VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY
INTERNATIONAL UNIVERSITY
SCHOOL OF INDUSTRIAL ENGINEERING & MANAGEMENT

GEYSER INSPIRED ALGORITHM: A NEW GEOLOGICAL-INSPIRED META-HEURISTIC FOR REAL-PARAMETER AND CONSTRAINED ENGINEERING OPTIMIZATION

NUMERICAL METHOD
Lecturer: Dao Truong Son
Group Number:
Class: G01_WED_456

Student ID	Member name	% Contribution
IELSIU21163	Nguyễn Minh Đức	100%
IELSIU21287	Dương Bình Dương	100%
IELSIU21084	Nguyễn Thị Bích Ngọc	100%
IELSIU21316	Huỳnh Như	100%
IELSIU22387	Nguyễn Văn Phong	100%

02/2024

TABLE OF CONTENTS

CHAPTER I: INTRODUCTION

As time goes by, many fast and accurate meta-heuristic algorithms have been developed in order to optimize real-life problems that occur every single day. This study presents a new optimization method based on an unusual geological phenomenon in nature, named Geyser inspired Algorithm (GEA). The mathematical modeling of this geological phenomenon is carried out to have a better understanding of the optimization process. The efficiency and accuracy of GEA are verified using statistical examination and convergence rate comparison on numerous CEC 2005, CEC 2014, CEC 2017, and real-parameter benchmark functions. To validate the GEA method, our group has decided to make a comparison between GEA and GWO by conducting each algorithm and evaluating them based on 13 Benchmark functions, each function would be conducted 20 times (20 for GEA and 20 for GWO). The precise steps for the test and observing results will be analyzed as below.

CHAPTER II: METHODOLOGY

1. Geyser Algorithm (GEA)

Optimization is an act, process, or methodology of making a design, system, or decision as fully perfect, functional, or effective as possible, especially the mathematical procedures such as finding the maximum of a function f(X). To describe how the proposed GEA was modeled as an optimization method, below are some detailed descriptions. In GEA, the initial sample is created at random. The basic idea for the GEA is to discover the best solution in each iteration.

In the first step, searching for channels is required.

The members (particles) which contain more optimal fitness functions as the channels will be defined. Through Roulette Wheel selection, the ith particle (Xi) chooses an appropriate channel and moves towards it with the help of its neighbor. The selection of channels is conducted twice in order to achieve optimal algorithm results and prevent becoming trapped in local optima. During the first selection, members with higher objective function values will be prior. In the second selection, grants greater chances of being chosen as channels to those members who have demonstrated better performance.

Next, Roulette Wheel selection is taken into consideration.

This is a stochastic approach in which the probability of potential selections is proportional to their fitness value. The wheel is split into multiple components, each of them is dedicated to a certain probability.

2. Approach Methods

To test whether the Geyser Algorithm is similar to an optimization algorithm, our group has conducted the test. As mentioned above, for each function, 20 runs were performed for GEA, 20 times for GWO. Each run performed 5000 intergrations, the result of each integration was the value of the CostFunction function corresponding to the result of the selected algorithm.

The results of running the two algorithms and their plots can be found in the attached files.

CHAPTER III: RESULT ANALYSIS

To compare the two algorithms, we use the results of the first run for each function of both for comparison:

- For the first function, there is no significant difference in results, with GEA producing 0.08829 and GWO producing 0
- For the second function, there is no significant difference in results, with GEA producing 0.11910 and GWO producing 0
- For the third function, there is a significant difference in results between the two algorithms, with GEA producing 95.85100 and GWO producing 0
- For the fourth function, there is no significant difference in results, with GEA producing 0.81292 and GWO producing 0
- For the fifth function, there is a significant difference in results between the two algorithms, with GEA producing 245.83510 and GWO producing 17.16700
- For the sixth function, there is no significant difference in results, with GEA producing 0.21028 and GWO producing 0
- For the seventh function, there is no significant difference in results, with GEA producing 0.01234 and GWO producing 0.00010

- For the eighth function, there is a significant difference in results between the two algorithms, with GEA producing -5080.08490 and GWO producing -7013.42160
- For the ninth function, there is a significant difference in results between the two algorithms, with GEA producing 106.59370 and GWO producing 0
- For the tenth function, there is no significant difference in results, with GEA producing 0.41203 and GWO producing 0
- For the eleventh function, there is no significant difference in results, with GEA producing 0.79595 and GWO producing 0
- For the twelfth function, there is no significant difference in results, with GEA producing 7.90150 and GWO producing 0.00980
- For the twelfth function, there is no significant difference in results, with GEA producing 0.57580 and GWO producing 0.

CHAPTER IV: IMPROVEMENTS

1. Dynamic Nc (Number of channel base on the diversity of population)

a. Adjustment

Firstly, added a method to calculate the diversity of the population by computing the average distance between the positions of geysers. Then, adjusted Nc based on the measured diversity. Used an exponential function to dynamically set Nc, increasing it when the population is clustered (low diversity) and decreasing it when the population is spread out (high diversity).The last step is Incorporating the diversity calculation and dynamic Nc adjustment into the main optimization loop. Added a conditional to adjust Nc in each iteration based on the current diversity of the population.

b. Objective

To dynamically adjust the number of channels (Nc), it is necessary to understand how diverse the population is. Diversity helps in determining whether the algorithm is exploring new areas or exploiting known good areas. A dynamic Nc allows the algorithm to adapt to the search landscape. When diversity is low (population is clustered), increasing Nc helps in sharing more information and potentially escaping local optima.

