Thomas Satterly
AAE 550, HW 3
11/14/17

Foreword: The provided Simulated Annealing (SA) and Genetic Algorithm (GA) code was modified to accept function handle inputs for the fitness function, rather than an execution string. This was to make the code more portable (don't have to add folders to search path) and executes much quicker than using "eval" statements. Modified code is available on request, but has been omitted from this homework as the modifications were very minor.

I. Nelder-Mead Simplex

The Matlab routing "fminsearch" uses the Nelder-Mead Simplex algorithm. Use the algorithm to solve the Rastrigin function is defined as follows:

$$f(x) = 10d + \sum_{i=1}^{d}(x_i^2 - 10\cos(2\pi x_i))$$

$$-5.12 \le x_i \le 5.12$$

1) Formulate an appropriate objective function for the Nelder-Mead Simplex to account for the bounds of the design variables. Recall that the function and/or slope continuity is not a requirement when developing the penalty function.
2) Solve the problem starting from $x^0 = [0.3\ 0.3]^T$. Be sure to use the first x* as x^0 for a re-start of the algorithm. Summarize in a table.
3) Solve the problem again starting from $x^0=[-2.5\ 0]^T$. As before, use the first x* as x^0 for a re-start of the algorithm and summarize in a table
4) Solve the problem again starting from $x^0=[3.5\ -3.5]^T$. As before, use the first x* as x^0 for a re-start of the algorithm and summarize in a table
5) Recall that the Nelder-Mead Simplex is generally better at non-smooth optimization than it is global optimization. Do the results indicate this?

Solution

1) See attached for solution.
2) The results table is shown below:

	x^0	Function Evaluations	x*	f(x*)
Run 1	[0.3; 0.3]	73	[-4.276e-5; 0.2534e-5]	4.90E-07
Re-Start	[-4.276e-5; 0.2534e-5]	3	[-4.276e-5; 0.2534e-5]	4.90E-07

AAE550 11/15/17 HW#3 Thomas [illegible]

1. Nelder-Mead Simplex

1) Minimize : $f(x) = 10d + \sum_{i=1}^{d}(x_i^2 - 10\cos(2\pi x_i))$, $-5.12 \le x_i \le 5.12$

For this problem, $x = [x_1 \ x_2]^T$

The constraints are as follows:

$x_1 \ge -5.12 \Rightarrow -5.12 - x_1 \le 0 \Rightarrow \frac{x_1}{-5.12} - 1 \le 0 \Rightarrow g_1(x) = \frac{x_1}{-5.12} - 1$

$x_1 \le 5.12 \Rightarrow x_1 - 5.12 \le 0 \Rightarrow \frac{x_1}{5.12} - 1 \le 0 \Rightarrow g_2(x) = \frac{x_1}{5.12} - 1$

$x_2 \ge -5.12 \Rightarrow g_3(x) = \frac{x_2}{-5.12} - 1$

$x_2 \le 5.12 \Rightarrow g_4(x) = \frac{x_2}{5.12} - 1$

For this problem, use a linear exterior penalty method

$P(x) = \sum_{i=1}^{n} \max(0, g_i(x))$

$\phi(x) = f(x) + r_p P(x)$

$f(x) = 20 + [(x_1^2 - 10\cos(2\pi x_1)) + (x_2^2 - 10\cos(2\pi x_2))]$

$P(x) = \sum_{i=1}^{n} \max[0, g_i(x)]$

$g_1(x) = \frac{x_1}{-5.12} - 1$

$g_2(x) = \frac{x_1}{5.12} - 1$

$g_3(x) = \frac{x_2}{-5.12} - 1$

$g_4(x) = \frac{x_2}{-5.12} - 1$

$\phi = f(x) + r_p P(x)$ (Choose r_p)

Minimize ϕ

3) The results table is shown below:

	x^0	Function Evaluations	x*	f(x*)
Run 1	[-2.5; 0]	73	[-4.276e-5, 0.2534e-5]	4.90E-07
Re-Start	[-4.276e-5, 0.2534e-5]	3	[-4.276e-5, 0.2534e-5]	4.90E-07

