length(tree1) && k == tree1(Z)
        child_1{k}= FirstTree{Z};
        Z = Z+1;
    end
    if L <= length(tree2) && k == tree2(L)
        child_2{k} = SecondTree{L};
        L = L+1;
    end

end






equation_child_1 = construct(child_1);
equation_child_2 = construct(child_2);
equation_parent1 = construct(Best_Parents(:,selection_vector(j,1)));
equation_parent2 = construct(Best_Parents(:,selection_vector(j,2)));

data_parent1 = [x, arrayfun(equation_parent1,x)];
data_parent2 = [x, arrayfun(equation_parent2,x)];
data_child_1 = [x, arrayfun(equation_child_1,x)];
data_child_2 = [x, arrayfun(equation_child_2,x)];

        if isnan(data_child_1(1,2)) || isinf(data_child_1(1,2)) %some functions are not well
behaved around zero, deletes zeros if this is the case
            data_child_1 = data_child_1(2:end,:);
        end
         if isnan(data_child_2(1,2)) || isinf(data_child_2(1,2)) %some functions are not well
behaved around zero, deletes zeros if this is the case
            data_child_2 = data_child_2(2:end,:);
        end
         if isnan(data_parent1(1,2)) || isinf(data_parent1(1,2)) %some functions are not well
behaved around zero, deletes zeros if this is the case
            data_parent1 = data_parent1(2:end,:);
         end
         if isnan(data_parent2(1,2)) || isinf(data_parent2(1,2)) %some functions are not well
behaved around zero, deletes zeros if this is the case
            data_parent2 = data_parent2(2:end,:);
         end
[fitness_child_1,~] = checkFitness(data_child_1, checkFunc);
[fitness_child_2,~] = checkFitness(data_child_2, checkFunc);

[fitness_parent1, ~] = checkFitness(data_parent1,checkFunc);
[fitness_parent2,~] = checkFitness(data_parent2,checkFunc);


evaluations = evaluations + 4;




if (fitness_child_1 > fitness_parent1 || fitness_child_1 > fitness_parent2) && fitness_child_1 >=
fitness_child_2
    children(:,j) = child_1;
    stop = 1;
elseif (fitness_child_2 > fitness_parent1 || fitness_child_2 > fitness_parent2) &&
fitness_child_2 >= fitness_child_1
    children(:,j) = child_2;
    stop = 1;
```

```
else
    stop = 0;
    counter = counter+1;
end

if counter == stopcounter
    if fitness_parent1 > fitness_parent2
        children(:,j) = Best_Parents(:,selection_vector(j,1));
    else
       children(:,j) = Best_Parents(:,selection_vector(j,2));
    end

end



end
end


end



function [fitness , mean_absolute_error] = checkFitness(data,checkFunc)
%calculate mean absolute error

for i = 1:length(data)
    error_node(i) = abs(checkFunc(i,2)-data(i,2));

end
mean_absolute_error = sum(error_node)/length(data);
fitness = 1/mean_absolute_error;

end
```

Code: hillclimber

```
clc;close all;clear all;
checkFunc = csvread('function1.csv');

%y = (0:0.01:9.99)';
%fx = sin(y);
%checkFunc = [y, fx];
generations = 1;
popsize = 100;
number_of_parents = 20;
number_of_evals = 1;
tic;


list(1:11) = formlist();
equation = construct(list);

x = (0:0.01:9.99)';

data = [x, arrayfun(equation,x)];

        if isnan(data(1,2)) || isinf(data(1,2)) %some functions are not well behaved around zero,
deletes zeros if this is the case
              data = data(2:end,:);
        end