When diversity is high (population is diverse), decreasing Nc focuses the search, allowing more thorough exploitation of promising areas. Continuous adjustment of Nc based on real-time diversity ensures that the algorithm remains adaptive throughout the optimization process.

c. Expected Impact

With a diversity measure, the algorithm can adapt Nc to balance between exploration and exploitation, potentially improving the search efficiency and avoiding premature convergence. Dynamic Adjustment of Channel Size (Nc) should help the algorithm maintain a balance between exploration and exploitation, leading to a more robust search process and potentially better optimization results. Last, integration ensures that the algorithm can continuously adapt to the changing landscape of the optimization problem, potentially leading to better convergence behavior and improved final results.

d. Solving real problems

Without going through the GEA algorithm, from analyzing the topic of the current paper, we have come up with some improvements for a real problem below:

Problem: Finding the optimal route to transport goods from the warehouse to many different delivery points.

Model:

2. Decision variable

- nVar = 30: 30 variables representing stops on the transport route. Each variable is a geographic coordinate (e.g., longitude, latitude) of a stop.
- VarMin = -100, VarMax = 100: Range of values for each decision variable, corresponding to the latitude and longitude range of the shipping area.

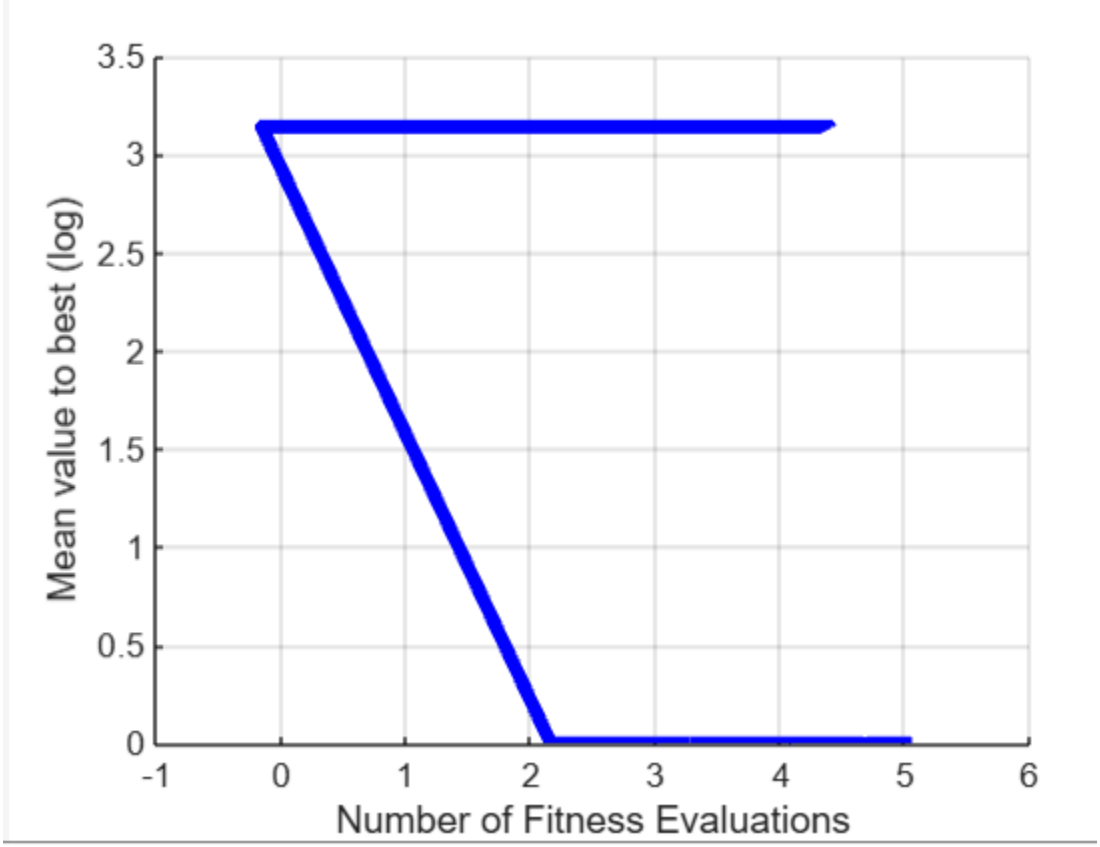
Objective function

- CostFunction = @(x) sum(log(epsilon + x.^2), 2): This function represents the total transportation time (or transportation cost) on the route, calculated based on the distance between stops and travel times.
- epsilon = 1e-6: A small value to avoid logarithmic errors when the distance between stops is very small.
- Geyser algorithm: Optimization algorithm based on the operating model of geyser (hot water volcanic eruption) with main concepts:
- Geysers (nPop = 60): Potential transportation routes, each route is a series of stops.
- Channels: Optimal search streams, each representing a movement method or pathfinding strategy.
- Tournament Selection: Select efficient channels based on their performance, i.e. ability to find the shortest route or lowest cost.
- Roulette Wheel Selection: Select geysers based on their performance, i.e. transit time or cost of the route.

- Eq(3), Eq(5), Eq(6), Eq(8): Mathematical equations that describe how to move and improve geysers and channels, based on information about distance, time and expense.
- FEs = 5000: Maximum number of calculations to find the optimal route.

Results: The Geyser algorithm will search for an optimal freight route, helping to minimize total shipping time (or shipping cost) and ensure delivery efficiency.

Here are the results after we adjusted the transportation problem.



