AAE 550 Homework Assignment #1

Alex Hagen

October 17, 2016

1 Engineering Problem in N Variables

For the three-bar truss presented below (not to scale; including angles and lengths), minimizing the total potential energy as a function of the displacement can determine the equilibrium position of the "free" node (i.e. the node not attached to a support) under the applied load P. For this, neglect the self-weight of the truss elements.

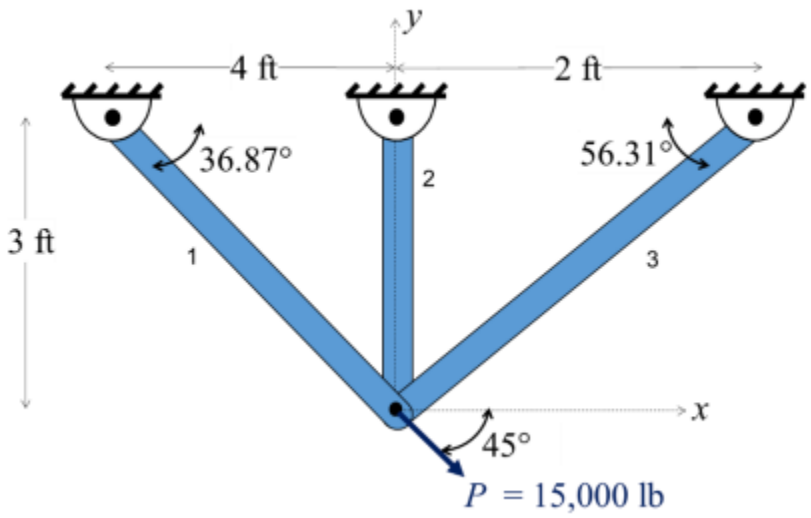

Figure 1: Truss Diagram for Problem 1

The potential energy function is:

$$\Pi(\vec{u}) = \frac{1}{2}\vec{u}^T\mathbb{K}\vec{u} - \vec{p}^T\vec{u}$$

Here, $\vec{u}$ is the displacement vector; this will be the design variables vector - u_i is the displacement in the horizontal (x-axis) direction and u_2 is the displacement in the vertical (y-axis) direction. $\mathbb{K}$ is the stiffness matrix, and $\vec{p}$ is the applied load vector. The global stiffness matrix is the sum of the stiffness matrices for the individual elements. The following expression describe the element stiffness matrices in terms of the x- and y- coordinates.

$$\mathbb{K}_1 = \left\{ \begin{array}{c} \cos(-36.87^o) \\ \sin(-36.87^o) \end{array} \right\} \frac{EA_1}{L_1} \left\{ \begin{array}{cc} \cos(-36.87^o) & \sin(-36.87^o) \end{array} \right\}$$

$$\mathbb{K}_2 = \left\{ \begin{array}{c} 0 \\ \sin(-90^o) \end{array} \right\} \frac{EA_2}{L_2} \left\{ \begin{array}{cc} 0 & \sin(-90^o) \end{array} \right\}$$

$$\mathbb{K}_3 = \left\{ \begin{array}{c} \cos(-123.69^o) \\ \sin(-123.69^o) \end{array} \right\} \frac{EA_3}{L_3} \left\{ \begin{array}{cc} \cos(-123.69^o) & \sin(-123.69^o) \end{array} \right\}$$

E is Young's modulus; use a value of 16×10^6 psi. $A_1$is the cross-sectional area of a truss member on the left; this element has a solid circular cross-section with a diameter of 1.75 in. Similarly, A_2 is the cross-sectional area

of the member in the center; this element has a solid circular cross-section with a diameter of 1.05 in. A_3 is the cross-sectional area of the member on the right; this element has a solid circular cross-section wtih a diameter of 1.20 in. L_1, L_2, and $L_3$are the lengths of the three elements; use the dimensions on the sketch above to compute these values. With the element stiffness matrices computed, the stiffness matrix for the truss is $\mathbb{K} = \mathbb{K}_1 + \mathbb{K}_2 + \mathbb{K}_3$.

Minimizing $\Pi(\vec{u})$ will provide the displacements $\vec{u}$ under the applied load vector, $\vec{p}$. Here, the load vector is

$$\vec{p} = \left\{ \begin{array}{c} P\cos(-45^o) \\ P\sin(-45^o) \end{array} \right\}$$

When you submit your work, include a copy of your Matlab function files and scripts used to call fminunc. These should be included, even if your have simply modified the provided examples. For the Excel problems, you can cut and paste your spreadsheet at the $\vec{x}^0$ values, the answer report and the sensititivity report as objects for tables in your submittal. You can use the example submittal from HW 0 as a template for this assignment, too.

1.1 Develop analytic expressions for the gradient vector components and the Hessian matrix components. You may do this either in matrix form, or you may wish to expand $\Pi(\vec{u})$ into a polynomial and then find derivatives of the polynomial. Provide the Matlab script file that includes the gradient and hessian.

With

$$\begin{aligned} \Pi(\vec{u}) &= \frac{1}{2}\vec{u}^T\mathbb{K}\vec{u} - \vec{p}^T\vec{u} \\ &= \frac{1}{2}\begin{bmatrix} u_1 & u_2 \end{bmatrix}\mathbb{K}\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} - \begin{bmatrix} p_1 & p_2 \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \end{aligned}$$

where $\mathbb{K}$ is defined earlier as $\mathbb{K} = \mathbb{K}_1 + \mathbb{K}_2 + \mathbb{K}_3$. We can then find the values of $\mathbb{K}$:

$$\mathbb{K} = \begin{bmatrix} & \\ & \end{bmatrix} + \begin{bmatrix} & \\ & \end{bmatrix} + \begin{bmatrix} & \\ & \end{bmatrix} = 1\times 10^5 \begin{bmatrix} 7.2561 & 5.2131 \\ 5.2131 & 6.5148 \end{bmatrix}$$

and the two values for $\vec{p}$

$$\vec{p} = \begin{bmatrix} 15000\cos(-45) \\ 15000\sin(-45) \end{bmatrix} = 1\times 10^4 \begin{bmatrix} 0.7880 \\ -1.2764 \end{bmatrix}$$

and thus we can start to simplify the expression

$$\begin{aligned} \Pi(\vec{u}) &= \frac{1}{2}\begin{bmatrix} (u_1K_{11} + u_1K_{12}) & (u_2K_{21} + u_2K_{22}) \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} - \begin{bmatrix} p_1 & p_2 \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \\ &= \frac{1}{2}\left(u_1^2K_{11} + u_2u_1K_{12} + u_1u_2K_{21} + u_2^2K_{22}\right) - (p_1u_1 + p_2u_2) \end{aligned}$$

