AAE550: HW3

Rahul Deshmukh
deshmuk5@purdue.edu
PUID: 0030004932

November 16, 2018

I Nelder-Mead Simplex

For this problem we are required to carry out several iterations of Nelder-Mead Simplex algorithm on the Goldstein-Price Function with different initial guesses.

As Nelder-Mead Simplex cannot handle bounds/constraints explicitly. We need to account for them using penalty approach.I have used exterior penalty method here without squaring as nelder-mead can handle non-smooth functions.

1) Objective Function formulation
As we have only side-constraints for this problem, I have reduced these constraints to the order of one and then incorporated into the objective function as follows:

Goldstein-Price Function

$f(x) = [1+(x_1+x_2+1)^2 * (19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)] * [30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)]$

Subject to:

$$\begin{aligned} g_1(x) &= -x_1/2 - 1 \\ g_2(x) &= -x_2/2 - 1 \\ g_3(x) &= x_1/2 - 1 \\ g_4(x) &= x_1/2 - 1 \end{aligned}$$

Pseudo Objective Function:

$$\min_{\vec{x}} \Phi(x) = f(x) + rp * \Sigma_i max(0, g_i(x))$$

I have chosen two different values for my penalty multiplier as 100 and 10^6. The matlab script for the Pseudo objective function is:

```
function phi = NMfunc(x)
% objective function: Goldstein Price Function
% linear exterior penalty to enforce bounds

f = ...
    [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)];
% constraints
g(1) = -x(1) / (2) - 1;  % enforces lower bound
g(2) = -x(2) / (2) - 1;
g(3) = x(1) / (2) - 1; % enforces upper bound
g(4) = x(2) / (2) - 1;
% Pseudo Obj
P = 0.0;    % initialize penalty function
for i = 1:4
    P = P + 1 * max(0,g(i));  % use c_j = 10 for all bounds
end

rp=100;
phi = f + rp*P;
end
```

The Matlab script for call to NM is:

```
clc;
close all;
clear all;

options = optimset('Display','iter');

x0 = [3,-2];

[x,fval,exitflag,output] = fminsearch('NMfunc',x0,options)

fprintf('------------------------------------------------------------');
% restart: second call to fminsearch
x0=x;
[x,fval,exitflag,output] = fminsearch('NMfunc',x0,options)

%  plot of function and optimum point
t=-2:1/10:2;
[X,Y]=meshgrid(t,t);
f =@(x) ...
    [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)];

Z=zeros(size(t,2),size(t,2));
for i=1:size(Z,1)
    for j = 1:size(Z,2)
        Z(i,j)=f([X(i,j),Y(i,j)]);
    end
end

figure(1);
surf(X,Y,Z)
```

```
xlabel('x1');
ylabel('x2');
zlabel('function');

msize=100;
figure(2);
contour(X,Y,Z,50);hold on;
colorbar;
scatter(x0(1),x0(2),msize,'o','g');ci
scatter(x(1),x(2),msize,'^','b');
x_star=[0,-1];
scatter(x_star(1),x_star(2),msize,'*','r');
xlabel('x1');
ylabel('x2');
```

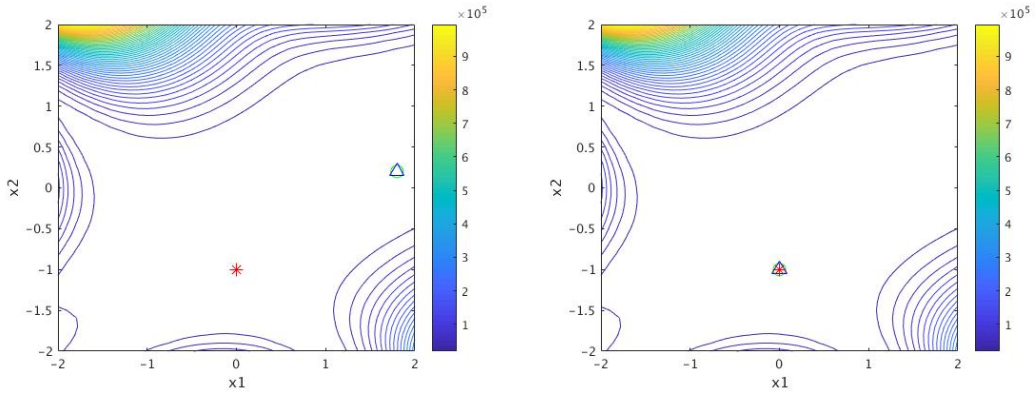
2) Summary for $x_0 = [2, 2]^T$:

For Penalty Multiplier as 100

	x0	Function Eval	x*	f(x*)
Run 1	2 2	183	1.8 0.2	84
Run 2	1.8 0.2	44	1.8 0.2	84

For Penalty Multiplier as 10^6

	x0	Function Eval	x*	f(x*)
Run 1	2 2	90	0 -1	3
Run 2	0 -1	21	0 -1	3



(a) rp=100 (b) rp=100

Figure 1: contour plot of goldstein-price function with first phase solution (circle), second phase solution (triangle) and original optimal solution(asteriks)