It seem that the result is not make sense a lot in practical transportation model, we suggest it could be improved in the future by these points :

- Define a Real Cost Function: Use a cost function that directly reflects the costs associated with transporting goods
- Tune Algorithm Parameters: Experiment with different values for nPop, Nc, FEs, and alpha to see how they affect the performance and solution quality.
- Consider Alternative Algorithms: The Geyser Algorithm might not be the most suitable for your specific problem. Consider exploring other optimization algorithms (like Genetic Algorithms, Simulated Annealing, Ant Colony Optimization) that might be better suited to finding global optima.
- Visualization: Plot the routes generated by the algorithm. This visual feedback can help you understand whether the routes make practical sense.

CHAPTER V: CONCLUSION

The practical evaluation of the two algorithms was conducted using 12 functions, with 20 runs for each function, and 5000 iterations per run. Based on the obtained results, it can be concluded that there is a significant similarity between the two algorithms, with the results not differing substantially for most benchmark functions, except for functions such as the third, fifth, eighth, and ninth. Possible reasons for different results between runs of the same algorithm on the same function could stem from different random initializations, and the ability to find local minima where some algorithms excel in local search, while others may struggle to avoid getting stuck.By translating and explaining through algorithm comparisons, we can identify combined methods or improvements to enhance the efficiency of the optimization process.

REFERENCES

Ghasemi, M., Zare, M., Zahedi, A., Akbari, M.-A., Mirjalili, S., & Abualigah, L. (2023a). Geyser inspired algorithm: A new geological-inspired meta-heuristic for real-parameter and constrained engineering optimization. *Journal of Bionic Engineering*, *21*(1), 374–408. https://doi.org/10.1007/s42235-023-00437-8

APPENDIX

1. MATLAB code for function 1 using GEA.

a. Main

```
clc;
close all
clear all;
tic;
for NF=1                    % NF: function number
    %% Problem Definition
    VarMin=-50;             % Decision Variables Lower Bound
    VarMax= -VarMin;        % Decision Variables Upper Bound
    nVar=30;                % Number of Decision Variables
    VarSize=[1 nVar];       % Decision Variables Matrix Size
    Nc=40;                  %Number of Channles
    FEs=5000;               % Maximum Number of Function Evoulations
    nPop=40;                % Number of Geyser (Swarm Size)
    %% Initialization
    Geyser.Position=[];     % Empty Geyser Structure
    Geyser.Cost=[];
    pop=repmat(Geyser,nPop,1);   % Initialize Population Array
    BestSol.Cost=inf;       % Initialize Best Solution Ever Found
    % Create Initial population of Geysers
    for i=1:nPop
       pop(i).Position=unifrnd(VarMin,VarMax,VarSize);
       pop(i).Cost=CostFunction(pop(i).Position);
       if pop(i).Cost<=BestSol.Cost
          BestSol=pop(i);
       end
       BestCost(i)=BestSol.Cost;
    end
```

```
   it=0;
   while it<=FEs
     for i=1:nPop
       %%%%%%%%%%%%%%%  Implementing Eq(3)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
       for ii=1:nPop
         D1(ii)=sum(pop(i).Position.*pop(ii).Position)/
(((sum(pop(i).Position.^2)*sum(pop(ii).Position.^2)))^0.5); % Eq(3)
         if ii==i
           D1(ii)=inf;
         end
       end
       S1=min(D1);
       [~,j1]=find(S1==D1);

       %%%% Calculating the pressure value in Eq(6)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

       [CS, Sortorder]=sort([pop.Cost]);
       dif=(CS(end)-CS(1));          % fmax-fmin  Eq(6)
       G=((pop(i).Cost-CS(1))/dif);   % f(i)-fmin  Eq(6)
       if dif==0;
         G=0;
       end
       if it==1
         it=2;
       end
       P_i(i)=(((G^(2/it))-(G^((it+1)/it)))^0.5)*((it/(it-1))^0.5);  % Pi Eq(6)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
       %%%%%%%%%%%%%%%  Search for channels using Roulette wheel selection   %%%%%
```

```
            ImpFitness=0;
            for ii=1:Nc
               ImpFitness=ImpFitness+pop(ii).Cost;
            end
            p=[];
            if ImpFitness==0
               ImpFitness=1e-320;
            end
            for ii=1:Nc
               p(ii)=pop(ii).Cost/ImpFitness;    % Eq(1)
            end
            if sum(p)==0
               i1=1;
            else
               i1=RouletteWheelSelection((p));
            end
            flag=0;
            %%%%%%%%%%%%%%%%  Implementing Eq(5)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

newsol.Position=pop(j1(1)).Position+(rand(VarSize)).*(pop(i1).Position-pop(j1(1)).Position)+(rand(VarSi
ze)).*(pop(i1).Position-pop(i).Position);
            newsol.Position=max(newsol.Position,VarMin);
            newsol.Position=min(newsol.Position,VarMax);
            newsol.Cost=CostFunction(newsol.Position);

            if newsol.Cost<=pop(i).Cost
               pop(i)=newsol;
               flag=0;
            end
            if newsol.Cost<=BestSol.Cost
               BestSol=newsol;
            end
```

```
            it = it + 1;  % Tăng it lên 1 sau mỗi lần cập nhật pop và BestSol
            BestCost(it)=BestSol.Cost;
            disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCost(it))]);
            if   flag==0
              i2=RouletteWheelSelection((1-p));  % Implementing Roulette wheel selection by Eq(7)
            %%%%%%%%%%%%%%%  Implementing Eq
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
              newsol.Position=pop(i2).Position+rand*(P_i(i)-rand)*unifrnd(1*VarMin,1*VarMax,VarSize);
              newsol.Position=max(newsol.Position,VarMin);
              newsol.Position=min(newsol.Position,VarMax);
              newsol.Cost=CostFunction(newsol.Position);

              if newsol.Cost<=pop(i).Cost
                 pop(i)=newsol;
              end
              if newsol.Cost<=BestSol.Cost
                 BestSol=newsol;
              end
              BestCost(it)=BestSol.Cost;
            end
          end
          [~, Sortorder]=sort([pop.Cost]);
          pop=pop(Sortorder);
        % disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCost(it))]);
      end
elapsedTime = toc;
fprintf('Thời gian chạy toàn bộ chương trình: %.4f giây\n', elapsedTime);
      % hold on
      % plot(log(BestCost),'b','LineWidth',2);
      Cost_Rsult(1)=BestSol.Cost;
      Rsult(1:FEs)=BestCost(:,1:FEs);
```

```
  Mean(NF)=mean(Cost_Rsult)
  Std(NF)=std(Cost_Rsult)
  hold on
  [r_res,~]=size(Rsult);
  if r_res>1
     plot(log(mean(Rsult)),'b','LineWidth',4); hold on
  else
     plot(log((Rsult)),'b','LineWidth',4); hold on
  end
  xlabel('Number of Fitness valuations')
  ylabel('Mean value to best (log)')
  grid on
end
```