Now we can actually calculate the gradient components

$$\nabla\Pi = \begin{bmatrix} \frac{\partial\Pi}{\partial u_1} \\ \frac{\partial\Pi}{\partial u_2} \end{bmatrix} = \begin{bmatrix} u_1K_{11} + \frac{u_2}{2}(K_{12} + K_{21}) - p_1 \\ \frac{u_1}{2}(K_{12} + K_{21}) + u_2K_{22} - p_2 \end{bmatrix}$$

and the Hessian components

$$\mathbb{H}\Pi = \begin{bmatrix} \frac{\partial^2\Pi}{\partial u_1^2} & \frac{\partial^2\Pi}{\partial u_1\partial u_2} \\ \frac{\partial^2\Pi}{\partial u_2\partial u_1} & \frac{\partial^2\Pi}{\partial u_2^2} \end{bmatrix} = \begin{bmatrix} K_{11} & \frac{K_{12}+K_{21}}{2} \\ \frac{K_{12}+K_{21}}{2} & K_{22} \end{bmatrix}$$

And these are defined in the listing below:

```
function [f, nabla, H] = Pi(u)
    E = 16E6; % psi
    A_1 = 1.75; % inches
    L_1 = sqrt((3*12)^2 + (4*12)^2); % inches
    theta_1 = -36.87;
    K_1 = [cos(theta_1); sin(theta_1)] * E * A_1 * ...
          [cos(theta_1) sin(theta_1)] / L_1;
    A_2 = 1.05; % inches
```

```
    L_2 = 3*12; % inches
    theta_2 = -90;
    K_2 = [cos(theta_2); sin(theta_2)] * E * A_2 * ...
          [cos(theta_2) sin(theta_2)] / L_2;
    A_3 = 1.20; % inches
    L_3 = sqrt((3*12)^2 + (2*12)^2); % inches
    theta_3 = -56.31;
    K_3 = [cos(theta_3); sin(theta_3)] * E * A_3 * ...
          [cos(theta_3) sin(theta_3)] / L_3;

    K = K_1 + K_2 + K_3;

    p = [15000 * cos(-45); 15000 * sin(-45)];

    f = u' * K * u - p' * u;

    nabla = [u(1) * K(1, 1) + (u(2)/2) * (K(1, 2) + K(2, 1)) - p(1);
             (u(1)/2) * (K(1, 2) + K(2, 1)) + u(2) * K(2, 2) - p(2)];

    H = [K(1, 1) (K(1, 2) + K(2, 1))/2;
         (K(1, 2) + K(2, 1))/2 K(2, 2)];
end
```

5C__Users_Alex_Desktop_aae550_hmwk_1_src_Pi.m

1.2 Use "fminunc" in Matlab's Optimization Toolbox to solve this problem for the equilibrium position of the free node. Pay attention to the "exitflag" and "message" information to determine if the algorithm has converged. The default algorithm uses the BFGS update.

The results for fminunc with BFGS are listed in table 1, and comments about the results as well as code for running them are listed under the problem prompts.

1.2.1 First solve the problem using finite difference gradients. Use fminunc with options = optimset('LargeScale', 'off', 'GradObj', 'off', 'Display', 'iter'). Record $\vec{x}^0$, $\vec{x}^*$, $f(\vec{x}^*)$, $\nabla f(\vec{x}^*)$, the number of iterations needed, the number of function evaluations needed, and the values of exitflag. These will be included in the table discussed in part 6.

Using the lines shown below, the optimal position was calculated and the results tabulated in Row 1 of table 1.

```
x_0 = [0; 0];
options = optimset('LargeScale', 'off', 'GradObj', 'off', ...
                   'Display', 'iter');
[x,fval,exitflag,output,grad,hessian] = fminunc(@(x) Pi(x), x_0, options)
```

6C__Users_Alex_Desktop_aae550_hmwk_1_src_minimize_Pi_num_grad.m

1.2.2 Solve the problem using analytic gradients. Here, use fminunc with options = optimset('LargeScale', 'off', 'GradObj', 'on', 'Display', 'iter'). Record $\vec{x}^0$, $\vec{x}^*$, $f(\vec{x}^*)$, $\nabla f(\vec{x}^*)$, the number of iterations needed, the number of function evaluations needed, and the values of exitflag.

For this, the optimal position was calculated with an analytical gradient using the below code, and tabulated in Row 2 of table 1.

```
x_0 = [0; 0];
options = optimset('LargeScale', 'off', 'GradObj', 'on', ...
                   'Display', 'iter');
[x,fval,exitflag,output,grad,hessian] = fminunc(@(x) Pi(x), x_0, options)
```

7C__Users_Alex_Desktop_aae550_hmwk_1_src_minimize_Pi_analytic_grad.m

1.3 Matlab offers two other first-order models, the DFP update and steepest descent. Explore these to solve the problem for the equilibrium position of the free node. Pay attention to the "exitflag" and "message" information to determine if the algorithm has converged.

1.3.1 Solve the problem using analytic gradient with the DFP update. Here, use fminunc with options = optimset('LargeScale', 'off', 'GradObj', 'on', 'Display', 'iter', 'HessUpdate', 'dfp'). As before, record $\vec{x}^0$, $\vec{x}^*$, $f(\vec{x}^*)$, $\nabla f(\vec{x}^*)$, the number of iterations needed, the number of function evaluations needed, and the values of exitflag.