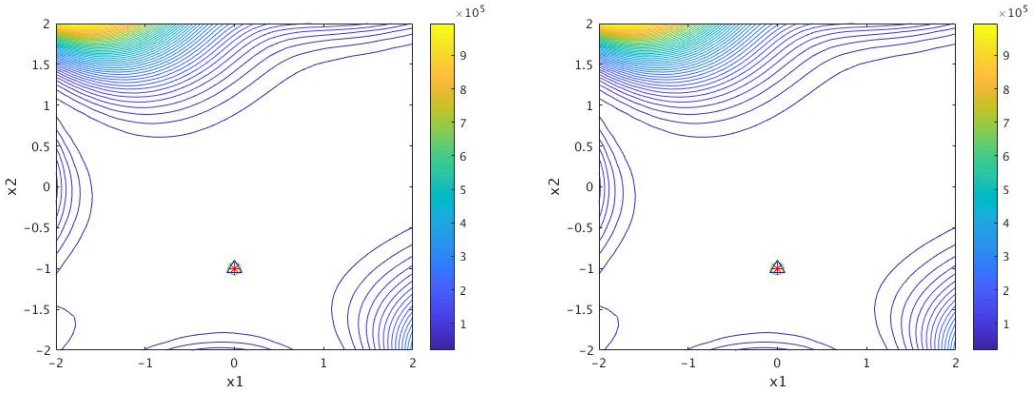
3) Summary for $x_0 = [-2, 2]^T$:

For Penalty Multiplier as 10

	x0	Function Eval	x*	f(x*)
Run 1	-2 2	118	0 -1	3
Run 2	0 -1	21	0 -1	3

For Penalty Multiplier as 10^6

	x0	Function Eval	x*	f(x*)
Run 1	-2 2	90	0 -1	3
Run 2	0 -1	21	0 -1	3



(a) rp=100 (b) rp=100

Figure 2: contour plot of goldstein-price function with first phase solution (circle), second phase solution (triangle) and original optimal solution(asteriks)

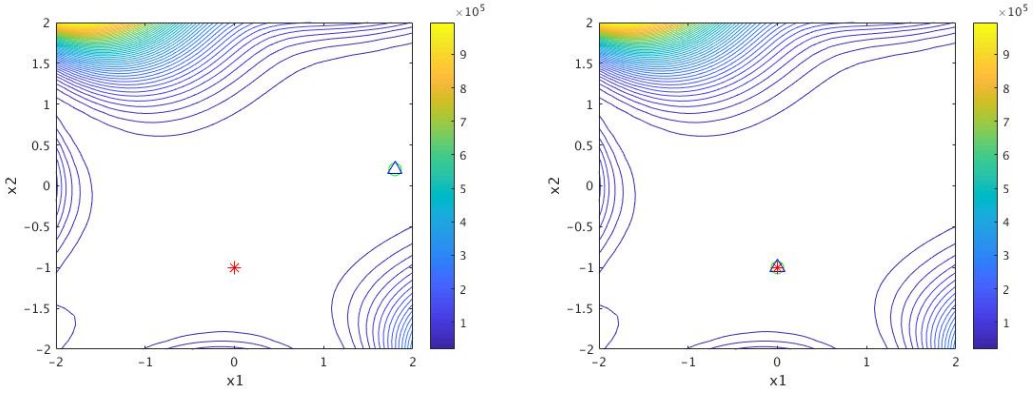
4) Summary for $x_0 = [3, -2]^T$:

For Penalty Multiplier as 10

	x0	Function Eval	x*	f(x*)
Run 1	3 -2	91	1.8 0.2	84
Run 2	1.8 0.2	44	1.8 0.2	84

For Penalty Multiplier as 10^6

	x0	Function Eval	x*	f(x*)
Run 1	3 -2	84	0 -1	3
Run 2	0 -1	21	0 -1	3



(a) rp=100 (b) rp=100

Figure 3: contour plot of goldstein-price function with first phase solution (circle), second phase solution (triangle) and original optimal solution(asteriks)

5) As can be observed from the results of the three different starting solutions and for different penalty multipliers, The Nelder-Mead Algorithm is clearly not able to find the Global minimum for smaller penalty multiplier. Form the contour plot of goldstein-price function we can see that the goldstein-price function is pretty flat in a wide region and Nelder-Mead is getting stuck in this region at a local minimum.

   Also, for a higher value of penalty multiplier, NM is doing a good job and gets us the global optimum. This indicates that for NM choosing a high value of penalty multiplier will have better chance of finding the global optimum.

   Although this function is a continuous function, Nelder-Mead performs well for non-smooth functions as well because it does not require any gradients.

II Simulated Annealing

For this problem we have to use simulated annealing algorithm for minimization of the Goldstein-Price Function. I have made use of the provided code `SA_550.m` for this task without making any changes to it.

