AAE 550 Homework Assignment #3

Alex Hagen

November 30, 2016

1 Nelder-Mead Simplex

1.1 Objective Function Formulation

$$\text{minimize} \quad f(x) = \sum_{i=1}^{d} \left[-x_i \sin\left(\sqrt{|x_i|} \right) \right] + \alpha d$$

where $\alpha = 418.982887$ and $-512 \leq x_i \leq 512$, and $i = 1, \ldots, d$.

First, we must convert the bounds to $g_j(\vec{x}) \leq 0$, so we get

$$\vec{g}(\vec{x}) = \begin{cases} -512 - x_i & \leq 0 \\ x_i - 512 & \leq 0 \end{cases}$$

Then, we have to include these as an exterior penalty function

$$P(\vec{x}) = \sum_{j=1}^{m} \{\max[0,\, g_j(\vec{x})]\}^2$$

and create our pseudo objective function

$$\phi(\vec{x}) = f(\vec{x}) + r_p P(\vec{x})$$

and so, to combine these, we get

$$\phi(\vec{x}) = \sum_{i=1}^{2} \left[-x_i \sin\left(\sqrt{|x_i|} \right) + r_p \left(\max[0,\, -512 - x_i] \right)^2 + r_p \left(\max[0,\, x_i - 512] \right)^2 \right] + \alpha 2$$

and we are asked to choose a large r_p, so we will choose $r_p = 10$.

1.2 Solution from $\mathbf{x}^0 = \begin{bmatrix} 100 & 400 \end{bmatrix}^T$

The Nelder-Mead Simplex for this starting point was called using the source code in listing 1, 2, and 3 on the following page with results in table 1 on the next page.

```
% Alpha from the problem statement
a = 418.982887;
% Bounds from the problem statement
lb = -512;
ub = 512;
% Dimensionality from the problem statement
d = 2;
% Penalty Multiplier
rp = 10;
```

Listing 1: Header Parameters for Egg Crate Function

```
function phi = NMfunc(x)
% objective function: egg-crate function homework assignment
% linear exterior penalty to enforce bounds
hmwk3_header
f = 0.0;
```

```
for i = 1:d
    f = f + (-x(i) * sin(sqrt(abs(x(i)))));
end
f = f + a * d;
g = zeros(1, 2*d);
for i = 1:d
    g(2*i-1) = x(i) / (lb) - 1;  % enforces lower bound
    g(2*i) = x(i) / (ub) - 1; % enforces upper bound
end
P = 0.0;     % initialize penalty function
for i = 1:2*d
    P = P + rp * max(0,g(i));
end
phi = f + P;
```

Listing 2: Nelder-Mead Function and Penalties

```
% this file provides input for and calls fminsearch to use the Nelder-Mead
% Simplex method
% Modified last on 11/28/16 by Alex Hagen.

close all;
clear all;

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

x0 = [100; 400];

[x,fval,exitflag,output] = fminsearch('NMfunc',x0,options);
table(1, :) = {'Run 1', x0, output.funcCount, x, fval, exitflag};
x0 = x;
[x,fval,exitflag,output] = fminsearch('NMfunc',x0,options);
table(2, :) = {'Re-start', x0, output.funcCount, x, fval, exitflag};

format short
header = {'', '$\vec{x}^{0}$', '\code{feval}', '$\vec{x}^{*}$', ...
          '$f\left( \vec{x}^{*} \right)$', 'exitflag'};
matrix2lyx(table,'prob1_2.lyx',header);
```

Listing 3: Source for calling Nelder Mead Simplex with starting point of 100, 400

Table 1: Optimization summary for Nelder-Mead Simplex using starting point $\vec{x}^0 = \begin{bmatrix} 100 \\ 400 \end{bmatrix}$

	$\vec{x}^0$		$\vec{x}^*$	$f(\vec{x}^*)$	exitflag
Run 1	$\begin{bmatrix} 100 \\ 400 \end{bmatrix}$	91	$\begin{bmatrix} 65.5479 \\ 420.969 \end{bmatrix}$	355.348	1
Re-start	$\begin{bmatrix} 65.5479 \\ 420.969 \end{bmatrix}$	78	$\begin{bmatrix} 65.5479 \\ 420.969 \end{bmatrix}$	355.348	1