The analytic gradient with DFP update was calculated for an optimal solution, with the results recorded in Row 3 of table 1, and the code used to run this listed below.

```
x_0 = [0; 0];
options = optimset('LargeScale', 'off', 'GradObj', 'on', ...
                   'Display', 'iter', 'HessUpdate', 'dfp');
[x,fval,exitflag,output,grad,hessian] = fminunc(@(x) Pi(x), x_0, options)
```

8C__Users_Alex_Desktop_aae550_hmwk_1_src_minimize_Pi_analytic_grad_dfp.m

1.3.2 Solve the problem using analytic gradient with the steepest descent update. Here, use fminunc with options = optimset('LargeScale', 'off', 'GradObj', 'on', 'Display', 'iter', 'HessUpdate', 'steepdesc'). Again, record $\vec{x}^0$, $\vec{x}^*$, $f(\vec{x}^*)$, $\nabla f(\vec{x}^*)$, the number of iterations needed, the number of function evaluations needed, and the values of exitflag.

Calculation of the optimal solution was done with the steepest descent, and the results again tabulated in Row 4 of table 1, with the code again listed below.

```
x_0 = [0; 0];
options = optimset('LargeScale', 'off', 'GradObj', 'on', ...
                   'Display', 'iter', 'HessUpdate', 'steepdesc');
[x,fval,exitflag,output,grad,hessian] = fminunc(@(x) Pi(x), x_0, options)
```

9C__Users_Alex_Desktop_aae550_hmwk_1_src_minimize_Pi_analytic_grad_steep_desc.m

1.3.3 Use Matlab's fminunc with options = optimset('LargeScale', 'on', 'GradObj', 'on', 'Hessian', 'on') to use a modified Newton's Method with user supplied gradient and Hessian values. You will need to modify your function to now return both first and second partial derivatives of the objective function. Record $\vec{x}^0$, $\vec{x}^*$, $f(\vec{x}^*)$, $\nabla f(\vec{x}^*)$, the number of iterations needed, and the number of function evaluations needed. Pay attention to the "exitflag" and "message" information to determine if the algorithm has converged.

The calculation with the analytical hessian and gradient was calculated with the code listed below, and the results listed in Row 5 of table 1.

```
x_0 = [0; 0];
options = optimset('LargeScale', 'on', 'GradObj', 'on', ...
                   'Hessian', 'on');
[x,fval,exitflag,output,grad,hessian] = fminunc(@(x) Pi(x), x_0, options)
```

10C__Users_Alex_Desktop_aae550_hmwk_1_src_minimize_Pi_analytic_grad_hess.m

1.4 The Excel Solver add-in can also be used to solve this problem. Excel only uses numerical gradients, although you can choose to use forward or central differencing schemes to do this. Solve the problem using most of the default options in Solver. This will use the quasi-Newton method to find search directions and a forward differencing scheme to compute gradients. If you are using Excel 2010 select GRG Nonlinear. In the Solver window, specify to minimize the cell value containing the objective function by changing the cells containing the design variable values. Click the "options" button and select "Show Iteration Results". You will have to count the number of iterations, because Solver does not record this information. When solver reaches a solution, also create an answer report and a sensitivity report. The sensitivity report includes a column for the "reduced gradient" values, which for an unconstrained problem are simply the elements of $\nabla f(\vec{x}^*)$. Record $\vec{x}^0$, $\vec{x}^*$, $f(\vec{x}^*)$, $\nabla f(\vec{x}^*)$, and the number of iterations needed. Solver does not report the number of function evaluations.

The solver was created and used with the following equation and solved using automatic scaling, constraint precision of 0.000001, and the GRG Nonlinear method.

$$C2 = 12763.5529 * B3 - 7879.8298 * B2 + B3 * (521312.621 * B2 + 651482.3626 * B3) + B2 * (725611.1277 * B2 + 521312.621 * B3)$$

Results are reported in table 1.

1.5 Make a table that compares the various approaches. It should have a format something like the table shown below. Here, use a reasonable number of significant digits. In some cases, if you want to see a difference between methods, you may need more digits than Matlab's default "format short". Following the table, include a short paragraph that answers the following: What conclusions can you make about these unconstrained minimization approaches for this problem? Is there a significant difference in cost and / or accuracy when using numerical derivatives and when using analytic derivatives? How or why might the form of this problem be better suited to one of the above solution techniques?

The table requested is shown in Table 1 on the following page.

1.6 State the optimality conditions for an unconstrained minimization problem and show why the displacement of the free node is usually found by solving $\mathbb{K}\vec{u} = \vec{p}$ for $\vec{u}$. Using this strategy, what is the optimal displacement of the free node and the resulting potential energy? Show that this value is indeed optimal. How does this answer compare to the answers found previously?

The first and second order necessary conditions for optimality are:

1. $\nabla f(\vec{x}) = 0$
2. $\nabla^2 f(\vec{x})$ is positive semi-definite

Solving for the displacement of the free node and the resulting potential energy was done with the code below, giving the results of

$$\vec{x}^* = \begin{bmatrix} 0.0587 \\ -0.0665 \end{bmatrix}$$

$$f(\vec{x}^*) = 4.5475 \times 10^{-13}$$

Table 1: Results Table for different minimizations of Truss Problem

Method / Program	$\vec{x}^0$	$\vec{x}^*$	$f(\vec{x}^*)$	$\nabla f(\vec{x}^*)$	N_{iter}	N_f	exit flag
BFGS / Matlab fminunc, numerical gradient	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0.0293 \\ -0.0333 \end{bmatrix}$	-327.835	$\begin{bmatrix} 0.0189 \\ 0.0001 \end{bmatrix}$	3	39	5
BFGS / Matlab fminunc, user-defined gradient	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0.0289 \\ -0.0342 \end{bmatrix}$	-326.7543	$\begin{bmatrix} -4722.1 \\ 5563.5 \end{bmatrix}$	4	34	5
DFP / Matlab fminunc, user-defined gradient	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0.0283 \\ -0.0349 \end{bmatrix}$	-323.4988	$\begin{bmatrix} -5548.7 \\ 4764.5 \end{bmatrix}$	4	32	5
Steepest Descent / Matlab fminunc, user-defined gradient	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0.0286 \\ -0.0344 \end{bmatrix}$	-325.6678	$\begin{bmatrix} -5086.9 \\ 5244.7 \end{bmatrix}$	4	41	5
Modified Newton's method / Matlab fminunc, user-defined gradient	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0.0285 \\ -0.0340 \end{bmatrix}$	-326.3445	$\begin{bmatrix} -4929.5 \\ 5473.5 \end{bmatrix}$	12	13	2
Quasi-Newton method / Excel Solver	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0.0293 \\ -0.0333 \end{bmatrix}$	-327.835	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	7	N/A	N/A