Unlike Nelder-Mead where we had to create a pseudo objective function to incorporate the bounds, in simulated annealing the problem bounds are handled directly.

The matlab code for my objective function is:

```
function f = SAfunc(x)
% objective function: egg-crate function from class
f = ...
    [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
```

The matlab code for calling simulated annealing is:

```
clc;
close all;
clear all;
bounds = [-2 2;
          -2 2];% upper and lower bounds for each of the two variables
X0 = [2; 2];% initial design NOTE: this is a column vector
options = zeros(1,9);% set up options array for non-default inputs
options(1) = 26;        % initial temperature (default = 50)
options(6) = 0.6;       % cooling rate (default = 0.5)
[xstar,fstar,count,accept,oob]=SA_550('SAfunc',bounds,X0,options);
```

1) Effect of random number generator on the solution by SA:

Summary Table for two different initial design.

	x0	T0	rT	Function Eval	final T	x*	f(x*)
Run 1	2 2	28	0.7	43201	1.72725e-07	1.8 0.2	84
Run 2	2 2	28	0.7	43201	1.72725e-07	0 -1	3
Run 1	-2 2	28	0.7	43201	1.72725e-07	0 -1	3
Run 2	-2 2	28	0.7	41601	3.52499e-07	0 -1	3

As can be observed from the above table for the same initial design we can get different optimal solutions just by running the program again but at nearly same computational cost.
Also, when running SA with second initial solution $[-2, 2]^T$ I observed that i am getting the same optimal solution for several runs but with changing computational cost.

2) Effect of Initial Temperature on the results:

Summary table for the runs carried out:

	x0	T0	rT	Function Eval	final T	x*	f(x*)
Run 1	2 2	29	0.7	40801	5.21555e-07	0 -1	3
Run 2	2 2	28	0.7	43201	1.72725-07	0 -1	3
Run 1	2 2	27	0.7	44801	8.16124e-08	0 -1	3

It can be observed that the higher the initial temperature, we have more chances of finding the global minimum with a lower computational cost as SA gets to do early exploration.

NOTE: I had to run SA several times for these temperature and rate values so as to get the same optimal solution, because then only this comparison table would have made sense.

3) Effect of cooling rate

Summary table for the runs carried out:

	x0	T0	rT	Function Eval	final T	x*	f(x*)
Run 1	2 2	28	0.8	66401	3.16619e-07	0 -1	3
Run 2	2 2	28	0.7	43201	1.72725-07	0 -1	3
Run 1	2 2	28	0.6	29601	2.88804e-07	0 -1	84

As can be observed from the above table, for a slower cooling rate the we have higher chances of finding the global minimum but at a cost of higher function evaluations. Also. for a faster cooling rate, we have lower function evaluations but we missed the global minimum.

III Genetic Algorithm

For this problem, we are again required to carry out the minimization of Goldstein-Price Function using genetic algorithm. I have used the in-class provided code `GA550.m` without any changes. This file had one function missing by the name 'b10to2' in the encode function block. I have coded up this part for the sake of completion, code placed at appendix.

Similar to SA, GA handles the bounds explicitly, In fact bounds are used to generate the initial generation. The code for objective function for GA is:

```
function f = GAfunc(x)
% objective function: egg-crate function from class
f = ...
    [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
```

The code for calling GA is:

```
clc;
close all;
clear all;

options = goptions([]);

vlb = [-2 -2];      %Lower bound of each gene - all variables
vub = [2 2];        %Upper bound of each gene - all variables
bits =[20    20];   %number of bits describing each gene - all variables

[x,fbest,stats,nfit,fgen,lgen,lfit]= GA550('GAfunc',[ ...
    ],options,vlb,vub,bits);

l=sum(bits);                % length of chromosome
Npop=4*l                    % total Population size
Mut_rate= (l+1)/(2*Npop*l) % probability for mutation
Ngen=size(stats,1) - 1        % number of generations ran by GA
nfit                        % number of fitness evaluations
x                           % optimal design found by GA
fbest                       % fitness value for optimal solution
```

1) Effect of random number generator:
Summary table of the runs carried out:

bits		Pop Size	Mut Rate	Num Gen	Fun Eval	x*	f(x*)
10	Run 1	80	0.0066	33	2720	0.0020 -0.9990	3.001
10	Run 2	80	0.0066	34	2800	0.0020 -0.9990	3.0010
10	Run 3	80	0.0066	32	2640	-0.2366 -0.7879	25.0805

As can be observed from the above table, for same parameters ie Population Size and Mutation Rate the number of generations and the number of function evaluations can be different and the optimal solution found might also be different.