1.3 Solution from $\mathbf{x}^0 = \begin{bmatrix} 300 & 300 \end{bmatrix}^T$

The Nelder-Mead Simplex for this starting point was called using the source code in listing 4 with results in table 2 on the next page.

```
% this file provides input for and calls fminsearch to use the Nelder-Mead
% Simplex method
% Modified last on 11/28/16 by Alex Hagen.

close all;
clear all;

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

x0 = [300; 300];
```

```
[x,fval,exitflag,output] = fminsearch('NMfunc',x0,options);
table(1, :) = {'Run 1', x0, output.funcCount, x, fval, exitflag};
x0 = x;
[x,fval,exitflag,output] = fminsearch('NMfunc',x0,options);
table(2, :) = {'Re-start', x0, output.funcCount, x, fval, exitflag};

format short
header = {'', '$\vec{x}^{0}$', '\code{feval}', '$\vec{x}^{*}$', ...
          '$f\left( \vec{x}^{*} \right)$', 'exitflag'};
matrix2lyx(table,'prob1_3.lyx',header);
```

Listing 4: Source for calling Nelder Mead Simplex with starting point of 300, 300

Table 2: Optimization summary for Nelder-Mead Simplex using starting point $\vec{x}^0 = \begin{bmatrix} 300 \\ 300 \end{bmatrix}$

	$\vec{x}^0$		$\vec{x}^*$	$f(\vec{x}^*)$	exitflag
Run 1	$\begin{bmatrix} 300 \\ 300 \end{bmatrix}$	108	$\begin{bmatrix} 420.969 \\ 420.969 \end{bmatrix}$	-5.44761e-07	1
Re-start	$\begin{bmatrix} 420.969 \\ 420.969 \end{bmatrix}$	77	$\begin{bmatrix} 420.969 \\ 420.969 \end{bmatrix}$	-5.44761e-07	1

1.4 Solution from $\mathbf{x}^0 = \begin{bmatrix} 520 & 520 \end{bmatrix}^T$

The Nelder-Mead Simplex for this starting point was called using the source code in listing 5 with results in table 3.

```
% this file provides input for and calls fminsearch to use the Nelder-Mead
% Simplex method
% Modified last on 11/28/16 by Alex Hagen.

close all;
clear all;

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

x0 = [520; 520];

[x,fval,exitflag,output] = fminsearch('NMfunc',x0,options);
table(1, :) = {'Run 1', x0, output.funcCount, x, fval, exitflag};
x0 = x;
[x,fval,exitflag,output] = fminsearch('NMfunc',x0,options);
table(2, :) = {'Re-start', x0, output.funcCount, x, fval, exitflag};

format short
header = {'', '$\vec{x}^{0}$', '\code{feval}', '$\vec{x}^{*}$', ...
          '$f\left( \vec{x}^{*} \right)$', 'exitflag'};
matrix2lyx(table,'prob1_4.lyx',header);
```

Listing 5: Source for calling Nelder Mead Simplex with starting point of 520, 520

Table 3: Optimization summary for Nelder-Mead Simplex using starting point $\vec{x}^0 = \begin{bmatrix} 520 \\ 520 \end{bmatrix}$

	$\vec{x}^0$		$\vec{x}^*$	$f(\vec{x}^*)$	exitflag
Run 1	$\begin{bmatrix} 520 \\ 520 \end{bmatrix}$	107	$\begin{bmatrix} 420.969 \\ 420.969 \end{bmatrix}$	-5.4485e-07	1
Re-start	$\begin{bmatrix} 420.969 \\ 420.969 \end{bmatrix}$	76	$\begin{bmatrix} 420.969 \\ 420.969 \end{bmatrix}$	-5.4485e-07	1

1.5 Results and Conclusions

My results do indicate that global optimization is not Nelder-Mead's strong point. While table 2 and 3 show that a minimum close to the global n-dimensional minimum is found, the table 1 on the preceding page shows that for

starting points off of the line $y = x$ (such as $\vec{x} = \begin{bmatrix} 100 \\ 400 \end{bmatrix}$), a global minimum is not achieved. On the line $y = x$, the Nelder Mead Simplex does a good job of finding a non-smooth optimization, as it goes through multiple "troughs" and still arrives at the minimum (which is on that line).