[fitness_previous , ~] = checkFitness(data,checkFunc);


while number_of_evals < 50000


equation = construct(list);
master_list = ["*","/", "+", "-", "sin", "const", "exp"];
for i = 1:11
    prevlist = list;
    if i == 1
       for L = 1:4
          symList = [ "*","/", "+", "-"];
          list{i} = symList{L};
            equation = construct(list);
                x = (0:0.01:9.99)';

                data = [x, arrayfun(equation,x)];

                    if isnan(data(1,2)) || isinf(data(1,2)) %some functions are not well behaved
around zero, deletes zeros if this is the case
                        data = data(2:end,:);
                     end

    [fitness , ~] = checkFitness(data,checkFunc);
    fitness_vector(number_of_evals) = fitness_previous;
    number_of_evals = number_of_evals+1;
            if fitness>fitness_previous
                prevlist = list;
                fitness_previous = fitness;
            else
                list = prevlist;
             end
       end



    end
    if i<=5 && i>1
        for k = 1:7
            list{i} = master_list{k};
```

```
                if  list{i} == "const"
               list{i} = string(-10+20*rand(1));
                end
                equation = construct(list);
                x = (0:0.01:9.99)';

                data = [x, arrayfun(equation,x)];

                    if isnan(data(1,2)) || isinf(data(1,2)) %some functions are not well behaved
around zero, deletes zeros if this is the case
                        data = data(2:end,:);
                     end

    [fitness , ~] = checkFitness(data,checkFunc);
    fitness_vector(number_of_evals) = fitness_previous;
    number_of_evals = number_of_evals+1;
            if fitness>fitness_previous
               prevlist = list;
                fitness_previous = fitness;
            else
                list = prevlist;
             end
        end
    elseif i>5
        chance = rand(1);
        if chance>0.5
        list{i} = string(-10+rand(1)*20);
        elseif chance>0.25 && chance<0.5
            list{i}= "x";
        else
            list{i} = "-x";


        end
         equation = construct(list);
                x = (0:0.01:9.99)';

                data = [x, arrayfun(equation,x)];

                    if isnan(data(1,2)) || isinf(data(1,2)) %some functions are not well behaved
around zero, deletes zeros if this is the case
                        data = data(2:end,:);
                     end

    [fitness , ~] = checkFitness(data,checkFunc);
    fitness_vector(number_of_evals) = fitness_previous;
    number_of_evals = number_of_evals+1;
            if fitness>fitness_previous
                prevlist = list;
                fitness_previous = fitness;
            else
                list = prevlist;

            end


    end
end



end

plot(checkFunc(:,1),checkFunc(:,2))
hold on
plot(data(:,1),data(:,2))
```

```
function list = formlist()
symbolic_table = zeros(2,5);
vars_table = zeros(2,6);
master_list = ["*","/", "+", "-", "sin", "const", "exp"];
nums_table = cell(1,6);



symbolic_table = master_list(randi([1 7],5,1));
%nums_table = 20*rand(6,1)-10;
x_nums = randi([1 6],1);
choose_switch = randperm(length(nums_table));
choose_switch = choose_switch(1:x_nums);
%following loops ensures list is "well behaved"
checkvals = find(ismember(symbolic_table,{'sin','const','exp'}));
if ~isempty(checkvals) && checkvals(1) == 1 && length(checkvals) < 5
   replace_val = find(ismember(symbolic_table,{'+','-','/','*'}));
   storeconst = symbolic_table(replace_val(1));
   symbolic_table(2) = symbolic_table(1);
   symbolic_table(1) = storeconst;

elseif ~isempty(checkvals) && checkvals(1) == 1 && length(checkvals) == 5
    randreplace = master_list(randi([1 4],1));
    symbolic_table(1) = cellstr(randreplace);
end

for i = 1:length(choose_switch)
chance = rand(1);
    if chance<0.5
        nums_table{choose_switch(i)} = 'x';
    else
        nums_table{choose_switch(i)} = '-x';
    end
end
for j = 1:length(nums_table)
if strcmp(nums_table{j},'x')==false && strcmp(nums_table{j},'-x') == false
    nums_table{j} = num2str(20*rand(1,'double')-10);
end
end