4) The results table is shown below

	x^0	Function Evaluations	x*	f(x*)
Run 1	[-2.5; 0]	48	[-2.984872; -0.000139]	8.95E+00
Re-Start	[-2.984872; -0.000139]	25	[-2.984872; -0.000139]	8.95E+00

5) A known global minimum for the Rastrigin function is at $x=[0\ 0]^T$. While part (2) manages to find this global minimum, parts (3) and (4) settle upon different local minima, indicating that the Nelder-Mead Simplex is more apt at finding local minima over a non-smooth surface than a global minimum.

```matlab
function phi = Rastrigin(x)
%RASTRIGIN Implemetation of the Rastrigin function, linear external
 penalty
% Thomas Satterly
% AAE 550, HW 3

f = 10 * numel(x);
for i = 1:numel(x)
    f = f + (x(i)^2 - 10 * cos(2 * pi * x(i)));
end

% Penalty function
g(1) = (x(1) / -5.12) - 1;
g(2) = (x(1) / 5.12) - 1;

g(3) = (x(2) / -5.12) - 1;
g(4) = (x(2) / 5.12) - 1;

P = 0;
for i = 1:numel(g)
    P = P + 10 * max(0,g(i));
end
phi = f + P;

end
```

Published with MATLAB® R2016a

```
% Thomas Satterly
% AAE 550, HW 3
% Problem 1

close all;
clear all;

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

x0 = [3.5; -3.5];

[x_star_1,fval_1,exitflag_1,output_1] = fminsearch(@(x)
 aae550.hw3.Rastrigin(x),x0,options);
[x_star_2,fval_2,exitflag_2,output_2] = fminsearch(@(x)
 aae550.hw3.Rastrigin(x),x_star_1,options);
fprintf('Run 1: x0=[%0.6f, %0.6f], numEvals = %d, x* = [%0.6f, %0.6f],
 f(x*) = %0.6f\n', x0(1), x0(2), output_1.funcCount, x_star_1(1),
 x_star_1(2), fval_1);
fprintf('Run 2: x0=[%0.6f, %0.6f], numEvals = %d, x* = [%0.6f, %0.6f],
 f(x*) = %0.6f\n', x_star_1(1), x_star_1(2), output_2.funcCount,
 x_star_2(1), x_star_2(2), fval_2);
```

Published with MATLAB® R2016a

II. Simulated Annealing

Minimize the Rastrigin function as defined in part (I) using the Simulated Annealing (SA) Algorithm..

1) Start from $x^0 = [3\ 3]^T$. Choose a value of T_0 and r_T. Run the SA algorithm again to see the effect of the random number generator. Adjust the maximum number of iterations so it does not effect the resulting answer. Use a different starting point, and solve using the SA algorithm twice again. Summarize the results in a table
2) Run the SA algorithm from one of the x^0 points in part (1), first with a higher initial temperature, than a lower initial temperature. Summarize the results in a table. What impact does temperature have on the quality of the results? What effect does initial temperature have on the computational cost?
3) Choose the best initial temperature from part (2) and run the SA algorithm at two different cooling rates than used in part (1) and (2). Summarize the results in a table. What impact does the cooling rate have on the quality of the result and the computational expense?

Solution

1) The results table is shown below:

	x^0	T_0	r_T	Function Evaluations	Final *T*	x*	*f*(x*)
Run 1	[3; 3]	15	0.85	88801	2.58E-07	3.9897e-7; -9.9313e-7]	2.27E-10
Run 2	[3; 3]	15	0.85	84801	5.82E-07	[9.5862e-7; 7.643598e-8]	1.83E-10
Run 1	[-5, 5]	15	0.85	83201	8.06E-07	[-5.047281e-7; 1.715971e-6]	6.35E-10
Run 2	[-5, 5]	15	0.85	81601	1.12E-06	[-3.09875e-7; 1.943226e-6]	7.7E-10

2) The results table is shown below:

	x^0	T_0	r_T	Function Evaluations	Final *T*	x*	*f*(x*)
Run 1	[3; 3]	50	0.85	92001	4.49E-07	[-1.41496e-6; 4.46457e-7]	4.37E-10
Run 2	[3; 3]	15	0.85	84801	5.82E-07	[9.58620e-7; 7.64358e-8]	1.83E-10
Run 3	[3; 3]	2	0.85	74401	6.42E-07	[-1.42215e-5, 0.99496]	0.995

Based on the results shown above, the higher the initial temperature, the more function evaluations required to generate a solution, meaning a higher computational cost. Also shown is how choosing too low of an initial temperature can prevent the global optimum from being found, as demonstrated by run 3. The higher initial temperature in run 1 produced a very slightly higher optimum value compared to run 2, but this difference is negligible and most likely due to the random nature of the algorithm.