$$\nabla f(\vec{x}^*) = 1.0 \times 10^{-11} \begin{bmatrix} 0.0909 \\ 0.1819 \end{bmatrix}$$

$$\nabla^2 f(\vec{x}^*) = 1.0 \times 10^{5} \begin{bmatrix} 7.2561 & 5.2131 \\ 5.2131 & 6.5148 \end{bmatrix}$$

which meets all of the optimality conditions ($\nabla f \approx 0$, $\nabla^2 f$ is positive definite).

The value gives much lower potential energy, and while the location is close to within the $^1/_{100}$ in, they are a ways off. Perhaps the newton solvers were caught in a local minimum which was not the global minimum. The exitflag given by all of the fminunc solvers were either 5, which states that the predicted decrease in the objective function was less than the functiontolerance tolerance, or that the change in x was smaller than the steptolerance, meaning that the solver was unable to approach so closely the true optimal solution because of tolerances set by default.

2 Engineering Application of SUMT Approach

NOTE: This problem is based upon problem 2.24 from Arora, Introduction to Optimum Design, third edition. You do not need the textbook to solve this problem, but I wanted to attribute its source.

This problem presents an engineering problem in a textual format that you must convert into an optimization problem statement. For those of you without a structures background, I have listed the important governing equations below. For those of you with a structures background, you will recognize many of these equations.

Design a hollow circular beam-column, as illustrated below, for the two conditions:

1. When $P = 75\,\text{kN}$, the axial stress σ must not exceed an allowable value σ_a.
2. When $P = 0$, the deflection δ due to the self-weight should satisfy $\delta \leq 0.002L$.

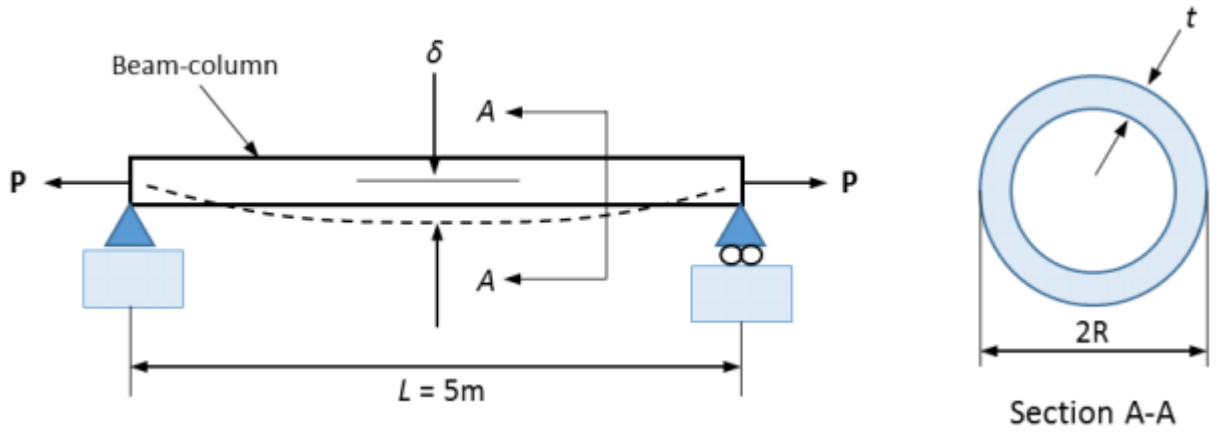

Figure 2: Truss Diagram for Problem 2

The manufacturing process limits the radius and thickness of the beam and the ratio of the radius and thickness of the beam as follows:

$$3.0\,\text{cm} \leq R \leq 30.0\,\text{cm}$$

$$0.20\,\text{cm} \leq t \leq 2.0\,\text{cm}$$

$$\frac{R}{t} \leq 15$$

The beam column shall be fabricated with the following material properties: Density, $\rho = 7800\,\frac{\text{kg}}{\text{m}^3}$; Allowable axial stress, $\sigma_a = 250\,\text{MPa}$; Modulus of Elasticity, $E = 210\,\text{GPa}$.

Use the following expressions:

Axial stress $\sigma = \frac{P}{A}$ where $A = \pi t\,(2R - t)$, the cross-sectional area of the beam column.

Deflection due to the self-weight $\delta = \frac{5wL^4}{384EI}$

$w = \frac{mg}{L}$, the self-weight force per unit length
$m = \rho AL$, mass of the beam
$I = \pi R^3 t$, moment of inertia
Use $g = 9.80\ \frac{\text{m}}{\text{s}^2}$

2.1 Formulate and explicitly state the optimization design problem.

We want to minimize the thickness while following both the manufacturing and safety (deflection under no load and stress under load) constraints. The minimization and constraints will be developed in the below subsections.

2.1.1 Write an objective function, $f(\vec{x})$, in terms of R and t. This objective function should minimize the mass of the beam. If bounds should be included, include these in your problem formulation, too. When writing the objective function, keep in mind that minimizing $Cf(\vec{x})$, where C is a constant, is the same as minimizing $f(\vec{x})$. Also, recall that variable scaling is sometimes required to improve conditioning. You might want to use these concepts to your advantage.