for z = 1:11
    if z<=5
   list{z} = symbolic_table{z};
    else
        list{z} =nums_table{z-5};
    end


end

for s = 1:5
    if list{s} == "const"
        list{s} = string(-10+rand(1)*20);

    end

end

end
```

```
function equation = construct(list)
P = 1;
N = 1;
for s = 1:5
   if ~isnan(str2double(list{s}))

        conVal{N} = list{s};
        list{s} = "const";
        N = N+1;
   end
end


for k = 1:5
 if isempty(list{k})
     continue

 elseif   list{k} == "const"
                list{2*k} = [];
                list{2*k+1} = [];
 elseif list{k} == "exp" || list{k} == "sin"
            list{2*k+1} = [];

 end


 end

empty_list = find(cellfun(@isempty,list(1:5)));

if isempty(empty_list)
   i = 5;
else
    i = empty_list(1)-1;
end



Z = 1;




if i == 2 && list{2} == "const"
   S_eqn{Z} = list{1};
   S_eqn{Z+1} = list{2};
elseif i==2 && (list{2} =="sin" || list{2} == "exp")
   S_eqn{Z} = list{1};
   S_eqn{Z+1} = list{2};
   if list{4} == "const"
        S_eqn{Z+2} = list{4};

   else
       S_eqn{Z+2} = list{4};
       S_eqn{Z+3} = list{8};

   end

end






while i > 2
     if list{i} == "sin" || list{i} == "exp"
         if mod(i,2) == 0
         connecting_sign = list{i/2};
         checkConst = "noCheck";
         else
         connecting_sign = list{(i-1)/2};
```

```
         checkConst = list{i-1};
         end

         if strcmp(checkConst,"const")
             try

             connecting2 = list{(i-1)/4};
             S_eqn{Z} = connecting2;
             S_eqn{Z+1} = "const";
             S_eqn{Z+2} = connecting_sign;
             S_eqn{Z+3} = list{i};
             S_eqn{Z+4} = list{2*i};
             Z=Z+5;
             i = i-2;
             catch

             S_eqn{Z} = "const";
             S_eqn{Z+1} = connecting_sign;
             S_eqn{Z+2} = list{i};
             S_eqn{Z+3} = list{2*i};
             Z=Z+4;
             i = i-2;
             end
         else
         S_eqn{Z} = connecting_sign;
         S_eqn{Z+1} = list{i} ;
         S_eqn{Z+2} = list{2*i};
         Z = Z+3;
         i = i-1;
         end
     elseif list{i} == "const"
         if mod(i,2) == 0
            S_eqn{Z} = list{i/2};
         else
            S_eqn{Z} = list{(i-1)/2};
         end
        S_eqn{Z+1} = "const";
        Z = Z+2;
        i=i-1;
     elseif list{i} == '*' || list{i} == '-' || list{i} == '+' || list{i} == '/'
         if mod(i,2) == 0
             S_eqn{Z} = list{i/2};

         else
             S_eqn{Z} = list{(i-1)/2};
         end
            S_eqn{Z+1} = list{2*i};
            S_eqn{Z+2} = list{i};
            S_eqn{Z+3} = list{2*i+1};
            Z = Z+4;
            i=i-1;
     end


end


check = S_eqn{1};
%deletes algebraic opporations at the beginning of string
if strcmp(check,'*') || strcmp(check,'/') || strcmp(check,'+') || strcmp(check,'-')
   S_eqn(1) = [];
end

%change "const" to random number (arbitrary constant)
for F = 1:length(S_eqn)
   if strcmp(S_eqn{F},"const")
       %S_eqn{F} = string(-10+20*rand(1));
        S_eqn{F} = string(conVal(P));
        P = P+1;
   end
   if strcmp(S_eqn{F}, '*')
```

```
        S_eqn{F} = '.*';
    elseif strcmp(S_eqn{F},'/')
        S_eqn{F} = './';
    end
end
equation_0 = string(cellstr(S_eqn));