b. CostFunc

```
function z =CostFunction(x)
dim=size(x,2);
z=.1*((sin(3*pi*x(1)))^2+sum((x(1:dim-1)-1).^2.*(1+(sin(3.*pi.*x(2:dim))).^2))+...
((x(dim)-1)^2)*(1+(sin(2*pi*x(dim)))^2))+sum(Ufun(x,5,100,4));
end
function o=Ufun(x,a,k,m)
o=k.*((x-a).^m).*(x>a)+k.*((-x-a).^m).*(x<(-a));
end
```

c. RouletteWheelSelection

```
function i=RouletteWheelSelection(p)
  c=cumsum(p);
  r=rand();
  i=find(r<=c,1,'first');
end
```

2. MATLAB code for function 1 using GWO.

a. main

```
clear all

clc

tic;

SearchAgents_no=40; % Number of search agents

Function_name='F10'; % Name of the test function that can be from F1 to F23 (Table 1,2,3 in the paper)

Max_iteration=5000; % Maximum numbef of iterations

% Load details of the selected benchmark function

[lb,ub,dim,fobj]=Get_Functions_details(Function_name);

[Best_score,Best_pos,GWO_cg_curve]=GWO(SearchAgents_no,Max_iteration,lb,ub,dim,fobj);

% In ra giá trị objective function cho mỗi iteration

for it = 1:Max_iteration

    fprintf('Iteration %d: Best Cost = %.4f\n', it, GWO_cg_curve(it));

end

figure('Position',[500 500 660 290])

elapsedTime = toc; % Kết thúc đo thời gian

fprintf('Thời gian chạy toàn bộ chương trình: %.4f giây\n', elapsedTime);

%Draw search space

subplot(1,2,1);

func_plot(Function_name);
```

```
title('Parameter space')

xlabel('x_1');

ylabel('x_2');

zlabel([Function_name,'( x_1 , x_2 )'])

%Draw objective space

subplot(1,2,2);

semilogy(GWO_cg_curve,'Color','r')

title('Objective space')

xlabel('Iteration');

ylabel('Best score obtained so far');

axis tight

grid on

box on

legend('GWO')

display(['The best solution obtained by GWO is : ', num2str(Best_pos)]);

display(['The best optimal value of the objective funciton found by GWO is : ', num2str(Best_score)]);
```