2 Simulated Annealing

2.1 Solution from $\mathbf{x}^0 = \begin{bmatrix} 100 & 400 \end{bmatrix}^T$

I modified SA_550.m to include the temperature output (simply by putting T in the output vector), and then used the listings 6 and 7 to call it and write values to a table. The output of the random number generator can be seen in the table 4 on the next page.

```
function f = SAfunc(x)
    hmwk3_header;
    f = 0.0;
    for i = 1:d
        f = f + (-x(i) * sin(sqrt(abs(x(i)))));
    end
    f = f + a * d;
```

Listing 6: Egg Crate function for calling Simulated Annealing

```
close all;
clear all;

% Bring in the header parameters and set the bounds into an input format
% that SA_550 knows
hmwk3_header
bounds = [lb ub;
    lb ub];
% Set our initial design
x0 = [100; 400];
% Set the default temperature and cooling
T0 = 50;
r0 = 0.5;
% Zero out the options array
options = zeros(1,9);
% Put the temperature and cooling rate in the appropriate places
options(1) = T0;
options(6) = r0;
% Run SA_550 twice and save the values to a cell array
[xstar, fstar, count, ~, ~, T] = SA_550('SAfunc', bounds, x0, options);
table(1, :) = {'Run 1', x0, T0, r0, count, T, xstar, fstar};
[xstar, fstar, count, ~,  ~, T] = SA_550('SAfunc', bounds, x0, options);
table(2, :) = {'Run 2', x0, T0, r0, count, T, xstar, fstar};

% Change the starting point
x0 = [-400; 100];
% Run SA_550 twice again and then save the values to a cell array
[xstar, fstar, count, ~, ~, T] = SA_550('SAfunc', bounds, x0, options);
table(3, :) = {'Run 1', x0, T0, r0, count, T, xstar, fstar};
[xstar, fstar, count, ~,  ~, T] = SA_550('SAfunc', bounds, x0, options);
table(4, :) = {'Run 2', x0, T0, r0, count, T, xstar, fstar};

% Save our results to a latex file so we can use it and update it easily
format short
header = {'', '$\vec{x}^{0}$', '$T_{0}$', '$r_{T}$', '\code{feval}', ...
          '$T$', '$\vec{x}^{*}$', '$f\left( \vec{x}^{*} \right)$'};
matrix2lyx(table,'prob2_1.lyx',header);
```

Listing 7: Source for calling Simulated Annealing for effect of random numbers

Table 4: Effect of starting point and random number generator on Simulated Annealing

	$\vec{x}^0$	T_0	r_T		T	$\vec{x}^*$	$f(\vec{x}^*)$
Run 1	$\begin{bmatrix} 100 \\ 400 \end{bmatrix}$	50	0.5	25601	2.32831e-08	$\begin{bmatrix} 203.814 \\ 420.969 \end{bmatrix}$	217.14
Run 2	$\begin{bmatrix} 100 \\ 400 \end{bmatrix}$	50	0.5	23201	1.86265e-07	$\begin{bmatrix} -124.829 \\ 420.969 \end{bmatrix}$	296.107
Run 1	$\begin{bmatrix} -400 \\ 100 \end{bmatrix}$	50	0.5	25601	2.32831e-08	$\begin{bmatrix} -512 \\ 203.814 \end{bmatrix}$	331.893
Run 2	$\begin{bmatrix} -400 \\ 100 \end{bmatrix}$	50	0.5	24001	9.31323e-08	$\begin{bmatrix} -302.525 \\ 420.969 \end{bmatrix}$	118.438