3) The results table is shown below:

	x^0	T_0	r_T	Function Evaluations	Final *T*	x*	$f(x^*)$
Run 1	[3; 3]	15	0.98	663201	8.15E-07	[3.68325e-7; 3.84754e-7]	5.63E-11
Run 2	[3; 3]	15	0.85	84801	5.82E-07	[9.58620e-7; 7.64398e-8]	1.83E-10
Run 3	[3; 3]	15	0.3	15201	5.81E-09	[0.99495; -2.01078e-7]	0.995

As the cooling rate increases, the number of function evaluations increases, with a much greater effect than simply increasing the temperature. However, it also can produce a better quality result. Decreasing the cooling rate too much can prevent the global minimum from being found, as shown in run 3.

```
% Thomas Satterly
% AAE 550, HW 3
% Problem 2

close all;
clear all;

bounds = [-5.12, 5.12;
    -5.12, 5.12]; % upper and lower bounds

X0 = [-5; 5]; % initial design NOTE: this is a column vector

options = zeros(1,9); % set up options array for non-default inputs

options(1) = 15;  % initial temperature (default = 50)
options(6) = 0.3; % cooling rate (default = 0.5)
options(8) = 1e6; % Maximum number of iterations

[xstar,fstar,count,accept,oob]= aae550.hw3.p2.SA_550(@(x)
 aae550.hw3.Rastrigin(x),bounds,X0,options);
fprintf('\n');
disp(xstar);
```

Published with MATLAB® R2016a

III. Genetic Algorithm

Minimize the Rastrigin function as defined in part (I) using the Genetic Algorithm.

1) Start by using 10 bits to encode each design variable. Use the guidelines discussed in class for population size and mutation rates. Examine the effect of the random number generator by running the program at least two more times.
2) Repeat part (1) using 5 bits for each variable. Repeat using 20 bits for each variable. Change the population and mutation rates according to the guidelines. How do these different coding resolutions affect the quality of the results and the runtime of the algorithm?

Solution

1) The results are shown below:

	Population Size	Mutation Rate	Number of Generations	Function Evaluations	x*	$f(x^*)$
Run 1	80	6.56E-03	23	1920	[5.00489e-3; 0.985963]	1.016
Run 2	80	6.56E-03	29	2400	[-1.0360117; 4.504399e-2]	1.7281
Run 3	80	6.56E-03	25	2080	[-3.50342e-2; 5.004887e-3]	0.2475
Run 4	80	6.56E-03	25	2080	[5.00489e-3; 0.995973]	1.0001

The random effect of the GA can produce very different results. In this case, only two of the four runs had results that are reasonably similar (run 1 and run 4). It should be noted that only one results, run 3, was close to the true optimum solution of $[0, 0]^T$. However, because of the 10 bit resolution produces an effective variable step size of approximately 0.01, with the closest value to zero at ~0.005, so the true optimum solution of $[0, 0]^T$ cannot be represented in the chromosome.

2) The results are shown below:

		Population Size	Mutation Rate	Number of Generations	Function Evaluations	x*	$f(x^*)$
5 bits	Run 1	40	1.38E-02	33	1360	[0.1651613; -0.1651613]	9.8912
	Run 2	40	1.38E-02	16	680	[-0.1651613; -0.1651613]	9.8912
	Run 3	40	1.38E-02	77	3120	[-0.1651613; 0.1651613]	9.8912
	Run 4	40	1.38E-02	101	4080	[-0.1651613; 0.1651613]	9.8912
20 bits	Run 1	160	3.20E-03	56	9120	[0.01422364; 0.00229806	0.0412
	Run 2	160	3.20E-03	54	8800	[-0.0452978; -0.0089795]	0.4203
	Run 3	160	3.20E-03	79	12800	[1.01970312; -8.92090e-3]	1.1321
	Run 4	160	3.20E-03	53	8640	[5.57129e-3; -4.340824e-3]	0.0099