%"package" equation so it can be converted to function
X = 1;
Z = 1;

while X <= length(equation_0)
    if strcmp(equation_0{X},"sin") || strcmp(equation_0{X}, "exp")
        if (X+2)<=length(equation_0)  && (strcmp(equation_0{X+2},'+') ||
strcmp(equation_0{X+2},'./') || strcmp(equation_0{X+2},'.*') || strcmp(equation_0{X+2},'-'))
                if strcmp(equation_0{X+3},"sin") || strcmp(equation_0{X+3},"exp")
                    equation_1{Z} = equation_0{X}+"(" +
equation_0{X+1}+")"+equation_0{X+2}+equation_0{X+3}+"("+equation_0{X+4} + ")";
                    Z = Z+1;
                    X = X+5;

                else
                 equation_1{Z} = equation_0{X}+"(" +
equation_0{X+1}+equation_0{X+2}+equation_0{X+3} + ")";
                 Z = Z+1;
                 X = X+4;
                end

        elseif strcmp(equation_0{X+1},"sin") || strcmp(equation_0{X+1},"exp")
            if strcmp(equation_0{X+2},"sin") || strcmp(equation_0{X+2},"exp")
                equation_1{Z} =
equation_0{X}+"("+equation_0{X+1}+"("+equation_0{X+2}+"("+equation_0{X+3}+")"+")"+")";
                Z = Z+1;
                X = X+4;
            else

            equation_1{Z} = equation_0{X}+"("+equation_0{X+1}+"("+equation_0{X+2}+")"+")";
            X = X+3;
            Z = Z+1;
            end
        else


            equation_1{Z} = equation_0{X}+"(" +equation_0{X+1}+")";

            Z = Z+1;
            X = X+2;

        end

    else
          equation_1{Z} = equation_0{X};
          Z = Z+1;
          X = X+1;



    end


end




string_equation = join(string(cellstr(equation_1)));

var = "@(x)";

equation = str2func(var+string_equation);
```

```
end



function [fitness , mean_absolute_error] = checkFitness(data,checkFunc)
%calculate mean absolute error

for i = 1:length(data)
    error_node(i) = abs(checkFunc(i,2)-data(i,2));

end
mean_absolute_error = sum(error_node)/length(data);
fitness = 1/mean_absolute_error;

end
```

Code: random search

```
clc;close all;clear all;
checkFunc = csvread('function1.csv');

%y = (0:0.01:9.99)';
%fx = sin(y);
%checkFunc = [y, fx];
generations = 1;
popsize = 100;
number_of_parents = 20;
number_of_evals = 50000;
tic;
%make 100 lists initially

%fix error when forming equations, some times a sign drops


list(1:11) = formlist();
equation = construct(list);

x = (0:0.01:9.99)';

data = [x, arrayfun(equation,x)];

        if isnan(data(1,2)) || isinf(data(1,2)) %some functions are not well behaved around zero,
deletes zeros if this is the case
              data = data(2:end,:);
        end

[fitness_previous , ~] = checkFitness(data,checkFunc);


for j = 1:number_of_evals

list(1:11) = formlist();
equation = construct(list);
x = (0:0.01:9.99)';

data = [x, arrayfun(equation,x)];

        if isnan(data(1,2)) || isinf(data(1,2)) %some functions are not well behaved around zero,
deletes zeros if this is the case
              data = data(2:end,:);
        end

[fitness , ~] = checkFitness(data,checkFunc);

if fitness> fitness_previous

   best_list = list ;
   fitness_previous = fitness;

end
 fitness_vector(j) = fitness_previous;

end


equation = construct(best_list);
x = (0:0.01:9.99)';

data = [x, arrayfun(equation,x)];

        if isnan(data(1,2)) || isinf(data(1,2)) %some functions are not well behaved around zero,
deletes zeros if this is the case
              data = data(2:end,:);
        end
```