2.2 Solution from $\mathbf{x}^0 = \begin{bmatrix} 100 & 400 \end{bmatrix}^T$ at $T_0 = 50$, 85, or 15

I called the source code in listing 8 to get the results in table 5. Even with different temperatures, the solution does not reach near the global minimum, the best being at $T_0 = 85$. Computationally, there is not much expense change in the initial temperatures, with near 23000 function evalutations each.

```
close all;
clear all;

% Bring in the header parameters and set the bounds into an input format
% that SA_550 knows
hmwk3_header
bounds = [lb ub;
          lb ub];
% Set our initial design
x0 = [100; 400];
% Set the cooling
r0 = 0.5;
% Zero out the options array
options = zeros(1,9);
% set a counter for our results array
i = 1;
for T0 = [15, 50, 85]
    % Put the temperature and cooling rate in the appropriate places
    options(1) = T0;
    options(6) = r0;
    % Run SA_550 save the values to a cell array
    [xstar, fstar, count, ~, ~, T] = SA_550('SAfunc', bounds, x0, options);
    table(i, :) = {'Run 1', x0, T0, r0, count, T, xstar, fstar};
    % increment the table
    i = i + 1;
end

% Save our results to a latex file so we can use it and update it easily
format short
header = {'', '$\vec{x}^{0}$', '$T_{0}$', '$r_{T}$', '\code{feval}', ...
          '$T$', '$\vec{x}^{*}$', '$f\left( \vec{x}^{*} \right)$'};
matrix2lyx(table,'prob2_2.lyx',header);
```

Listing 8: Source for calling Simulated Annealing for effect of initial temperature

Table 5: Effect of initial temperature on Simulated Annealing

	$\vec{x}^0$	T_0	r_T		T	$\vec{x}^*$	$f(\vec{x}^*)$
Run 1	$\begin{bmatrix} 100 \\ 400 \end{bmatrix}$	15	0.5	24001	2.79397e-08	$\begin{bmatrix} 65.5479 \\ 420.969 \end{bmatrix}$	355.348
Run 1	$\begin{bmatrix} 100 \\ 400 \end{bmatrix}$	50	0.5	22401	3.72529e-07	$\begin{bmatrix} 203.814 \\ 420.969 \end{bmatrix}$	217.14
Run 1	$\begin{bmatrix} 100 \\ 400 \end{bmatrix}$	85	0.5	24801	7.91624e-08	$\begin{bmatrix} -302.525 \\ 420.969 \end{bmatrix}$	118.438

2.3 Best initial temperature

Using the best initial temperature of $T_0 = 85$, I called the source code in listing 9 to get the results in table 6. The computational expense of the algorithm clearly increases with the high cooling rate ($r_T = 0.15$ requires 12001 function evaluations, whereas $r_T = 0.85$ requires 96001, eight times higher). However, the higher cooling rate did give a much better result, finally converging near the n-dimensional minimum.

```
close all;
clear all;

% Bring in the header parameters and set the bounds into an input format
% that SA_550 knows
hmwk3_header
bounds = [lb ub;
          lb ub];
% Set our initial design
x0 = [100; 400];
% Set the cooling
T0 = 85;
% Zero out the options array
options = zeros(1,9);
% set a counter for our results array
i = 1;
for r0 = [0.15, 0.5, 0.85]
    % Put the temperature and cooling rate in the appropriate places
    options(1) = T0;
    options(6) = r0;
    % Run SA_550 save the values to a cell array
    [xstar, fstar, count, ~, ~, T] = SA_550('SAfunc', bounds, x0, options);
    table(i, :) = {'Run 1', x0, T0, r0, count, T, xstar, fstar};
    % increment the table
    i = i + 1;
end

% Save our results to a latex file so we can use it and update it easily
format short
header = {'', '$\vec{x}^{0}$', '$T_{0}$', '$r_{T}$', '\code{feval}', ...
          '$T$', '$\vec{x}^{*}$', '$f\left( \vec{x}^{*} \right)$'};
matrix2lyx(table,'prob2_3.lyx',header);
```

Listing 9: Source for calling Simulated Annealing for effect of cooling rate

Table 6: Effect of cooling rate on Simulated Annealing

	$\vec{x}^0$	T_0	r_T		T	$\vec{x}^*$	$f(\vec{x}^*)$
Run 1	$\begin{bmatrix} 100 \\ 400 \end{bmatrix}$	85	0.15	12001	2.4814e-10	$\begin{bmatrix} -302.525 \\ 420.969 \end{bmatrix}$	118.438
Run 1	$\begin{bmatrix} 100 \\ 400 \end{bmatrix}$	85	0.5	24001	1.58325e-07	$\begin{bmatrix} 203.814 \\ 420.969 \end{bmatrix}$	217.14
Run 1	$\begin{bmatrix} 100 \\ 400 \end{bmatrix}$	85	0.85	96001	3.39056e-07	$\begin{bmatrix} 420.969 \\ 420.969 \end{bmatrix}$	-5.44864e-07