b. Get_functions_detail

```
% This function containts full information and implementations of the benchmark

% functions in Table 1, Table 2, and Table 3 in the paper


% lb is the lower bound: lb=[lb_1,lb_2,...,lb_d]
```

```
% up is the uppper bound: ub=[ub_1,ub_2,...,ub_d]
% dim is the number of variables (dimension of the problem)

function [lb,ub,dim,fobj] = Get_Functions_details(F)
switch F
    case 'F1'
        fobj = @F1;
        lb=-100;
        ub=100;
        dim=20;

    case 'F2'
        fobj = @F2;
        lb=-10;
        ub=10;
        dim=20;

    case 'F3'
        fobj = @F3;
        lb=-100;
        ub=100;
        dim=20;

    case 'F4'
        fobj = @F4;
        lb=-100;
```

```
    ub=100;
    dim=20;

case 'F5'
    fobj = @F5;
    lb=-30;
    ub=30;
    dim=20;

case 'F6'
    fobj = @F6;
    lb=-100;
    ub=100;
    dim=20;

case 'F7'
    fobj = @F7;
    lb=-1.28;
    ub=1.28;
    dim=20;

case 'F8'
    fobj = @F8;
    lb=-500;
    ub=500;
    dim=20;
```

```
case 'F9'
    fobj = @F9;
    lb=-5.12;
    ub=5.12;
    dim=20;

case 'F10'
    fobj = @F10;
    lb=-32;
    ub=32;
    dim=20;

case 'F11'
    fobj = @F11;
    lb=-600;
    ub=600;
    dim=20;

case 'F12'
    fobj = @F12;
    lb=-50;
    ub=50;
    dim=20;

case 'F13'
```

```
    fobj = @F13;
    lb=-50;
    ub=50;
    dim=2;

case 'F14'
    fobj = @F14;
    lb=-65.536;
    ub=65.536;
    dim=20;

case 'F15'
    fobj = @F15;
    lb=-5;
    ub=5;
    dim=4;

case 'F16'
    fobj = @F16;
    lb=-5;
    ub=5;
    dim=2;

case 'F17'
    fobj = @F17;
    lb=[-5,0];
```

```
    ub=[10,15];
    dim=2;

case 'F18'
    fobj = @F18;
    lb=-2;
    ub=2;
    dim=2;

case 'F19'
    fobj = @F19;
    lb=0;
    ub=1;
    dim=3;

case 'F20'
    fobj = @F20;
    lb=0;
    ub=1;
    dim=6;

case 'F21'
    fobj = @F21;
    lb=0;
    ub=10;
    dim=4;
```

```
    case 'F22'
        fobj = @F22;
        lb=0;
        ub=10;
        dim=4;

    case 'F23'
        fobj = @F23;
        lb=0;
        ub=10;
        dim=4;
end

end

% F1
function o = F1(x)
o=sum(x.^2);
end

% F2
function o = F2(x)
o=sum(abs(x))+prod(abs(x));
end
```

```
% F3
function o = F3(x)
dim=size(x,2);
o=0;
for i=1:dim
   o=o+sum(x(1:i))^2;
end
end

% F4
function o = F4(x)
o=max(abs(x));
end

% F5
function o = F5(x)
dim=size(x,2);
o=sum(100*(x(2:dim)-(x(1:dim-1).^2)).^2+(x(1:dim-1)-1).^2);
end

% F6
function o = F6(x)
o=sum(abs((x+.5)).^2);
end

% F7
```

```
function o = F7(x)
dim=size(x,2);
o=sum([1:dim].*(x.^4))+rand;
end

% F8
function o = F8(x)
o=-418.9829*5 + sum(-x.*sin(sqrt(abs(x))));
end

% F9
function o = F9(x)
dim=size(x,2);
o=sum(x.^2-10*cos(2*pi.*x))+10*dim;
end

% F10
function o = F10(x)
dim=size(x,2);
o=-20*exp(-.2*sqrt(sum(x.^2)/dim))-exp(sum(cos(2*pi.*x))/dim)+20+exp(1);
end

% F11
function o = F11(x)
dim=size(x,2);
o=sum(x.^2)/4000-prod(cos(x./sqrt([1:dim])))+1;
```

```
end

% F12
function o = F12(x)
dim=size(x,2);
o=(pi/dim)*(10*((sin(pi*(1+(x(1)+1)/4)))^2)+sum((((x(1:dim-1)+1)./4).^2).*...
(1+10.*((sin(pi.*(1+(x(2:dim)+1)./4)))).^2))+((x(dim)+1)/4)^2)+sum(Ufun(x,10,100,4));
end

% F13
function o = F13(x)
dim=size(x,2);
o=.1*((sin(3*pi*x(1)))^2+sum((x(1:dim-1)-1).^2.*(1+(sin(3.*pi.*x(2:dim))).^2))+...
((x(dim)-1)^2)*(1+(sin(2*pi*x(dim)))^2))+sum(Ufun(x,5,100,4));
end

% F14
function o = F14(x)
aS=[-32 -16 0 16 32 -32 -16 0 16 32 -32 -16 0 16 32 -32 -16 0 16 32 -32 -16 0 16 32;,...
-32 -32 -32 -32 -32 -16 -16 -16 -16 -16 0 0 0 0 0 16 16 16 16 16 32 32 32 32 32];
for j=1:25
    bS(j)=sum((x'-aS(:,j)).^6);
end
o=(1/500+sum(1./([1:25]+bS))).^(-1);
end
```

```
% F15
function o = F15(x)
aK=[.1957 .1947 .1735 .16 .0844 .0627 .0456 .0342 .0323 .0235 .0246];
bK=[.25 .5 1 2 4 6 8 10 12 14 16];bK=1./bK;
o=sum((aK-((x(1).*(bK.^2+x(2).*bK))./(bK.^2+x(3).*bK+x(4)))).^2);
end

% F16
function o = F16(x)
o=4*(x(1)^2)-2.1*(x(1)^4)+(x(1)^6)/3+x(1)*x(2)-4*(x(2)^2)+4*(x(2)^4);
end

% F17
function o = F17(x)
o=(x(2)-(x(1)^2)*5.1/(4*(pi^2))+5/pi*x(1)-6)^2+10*(1-1/(8*pi))*cos(x(1))+10;
end

% F18
function o = F18(x)
o=(1+(x(1)+x(2)+1)^2*(19-14*x(1)+3*(x(1)^2)-14*x(2)+6*x(1)*x(2)+3*x(2)^2))*...
   (30+(2*x(1)-3*x(2))^2*(18-32*x(1)+12*(x(1)^2)+48*x(2)-36*x(1)*x(2)+27*(x(2)^2)));
end

% F19
function o = F19(x)
aH=[3 10 30;.1 10 35;3 10 30;.1 10 35];cH=[1 1.2 3 3.2];
```

```
pH=[.3689 .117 .2673;.4699 .4387 .747;.1091 .8732 .5547;.03815 .5743 .8828];
o=0;
for i=1:4
    o=o-cH(i)*exp(-(sum(aH(i,:).*((x-pH(i,:)).^2))));
end
end

% F20
function o = F20(x)
aH=[10 3 17 3.5 1.7 8;.05 10 17 .1 8 14;3 3.5 1.7 10 17 8;17 8 .05 10 .1 14];
cH=[1 1.2 3 3.2];
pH=[.1312 .1696 .5569 .0124 .8283 .5886;.2329 .4135 .8307 .3736 .1004 .9991;...
.2348 .1415 .3522 .2883 .3047 .6650;.4047 .8828 .8732 .5743 .1091 .0381];
o=0;
for i=1:4
    o=o-cH(i)*exp(-(sum(aH(i,:).*((x-pH(i,:)).^2))));
end
end

% F21
function o = F21(x)
aSH=[4 4 4 4;1 1 1 1;8 8 8 8;6 6 6 6;3 7 3 7;2 9 2 9;5 5 3 3;8 1 8 1;6 2 6 2;7 3.6 7 3.6];
cSH=[.1 .2 .2 .4 .4 .6 .3 .7 .5 .5];

o=0;
for i=1:5
```

```
    o=o-((x-aSH(i,:))*(x-aSH(i,:))'+cSH(i))^(-1);
end
end

% F22
function o = F22(x)
aSH=[4 4 4 4;1 1 1 1;8 8 8 8;6 6 6 6;3 7 3 7;2 9 2 9;5 5 3 3;8 1 8 1;6 2 6 2;7 3.6 7 3.6];
cSH=[.1 .2 .2 .4 .4 .6 .3 .7 .5 .5];

o=0;
for i=1:7
    o=o-((x-aSH(i,:))*(x-aSH(i,:))'+cSH(i))^(-1);
end
end

% F23
function o = F23(x)
aSH=[4 4 4 4;1 1 1 1;8 8 8 8;6 6 6 6;3 7 3 7;2 9 2 9;5 5 3 3;8 1 8 1;6 2 6 2;7 3.6 7 3.6];
cSH=[.1 .2 .2 .4 .4 .6 .3 .7 .5 .5];

o=0;
for i=1:10
    o=o-((x-aSH(i,:))*(x-aSH(i,:))'+cSH(i))^(-1);
end
end
```

```
function o=Ufun(x,a,k,m)
o=k.*((x-a).^m).*(x>a)+k.*((-x-a).^m).*(x<(-a));
end
```