To minimize the mass of the beam, we have to minimize the radius and the thickness. Giving

$$m = \rho AL$$

where ρ is density (a constant), L is length of the beam (a constant), and A is the cross sectional area of the tube, given by

$$A = \pi t (2R - t)$$

By putting these two together, we can define a function, $f(\vec{x})$ which we will need to minimize to minimize the mass of the beam, given by

$$f(\vec{x}) = f\left(\begin{bmatrix} R \\ t \end{bmatrix}\right) = \rho \pi t (2R - t) L$$

2.1.2 Write the inequality constraint functions, $g_j(\vec{x})$, in terms of R and t. The text above describes the various inequality constraints required. Start by writing $\delta \geq \Delta$ (for example), and then convert this to an appropriate form for $g_j(\vec{x}) \leq 0$. If you find that your appropriate form of $g_j(\vec{x})$ has design variables in the denominator, make a note of this. You may need to change this when you implement the SUMT approaches. There are no equality constraints in this problem.

There are multiple contraints in this problem, the first of which is the stress constraint

$$\sigma < \sigma_a$$

where the stress is defined by

$$\sigma = \frac{P}{A}$$

where P is the load (a constant), and A is the cross sectional area, as given above. We will define this constraint as the first constraint

$$\begin{aligned} \sigma &< \sigma_a \\ \frac{P}{\pi t (2R - t)} &< \sigma_a \\ \frac{P}{\pi t (2R - t)} - \sigma_a &< 0 \\ g_1(\vec{x}) &< 0 \end{aligned}$$

The second constraint deals with the deflection under no load

$$\delta < \Delta$$

where δ is defined by

$$\delta = \frac{5wL^4}{384EI}$$

where L is the length (a constant), E is the modulus of elasticity (a constant), w is the self weight, defined below, and I is the moment of intertia, defined below.

$$w = \frac{mg}{L}$$

where g is the gravitational constant, L is the length (a constant), and m is the mass of the element, given by

$$m = \rho AL = \rho\pi t\left(2R - t\right)L$$

The moment of interia is given by

$$I = \pi R^3 t$$

Thus, the deflection constraint is given by

$$\begin{aligned}
\delta &< \Delta \\
\frac{5wL^4}{384EI} &< \Delta \\
\frac{5\frac{mg}{L}L^4}{384E\pi R^3 t} &< \Delta \\
\frac{5\rho\pi t\left(2R - t\right)gL^4}{384E\pi R^3 t} &< \Delta \\
\frac{5\rho gL^4\left(2R - t\right)}{384ER^3} &< \Delta \\
\frac{5\rho gL^4\left(2R - t\right)}{384ER^3} - \Delta &< 0 \\
g_2\left(\vec{x}\right) &< 0
\end{aligned}$$

2.1.3 Write any side constraints or bounds on the design variables. Some appear explicitly in the problem description, and there are obvious bounds for others. Convert these into additional $g_j\left(\vec{x}\right) \leq 0$ for use in the SUMT approach. Provide the Matlab script file.

The manufacturing constraints give some bounds for the design variables, with

$$3.0\,\text{cm} \leq R \leq 30.0\,\text{cm}$$

which can be split into two bounding constraints

$$\begin{aligned}
3.0\,\text{cm} &\leq R \\
3.0\,\text{cm} - R &\leq 0 \\
g_3\left(\vec{x}\right) &\leq 0
\end{aligned}$$

and

$$\begin{aligned}
R &\leq 30.0\,\text{cm} \\
R - 30.0\,\text{cm} &\leq 0 \\
g_4\left(\vec{x}\right) &\leq 0
\end{aligned}$$

This same can be done for the thickness

$$0.20\,\text{cm} \leq t \leq 2.0\,\text{cm}$$

which can be split into two constraints again

$$\begin{aligned}
0.20\,\text{cm} &\leq t \\
0.20\,\text{cm} - t &\leq 0 \\
g_5\left(\vec{x}\right) &\leq 0
\end{aligned}$$

and

$$\begin{aligned} t &\leq 2.0\,\text{cm} \\ t - 2.0\,\text{cm} &\leq 0 \\ g_6\left(\vec{x}\right) &\leq 0 \end{aligned}$$

The last explicitly defined bound is

$$\begin{aligned} \frac{R}{t} &\leq 15 \\ \frac{R}{t} - 15 &\leq 0 \\ g_7\left(\vec{x}\right) &\leq 0 \end{aligned}$$

And finally, the thickness cannot be bigger than the radius, which gives

$$\begin{aligned} t &\leq R \\ t - R &\leq 0 \\ g_8\left(\vec{x}\right) &\leq 0 \end{aligned}$$

2.2 Use the fminunc function from Matlab with the following penalty methods to solve this problem via the SUMT approach. It is acceptable to use the default BFGS update and numerical gradients. If you use something other than these options, clearly state that in your submittal.
For each, you will need to choose the value of ε you wish to use for convergence of the objective function between successive minimizations. Be sure to mention this value in your responses to this question.
Record the total number of unconstrained minimizations, the total number of iterations (each individual unconstrained minimization will require some number of its own iterations), the final design solution, $\vec{x}^*$ (where $\vec{x} = \left\{ \begin{array}{cc} R & t \end{array} \right\}^T$), the values of $f(\vec{x}^*)$ and $g_j(\vec{x}^*)$, and the exitflag value provided by fminunc. Prepare a table.
At the $\vec{x}^*$ that you find, be sure that the constraints, if slightly violated, are acceptable. For instance, you may need to directly compare $\sigma(\vec{x}^*)$ to σ_a or $\delta(\vec{x}^*)$ to maximum allowed deflection.
To obtain good results, you may need to try different choices of x_0; this is because the penalty functions can have poor condition if $\vec{x}_0$ is infeasible. The exitflag value returned by fminunc may help you determine if you have a good result. You may also need to experiment with the initial value of r (or $r\prime$). Also, be aware that the design variables likely have different magnitudes, if you use consistent units. This can have an effect on your solution. Pay attention to constraints with design variables in the denominator; if it is numerically possible for the design variable to have a zero value (even if this is an infeasible design – recall, the SUMT methods often need to evaluate infeasible $\vec{x}$ during the search), this will cause problems for some of the SUMT approaches. The interior penalty method must have a feasible $\vec{x}_0$. You may also need to consider using constraint scaling terms, c_j, in your pseudo-objective formulation.
Even with the improved conditioning and scaling, the available optimization routines may not be able to solve a problem using one or more of the penalty methods. If this is the case, be sure to explain what you have tried to make the method work and what you have deduced as the reason why the problem cannot be solved using the available optimization routine.
Along with the table for each of the following methods, include the *.m file used to create your pseudoobjective function.