3 Genetic Algorithm

3.1 Solution with 10 bits decoding

The genetic algorithm was called using listing 11, which uses the egg crate function as in listing 10. These results are shown in table 7. The random number generator does make a small difference, especially in the number of evaluations, with some taking longer (up to 5 generations longer) than the others. They do converge to the same value, though.

```
function f = GAfunc(x)

    hmwk3_header;
```

```
    f = 0.0;
    for i = 1:d
        f = f + (-x(i) * sin(sqrt(abs(x(i)))));
    end
    f = f + a * d;
```

Listing 10: Egg Crate function for calling Genetic Algorithm

```
close all;
clear all;


hmwk3_header;
vlb = [lb lb];  %Lower bound of each gene - all variables
vub = [ub ub];  %Upper bound of each gene - all variables
bits =[10 10];  %number of bits describing each gene - all variables
l = sum(bits);
% Finding the population size using criteria given in class
N_pop = 4 * l;
% Finding the mutation rate using criteria given in class
P_m = (l + 1) / (2 * N_pop * l);
% Call default options with a modified population size and mutation
% probability
options = goptions([1, 0.9, 0, 0, 0, 0, 0, 0, 0, 0, N_pop, 0, P_m, 200]);

for i = 1:3
    [x, fbest, stats, nfit, fgen, lgen, lfit]= GA550('GAfunc', [ ], ...
                                                    options, vlb, vub, ...
                                                    bits);
    name = sprintf('Run %d (10 bits)', i);
    table(i, :) = {name, N_pop, P_m, max(stats(:,1)), nfit, x, fbest};
end

% Save our results to a latex file so we can use it and update it easily
format short
header = {'', '$N_{pop}$', '$P_{m}$', '$N_{gen}$', '\code{feval}', ...
          '$\vec{x}^{*}$', '$f\left( \vec{x}^{*} \right)$'};
matrix2lyx(table,'prob3_1.lyx',header);
```

Listing 11: Source for calling Genetic Algorithm for effect of random numbers

Table 7: Effect of random number generator on Genetic Algorithm

	N_{pop}	P_m	N_{gen}		$\vec{x}^*$	$f(\vec{x}^*)$
Run 1 (10 bits)	80	0.0065625	25	2080	$\begin{bmatrix} 420.911 \\ 420.911 \end{bmatrix}$	0.000839652
Run 2 (10 bits)	80	0.0065625	26	2160	$\begin{bmatrix} 420.911 \\ 420.911 \end{bmatrix}$	0.000839652
Run 3 (10 bits)	80	0.0065625	28	2320	$\begin{bmatrix} 420.911 \\ 420.911 \end{bmatrix}$	0.000839652