2) Effect of change of Population Size and Mutation Rate:

Summary table for the runs carried out:

bits		Pop Size	Mut Rate	Num Gen	Fun Eval	x*	f(x*)
5	Run 1	40	0.0138	19	800	0.0645 -0.9677	4.0082
5	Run 2	40	0.0138	16	600	0.0645 -0.9677	4.0082
5	Run 3	40	0.0138	17	720	0.0645 -0.9677	4.0082
20	Run 1	160	0.0032	61	9920	0.0000 -1.0000	3.0000
20	Run 2	160	0.0032	60	9760	0.0000 -1.0000	3.0000
20	Run 3	160	0.0032	84	13600	0.0000 -1.0000	3.0000

From the above table we can observe that as the bit sizes per variable increases, the Population Size increases and the mutation rate decreases (when set to values as per the guidelines given in class and not as hard coded).

We can also notice that with a larger bit size, the optimal solution found looks similar to the global optimum (in this case for 20 bits we get the exact solution), but this comes at a cost of much higher function evaluations thus increasing the run-time of the algorithm.

IV Combinatorial Problem with GA

For this problem , we are required to solve a combinatorial problem (with discrete and continuous variables) using GA. The aim of the problem is to minimize the weight of the Truss by varying the cross-sectional area and material choice while avoiding yielding of any member.

For this problem I am again using the in-class provided code `GA550.m` and for constraints `stressHW3.m` without making any changes to either one of them.

For this problem, my design variable is as follows:

$$\vec{x} = \begin{bmatrix} material for truss1 \\ material for truss2 \\ material for truss3 \\ A1 \\ A2 \\ A3 \end{bmatrix}$$

Where the material values will always be an integer in the range 1-4 and i have assigned their numbers in the same manner as described in the homework prompt.

Note: I will be using mm^2 as my units for the cross sectional areas A1, A2, A3 in the design variable. Rest of the internal computation (for material properties and stresses) will take place in SI units.

1) Parameters chosen for the simulation:

Lower bounds for area variables $(LB) = 100\ [mm^2]$
Upper Bounds for area variables $(UB) = 1000\ [mm^2]$
Number of bits for area variables $(b) = 10$
Resolution for area variables $= (UB - LB)/(2^b - 1) = 0.8798\ [mm^2]$
Length of chromosome (l) = sum of bits = 36
Population Size $(N_{pop}) = 4 * l = 144$
Mutation Rate $(P_m) = (l + 1)/(2N_{pop}l) = 0.0036$

Comment on Bounds: The lower bound can be set to even smaller value which will result in a smaller total mass, but then such a value of cross sectional area for a truss will not be easily available (also can be very thin to even manufacture). For example when I set the lower bound to 1 $[mm^2]$, I get the final optimal solution as 1 $[mm^2]$ as cross-sectional area for Truss-1. I have therefore assumed that the smallest truss size will a square cross-section of side 10mm (i.e. 100 $[mm^2]$ in area)

2) Fitness Function

For developing the fitness function we need to account for constraints on the stresses for each truss into the objective function. I have scaled the constraints to an order of 1 as follows (this also makes my constraints unitless):

$$g_i(x) = |\sigma_i|/\sigma_i^{Yield} - 1 \leq 0 \qquad \forall i = 1, 2, 3$$

Now, as all my constraints are of the order of 1. For every constraint i have incorporated a constraint scaling parameters c_j.
Then I am incorporating the constraints into the objective function using Exterior Step-Linear penalty with a penalty multiplier r_p.

Comment on Penalty Multipliers: For this problem I tried out several penalty multipliers (r_p) and scaling factors (c_j) but it was observed that the constraint value for Truss-1 always remained closer to 1 and for Truss-2 always close to 0. This would indicate that I should lower the c_1 and increase c_2 but as it turned out with several combinations that it did not help. So the final values chosen were $c_j = \begin{bmatrix} 1 & 100000 & 100 \end{bmatrix}$ and $r_p = 1000$.
The code for objective function is:

```
function f = GAfunc(x)
% Objective function for AAE550 : HW 3  Problem 4
% Truss Optimization: Minimize Weight
% Design variables: x(1:3): Material: type =discrete
%                   x(4:6): cross sectional area in mm^2: type= ...
    continuous
% Created By Rahul Deshmukh
% email : deshmuk5@purdue.edu
% PUID: 0030004932

A = 1E-6*x(4:6); % converted cross sectional areas to [m^2]

% set Geometric entities
% Lengths and Angles of Bars
L(1) = (5*cosd(75)-3.3*sind(75))/(cosd(124));   % length of bar 1 [m]
theta2 = atand(L(1)*sind(49)/(L(1)*cosd(49)-3))+180;
L(2) = L(1)*sind(49)/sind(theta2);               % length of bar 2 [m]
L(3) = 3.3/cosd(75)-L(1)*sind(49)/cosd(75);   % length of bar 3 [m]

% set Material properties based on x(1:3)
rho=[];
E=[];
sigma_y=[];
for i=1:3
    if x(i) == 1
        % Aluminum
        E = [E, 68.9E9]; % [Pa]
        rho = [rho,2700]; % [km/m^3]
        sigma_y = [sigma_y,55.2E6]; % [Pa]
    elseif x(i) == 2
        % Titanium
```

```
        E = [E,116E9];    % [Pa]
        rho = [rho,4500];    % [kg/m^3]
        sigma_y = [sigma_y,140E6];    % [Pa]
    elseif x(i) == 3
        % Steel
        E = [E,205E9];    % [Pa]
        rho = [rho,7872];    % [kg/m^3]
        sigma_y = [sigma_y,285E6];    % [Pa]
    elseif x(i)==4
        % Nickel
        E = [E,207E9];    % [Pa]
        rho = [rho,8800];    % [kg/m^3]
        sigma_y = [sigma_y,59E6];    % [Pa]
    end
end

% Total Mass
mass=sum(rho.*A.*L);

% account for constraints using penalty approach
sigma = stressHW3(A,E); % row vector
g = abs(sigma)./sigma_y-1;

cj= [1 100000 100];
% using Exterior Quadratic
% phi= sum(cj.*(max(0,g).^2));

% using Exterior Linear
% phi= sum(cj.*max(0,g));

% using Exterior step-Linear
Pj=zeros(length(g),1);
for j=1:length(g)
    if g(j)≤0
        Pj(j)=0;
    else
        Pj(j)=cj(j)*(1+g(j));
    end
end
phi= sum(Pj);

rp=1000;
f= mass +rp*phi;
end
```