2.2.1 Use the exterior penalty method. Comment on values used for c_j, if any.

The scaling was performed with the below vector.

$$c_j = \left[\begin{array}{cccccccc} 1 \times 10^{-7} & 1 \times 10^{-3} & 10 & 10 & 100 & 100 & 0.01 & 10 \end{array} \right]$$

This was then run using the files listed below, and the results tabulated in table 2.

```
function phi = sumt_phi(x,r_p)

% compute values of the objective function and constraints at the current
% value of x
f = sumt_fun(x);
g = sumt_con(x);

% exterior penalty function
ncon = length(g);    % number of constraints
P = 0;               % intialize P value to zero
for j = 1:ncon
    P = P + max(0,g(j))^2;  % note: no c_j scaling parameters
```

```
end
phi = f + r_p * P;
```

12C__Users_Alex_Desktop_aae550_hmwk_1_src_sumt_phi.m

```
clear all
format long     % use format long to see differences near convergence

table = [];
x0 = [0.03; 0.02];    % initial design
p = 0;          % initial value of minimization counter
r_p = 1.0;      % initial value of penalty multiplier

% compute function value at x0, initialize convergence criteria
f = sumt_fun(x0);
f_last = 2 * f;    % ensure that first loop does not trigger convergence
% set optimization options - use default BFGS with numerical gradients
% provide display each iteration
options = optimset('LargeScale', 'off', 'Display', 'iter');

% begin sequential minimizations  - note tolerances chosen here
% absolute tolerance for change in objective function, absolute tolerance
% for constraints
while ((abs((f-f_last)/f_last) >= 1e-3) || (max(g) >= 1e-5))
    f_last = f;  % store last objective function value
    p            % display current minimization counter
    r_p          % display current penalty multiplier
    % call fminunc - use "phi" pseudo-objective function, note that r_p is
    % passed as a "parameter", no semi-colon to display results
    [xstar,phistar,exitflag,output] = fminunc(@sumt_phi,x0,options,r_p)
    % compute objective and constraints at current xstar
    f = sumt_fun(xstar);
    g = sumt_con(xstar);
    N = output.funcCount;
    table = [table; p, r_p, xstar(1), xstar(2), f, phistar, N, exitflag];
    p = p + 1;      % increment minimization counter
    r_p = r_p * 5; % increase penalty multiplier
    x0 = xstar;     % use current xstar as next x0
end
% display function and constraint values at last solution
f = sumt_fun(xstar)
g = sumt_con(xstar)
format short
header = {'$p$', '$r_{p}$', '$R$', '$t$', ...
          '$f\left(\vec{x}^{*}\right)$', ...
          '$\phi\left(\vec{x}^{*}\right)$', '$N$', 'exitflag'};
matrix2lyx(table,'epm.lyx',header);
```

13C__Users_Alex_Desktop_aae550_hmwk_1_src_sumt_epm.m

2.2.2 Use the interior penalty method. Comment on values used for c_j, if any.

The scaling was performed with the below vector.

$$c_j = \begin{bmatrix} 1\times10^{-7} & 1\times10^{-3} & 10 & 10 & 100 & 100 & 0.01 & 10 \end{bmatrix}$$

This was then run using the files listed below, and the results tabulated in table 3.

```
function phi = sumt_phi_ipm(x,r_p)

% compute values of the objective function and constraints at the current
% value of x
f = sumt_fun(x);
g = sumt_con(x);

% exterior penalty function
ncon = length(g);    % number of constraints
P = 0;               % intialize P value to zero
for j = 1:ncon
```

Table 2: Minimization Table and Resulting Scaled Constraint Values for Exterior Penalty Method

p	r_p	R	t	$f(\vec{x}^*)$	$\phi(\vec{x}^*)$	N	exitflag
0	1	$-3.31319 \times 10^{+07}$	$3.22597 \times 10^{+07}$	$-3.89416 \times 10^{+20}$	$-3.78472 \times 10^{+20}$	45	-3
1	5	$-3.31319 \times 10^{+07}$	$3.22597 \times 10^{+07}$	$-3.89416 \times 10^{+20}$	$-3.34695 \times 10^{+20}$	3	-3
2	25	$-3.31319 \times 10^{+07}$	$3.22597 \times 10^{+07}$	$-3.89416 \times 10^{+20}$	$-1.1581 \times 10^{+20}$	3	-3
3	125	492.848	-53.3306	$-6.78919 \times 10^{+09}$	$-2.01182 \times 10^{+08}$	45	1
4	625	0.308077	-0.00412018	-313.123	-74.9414	42	1
5	3125	0.307651	-7.21084×10^{-10}	-5.43611×10^{-05}	143.292	201	0
6	15625	0.307651	-7.21084×10^{-10}	-5.43611×10^{-05}	716.461	63	5
7	78125	0.0299364	0.00199578	14.1525	14.198	123	2
8	390625	0.0299927	0.00199925	14.2039	14.2083	57	2
9	$1.95313 \times 10^{+06}$	0.0299928	0.00200002	14.2092	14.2194	36	2
10	$9.76563 \times 10^{+06}$	0.0299933	0.00199977	14.2077	14.2573	27	2

$$\vec{c}g(\vec{x}^*) = \begin{bmatrix} -4.4251 & -2.8725 \times 10^{7} & -4.1101 \times 10^{5} & -2.7 & -2.6 \times 10^{-5} & -1.8 & 1.0112 \times 10^{-6} & -0.28 \end{bmatrix}$$

```
    P = P - log(-g(j));  % note: no c_j scaling parameters
end
phi = f + r_p * P;
```