c. GWO

```
% Grey Wolf Optimizer
function [Alpha_score,Alpha_pos,Convergence_curve]=GWO(SearchAgents_no,Max_iter,lb,ub,dim,fobj)
% initialize alpha, beta, and delta_pos
Alpha_pos=zeros(1,dim);
Alpha_score=inf; %change this to -inf for maximization problems
Beta_pos=zeros(1,dim);
Beta_score=inf; %change this to -inf for maximization problems
Delta_pos=zeros(1,dim);
Delta_score=inf; %change this to -inf for maximization problems
%Initialize the positions of search agents
Positions=initialization(SearchAgents_no,dim,ub,lb);
Convergence_curve=zeros(1,Max_iter);
l=0;% Loop counter
% Main loop
while l<Max_iter
    for i=1:size(Positions,1)
       % Return back the search agents that go beyond the boundaries of the search space
        Flag4ub=Positions(i,:)>ub;
        Flag4lb=Positions(i,:)<lb;
        Positions(i,:)=(Positions(i,:).*(~(Flag4ub+Flag4lb)))+ub.*Flag4ub+lb.*Flag4lb;
```

```
    % Calculate objective function for each search agent
    fitness=fobj(Positions(i,:));
    % Update Alpha, Beta, and Delta
    if fitness<Alpha_score
        Alpha_score=fitness; % Update alpha
        Alpha_pos=Positions(i,:);
    end
    if fitness>Alpha_score && fitness<Beta_score
        Beta_score=fitness; % Update beta
        Beta_pos=Positions(i,:);
    end
    if fitness>Alpha_score && fitness>Beta_score && fitness<Delta_score
        Delta_score=fitness; % Update delta
        Delta_pos=Positions(i,:);
    end
end
a=2-l*((2)/Max_iter); % a decreases linearly fron 2 to 0
% Update the Position of search agents including omegas
for i=1:size(Positions,1)
    for j=1:size(Positions,2)
        r1=rand(); % r1 is a random number in [0,1]
        r2=rand(); % r2 is a random number in [0,1]
        A1=2*a*r1-a; % Equation (3.3)
        C1=2*r2; % Equation (3.4)
```

```
            D_alpha=abs(C1*Alpha_pos(j)-Positions(i,j)); % Equation (3.5)-part 1
            X1=Alpha_pos(j)-A1*D_alpha; % Equation (3.6)-part 1
            r1=rand();
            r2=rand();
            A2=2*a*r1-a; % Equation (3.3)
            C2=2*r2; % Equation (3.4)
            D_beta=abs(C2*Beta_pos(j)-Positions(i,j)); % Equation (3.5)-part 2
            X2=Beta_pos(j)-A2*D_beta; % Equation (3.6)-part 2
            r1=rand();
            r2=rand();
            A3=2*a*r1-a; % Equation (3.3)
            C3=2*r2; % Equation (3.4)
            D_delta=abs(C3*Delta_pos(j)-Positions(i,j)); % Equation (3.5)-part 3
            X3=Delta_pos(j)-A3*D_delta; % Equation (3.5)-part 3
            Positions(i,j)=(X1+X2+X3)/3;% Equation (3.7)
        end
    end
    l=l+1;
    Convergence_curve(l)=Alpha_score;
end
```

d. initialization

```
% This function initialize the first population of search agents
function Positions=initialization(SearchAgents_no,dim,ub,lb)
```

```
Boundary_no= size(ub,2); % numnber of boundaries


% If the boundaries of all variables are equal and user enter a signle
% number for both ub and lb
if Boundary_no==1
    Positions=rand(SearchAgents_no,dim).*(ub-lb)+lb;
end


% If each variable has a different lb and ub
if Boundary_no>1
    for i=1:dim
        ub_i=ub(i);
        lb_i=lb(i);
        Positions(:,i)=rand(SearchAgents_no,1).*(ub_i-lb_i)+lb_i;
    end
end
```