The code for Call to GA is:

```
clc;
close all;
clear all;

options = goptions([]);

%lb ub for area in [mm^2]
lb= 100;
```

```
ub= 1000;

vlb = [1 1 1 lb*ones(1,3)]; %Lower bound of each gene - all variables
vub = [4 4 4 ub*ones(1,3)]; %Upper bound of each gene - all variables
bits =[2 2 2 10 10 10]; %number of bits describing each gene - all ...
    variables


[x,fbest,stats,nfit,fgen,lgen,lfit]= GA550('GAfunc',[ ...
    ],options,vlb,vub,bits);

format short;
x
A = 1E-6*x(4:6); % converted cross sectional areas to [m^2]

% set Geometric entities
% Lengths and Angles of Bars
L(1) = (5*cosd(75)-3.3*sind(75))/(cosd(124));   % length of bar 1 [m]
theta2 = atand(L(1)*sind(49)/(L(1)*cosd(49)-3))+180;
L(2) = L(1)*sind(49)/sind(theta2);               % length of bar 2 [m]
L(3) = 3.3/cosd(75)-L(1)*sind(49)/cosd(75);   % length of bar 3 [m]

% set Material properties based on x(1:3)
rho=[];
E=[];
sigma_y=[];
for i=1:3
    if x(i) == 1
        % Aluminum
        E = [E, 68.9E9]; % [Pa]
        rho = [rho,2700]; % [km/m^3]
        sigma_y = [sigma_y,55.2E6]; % [Pa]
    elseif x(i) == 2
        % Titanium
        E = [E,116E9];    % [Pa]
        rho = [rho,4500];    % [kg/m^3]
        sigma_y = [sigma_y,140E6];    % [Pa]
    elseif x(i) == 3
        % Steel
        E = [E,205E9];    % [Pa]
        rho = [rho,7872];    % [kg/m^3]
        sigma_y = [sigma_y,285E6];    % [Pa]
    elseif x(i)==4
        % Nickel
        E = [E,207E9];    % [Pa]
        rho = [rho,8800];    % [kg/m^3]
        sigma_y = [sigma_y,59E6];    % [Pa]
    end
end

% resolution
resolution=(vub-vlb)./(2.^bits-1)

%chromosome length
lchrom=sum(bits)
Npop=4*lchrom
```

```
Pm=(lchrom+1)/(2*Npop*lchrom)

% Total Mass
mass=sum(rho.*A.*L) % in KG

% account for constraints using penalty approach
sigma = stressHW3(A,E) %  [PA]
g = abs(sigma)./sigma_y-1
g≤0

% figure(1)
% hold on;
% plot(stats(:,1),stats(:,2),'-or');
% plot(stats(:,1),stats(:,3),'-sg');
% plot(stats(:,1),stats(:,4),'-^b');
% xlabel('generations');
% ylabel('fitness');
% legend('min','av','max');
```

3) GA solutions:

Summary Table for runs carried out:

	Num Gen	Fun Eval	x*[-,-,-,mm^2,mm^2,mm^2]	Objective(x*)[Kg]	$g_i(x*)$	f(x*)
Run 1	47	6912	[1,3,1,100,421.9941,121.1144]	10.7293	[-0.9518,-0.0064,-0.0084]	10.7293
Run 2	42	6192	[1,3,1,100.8798,421.9941,121.1144]	10.7373	[-0.9519,-0.0064,-0.0082]	10.7373
Run 3	51	7488	[1,3,1,100,423.7537,121.1144]	10.7663	[-0.9539,-0.0105,-0.0098]	10.7663

We can note that for the three runs, the optimal solution found is different and so is the Objective function values.