Decreasing the bit resolution to 5 bits produces a variable resolution of ~0.33, with a closest-to-zero value of ~0.1651. While this prevents the GA from reaching the optimum solution, it also appears to make it easier for the GA to find the optimum solution over its entire design space, as all four runs produced the same optimum function value of 9.8912. While the x* values are different, the symmetrical nature of the Rastrigin function produces four points of complementary pairs that each have the same function value. Unsurprisingly, the number of function evaluations was reduced on average, as the population size and possible chromosome permutations are both reduced as resolution is reduced.

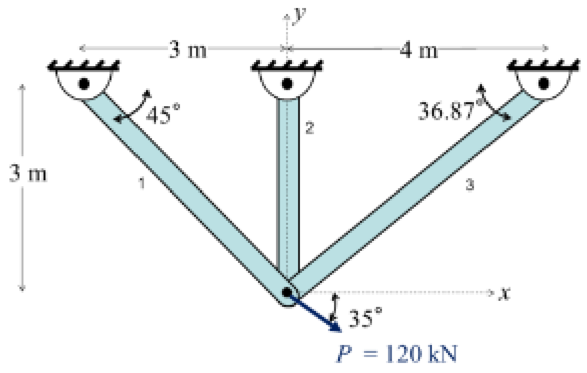
Increasing the resolution to 20 bits results in a variable resolution of $9.765*10^{-6}$, which allows the GA to find a solution incredibly close to the optimum $x=[0, 0]^T$. This is shown in run 4, which had the best quality results of any run. Using a 20 bit resolution also produced the most quality results, even though no two results can be considered reasonably similar. It should also be noted that, even at this higher resolution, the GA can still get stuck at a local minimum, as occurred in run 3. As expected, the computational cost increases with increasing the resolution.

```
% Thomas Satterly
% AAE 550, HW 3
% Problem 3

close all;
clear all;

options = aae550.hw3.p3.goptions([]);

vlb = [-5.12 -5.12]; % Lower bounds on genes
vub = [5.12 5.12]; % Upper bounds on genes
bits =[10 10]; % Number of bits per gene

l = sum(bits); % Length of the chromosome

% Basic guidelines for population and mutation rate
nPop = 4 * l;
pMutation = (l + 1) / (2 * nPop * l);

options(11) = nPop; % Set the population size
options(13) = pMutation; % Set the mutation probability

[x,fbest,stats,nfit,fgen,lgen,lfit]= aae550.hw3.p3.GA550(@(x)
 aae550.hw3.Rastrigin(x),[ ],options,vlb,vub,bits);

disp(x)
```

Published with MATLAB® R2016a

IV. A Combinatorial Problem with the Genetic Algorithm

Use the Genetic Algorithm (GA) to solve a combinatorial problem. The problem is to minimize the weight of the truss shown below by varying the cross-sectional area and material choice while avoiding yielding on any member.



Four possible materials are available for each beam member. The properties for each are given below. Assume that the allowable stress for both compression and tension is the same.

Material	E (Pa)	ρ (kg/m^3)	σ_y (Pa)
Aluminum	68.9×10^9	2700	55.2×10^6
Titanium	116×10^9	4500	140×10^6
Steel	205×10^9	7872	285×10^6
Nickel	207×10^9	8800	59.0×10^6