3.2 Solution with 5 bits and 20 bits

Using listing 12 to run several different bit sizes, with the results listed in table 8, we can see several differences between the runtime and also the results. With more bits, the quality is clearly better, however the runtime is almost 5× higher, whereas the decrease when dropping to 5 bits is over 100%. The 5 bit solution, though, is not the final answer, although it gets close.

```
close all;
clear all;


hmwk3_header;
vlb = [lb lb];  %Lower bound of each gene - all variables
vub = [ub ub];  %Upper bound of each gene - all variables
```

```
bits =[20 20];  %number of bits describing each gene - all variables
l = sum(bits);
% Finding the population size using criteria given in class
N_pop = 4 * l;
% Finding the mutation rate using criteria given in class
P_m = (1 + 1) / (2 * N_pop * l);
% Call default options with a modified population size and mutation
% probability
options = goptions([1, 0.9, 0, 0, 0, 0, 0, 0, 0, 0, N_pop, 0, P_m, 200]);

for i = 1:3
    [x, fbest, stats, nfit, fgen, lgen, lfit]= GA550('GAfunc', [ ], ...
                                                    options, vlb, vub, ...
                                                    bits);
    name = sprintf('Run %d (20 bits)', i);
    table(i, :) = {name, N_pop, P_m, max(stats(:,1)), nfit, x, fbest};
end

bits =[5 5];    %number of bits describing each gene - all variables
l = sum(bits);
% Finding the population size using criteria given in class
N_pop = 4 * l;
% Finding the mutation rate using criteria given in class
P_m = (1 + 1) / (2 * N_pop * l);
% Call default options with a modified population size and mutation
% probability
options = goptions([1, 0.9, 0, 0, 0, 0, 0, 0, 0, 0, N_pop, 0, P_m, 200]);

for i = 4:6
    [x, fbest, stats, nfit, fgen, lgen, lfit]= GA550('GAfunc', [ ], ...
                                                    options, vlb, vub, ...
                                                    bits);
    name = sprintf('Run %d (5 bits)', i - 3);
    table(i, :) = {name, N_pop, P_m, max(stats(:,1)), nfit, x, fbest};
end

% Save our results to a latex file so we can use it and update it easily
format short
header = {'', '$N_{pop}$', '$P_{m}$', '$N_{gen}$', '\code{feval}', ...
          '$\vec{x}^{*}$', '$f\left( \vec{x}^{*} \right)$'};
matrix2lyx(table,'prob3_2.lyx',header);
```

Listing 12: Source for calling Genetic Algorithm for effect of bit size

Table 8: Effect of bit size on Genetic Algorithm

	N_{pop}	P_m	N_{gen}		$\vec{x}^*$	$f(\vec{x}^*)$
Run 1 (20 bits)	160	0.00320312	53	8640	$\begin{bmatrix} 420.968 \\ 420.969 \end{bmatrix}$	-4.02285e-07
Run 2 (20 bits)	160	0.00320312	56	9120	$\begin{bmatrix} 420.969 \\ 420.969 \end{bmatrix}$	-5.43122e-07
Run 3 (20 bits)	160	0.00320312	54	8800	$\begin{bmatrix} 420.969 \\ 420.969 \end{bmatrix}$	-5.43122e-07
Run 1 (5 bits)	40	0.01375	21	880	$\begin{bmatrix} 412.903 \\ 412.903 \end{bmatrix}$	16.3123
Run 2 (5 bits)	40	0.01375	21	880	$\begin{bmatrix} 412.903 \\ 412.903 \end{bmatrix}$	16.3123
Run 3 (5 bits)	40	0.01375	15	640	$\begin{bmatrix} 412.903 \\ 412.903 \end{bmatrix}$	16.3123

4 Combinatorial Problem with Genetic Algorithm

As stated, we would like to minimize the combined mass of the trusses while not allowing yielding. This means that the objective function for minimization is

$$\begin{aligned} f(x) = m &= \rho_1 A_1 L_1 + \rho_2 A_2 L_2 + \rho_3 A_3 L_3 \\ &= \rho_1 w_1 h_1 L_1 + \rho_2 w_2 h_2 L_2 + \rho_3 w_3 h_3 L_3 \end{aligned}$$

and the constraint is simply

$$\vec{g}(\vec{x}) = \begin{cases} \sigma_{y,1} - \sigma_1 & \leq 0 \\ \sigma_{y,2} - \sigma_2 & \leq 0 \\ \sigma_{y,3} - \sigma_3 & \leq 0 \end{cases}$$

and the design variables are the area variables h and w for each truss, and the material choice for each element, which defines ρ, E, and σ_y for each element.

4.1 Population size and Mutation Rate

We will use two bits to encode each of the material choices. I will apply a lower bound on each area variable as 0.0001 m to 0.1 m) and encode them with 12 bits (for a resolution of 2.43896×10^{-5} m). Our input vector will be organized as

$$\vec{x} = \begin{bmatrix} \text{material}_1 \\ \text{material}_2 \\ \text{material}_3 \\ w_1 \\ w_2 \\ w_3 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix} \quad \text{with bit sizes} \quad \text{size}(\vec{x}) = \begin{bmatrix} 2 \\ 2 \\ 2 \\ 12 \\ 12 \\ 12 \\ 12 \\ 12 \\ 12 \end{bmatrix}$$