e. func_plot

```
% This function draw the benchmark functions
function func_plot(func_name)
[lb,ub,dim,fobj]=Get_Functions_details(func_name);
switch func_name
    case 'F1'
        x=-100:2:100; y=x; %[-100,100]
    case 'F2'
```

```
    x=-100:2:100; y=x; %[-10,10]
case 'F3'
    x=-100:2:100; y=x; %[-100,100]
case 'F4'
    x=-100:2:100; y=x; %[-100,100]
case 'F5'
    x=-200:2:200; y=x; %[-5,5]
case 'F6'
    x=-100:2:100; y=x; %[-100,100]
case 'F7'
    x=-1:0.03:1;  y=x  %[-1,1]
case 'F8'
    x=-500:10:500;y=x; %[-500,500]
case 'F9'
    x=-5:0.1:5;   y=x; %[-5,5]
case 'F10'
    x=-20:0.5:20; y=x;%[-500,500]
case 'F11'
    x=-500:10:500; y=x;%[-0.5,0.5]
case 'F12'
    x=-10:0.1:10; y=x;%[-pi,pi]
case 'F13'
    x=-5:0.08:5; y=x;%[-3,1]
case 'F14'
```

```
        x=-100:2:100; y=x;%[-100,100]
    case 'F15'
        x=-5:0.1:5; y=x;%[-5,5]
    case 'F16'
        x=-1:0.01:1; y=x;%[-5,5]
    case 'F17'
        x=-5:0.1:5; y=x;%[-5,5]
    case 'F18'
        x=-5:0.06:5; y=x;%[-5,5]
    case 'F19'
        x=-5:0.1:5; y=x;%[-5,5]
    case 'F20'
        x=-5:0.1:5; y=x;%[-5,5]
    case 'F21'
        x=-5:0.1:5; y=x;%[-5,5]
    case 'F22'
        x=-5:0.1:5; y=x;%[-5,5]
    case 'F23'
        x=-5:0.1:5; y=x;%[-5,5]
end
L=length(x);
f=[];

for i=1:L
```

```
    for j=1:L
        if strcmp(func_name,'F15')==0 && strcmp(func_name,'F19')==0 && strcmp(func_name,'F20')==0
&& strcmp(func_name,'F21')==0 && strcmp(func_name,'F22')==0 && strcmp(func_name,'F23')==0
            f(i,j)=fobj([x(i),y(j)]);
        end
        if strcmp(func_name,'F15')==1
            f(i,j)=fobj([x(i),y(j),0,0]);
        end
        if strcmp(func_name,'F19')==1
            f(i,j)=fobj([x(i),y(j),0]);
        end
        if strcmp(func_name,'F20')==1
            f(i,j)=fobj([x(i),y(j),0,0,0,0]);
        end
        if strcmp(func_name,'F21')==1 || strcmp(func_name,'F22')==1 ||strcmp(func_name,'F23')==1
            f(i,j)=fobj([x(i),y(j),0,0]);
        end
    end
end
surfc(x,y,f,'LineStyle','none');
end
```

3. MATLAB code for 13 provided functions.

a. Function 1.

```
function o = f1(x)
```

```
o=sum(x.^2);
end
```

b. Function 2.

```
function o = f2(x)
o=sum(abs(x))+prod(abs(x));
end
```

c. Function 3.

```
function o = f3(x)
dim=size(x,2);
o=0;
for i=1:dim
   o=o+sum(x(1:i))^2;
end
end
```

d. Function 4.

```
function o = f4(x)
o=max(abs(x));
end
```

e. Function 5.

```
function o = f5(x)
dim=size(x,2);
o=sum(100*(x(2:dim)-(x(1:dim-1).^2)).^2+(x(1:dim-1)-1).^2);
end
```

f. Function 6.

```
function o = f6(x)
o=sum(abs((x+.5)).^2);
end
```

g. Function 7.

```
function z=f1(x)
z=sum((1:length(x)).*x.^4) + rand();
end
```

h. Function 8.

```
function z=f8(x)
z=-sum(x .* sin(sqrt(abs(x))));
end
```

i. Function 9.

```
function z=f9(x)
z=sum(x.^2 - 10 * cos(2 * pi * x) + 10);
end
```

j. Function 10.

```
function o = f10(x)

    a = 20;
    b = 0.2;
    c = 2 * pi;

    sum_sq_term = -a * exp(-b * sqrt(sum(x.^2) / numel(x)));
    cos_term = -exp(sum(cos(c * x)) / numel(x));

    o = sum_sq_term + cos_term + a + exp(1) - 0;

end
```

k. Function 11.

```
function o =f11(x)
```

```
    sum_part = sum(x.^2) / 4000;
    prod_part = prod(cos(x ./ sqrt(1:numel(x))));

    z = sum_part - prod_part + 1;
end
```

l. Function 12.

```
function o = f12(x)
dim=size(x,2);
o=(pi/dim)*(10*((sin(pi*(1+(x(1)+1)/4)))^2)+sum((((x(1:dim-1)+1)./4).^2).*...
(1+10.*((sin(pi.*(1+(x(2:dim)+1)./4)))).^2))+((x(dim)+1)/4)^2)+sum(Ufun(x,10,100,4));
end

function o=Ufun(x,a,k,m)
o=k.*((x-a).^m).*(x>a)+k.*((-x-a).^m).*(x<(-a));
end
```

m. Function 13.

```
function o = f13(x)
dim=size(x,2);
o=.1*((sin(3*pi*x(1)))^2+sum((x(1:dim-1)-1).^2.*(1+(sin(3.*pi.*x(2:dim))).^2))+...
((x(dim)-1)^2)*(1+(sin(2*pi*x(dim)))^2))+sum(Ufun(x,5,100,4));
end

function o=Ufun(x,a,k,m)
o=k.*((x-a).^m).*(x>a)+k.*((-x-a).^m).*(x<(-a));
end
```