15C__Users_Alex_Desktop_aae550_hmwk_1_src_sumt_phi_ipm.m

```
clear all
format long     % use format long to see differences near convergence

table = [];
x0 = [0.15; 0.015];   % initial design
p = 0;          % initial value of minimization counter
r_p = 1.0;      % initial value of penalty multiplier
gamma = 10;

% compute function value at x0, initialize convergence criteria
f = sumt_fun(x0);
f_last = 2 * f;   % ensure that first loop does not trigger convergence
% set optimization options - use default BFGS with numerical gradients
% provide display each iteration
options = optimset('LargeScale', 'off', 'Display', 'iter');

% begin sequential minimizations  - note tolerances chosen here
% absolute tolerance for change in objective function, absolute tolerance
% for constraints
while ((abs((f-f_last)/f_last) >= 1e-3) || (max(g) >= 1e-5))
    f_last = f;  % store last objective function value
    p            % display current minimization counter
    r_p          % display current penalty multiplier
    % call fminunc - use "phi" pseudo-objective function, note that r_p is
    % passed as a "parameter", no semi-colon to display results
    [xstar,phistar,exitflag,output] = fminunc(@sumt_phi_ipm,x0,options,r_p)
    % compute objective and constraints at current xstar
    f = sumt_fun(xstar);
    g = sumt_con(xstar);
    N = output.funcCount;
    table = [table; p, r_p, xstar(1), xstar(2), f, phistar, N, exitflag];
    p = p + 1;      % increment minimization counter
    r_p = r_p / gamma; % solve rpp
    x0 = xstar;     % use current xstar as next x0
end
% display function and constraint values at last solution
f = sumt_fun(xstar)
g = sumt_con(xstar)
format short
```

Table 3: Minimization Table and Resulting Scaled Constraint Values for Interior Penalty Method

p	r_p	R	t	$f(\vec{x}^*)$	$\phi(\vec{x}^*)$	N	exitflag
0	1	0.0313985	0.0023836	17.6434	9.00734	105	2
1	0.1	0.0301379	0.00203576	14.5266	14.4572	65	2

$$\vec{cg}(\vec{x}^*) = \begin{bmatrix} -4.48645 & -3.1283 \times 10^7 & -0.0014 & -2.6986 & -0.0036 & -1.7964 & -0.0020 & -0.2810 \end{bmatrix}$$

```
header = {'$p$', '$r_{p}$', '$R$', '$t$', ...
          '$f\left(\vec{x}^{*}\right)$', ...
          '$\phi\left(\vec{x}^{*}\right)$', '$N$', 'exitflag'};
matrix2lyx(table,'ipm.lyx',header);
```

16C__Users_Alex_Desktop_aae550_hmwk_1_src_sumt_ipm.m

2.2.3 Use the extended-linear penalty method. Comment on values used for c_j, if any.

The scaling was performed with the below vector.

$$c_j = \begin{bmatrix} 1 \times 10^{-7} & 1 \times 10^{-3} & 10 & 10 & 100 & 100 & 0.01 & 10 \end{bmatrix}$$

This was then run using the files listed below, and the results tabulated in table 4.

```
function phi = sumt_phi_elpm(x,r_p,epsilon)

% compute values of the objective function and constraints at the current
% value of x
f = sumt_fun(x);
g = sumt_con(x);

% exterior penalty function
ncon = length(g);    % number of constraints
P = 0;               % intialize P value to zero
for j = 1:ncon
    if g(j) <= epsilon
        P = P - 1/g(j);
    else
        P = P - (2*epsilon - g(j))/(epsilon^2);
    end
end
phi = f + r_p * P;
```

18C__Users_Alex_Desktop_aae550_hmwk_1_src_sumt_phi_elpm.m

```
function phi = sumt_phi_elpm(x,r_p,epsilon)

% compute values of the objective function and constraints at the current
% value of x
f = sumt_fun(x);
g = sumt_con(x);

% exterior penalty function
ncon = length(g);    % number of constraints
P = 0;               % intialize P value to zero
for j = 1:ncon
    if g(j) <= epsilon
        P = P - 1/g(j);
    else
        P = P - (2*epsilon - g(j))/(epsilon^2);
    end
end
phi = f + r_p * P;
```

18C__Users_Alex_Desktop_aae550_hmwk_1_src_sumt_phi_elpm.m

Table 4: Minimization Table and Resulting Scaled Constraint Values for Extended Linear Penalty Method

p	r_p	ε	R	t	$f(\vec{x}^*)$	$\phi(\vec{x}^*)$	N	exitflag
0	26.1226	-0.1	0.0561148	0.0109379	135.745	628.135	66	1
1	2.61226	-0.0316228	0.0413666	0.00540127	51.1764	127.62	45	2
2	0.261226	-0.01	0.034548	0.00314065	25.3795	41.0491	54	2
3	0.0261226	-0.00316228	0.0316368	0.0023726	17.7037	21.6858	57	2
4	0.00261226	-0.001	0.0305462	0.00211952	15.3146	16.4676	63	2
5	0.000261226	-0.000316228	0.030176	0.00203798	14.5609	14.9145	60	2
6	2.61226×10^{-05}	-0.0001	0.030056	0.0020121	14.3232	14.4334	66	2
7	2.61226×10^{-06}	-3.16228×10^{-05}	0.030018	0.0020038	14.2475	14.2823	57	2

$$\vec{c}g(\vec{x}^*) = \begin{bmatrix} -4.4601 & -2.8925 \times 10^{7} & -1.7949 \times 10^{-4} & -2.6998 & -2.8002 \times 10^{-4} & -1.7997 & -1.2010 \times 10^{-4} & -0.2802 \end{bmatrix}$$

2.2.4 Use the Augmented Lagrange Multiplier Method for inequality-constrained problems. Comment on values used for c_j, if any. In the script for the pseudo-objective function, you will need to properly update the Lagrange multipliers and use Fletcher's substitution.