So, $\lambda = \text{sum}(\text{size}(\vec{x})) = 66$. Thus, our population size will be

$$N_{pop} = 4 \cdot \lambda = 312$$

and our mutation rate will be

$$P_m = \frac{\lambda + 1}{2 N_{pop} \lambda} = 0.0016$$

4.2 Fitness function

Now, we must define a pseudo-objective function using penalty multipliers for the constraints. I will apply the constraints as external penalties.

$$P(\vec{x}) = \sum_{j=1}^{m} \max[0,\, g_j(\vec{x})]$$

and create our pseudo objective function

$$\phi(\vec{x}) = f(\vec{x}) + r_p P(\vec{x})$$

So, the pseudo-objective function for this design will be

$$\phi(\vec{x}) = \rho_1 w_1 h_1 L_1 + \rho_2 w_2 h_2 L_2 + \rho_3 w_3 h_3 L_3 + r_p \left(\max[0,\, \sigma_{y,1} - \sigma_1] + \max[0,\, \sigma_{y,2} - \sigma_2] + \max[0,\, \sigma_{y,3} - \sigma_3]\right)$$

and to start, we will try an $r_p = 10$, so that the constraints will not be violated. Upon experimentation, it was found that a much higher penalty multiplier must be used in order to not violate constraints. An r_p value of 1×10^9 is the lowest value that doesn't allow for constraint violation, and so it was used. The files used for this are enumerated below. Listing 13 on the following page is used to define the parameters of the problem in several places, listing 14 on the next page is used to convert the input vector into the variables used in the problem statement in several places, listing 15 on the following page takes the material choice and outputs the material coefficients, Listing 18 on page 11 calculates the pseudoobjective function using the mass function in listing 16 on the following page, Listing 17 on the next page calculates constraints, and listing 19 on page 11 runs all of the analyses for the problem.

```
% Alpha from the problem statement
L_1 = 3*0.3048;                 % length of element 1 [m]
L_2 = 3*0.3048;                 % length of element 2 [m]
L_3 = 3.625*0.3048;             % length of element 3 [m]
% Bounds from the problem statement
lb = 0.0001;
ub = 0.1;
% Penalty Multiplier
rp = [1, 1, 1]*1e9;
```

Listing 13: Parameter Definitions for Truss Problem

```
w_1 = x(4); w_2 = x(5); w_3 = x(6);
h_1 = x(7); h_2 = x(8); h_3 = x(9);

[rho_1, E_1, sigma_y1] = material(x(1));
[rho_2, E_2, sigma_y2] = material(x(2));
[rho_3, E_3, sigma_y3] = material(x(3));
```

Listing 14: Input vector to variable conversion script

```
function [E, rho, sigma_y] = material(x)
    switch x
        case 1
            E=68.9e9;
            rho = 2700;
            sigma_y = 55.2e6;
        case 2
            E = 116e9;
            rho = 4500;
            sigma_y = 140e6;
        case 3
            E = 205e9;
            rho=7872;
            sigma_y = 285e6;
        case 4
            E = 207e9;
            rho = 8800;
            sigma_y = 59.0e6;
        otherwise
            error('We dont have that material information');
    end
end
```

Listing 15: Material switch case script

```
function f = massfunc(x)

    hmwk3_prob4_header;
    convert_bits;

    f = rho_1 * w_1 * h_1 * L_1 + ...
        rho_2 * w_2 * h_2 * L_2 + ...
        rho_3 * w_3 * h_3 * L_3;

end
```

Listing 16: Function for objective function calculation

```
function g = constraintsfunc(x)
    convert_bits;
    A = [w_1 * h_1, w_2 * h_2, w_3 * h_3];
    E = [E_1 E_2 E_3];
    sigma = stressHW3(A, E);
    sigma_1 = sigma(1);
    sigma_2 = sigma(2);
    sigma_3 = sigma(3);
    g = [ abs(sigma_1)/sigma_y1 - 1, ...
          abs(sigma_2)/sigma_y2 - 1, ...
```

```
        abs(sigma_3)/sigma_y3 - 1];
end
```