4. MATLAB code for GEA improvement

```
% Main GEA Improve
clc;
```

```
close all;
clear all;
tic;

for NF = 1  % NF: function number
    %% Problem Definition
    VarMin = -100;              % Decision Variables Lower Bound
    VarMax = -VarMin;           % Decision Variables Upper Bound
    nVar = 20;                  % Number of Decision Variables
    VarSize = [1 nVar];         % Decision Variables Matrix Size
    Nc = 40;                    % Initial Number of Channels
    FEs = 5000;                 % Maximum Number of Function Evaluations
    nPop = 40;                  % Number of Geysers (Swarm Size)

    %% Initialization
    Geyser.Position = [];       % Empty Geyser Structure
    Geyser.Cost = [];
    pop = repmat(Geyser, nPop, 1);  % Initialize Population Array
    BestSol.Cost = inf;         % Initialize Best Solution Ever Found

    % Create Initial Population of Geysers
    for i = 1:nPop
        pop(i).Position = unifrnd(VarMin, VarMax, VarSize);
        pop(i).Cost = CostFunction(pop(i).Position);
        if pop(i).Cost <= BestSol.Cost
            BestSol = pop(i);
        end
        BestCost(i) = BestSol.Cost;
    end

    it = 0;
    alpha = 0.1;  % Initial learning rate

    while it <= FEs
```

```
%% Calculate Diversity
positions = reshape([pop.Position], nVar, nPop)';
meanPosition = mean(positions);
diversity = mean(sqrt(sum((positions - meanPosition) .^ 2, 2)));

%% Dynamically Adjust Nc Based on Diversity
maxNc = nPop;
minNc = 5;
Nc = round(minNc + (maxNc - minNc) * exp(-diversity));  % Example adjustment formula

for i = 1:nPop
    % Implementing Eq(3)
    for ii = 1:nPop
        D1(ii) = sum(pop(i).Position .* pop(ii).Position) / (((sum(pop(i).Position .^ 2) * sum(pop(ii).Position
.^ 2))) ^ 0.5); % Eq(3)
        if ii == i
            D1(ii) = inf;
        end
    end
    S1 = min(D1);
    [~, j1] = find(S1 == D1);

    % Calculating the pressure value in Eq(6)
    [CS, Sortorder] = sort([pop.Cost]);
    dif = (CS(end) - CS(1));  % fmax-fmin  Eq(6)
    G = ((pop(i).Cost - CS(1)) / dif);  % f(i)-fmin  Eq(6)
    if dif == 0
        G = 0;
    end
    if it == 1
        it = 2;
    end
    P_i(i) = (((G^(2/it)) - (G^((it + 1) / it))) ^ 0.5) * ((it / (it - 1)) ^ 0.5);  % Pi Eq(6)
```

```
        % Adaptive Learning Rate Update
        alpha = 0.1 + 0.9 * (1 - it / FEs);  % Decrease alpha over time

        % Search for channels using Roulette wheel selection
        ImpFitness = 0;
        for ii = 1:Nc
            ImpFitness = ImpFitness + pop(ii).Cost;
        end
        p = [];
        if ImpFitness == 0
            ImpFitness = 1e-320;
        end
        for ii = 1:Nc
            p(ii) = pop(ii).Cost / ImpFitness;  % Eq(1)
        end
        if sum(p) == 0
            i1 = 1;
        else
            i1 = RouletteWheelSelection((p));
        end
        flag = 0;

        % Implementing Eq(5) with adaptive alpha
        newsol.Position = pop(j1(1)).Position + (rand(VarSize)) .* (pop(i1).Position - pop(j1(1)).Position) +
(rand(VarSize)) .* (pop(i1).Position - pop(i).Position);
        newsol.Position = max(newsol.Position, VarMin);
        newsol.Position = min(newsol.Position, VarMax);
        newsol.Cost = CostFunction(newsol.Position);

        if newsol.Cost <= pop(i).Cost
            pop(i) = newsol;
            flag = 0;
        end
        if newsol.Cost <= BestSol.Cost
```

```
            BestSol = newsol;
        end

        it = it + 1;  % Increment it after updating pop and BestSol
        BestCost(it) = BestSol.Cost;
        disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCost(it))]);

        if flag == 0
            i2 = RouletteWheelSelection((1 - p));  % Implementing Roulette wheel selection by Eq(7)
            % Implementing Eq
            newsol.Position = pop(i2).Position + rand * (P_i(i) - rand) * unifrnd(1 * VarMin, 1 * VarMax,
VarSize);
            newsol.Position = max(newsol.Position, VarMin);
            newsol.Position = min(newsol.Position, VarMax);
            newsol.Cost = CostFunction(newsol.Position);

            if newsol.Cost <= pop(i).Cost
                pop(i) = newsol;
            end
            if newsol.Cost <= BestSol.Cost
                BestSol = newsol;
            end
            BestCost(it) = BestSol.Cost;
        end
    end
    [~, Sortorder] = sort([pop.Cost]);
    pop = pop(Sortorder);
end

elapsedTime = toc;
fprintf('Total running time: %.4f seconds\n', elapsedTime);

% Plotting the results
Cost_Result(1) = BestSol.Cost;
```

```
    Result(1:FEs) = BestCost(:, 1:FEs);

    Mean(NF) = mean(Cost_Result);
    Std(NF) = std(Cost_Result);

    hold on;
    [r_res, ~] = size(Result);
    if r_res > 1
        plot(log(mean(Result)), 'b', 'LineWidth', 4); hold on;
    else
        plot(log((Result)), 'b', 'LineWidth', 4); hold on;
    end
    xlabel('Number of Fitness Evaluations');
    ylabel('Mean value to best (log)');
    grid on;
end
```