```matlab
plot(checkFunc(:,1),checkFunc(:,2))
hold on
plot(data(:,1),data(:,2))





function list = formlist()
symbolic_table = zeros(2,5);
vars_table = zeros(2,6);
master_list = ["*","/", "+", "-", "sin", "const", "exp"];
nums_table = cell(1,6);



symbolic_table = master_list(randi([1 7],5,1));
%nums_table = 20*rand(6,1)-10;
x_nums = randi([1 6],1);
choose_switch = randperm(length(nums_table));
choose_switch = choose_switch(1:x_nums);
%following loops ensures list is "well behaved"
checkvals = find(ismember(symbolic_table,{'sin','const','exp'}));
if ~isempty(checkvals) && checkvals(1) == 1 && length(checkvals) < 5
   replace_val = find(ismember(symbolic_table,{'+','-','/','*'}));
   storeconst = symbolic_table(replace_val(1));
   symbolic_table(2) = symbolic_table(1);
   symbolic_table(1) = storeconst;

elseif ~isempty(checkvals) && checkvals(1) == 1 && length(checkvals) == 5
    randreplace = master_list(randi([1 4],1));
    symbolic_table(1) = cellstr(randreplace);
end

for i = 1:length(choose_switch)
chance = rand(1);
    if chance<0.5
        nums_table{choose_switch(i)} = 'x';
    else
        nums_table{choose_switch(i)} = '-x';
    end
end
for j = 1:length(nums_table)
if strcmp(nums_table{j},'x')==false && strcmp(nums_table{j},'-x') == false
    nums_table{j} = num2str(20*rand(1,'double')-10);
end
end

for z = 1:11
    if z<=5
   list{z} = symbolic_table{z};
    else
        list{z} =nums_table{z-5};
    end


end

for s = 1:5
    if list{s} == "const"
        list{s} = string(-10+rand(1)*20);

    end
```

```
end

end




function equation = construct(list)
P = 1;
N = 1;
for s = 1:5
   if ~isnan(str2double(list{s}))

        conVal{N} = list{s};
        list{s} = "const";
        N = N+1;
   end
end


for k = 1:5
 if isempty(list{k})
     continue

 elseif   list{k} == "const"
                list{2*k} = [];
                list{2*k+1} = [];
 elseif list{k} == "exp" || list{k} == "sin"
            list{2*k+1} = [];

 end


 end

empty_list = find(cellfun(@isempty,list(1:5)));

if isempty(empty_list)
   i = 5;
else
    i = empty_list(1)-1;
end



Z = 1;




if i == 2 && list{2} == "const"
   S_eqn{Z} = list{1};
   S_eqn{Z+1} = list{2};
elseif i==2 && (list{2} =="sin" || list{2} == "exp")
   S_eqn{Z} = list{1};
   S_eqn{Z+1} = list{2};
   if list{4} == "const"
        S_eqn{Z+2} = list{4};

   else
       S_eqn{Z+2} = list{4};
       S_eqn{Z+3} = list{8};

   end

end
```

```
while i > 2
     if list{i} == "sin" || list{i} == "exp"
         if mod(i,2) == 0
         connecting_sign = list{i/2};
         checkConst = "noCheck";
         else
         connecting_sign = list{(i-1)/2};
         checkConst = list{i-1};
         end

         if strcmp(checkConst,"const")
             try

             connecting2 = list{(i-1)/4};
             S_eqn{Z} = connecting2;
             S_eqn{Z+1} = "const";
             S_eqn{Z+2} = connecting_sign;
             S_eqn{Z+3} = list{i};
             S_eqn{Z+4} = list{2*i};
             Z=Z+5;
             i = i-2;
             catch

