Supplementary Materials

In the Supplementary Materials, the core Matlab program code of the improved genetic algorithm used in this paper is mainly displayed, which mainly includes the main program (main.m) and four sub-functions (Rank.m, Select.m, Cross.m, Mutation.m).

1. main.m

```
clear;clc
Nind = 100;                                                    % Population size
Lind = 5;                                                      % Chromosome number
maxGen = 200;                                                  % Iterations
Xmin = [0.003,0.006,0.002,0.002,0.0006];                       % Upper limit of variable
Xmax = [0.01,0.02,0.007,0.006,0.0014];                         % Lower limit of variable
individuals = struct('fitness', zeros(Nind, 1), 'chrom', []);  % Population structure
for i = 1:Nind
    individuals.chrom(i,:) = rand(1, Lind).*(Xmax - Xmin)+Xmin;  % Randomly generated chromosome
    temp = individuals.chrom(i,:);
    individuals.fitness(i) = fun(temp);                        % Calculate individual fitness
end
[best_fit,best_index] = min(individuals.fitness);
best_chrom = individuals.chrom(best_index,:);                  % Look for the current optimal individual
trace = [];                                                    % Optimal fitness of past dynasties
% Start iteration
for gen = 1:maxGen
    % Rank
    individuals = Rank(individuals, Nind);
    % Select
    individuals = Select(individuals,Nind);
    % Cross
    individuals = Cross(individuals,Nind,Lind);
    % Mutation
    individuals = Mutation(individuals,Nind,Lind,Xmax,Xmin);
    % nonlinear programming
    if do_NLP(gen) == 1                                        % Judge whether to carry out fmincon
        A = []; b = []; Aeq = []; Beq = [];
        for i = 1:Nind
            x0 = individuals.chrom(i,:);
            individuals.chrom(i,:) = fmincon(@obj_fun, x0, A, b, Aeq, Beq, Xmin, Xmax,@yuesu);
        end
    end
    % Calculate individual fitness
    for i = 1:Nind
        temp = individuals.chrom(i,:);
        individuals.fitness(i) = fun(temp);                    % Calculate individual fitness
    end
    % Find the location of the optimal value
```

```
    [newbest_fit,newbest_index] = min(individuals.fitness);
    % Compared with the optimal value obtained in the last round of evolution
    if best_fit > newbest_fit
        best_fit = newbest_fit;
        best_index = newbest_index;
        best_chrom = individuals.chrom(best_index,:);
        iterations=gen;
    end
    trace = [trace;best_fit];                                   % Each generation of data records
end
disp(['The number of iterations when reaching the optimum is:',num2str(iterations)])
disp(['The optimal fitness is:',num2str(trace(end))])
disp(['The corresponding population is:',num2str(best_chrom)])
plot(trace, '-','LineWidth',1.5)
title('Trend of optimal point change','Fontname',' Times New Roman ','FontSize',15);
xlabel('Iterations','Fontname','Times New Roman','FontSize',15);
ylabel('Best objective','Fontname',' Times New Roman ','FontSize',15);
legend('optimum point');
```

2. Rank.m

```
function new_individuals = Rank(individuals, Nind)
for i=1:Nind
    max_index = i;
    for j = i+1:Nind
        if individuals.fitness(j) > individuals.fitness(max_index)
            max_index = j;
        end
    end
    if max_index ~= i % When I is not equal to true, perform the following operations.
        % Swap the values of individuals.fitness(i) and individuals.fitness(max_index)
        temp_fitness = individuals.fitness(i);
        individuals.fitness(i) = individuals.fitness(max_index);
        individuals.fitness(max_index) = temp_fitness;
        % At this time, the fitness of individuals.fitness(i) is the largest in [i,population_size]
        % Swap the chromosome strings of population(i) and population(min_index)
        temp_chrom(i,:) = individuals.chrom(i,:);
        individuals.chrom(i,:) = individuals.chrom(max_index,:);
        individuals.chrom(max_index,:) = temp_chrom(i,:);
    end
end
new_individuals = individuals;
end
```