Because this problem is combinatorial, the objective function must include the density of the material and the constraints must incorporate Young's modulus and yield stress. Because there are four materials, the material selection for each truss can be represented as 2 bits. The mass of the truss can be computed as follows:

$$m = \rho_1 A_1 L_1 + \rho_2 A_2 L_2 + \rho_3 A_3 L_3$$

The lengths of the truss members can be treated as constant. The stiffness matrices are:

$$\boldsymbol{K}_1 = \begin{Bmatrix} \cos(-45°) \\ \sin(-45°) \end{Bmatrix} \frac{E_1 A_1}{L_1} \{\cos(-45°) \quad \sin(-45°)\}$$

$$\boldsymbol{K}_2 = \begin{Bmatrix} \cos(-90°) \\ \sin(-90°) \end{Bmatrix} \frac{E_2 A_2}{L_2} \{\cos(-90°) \quad \sin(-90°)\}$$

$$\boldsymbol{K}_3 = \begin{Bmatrix} \cos(-143.13°) \\ \sin(-143.13°) \end{Bmatrix} \frac{E_3 A_3}{L_3} \{\cos(-143.13°) \quad \sin(-143.13°)\}$$

$$\mathbf{K} = \mathbf{K}_1 + \mathbf{K}_2 + \mathbf{K}_3$$

The load vector is defined as follows:

$$\boldsymbol{p} = \begin{Bmatrix} \mathrm{P}\cos(-35^\circ) \\ \mathrm{P}\sin(-35^\circ) \end{Bmatrix}$$

The displacement of the free node, u, is obtained by solving the system Ku=p. Therefore, the change in length of any element can be found:

$$\Delta L_1 = \sqrt{\{-L_1\cos(-45^\circ) - u_1\}^2 + \{-L_1\sin(-45^\circ) - u_2\}^2} - L_1$$

$$\Delta L_2 = \sqrt{\{-L_2\cos(-90^\circ) - u_1\}^2 + \{-L_2\sin(-90^\circ) - u_2\}^2} - L_2$$

$$\Delta L_3 = \sqrt{\{-L_3\cos(-143.13^\circ) - u_1\}^2 + \{-L_3\sin(-143.13^\circ) - u_2\}^2} - L_3$$

Finally, the stress in each element is the strain multiplied by Young's modulus:

$$\sigma_i = E_i \frac{\Delta L_i}{L_i}$$

1) Encode the material selection in two bits as suggested above. Choose upper and lower bound variables for the cross sectional area of the beams. Choose the number of bits used to encode the cross section area. What resolution does this provide between each adjacent value? Determine the population and mutation rate for this problem, following the guidelines for uniform crossover and tournament selection.
2) Develop a fitness function that uses and exterior penalty to enforce the constraints. Be sure to state and explain any penalty multipliers. Recall that the quadratic form is not necessary, linear or step-linear forms are acceptable.
3) Run the GA to find a solution, evaluate the objective, constraint and fitness function values at this best design. Adjust the penalty multiplier until the constraints behave well. Include the GA function and any necessary supporting functions in the report.
4) Run the GA three times. Record the number of generations, function evaluations, best design and objective, constraint and fitness functions values at the best design.
5) Consider the best x* found. Based on this solution and considering the constraints at or near the objective bounds, recommend a change to the basic truss configuration that would improve the truss structure needed to support the design load.

Solution

1) The bounds for cross sectional area were chosen at [$1*10^{-9}$ m^2, $5*10^{-4}$ m^2]. This corresponds to a maximum area similar to a 50 mm wide square tube with a 2.75mm wall, and a minimum area that, while physically impractical, prevents a zero term during truss evaluation. Based on the results of bit resolution effect in problem (3), a 40 bit resolution was chosen for the area cross section, producing a variable resolution of ~$4.55*10^{-16}$ m^2. This is an impractically small resolution when considering the manufacturing tolerances on structural beams, but the consistent performance of the GA as a result of this resolution justifies using such a small resolution to determine an optimum value. This produces a population of 504 with a mutation rate of ~$1*10^{-4}$. As is shown in the results in part (4), the maximum cross section area bound is not an active or near-active constraint. Additionally, the GA was run with high area bounds, but this produced widely varying results, none of which bested the result provided by using the bounds as stated above.
2) See attached for solution
3) See attached for Matlab code of the fitness function. After iterating on the penalty multiplier, a value of 700 was found to produce a very well-behaved optimization.
4) The results are shown below:

	Number of Generations	Function Evaluations	Material			Cross Section Area (m^2)			$f(x^*)$
			Beam 1	Beam 2	Beam 3	Beam 1	Beam 2	Beam 3	
Run 1	224	113400	3	1	2	4.0443E-04	1.0000E-09	1.5029E-04	16.8888
Run 2	257	130032	3	1	2	4.0443E-04	1.0000E-09	1.5029E-04	16.8888
Run 3	260	131544	3	1	2	4.0443E-04	1.0000E-09	1.5029E-04	16.8888

	g(1)	g(2)	g(3)	g(4)g	g(5)	g(6)
Run 1	-2.0000E+00	-4.1112E-11	-1.1923E+00	-8.0774E-01	-4.5693E-10	-2.0000E+00
Run 2	-2.0000E+00	-5.4363E-11	-1.1923E+00	-8.0774E-01	-2.5924E-08	-2.0000E+00
Run 3	-2.0000E+00	-3.4660E-12	-1.1923E+00	-8.0774E-01	-6.2177E-10	-2.0000E+00

5) The near-active constraints on the optimal design were the tension yield stress on beam 1 and compression yield stress on 3, and the lower limit of cross section area on beam 2. This intuitively makes sense, as the applied load is closest to be align with beam 1, while the remaining load would be most effective taken by a beam perpendicular to beam 1, of which beam 3 is the closest. In this way, only two load components need resolved, while three members exist to resolve those loads, implying that at least one beam is redundant. In order to improve the truss structure, beam 2 should be eliminated, and beam 1 should be aligned at 35°, collinear with the expected load. Beam 3 should remain to resolve any torque from self-weight or load variations not explicitly present in this problem.

IV. Combinatorial Problem with GA

2. Goal is to minimize the mass of the beam, given by:

$$m = f(x) = \rho_1 A_1 L_1 + \rho_2 A_2 L_2 + \rho_3 A_3 L_3$$

$x = [M_1 \quad A_1 \quad M_2 \quad A_2 \quad M_3 \quad A_3]$, L_i's are fixed

M_i = Material selection (1, 2, 3, or 4), such that:

$$\rho_i = \rho(M_i), \quad E_i = E(M_i), \quad \sigma_{y_i} = \sigma_y(M_i)$$

Constraints: $-\sigma_{y_i} \le \sigma_i \le \sigma_{y_i}$

$\sigma_1 \ge -\sigma_{y_1} \Rightarrow g_1(x) = \frac{\sigma_1}{-\sigma_{y_1}} - 1$

$\sigma_1 \le \sigma_{y_1} \Rightarrow g_2(x) = \frac{\sigma_1}{\sigma_{y_1}} - 1$

$\sigma_2 \ge -\sigma_{y_2} \Rightarrow g_3(x) = \frac{\sigma_2}{-\sigma_{y_2}} - 1$

$\sigma_2 \le \sigma_{y_2} \Rightarrow g_4(x) = \frac{\sigma_2}{\sigma_{y_2}} - 1$

$\sigma_3 \ge -\sigma_{y_3} \Rightarrow g_5(x) = \frac{\sigma_3}{-\sigma_{y_3}} - 1$

$\sigma_3 \le \sigma_{y_3} \Rightarrow g_6(x) = \frac{\sigma_3}{\sigma_{y_3}} - 1$