             S_eqn{Z} = "const";
             S_eqn{Z+1} = connecting_sign;
             S_eqn{Z+2} = list{i};
             S_eqn{Z+3} = list{2*i};
             Z=Z+4;
             i = i-2;
             end
         else
         S_eqn{Z} = connecting_sign;
         S_eqn{Z+1} = list{i} ;
         S_eqn{Z+2} = list{2*i};
         Z = Z+3;
         i = i-1;
         end
     elseif list{i} == "const"
         if mod(i,2) == 0
            S_eqn{Z} = list{i/2};
         else
            S_eqn{Z} = list{(i-1)/2};
         end
        S_eqn{Z+1} = "const";
        Z = Z+2;
        i=i-1;
     elseif list{i} == '*' || list{i} == '-' || list{i} == '+' || list{i} == '/'
         if mod(i,2) == 0
             S_eqn{Z} = list{i/2};

         else
             S_eqn{Z} = list{(i-1)/2};
         end
            S_eqn{Z+1} = list{2*i};
            S_eqn{Z+2} = list{i};
            S_eqn{Z+3} = list{2*i+1};
            Z = Z+4;
            i=i-1;
     end


end


check = S_eqn{1};
%deletes algebraic opporations at the beginning of string
if strcmp(check,'*') || strcmp(check,'/') || strcmp(check,'+') || strcmp(check,'-')
   S_eqn(1) = [];
end

%change "const" to random number (arbitrary constant)
```

```
for F = 1:length(S_eqn)
   if strcmp(S_eqn{F},"const")
       %S_eqn{F} = string(-10+20*rand(1));
        S_eqn{F} = string(conVal(P));
        P = P+1;
   end
   if strcmp(S_eqn{F}, '*')
        S_eqn{F} = '.*';
   elseif strcmp(S_eqn{F},'/')
       S_eqn{F} = './';
   end
end
equation_0 = string(cellstr(S_eqn));

%"package" equation so it can be converted to function
X = 1;
Z = 1;

while X <= length(equation_0)
    if strcmp(equation_0{X},"sin") || strcmp(equation_0{X}, "exp")
        if (X+2)<=length(equation_0)  && (strcmp(equation_0{X+2},'+') ||
strcmp(equation_0{X+2},'./') || strcmp(equation_0{X+2},'.*') || strcmp(equation_0{X+2},'-'))
                if strcmp(equation_0{X+3},"sin") || strcmp(equation_0{X+3},"exp")
                    equation_1{Z} = equation_0{X}+"(" +
equation_0{X+1}+")"+equation_0{X+2}+equation_0{X+3}+"("+equation_0{X+4} + ")";
                    Z = Z+1;
                    X = X+5;

                else
                 equation_1{Z} = equation_0{X}+"(" +
equation_0{X+1}+equation_0{X+2}+equation_0{X+3} + ")";
                 Z = Z+1;
                 X = X+4;
                end

        elseif strcmp(equation_0{X+1},"sin") || strcmp(equation_0{X+1},"exp")
            if strcmp(equation_0{X+2},"sin") || strcmp(equation_0{X+2},"exp")
                equation_1{Z} =
equation_0{X}+"("+equation_0{X+1}+"("+equation_0{X+2}+"("+equation_0{X+3}+")"+")"+")";
                Z = Z+1;
                X = X+4;
            else

            equation_1{Z} = equation_0{X}+"("+equation_0{X+1}+"("+equation_0{X+2}+")"+")";
            X = X+3;
            Z = Z+1;
            end
        else


            equation_1{Z} = equation_0{X}+"(" +equation_0{X+1}+")";

            Z = Z+1;
            X = X+2;

        end

    else
          equation_1{Z} = equation_0{X};
          Z = Z+1;
          X = X+1;



    end


end
```

```
string_equation = join(string(cellstr(equation_1)));

var = "@(x)";

equation = str2func(var+string_equation);

end



function [fitness , mean_absolute_error] = checkFitness(data,checkFunc)
%calculate mean absolute error

for i = 1:length(data)
    error_node(i) = abs(checkFunc(i,2)-data(i,2));

end
mean_absolute_error = sum(error_node)/length(data);
fitness = 1/mean_absolute_error;

end
```