4) Summary:

The best solution(minimum objective value) out of the three runs carried out is as follows:

$$x* = \begin{bmatrix} 1 \\ 3 \\ 1 \\ 100[mm^2] \\ 421.9941[mm^2] \\ 121.1144[mm^2] \end{bmatrix}$$

Objective function value at x*= 10.7293 [Kg]

Fitness Value at x* = 10.7293

$$\text{Constraints } g_j(x*) = \begin{bmatrix} -0.9518 \\ -0.0064 \\ -0.0084 \end{bmatrix}$$

This means that constraints g_2 and g_3 are nearly active.

Which means that we have the following material assignment for the trusses:
Truss-1 : Aluminum ($\sigma_y = 55.2$ [MPa])
Truss-2 : Steel ($\sigma_y = 285$ [MPa])
Truss-3 : Aluminum ($\sigma_y = 55.2$ [MPa])

And the stresses coming onto these trusses are:
Truss-1 : 2.55 [MPa]
Truss-2 : 282.01 [MPa]
Truss-3 : 54.66 [MPa]

This means that for our optimal solution the Truss-2 & 3 is bordering on its yield strength and whereas Truss-1 is nowhere close to yielding.

This indicates that for the current truss configuration Truss-1 is not contributing significantly in taking up the load. Therefore, we need to change the position of Truss-1 in such a way that it can take more load or we can also change the position of Truss-2 or 3 in such a way that they do not take up the whole load.

For example, If for the given problem we retain the given dimensions (3,5,3.3) but include the angles θ_1 and θ_3 into our design variables. Then we can use the same code with minor changes to get a more optimized configuration.

Now my design variables becomes:

$$\vec{x} = \begin{bmatrix} material for truss1 \\ material for truss2 \\ material for truss3 \\ A1 \\ A2 \\ A3 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$

I have chosen bounds for angles as $30 \leq \theta_1 \leq 60$ and $30 \leq \theta_1 \leq 85$ with 10 bits for each of them.

The matlab code for calling this new GA problem will be:

```
clc;
close all;
clear all;

options = goptions([]);

%lb ub for area in [mm^2]
lb= 100;
ub= 1000;

vlb = [1 1 1 lb*ones(1,3) 30 30];   %Lower bound of each gene - all ...
    variables
vub = [4 4 4 ub*ones(1,3) 60 85];   %Upper bound of each gene - all ...
    variables
bits =[2 2 2 10 10 10 10 10];   %number of bits describing each gene ...
    - all variables


[x,fbest,stats,nfit,fgen,lgen,lfit]= GA550('GAfunc',[ ...
    ],options,vlb,vub,bits);

format short;
x
A = 1E-6*x(4:6); % converted cross sectional areas to [m^2]

% set Geometric entities
% Lengths and Angles of Bars
L(1) = (5*cosd(x(end))-3.3*sind(x(end)))/(cosd(sum(x(end-1:end)))); % ...
    length of bar 1 [m]
theta2 = atand(L(1)*sind(x(end-1))/(L(1)*cosd(x(end-1))-3))+180;
L(2) = L(1)*sind(x(end-1))/sind(theta2);                % length of ...
    bar 2 [m]
L(3) = 3.3/cosd(x(end))-L(1)*sind(x(end-1))/cosd(x(end));   % length ...
    of bar 3 [m]

theta1 = x(end-1);
theta3 = x(end);

```

```
% set Material properties based on x(1:3)
rho=[];
E=[];
sigma_y=[];
for i=1:3
    if x(i) == 1
        % Aluminum
        E = [E, 68.9E9]; % [Pa]
        rho = [rho,2700]; % [km/m^3]
        sigma_y = [sigma_y,55.2E6]; % [Pa]
    elseif x(i) == 2
        % Titanium
        E = [E,116E9];    % [Pa]
        rho = [rho,4500];    % [kg/m^3]
        sigma_y = [sigma_y,140E6];    % [Pa]
    elseif x(i) == 3
        % Steel
        E = [E,205E9];    % [Pa]
        rho = [rho,7872];    % [kg/m^3]
        sigma_y = [sigma_y,285E6];    % [Pa]
    elseif x(i)==4
        % Nickel
        E = [E,207E9];    % [Pa]
        rho = [rho,8800];    % [kg/m^3]
        sigma_y = [sigma_y,59E6];    % [Pa]
    end
end

% resolution
resolution=(vub-vlb)./(2.^bits-1);

%chromosome length
lchrom=sum(bits);
Npop=4*lchrom;
Pm=(lchrom+1)/(2*Npop*lchrom);

% Total Mass
mass=sum(rho.*A.*L) % in KG

% account for constraints using penalty approach
sigma = stressHW3(A,E,x(end-1:end)) %  [PA]
g = abs(sigma)./sigma_y-1
g≤0

figure(2);
hold on;
plot([0, -L(3)*sind(theta3)],[-3.3,-3.3+L(3)*cosd(theta3)]) %truss-3
plot([-5+3, -5+3-L(2)*cosd(180-theta2)],[0, -L(2)*sind(180-theta2) ]) ...
    %truss-2
plot([-5, -5+3-L(2)*cosd(180-theta2)],[0, -L(2)*sind(180-theta2) ]) ...
    %truss-1
```