Listing 17: Function for calculating constraints

```
function phi = pseudoobjfunc(x)

    hmwk3_prob4_header;
    convert_bits;

    f = massfunc(x);

    g = constraintsfunc(x);

    P = 0.0;
    for i = 1:length(g)
        P = P + rp(i) * max(0, g(i));
    end

    phi = f + P;
end
```

Listing 18: Pseudo Objective Function for calculation of truss problem

```
close all;
clear all;

hmwk3_prob4_header;
vlb = [1 1 1 lb lb lb lb lb lb];     %Lower bound of each gene - all
    variables
vub = [4 4 4 ub ub ub ub ub ub];     %Upper bound of each gene - all
    variables
bits =[2 2 2 12 12 12 12 12 12];     %number of bits describing each gene -
    all variables
lambda = sum(bits);
% Finding the population size using criteria given in class
N_pop = 4 * lambda;
% Finding the mutation rate using criteria given in class
P_m = (lambda + 1) / (2 * N_pop * lambda);
% Call default options with a modified population size and mutation
% probability
options = goptions([1, 0.9, 0, 0, 0, 0, 0, 0, 0, 0, N_pop, 0, P_m, 200]);

for i = 1:3
    [x, fbest, stats, nfit, fgen, lgen, lfit]= GA550('pseudoobjfunc', ...
                                                    [ ], options, vlb, ...
                                                    vub, bits);
    name = sprintf('Run %d', i);
    g = constraintsfunc(x);
    mass = massfunc(x);
    table(i, :) = {name, N_pop, P_m, max(stats(:,1)), nfit, x, mass, ...
                   g, fbest};
end

% Save our results to a latex file so we can use it and update it easily
format short
header = {'', '$N_{pop}$', '$P_{m}$', '$N_{gen}$', '\code{feval}', ...
          '$\vec{x}^{*}$', '$m$', ...
          '$\vec{g}\left( \vec{x}^{*} \right)$', ...
          '$f\left( \vec{x}^{*} \right)$'};
m2lyx(table,'prob4.lyx',header);
```

Listing 19: Script for running truss problem analysis

4.3 Solution with best design

Three runs were performed and their results tabulated in table 9 on the next page.

Table 9: Truss optimization results table (masses - m and $f(\vec{x}^*)$ are given in kg, constraint loads - $g(\vec{x}^*)$ are given in Pa, and the input vector is given in a combination of m and material choice)

	N_{pop}	P_m	N_{gen}		$\vec{x}^*$	m	$\vec{g}(\vec{x}^*)$	$f(\vec{x}^*)$
Run 1	312	0.00162311	105	33072	$\begin{bmatrix} 2 \\ 2 \\ 1 \\ 0.00031956 \\ 0.0262033 \\ 0.020324 \\ 0.000173187 \\ 0.0159571 \\ 0.0204703 \end{bmatrix}$	7.6029e+07	$\begin{bmatrix} -0.000109207 \\ -8.59252e-05 \\ -0.00496509 \end{bmatrix}$	7.6029e+07
Run 2	312	0.00162311	117	36816	$\begin{bmatrix} 2 \\ 2 \\ 2 \\ 0.00249077 \\ 0.0308385 \\ 0.0348881 \\ 0.0126637 \\ 0.0120295 \\ 0.0129809 \end{bmatrix}$	1.00739e+08	$\begin{bmatrix} -0.0231607 \\ -0.0231463 \\ -0.605196 \end{bmatrix}$	1.00739e+08
Run 3	312	0.00162311	95	29952	$\begin{bmatrix} 2 \\ 2 \\ 1 \\ 0.00031956 \\ 0.0286916 \\ 0.018909 \\ 0.000148791 \\ 0.0145178 \\ 0.0221536 \end{bmatrix}$	7.60777e+07	$\begin{bmatrix} -0.000247414 \\ -0.000224289 \\ -0.0117763 \end{bmatrix}$	7.60777e+07

4.4 Conclusions

Table 9 on the preceding page shows that truss #3 is always the closest to the active constraint, however the GA does not always solve it as having the largest area. I would move the base of truss 3 to the negative x direction and extend it's length, so that it could resist the force P at 45^o with more of it's tensile strength, instead of shear strength.