The scaling was performed with the below vector.

$$c_j = \begin{bmatrix} 1 \times 10^{-7} & 1 \times 10^{-3} & 10 & 10 & 100 & 100 & 0.01 & 10 \end{bmatrix}$$

This was then run using the files listed below, and the results tabulated in table 5.

```
function A = sumt_phi_alm(x,r_p,lambda)

% compute values of the objective function and constraints at the current
% value of x
f = sumt_fun(x);
g = sumt_con(x);
psi = zeros(size(g));

% Fletcher's substitution
for j = 1:length(g)
    psi(j) = max(g(j), -lambda(j) / (2 * r_p));
end
%psi(1) = max(g(1), -lambda(1) / (2 * r_p));
%psi(2) = max(g(2), -lambda(2) / (2 * r_p));

% Augmented Lagrangian function
A = f + sum(lambda .* psi) + sum(r_p * psi.^2);
```

20C__Users_Alex_Desktop_aae550_hmwk_1_src_sumt_phi_alm.m

```
clear all
clc
format long     % use format long to see differences near convergence

table = [];

x0 = [0.15; 0.015];    % initial design
p = 0;          % initial value of minimization counter
gamma = 5;
g = sumt_con(x0);
r_p = 1;
lambda = zeros(size(g));
for j = 1:length(lambda)
        lambda(j) = lambda(j) + 2*r_p*(max([g(j); -lambda(j)/(2*r_p)]));
end


% compute function value at x0, initialize convergence criteria
```

Table 5: Minimization Table and Resulting Scaled Constraint Values for Augmented Lagrange Multiplier Method

p	r_p	R	t	$f(\vec{x}^*)$	$\phi(\vec{x}^*)$	N	exitflag
0	1	1.51377×10^{-08}	-1.335	-218362	-200486	201	0
1	5	$4.84276 \times 10^{+07}$	$-2.4674 \times 10^{+07}$	$-3.67396 \times 10^{+20}$	$-3.35783 \times 10^{+20}$	30	-3
2	25	$4.84276 \times 10^{+07}$	$-2.4674 \times 10^{+07}$	$-3.67396 \times 10^{+20}$	$-1.46105 \times 10^{+20}$	3	-3
3	125	$-1.70934 \times 10^{+07}$	$3.88408 \times 10^{+06}$	$-1.81174 \times 10^{+19}$	$-3.04392 \times 10^{+19}$	60	1
4	625	$2.60822 \times 10^{+06}$	-219195	$-1.45981 \times 10^{+17}$	$-1.8046 \times 10^{+18}$	57	1
5	3125	-27543.7	5442.48	$-4.03627 \times 10^{+13}$	$-2.62532 \times 10^{+15}$	54	1
6	15625	138.631	-12.2241	$-4.33571 \times 10^{+08}$	$-3.62183 \times 10^{+11}$	39	1
7	78125	0.0492297	0.0223746	208.577	$-2.20367 \times 10^{+11}$	42	1
8	390625	0.113852	0.0115106	304.898	$-2.20364 \times 10^{+11}$	30	1

$$\vec{cg}(\vec{x}^*) = \begin{bmatrix} -24.0406 & -1.0648 \times 10^{7} & -0.8384 & -1.8616 & -0.9512 & -0.8488 & -0.0511 & -1.0233 \end{bmatrix}$$

```
f = sumt_fun(x0);
f_last = 2 * f;   % ensure that first loop does not trigger convergence
% set optimization options - use default BFGS with numerical gradients
% provide display each iteration
options = optimset('LargeScale', 'off', 'Display', 'iter');

% begin sequential minimizations  - note tolerances chosen here
% absolute tolerance for change in objective function, absolute tolerance
% for constraints
while ((abs((f-f_last)/f_last) >= 1e-3) || (max(g) >= 1e-5))
    f_last = f;  % store last objective function value
    p            % display current minimization counter
    r_p          % display current penalty multiplier
    % call fminunc - use "phi" pseudo-objective function, note that r_p is
    % passed as a "parameter", no semi-colon to display results
    [xstar,phistar, exitflag, output] = fminunc(@sumt_phi_alm,x0,options,r_p,lambda)
    % compute objective and constraints at current xstar
    f = sumt_fun(xstar);
    g = sumt_con(xstar);
    N = output.funcCount;
    table = [table; p, r_p, xstar(1), xstar(2), f, phistar, N, exitflag];
    p = p + 1;      % increment minimization counter
    for j = 1:length(g)
        lambda(j) = lambda(j) + 2*r_p*(max([g(j); -lambda(j)/(2*r_p)]));
    end
    r_p = r_p * gamma; % update rpp
    x0 = xstar;     % use current xstar as next x0
end
% display function and constraint values at last solution
f = sumt_fun(xstar)
g = sumt_con(xstar)
format short
header = {'$p$', '$r_{p}$', '$R$', '$t$', ...
          '$f\left(\vec{x}^{*}\right)$', ...
          '$\phi\left(\vec{x}^{*}\right)$', '$N$', 'exitflag'};
matrix2lyx(table,'alm.lyx',header);
```

21C__Users_Alex_Desktop_aae550_hmwk_1_src_sumt_alm.m

2.3 Compare the total number of unconstrained minimizations and iterations needed for each method (i.e. which method used the most; which, the fewest?). Also, compare the solutions (i.e. are they all the same? Did one method find a lighter mass, feasible design?). Which of the methods was easiest for you to implement and use? Can you make any conclusions about the different penalty methods for this problem?

The interior penalty method used the least amount of iterations, with only 3 before converging. This was also the easiest to use for me, as the other two closest (the exterior penalty method and the extended linear penalty method) took significant amount of modification of the penalty multiplier for them to converge. The augmented Lagrange Multiplier Method found a different final solution, with a much heavier mass ($\sim 300\,\text{kg}$ vs. $14\,\text{kg}$). It also took very many function evaluations. The exterior penalty method and the augmented lagrange method also both started off with infeasible cases before finding a case which took $N > 200$ iterations to minimize, before then converging to a solution. These seem to be senstitive to the starting conditions in those cases.