The Matlab code for the new objective function becomes:

```
function f = GAfunc(x)
```

```
% Objective function for AAE550 : HW 3  Problem 4
% Truss Optimization: Minimize Weight
% Design variables: x(1:3): Material: type =discrete
%                   x(4:6): cross sectional area in mm^2: type= ...
    continuous
% Created By Rahul Deshmukh
% email : deshmuk5@purdue.edu
% PUID: 0030004932

A = 1E-6*x(4:6); % converted cross sectional areas to [m^2]

% set Geometric entities
% Lengths and Angles of Bars
L(1) = (5*cosd(x(end))-3.3*sind(x(end)))/(cosd(sum(x(end-1:end)))); % ...
    length of bar 1 [m]
theta2 = atand(L(1)*sind(x(end-1))/(L(1)*cosd(x(end-1))-3))+180;
L(2) = L(1)*sind(x(end-1))/sind(theta2);                % length of ...
    bar 2 [m]
L(3) = 3.3/cosd(x(end))-L(1)*sind(x(end-1))/cosd(x(end));   % length ...
    of bar 3 [m]

% set Material properties based on x(1:3)
rho=[];
E=[];
sigma_y=[];
for i=1:3
    if x(i) == 1
        % Aluminum
        E = [E, 68.9E9]; % [Pa]
        rho = [rho,2700]; % [km/m^3]
        sigma_y = [sigma_y,55.2E6]; % [Pa]
    elseif x(i) == 2
        % Titanium
        E = [E,116E9];    % [Pa]
        rho = [rho,4500];    % [kg/m^3]
        sigma_y = [sigma_y,140E6];    % [Pa]
    elseif x(i) == 3
        % Steel
        E = [E,205E9];    % [Pa]
        rho = [rho,7872];    % [kg/m^3]
        sigma_y = [sigma_y,285E6];    % [Pa]
    elseif x(i)==4
        % Nickel
        E = [E,207E9];    % [Pa]
        rho = [rho,8800];    % [kg/m^3]
        sigma_y = [sigma_y,59E6];    % [Pa]
    end
end

% Total Mass
mass=sum(rho.*A.*L);

% account for constraints using penalty approach
sigma = stressHW3(A,E,x(end-1:end)); % row vector
g = abs(sigma)./sigma_y-1;

```

```
% cj= [1 1000 100];
cj= [1 1 10000];
% using Exterior Quadratic
% phi= sum(cj.*(max(0,g).^2));

% using Exterior Linear
% phi= sum(cj.*max(0,g));

% using Exterior step-Linear
Pj=zeros(length(g),1);
for j=1:length(g)
    if g(j)≤0
        Pj(j)=0;
    else
        Pj(j)=cj(j)*(1+g(j));
    end
end
phi= sum(Pj);

rp=10E2;
f= mass +rp*phi;
end
```

And the New function for calculating stress becomes:

```
function sigma = stressHW3(A,E,x)
% This function assembles the stiffness matrix and computes the ...
    stress in
% each element of the three-bar truss in AAE 550, HW 3, part II, Fall ...
    2017.
%
% The function returns a three-element vector "sigma"; each element ...
    is the computed
% stress in each truss element.  The input is the three-element ...
    vector "A";
% each element is the cross-sectional area of each truss element.  Values
% for Young's modulus are input as parameters

% fixed values
P = 120E3;                      % applied load [N]

% Lengths and Angles of Bars
L(1) = (5*cosd(x(end))-3.3*sind(x(end)))/(cosd(sum(x)));    % length ...
    of bar 1 [m]
theta2 = atand(L(1)*sind(x(end-1))/(L(1)*cosd(x(end-1))-3))+180;

L(2) = L(1)*sind(x(end-1))/sind(theta2);                % length of ...
    bar 2 [m]
L(3) = 3.3/cosd(x(end))-L(1)*sind(x(end-1))/cosd(x(end));   % length ...
    of bar 3 [m]

theta1 = x(end-1);
theta3 = x(end);

% local stiffness matrices
```

```
K1 = [cosd(-theta1) sind(-theta1)]'*(E(1)*A(1)/L(1))*[cosd(-theta1) ...
    sind(-theta1)];
K2 = [cosd(-theta2) sind(-theta2)]'*(E(2)*A(2)/L(2))*[cosd(-theta2) ...
    sind(-theta2)];
K3 = [cosd(theta3+90) ...
    sind(theta3+90)]'*(E(3)*A(3)/L(3))*[cosd(theta3+90) sind(theta3+90)];

% global (total) stiffness matrix:
K = K1 + K2 + K3;

% load vector (note lower case to distinguish from P)
theta4 = -110;
p = P*[cosd(theta4) sind(theta4)]';

% compute displacements (u(1) = x-displacement on figure; u(2) =
% y-displacement on figure)
u = K \ p;

% change in element length under load
DL(1) = sqrt((-L(1)*cosd(-theta1)-u(1))^2 + ...
    (-L(1)*sind(-theta1)-u(2))^2) - L(1);
DL(2) = sqrt((-L(2)*cosd(-theta2)-u(1))^2 + ...
    (-L(2)*sind(-theta2)-u(2))^2) - L(2);
DL(3) = sqrt((-L(3)*cosd(theta3+90)-u(1))^2 + ...
    (-L(3)*sind(theta3+90)-u(2))^2) - L(3);

% stress in each element
sigma = E .* DL ./ L;
```