3. Select.m

```
function new_individuals = Select(individuals, Nind)
% individuals                    Population
% Nind                           Population size
% new_chrom                      New population generated by selection operator
temp =   1./(individuals.fitness);
```

```
sumfitness = sum (temp);                % Find the sum of fitness
sumf = temp./sumfitness;                % Find the proportion of fitness of each function value
avgf = sumfitness/Nind;                 % Find the average value of fitness
index = [];                             % Create a sequence
for i = 1:Nind                          % Turn Nind roulette
    pick = rand;
    while pick == 0
        pick = rand;
    end
    for j = 1:Nind
        pick = pick - sumf(j);
        if pick < 0
            index = [index j];          %The pick and sumf are compared
            break;
        end
    end
end
temp_chrom = individuals.chrom(index,:);
temp_fitness = individuals.fitness(index);
for i=1:Nind-0.05*Nind
    individuals.chrom(i,:) = temp_chrom(i,:);
    individuals.fitness(i) = temp_fitness(i);
end
new_individuals = individuals;
end
```

4. Cross.m

```
function new_individuals = Cross(individuals, Nind, Lind)
% individuals                           Population
% Pc                                    Cross probability
% Nind                                  Population size
% Lind                                  Chromosome number
% new_individuals                       A new population produced after crossover operation
% new_individuals = individuals;
temp = 1./(individuals.fitness);
sumfitness = sum(temp);                 % Find the sum of fitness
avgf = sumfitness/Nind;                 % Find the average value of fitness
maxf = max(temp);
% Adaptive adjustment of cross probability
for i = 1:Nind
    if temp(i)<=avgf
        Pc=0.9;
    else
        Pc=((exp(avgf)-exp(maxf))*0.9*0.4)/((exp(avgf)-exp(temp(i)))*0.9+(exp(temp(i))-exp(maxf))*0.4);
    end
    % Randomly select two individuals
    index = randi(Nind, [2,1]);
    temp1 = zeros(2, Lind);
    % Cross operation
```

```
    for j = 1:Lind
        Pick = rand;
        if Pick < Pc
            u = rand;
            % Cross
            temp1(1,j) = (1-u)*individuals.chrom(index(1),j) + u*individuals.chrom(index(2),j);
            temp1(2,j) = (1-u)*individuals.chrom(index(2),j) + u*individuals.chrom(index(1),j);
        else
            temp1(1,j) = individuals.chrom(index(1),j);
            temp1(2,j) = individuals.chrom(index(2),j);
        end
    end
    individuals.chrom(index(1),:) = temp1(1,:);
    individuals.chrom(index(2),:) = temp1(2,:);
%       new_individuals.chrom(index(1),:) = temp1(1,:);
%       new_individuals.chrom(index(2),:) = temp1(2,:);
end
new_individuals = individuals;
end
```

5. Mutation.m

```
function new_individuals = Mutation(individuals, Nind, Lind, Xmax, Xmin)
% Mutation                  Randomly select a point to mutate.
% individuals               Population
% Pm                        Mutation probability
% Nind                      Population size
% Lind                      Chromosome number
% Xmax                      Upper bound of independent variable
% Xmin                      Lower bound of independent variable
% new_individuals           A new population produced after crossover operation
% new_individuals = individuals;
temp = 1./(individuals.fitness);
sumfitness = sum(temp);     % Find the sum of fitness
avgf = sumfitness/Nind;     % Find the average value of fitness
maxf = max(temp);
% Adaptive adjustment of mutation probability
for i = 1:Nind
    if temp(i)<=avgf
        Pm=0.1;
    else
        Pm=((exp(avgf)-exp(maxf))*0.1*0.01)/((exp(avgf)-exp(temp(i)))*0.1+(exp(temp(i))-exp(maxf))*0.01);
    end
    % Perform mutation operation
    for j = 1:Lind
        Pick = rand;
        if Pick < Pm
            u = (Xmax(j)+Xmin(j))/2;
            sigma = (Xmax(j)-Xmin(j))/6;
            %           new_individuals.chrom(i,j) = u+sigma*(sum(rand([1,12]))-6);
```

```
                individuals.chrom(i,j) = u+sigma*(sum(rand([1,12]))-6);
            end
        end
end
new_individuals = individuals;
end
```