For this problem, use the linear exterior penalty method

$$P(x) = \sum_{i=1}^{n} \max[0, g_i(x)]$$

Minimize: $\phi(x) = f(x) + r_p P(x)$

$f(x) = \rho_1 A_1 L_1 + \rho_2 A_2 L_2 + \rho_3 A_3 L_3$

$g_1(x) = \frac{\sigma_1}{-\sigma_{y_1}} - 1$

$g_2(x) = \frac{\sigma_1}{\sigma_{y_1}} - 1$

$g_3(x) = \frac{\sigma_2}{-\sigma_{y_2}} - 1$

$g_4(x) = \frac{\sigma_2}{\sigma_{y_2}} - 1$

$g_5(x) = \frac{\sigma_3}{-\sigma_{y_3}} - 1$

$g_6(x) = \frac{\sigma_3}{\sigma_{y_3}} - 1$

```
% Thomas Satterly
% AAE 550, HW 3
% Problem 4


close all;
clear;

options = aae550.hw3.p3.goptions([]);

% Gene encoding:
%
%     |   Beam 1     |    Beam 2     |    Beam 3     |
% x = [Material, Area, Material, Area, Material, Area]
% Area in [m^2]
minArea = 1e-9;
maxArea = 5e-4;
vlb = [1 minArea 1 minArea 1 minArea]; %Lower bound of each gene
vub = [4 maxArea 4 maxArea 4 maxArea]; %Upper bound of each gene
bits =[2 40 2 40 2 40]; % Number of bits describing each gene

l = sum(bits); % Chromosome length

% Basic guidlines for population and mutation rate
nPop = 4 * l;
pMutation = (l + 1) / (2 * nPop * l);

% Set options
options(11) = nPop; % Set the population size
options(13) = pMutation; % Set the mutation probability
options(14) = 1e6; % Maximum number of generations

% Evaluate
pMult = 7e2; % Penalty multiplier
[x,fbest,stats,nfit,fgen,lgen,lfit]= aae550.hw3.p3.GA550(@(x)
 aae550.hw3.p4.evalTruss(x, pMult),[ ],options,vlb,vub,bits);

% Print results
resolution = (vub(2) - vlb(2)) / (2^bits(2));
fprintf('Cross sectional area resolution: %0.9f %sm^2 \n', resolution
 * 1e12, char(956));

disp(x)
[phi, g] = aae550.hw3.p4.evalTruss(x)
assert(all(g <= 0), 'Constraints violated!');
```

Published with MATLAB® R2016a

```
function [phi, g] = evalTruss(x, pMult)
%EVALTRUSS Fitness function for the truss, linear exterior penalty

% Thomas Satterly
% AAE 550, HW 3
% Problem 4

% Default penalty multiplier
if nargin < 2
    pMult = 10;
end

% Input format:
% x(1) = Beam 1 material
% x(2) = Beam 1 cross section area [m^2]
% x(3) = Beam 2 material
% x(4) = Beam 2 cross section area [m^2]
% x(5) = Beam 3 material
% x(6) = Beam 3 cross section area [m^2]

A = x(2:2:6); % Cross section area
materials = x(1:2:5); % Material selection

% Preallocate arrays
E = zeros(1, numel(materials));
rho = zeros(1, numel(materials));
sigma_y = zeros(1, numel(materials));

% Get material properties for each beam
for i = 1:numel(materials)
    [E(i), rho(i), sigma_y(i)] = getMaterial(materials(i));
end

% Get the stress on each beam
sigmas = aae550.hw3.p4.stressHW3(A,E);

% Check constraints
g = zeros(1, numel(sigmas));
for i = 1:numel(sigmas)
    for j = -1:2:1
        g(((i - 1) * 2) + (j + 1) / 2 + 1) = (sigmas(i) / (j *
 sigma_y(i))) - 1;
    end
end

% Calculate penalty
P = 0;
for i = 1:numel(g)
    % Linear exterior penalty
    P = P + max(0, g(i));
end
```

```
% Get mass
L(1) = sqrt(3^2 + 3^2); % [m]
L(2) = 3; % [m]
L(3) = sqrt(3^2 + 4^2); % [m]
mass = 0;
for i = 1:numel(materials)
    mass = mass + L(i) * rho(i) * A(i);
end

% Fitness function
phi = mass + pMult * P;

    function [E, rho, sigma_y] = getMaterial(x)
        % Gets the material properties
        switch x
            case 1 % Aluminum
                E = 68.9e9; % [Pa] Young's Modulus
                rho = 2700; % [kg/m^3] Density
                sigma_y = 55.2e6; % [Pa] Yeild Stress
            case 2 % Titanium
                E = 116e9; % [Pa] Young's Modulus
                rho = 4500; % [kg/m^3] Density
                sigma_y = 140e6; % [Pa] Yeild Stress
            case 3 % Steel
                E = 205e9; % [Pa] Young's Modulus
                rho = 7872; % [kg/m^3] Density
                sigma_y = 285e6; % [Pa] Yeild Stress
            case 4 % Nickel
                E = 207e9; % [Pa] Young's Modulus
                rho = 8800; % [kg/m^3] Density
                sigma_y = 59.0e6; % [Pa] Yeild Stress
            otherwise
                error('Unrecognized material!');
        end
    end

end
```

Published with MATLAB® R2016a