On Running this new problem, The final Optimal solution comes as:

$$x* = \begin{bmatrix} 2 \\ 3 \\ 1 \\ 100 \\ 406.1584 \\ 100 \\ 45.78 \\ 66.94 \end{bmatrix}$$

$mass(x*) = 9.3788$ [Kg]

$$g_j(x*) = \begin{bmatrix} -0.0832 \\ -0.0039 \\ -0.2622 \end{bmatrix}$$

Which translates to following material assignment:
Truss-1: Titanium ($\sigma_y = 140$ [MPa])
Truss-2: Steel ($\sigma_y = 285$ [MPa])
Truss-3: Aluminum ($\sigma_y = 55.2$ [MPa])

The angles are:
$\theta_1 = 45.78$ [deg]
$\theta_2 = 66.94$ [deg]

The new config will look something like this:

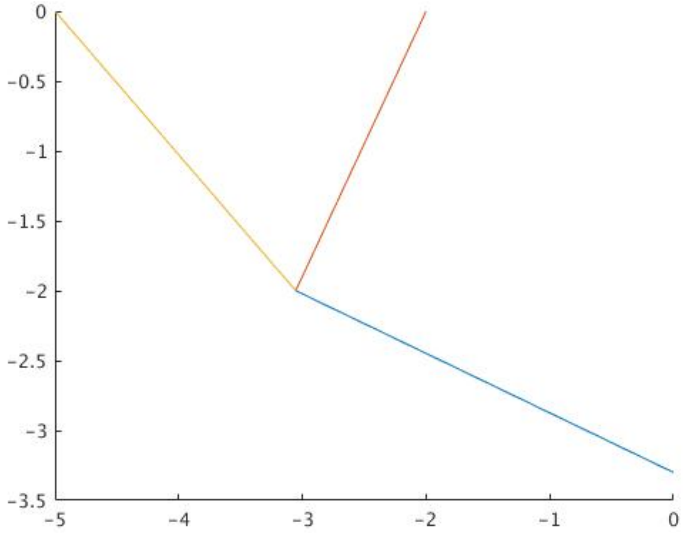


Figure 4: New Truss Configuration (Truss-1: Yellow, Truss-2: Red, Truss-3: blue)

And stresses coming onto them:

Truss-1: 128.35 [MPa]
Truss-2: 283.9 [MPa]
Truss-3: -40.73 [Mpa] (Compressive)

Clearly we have found a better solution with lower total mass and better load sharing by all three trusses.

Furthermore, when formulating the constraint equations we could incorporate known safety factor so that our trusses never reach their yield point for the optimal solution. Meaning formulate them as :

$g_j(x) = |\sigma_j|/(\eta_j \sigma_j^y) - 1$

where η_j will be the safety factor for j^{th} truss.

V Appendix:

Code for generating of gray-coded generation for Genetic Algorithm for the script `GA550.m`:

```
% Created by Rahul Deshmukh
% PUID: 0030004932
% email: deshmuk5@purdue.edu

function gen=b10to2(b10,bits)
% Function for generating gray code from an integer valued matrix b10 with
% bits as bit length per var
% Input: b10: Npop x 2 matrix of integer elements
%        bits: array of length Nvar with information of bit
%              length per design variable
% Output: gen: Npop x sum(bits) matrix of gray coded
%              generation,format: not string but numbers 0 and 1s
gen=[];
for i=1:size(b10,1)
    igen=[];
    for j=1:size(b10,2)
       bin=de2bi(b10(i,j),bits(j)); % first convert to binary
       gray=bi2gray(bin); % convert to gray-code
       igen=[igen,gray];
    end
    gen=[gen;igen];
end
end


function gray= bi2gray(bin)
% function for converting a binary array to gray code
% Input: bin: a binary array of 1 and 0 of length bits(i)
% Output: gray: gray coded equivalent of the binary word,
%               format:array of 1 and 0
bits=length(bin);
gray=[bin(1)];
for i=2:length(bin)
    gray=[gray,(xor(bin(i-1),bin(i)))];
end
end
```