Optimizing Truss Structures Using Genetic Algorithms

Clearing Workspace

```
clc;
clear;
close all;
```

Initialization of global variables

```
global_variables
% global population_size  VarMin VarMax
% global number_of_nodes nodes nodal_force fixed_nodes connectivity_matrix lengths_of_bars bars
% global E number_of_bars KG F U
```

Initialization Of variables bounds and Velocity Model (Young Model)

```
VarMin = 0.000001;        % Lower Bound of Decision Variables
VarMax =  1;          % Upper Bound of Decision Variables
E = 2e5;
```

Loading Data

```
nodes = load('nodes')
```

```
nodes = 6×2
         0         0
    0.5000         0
    1.0000         0
    1.0000    0.5000
    0.5000    0.5000
         0    0.5000
```

```
connectivity_matrix = load('connectivity_matrix')
```

```
connectivity_matrix = 6×6
     0     1     0     1     1     1
     1     0     1     1     1     1
     0     1     0     1     1     1
     1     1     1     0     1     0
     1     1     1     1     0     1
```

```
     1     1     1     0     1     0
```

```
nodal_force = load('nodal_force')
```

```
nodal_force = 6×2
           0           0
       -2000           0
           0       -2000
        2000           0
           0        2000
           0           0
```

```
fixed_nodes = load('fixed_nodes')
```

```
fixed_nodes = 1×4
     1     2    11    12
```

```
number_of_nodes = size(nodes, 1)
```

```
number_of_nodes = 6
```

```
number_of_bars = sum(sum(connectivity_matrix))/2
```

```
number_of_bars = 13
```

```
bars = zeros(2, number_of_bars);
lengths_of_bars = zeros(1, number_of_bars);
```

Calculating the length of each bar

```
k = 0;
for i=1:number_of_nodes
    for j=i+1:number_of_nodes
        if connectivity_matrix(i,j) == 1
            k = k + 1;
            lengths_of_bars(k) = sqrt((nodes(i,1)-nodes(j,1))^2+(nodes(i,2)-nodes(j,2))^2);
            bars(1,k)= i;
            bars(2,k)= j;
        end
    end
end
```

Creating Individuals data structure

every individual will have the structure (position, energy), where position is a number_of_bars-dimensional vector and energy if the deformation energy with respect to the lengths of bars.

```
agent.position=[];
agent.energy=[];
```

Initialization of genetic algorithm parameters

we will create variables:

- maximum_number_of_generations: The maximum number of generations
- population_size: The size of population (number of individuals in population)
- crossover_probability: Percentage of population that will be selected to crossover
- offspring_size: The number of individuals to crossover
- gamma: Extra range factor for crossover
- mutation_probability: Mutation probability
- number_of_mutantes: Number of mutants
- mu: Mutation rate

```
maximum_number_of_generations=100
```

```
maximum_number_of_generations = 100
```

```
population_size=25
```

```
population_size = 25
```

```
crossover_probability=0.75
```

```
crossover_probability = 0.7500
```

```
offspring_size=2*round(crossover_probability*population_size/2)
```

```
offspring_size = 18
```

```
gamma=0.4
```

```
gamma = 0.4000
```

```
mutation_probability=0.1
```

```
mutation_probability = 0.1000
```

```
number_of_mutantes=round(mutation_probability*population_size)
```

```
number_of_mutantes = 3
```

```
mu=0.1
```

```
mu = 0.1000
```

now we will create the population with population_size is the number of agents (i.e individuals)

```
population=repmat(agent,population_size,1);
```

Selection method

a dialog will pop-up asking to select a selection method between:

1. Roulette Wheel
2. Tournament
3. Random

```
ANSWER=questdlg('Choose selection method:', 'Genetic Algorith', 'Roulette Wheel', 'Tournament',

UseRouletteWheelSelection=strcmp(ANSWER, 'Roulette Wheel');
UseTournamentSelection=strcmp(ANSWER, 'Tournament');
UseRandomSelection=strcmp(ANSWER, 'Random');
```

if the user select " Roulette Wheel " we will add a variable beta to determinate the selection pressure.

if the user select " Tournament " we will add a variable TournamentSize to determinate the number of individuals putted into tournament.

```
if UseRouletteWheelSelection
    beta=8;
end

if UseTournamentSelection
    TournamentSize=3;
end
```

Initializing the population

- Generating positions randomly from the uniform distribution.
- Calculation the fitness for each generated position.

```
cost_function = @compliance;
for i=1:population_size
    population(i).position = unifrnd(VarMin, VarMax, [1 number_of_bars]);
    population(i).energy=cost_function(population(i).position);
end
```

```
costs = [population.energy];
[costs, SortOrder] = sort(costs);
population = population(SortOrder);
```

We will select the agent who has the best fitness (minimum fitness)

```
best_agent = population(1);
```

best cost value in each generation

```
best_agents=zeros(maximum_number_of_generations,1);
```

```
worst_cost = population(end).energy;
```

```
for it=1:maximum_number_of_generations

    % Calculate Selection Probabilities
    if UseRouletteWheelSelection
        P=exp(-beta*costs/worst_cost);
        P=P/sum(P);
    end

    % Crossover
    popc=repmat(agent,offspring_size/2,2);
    for k=1:offspring_size/2

        % Select Parents Indices
        if UseRouletteWheelSelection
            i1=RouletteWheelSelection(P);
            i2=RouletteWheelSelection(P);
        end
        if UseTournamentSelection
            i1=TournamentSelection(population,TournamentSize);
            i2=TournamentSelection(population,TournamentSize);
        end
        if UseRandomSelection
```

```
        i1=randi([1 population_size]);
        i2=randi([1 population_size]);
    end

    % Select Parents
    p1=population(i1);
    p2=population(i2);

    % Apply Crossover
    [popc(k,1).position, popc(k,2).position]=Crossover(p1.position,p2.position,gamma,VarMi

    % Evaluate Offsprings
    popc(k,1).energy=cost_function(popc(k,1).position);
    popc(k,2).energy=cost_function(popc(k,2).position);

end
popc=popc(:);


% Mutation
popm=repmat(agent,number_of_mutantes,1);
for k=1:number_of_mutantes

    % Select Parent
    i=randi([1 population_size]);
    p=population(i);

    % Apply Mutation
    popm(k).position=Mutate(p.position,mu,VarMin,VarMax);

    % Evaluate Mutant
    popm(k).energy=cost_function(popm(k).position);

end

% Create Merged Population
population =[population
             popc
             popm]; % #ok

% Sort Population
costs=[population.energy];
[costs, SortOrder]=sort(costs);
population=population(SortOrder);

% Update Worst Cost
worst_cost=max(worst_cost,population(end).energy);

% Truncation
population=population(1:population_size);
costs=costs(1:population_size);

% Store Best Solution Ever Found
```

```
    best_agent=population(1);

    % Store Best Cost Ever Found
    best_agents(it)=best_agent.energy;

    % Show Iteration Information
    %disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(best_agents(it))]);

end
```

```
figure;
semilogy(best_agents,'LineWidth',2);
% plot(BestCost,'LineWidth',2);
xlabel('Generations');
title('Change in fitness over generations')
ylabel('Fitness');
grid on;
```