Assignment, Module III:
Shape optimization of a simply supported beam

Hand in a written report containing a short presentation of the problem, results, discussion, and
source code. Remember to answer all questions and comment your results.

Each student should hand in individual solutions; state name and personal number. Discussions
between students are encouraged, but write your solutions and programs individually. Do not copy
solutions from others. I prefer answering questions by phone (instead of email). I do not wish to
debug your code. It is easiest to reach me on my cellular 070-732 8111

The Matlab m-files assemble.m, force.m, compl.m, and grad.m, available for download on
the home page, are necessary for this assignment.

The problem
A simply supported beam (a beam whose vertical displacement but not moment is fixed at the end-
points) with variable thickness \(0 \leq \phi(x)\) is subject to a uniform load \(f\) (figure i); without loss of
generality, we may assume that \(f \equiv 1\). Given a given maximum weight of the beam, the problem is to
determine the thickness distribution \(\phi\) such that the beam is as stiff as possible.

A simple model for the vertical displacement \(u(x)\) is given by the boundary-value problem

\[
\begin{align*}
(\phi^3 u''')'' &= f \quad \text{in } (0,1) \\
u(0) &= u(1) = 0 \\
\phi^3(0)u''(0) &= \phi^3(1)u''(1) = 0
\end{align*}
\]

For a given vertical displacement \(u\) and load \(f\), the compliance of the beam is defined by

\[
J(u) = \int_0^1 uf \, dx.
\]

The lower the compliance, the stiffer the beam. The above design problem can thus be stated as the
optimization problem

\[
\begin{align*}
\text{minimize}_{\phi \in L^\infty((0,1))} J(u) \text{ subject to} \\
(\phi^3 u''')'' &= f \quad \text{in } (0,1) \\
u(0) &= u(1) = 0 \\
\phi(0)^3 u''(0) &= \phi(1)^3 u''(1) = 0 \\
\int_0^1 \phi \, dx &\leq \gamma \\
0 &\leq \phi \leq \overline{\phi}
\end{align*}
\]

(state equation)

(weight constraint)

(feasibility)

where \(\phi\) and \(\overline{\phi}\) are lower and upper bounds on the beam thickness. For \(\phi = 0, \overline{\phi} = +\infty, \gamma = 1\), the
solution to problem (P) is simply

\[
\phi(x) = \frac{8}{\pi} \sqrt{x(1-x)}.
\]

For the finite-element approximation of the state equation, the unit interval is divided into \(n_{el}\)
subintervals, each of length \(h = 1/n_{el}\). The interval will thus contain \(n_{el} + 1\) grid points, including the
boundaries $x = 0, 1$. Since the state equation is a fourth-order equation, we use the space $V^h_0$ of continuously differentiable ($C^1([0,1])$) functions that are third-order polynomials on each subinterval (that is, cubic splines) and vanish at $x = 0$ and 1. The degrees of freedom for each $\psi_h \in V^h_0$ are the values and the derivatives of $\psi_h$ at each internal grid point and the derivatives of $\psi_h$ at the boundary points (the values at the end points are known and not stored). Totally, there are thus $2n_d$ degrees of freedom to represent each $\psi_h \in V^h_0$. Let $\nu$ be a Matlab column vector representing $\psi_h$. The supplied Matlab files assume the following order of the degrees of freedom: $\nu(1) = \psi'_h(0)$, $\nu(2) = \psi_h(h)$, $\nu(3) = \psi'_h(h)$, $\nu(4) = \psi_h(2h)$, ..., $\nu(2n_\text{nel}-2) = \psi_h(1-h)$, $\nu(2n_\text{nel}-1) = \psi'_h(1-h)$, $\nu(2n_\text{nel}) = \psi'_h(1)$. The values of $\psi_h$ at the internal node points can thus be extracted from $\nu(2:2:2n_\text{nel}-2)$.

The thickness function $\phi$ is approximated with a function $\phi_h$ that is piecewise constant on each subinterval. The degrees of freedom for $\phi_h$ is thus its value on subintervals $1, \ldots, n_\text{nel}$.

The finite-element approximation of the state equation leads to a linear system $Au = b$ where the entries of matrix $A$ depends on the thickness function $\phi_h$. The Matlab function assemble, contained in the file assemble.m, returns the stiffness matrix $A$ and the load vector $b$. assemble requires a callable function force with a single input argument $x$ that returns the force at the coordinate $x$; force is also available.

The function compl, contained in the file compl.m, computes the compliance by Gaussian quadrature of integral (2), and grad returns the gradient of the compliance with respect to the thickness distribution.

**Tasks**

We will in this assignment solve the discretized version of problem (P) using the routine fmincon in Matlab’s Optimization Toolbox. Read the documentation by typing doc fmincon (or help fmincon) at the Matlab prompt. The variable supplied as the last input argument to fmincon specifies the desired options for the algorithm. This variable is set by calling the function optimset. The following options are recommended throughout the assignment:

```matlab
options = optimset('DerivativeCheck','off', 'Hessian','off', ...
                   'LargeScale','off');
```

These options turn off an initial check of the accuracy of supplied derivatives, tells that analytic Hessians are not available, and selects the medium-scale version of the algorithm (which is an SQP method). More options are sometimes needed, as discussed below. You can use the general form

```
```
\[
[\text{phi\_numopt}, fval, \text{exitflag}, \text{output}] = \ldots
\]
\[
fmincon(@\text{fun}, \text{phi\_0}, A, b, Aeq, beq, \text{lb}, \text{ub}, [], \text{options});
\]
when calling \texttt{fmincon}. The \texttt{f} in front of \texttt{fun} indicates a \textit{function handler}, and tells \texttt{fmincon} to use the Matlab function \texttt{fun} to evaluate function values and possibly gradients. Read the documentation to find out the meaning of the other arguments. Note that the input arguments that are not needed can be defined as the empty matrix \([]\).

1. Write a Matlab function \texttt{fun} that returns the compliance and the gradient of the compliance for a given input thickness distribution. You may use functions \texttt{compl} and \texttt{grad}, available in the files \texttt{compl.m} and \texttt{grad.m} to compute the compliance and the gradient. To avoid unnecessary computations, use the build-in Matlab function \texttt{nargout}:

\[
\begin{align*}
\text{function} \ [c, gc] &= \text{fun}(\phi) \\
\text{...} \\
c &= \ldots & \% \text{Compute the objective function value at } \phi \\
\text{if} & \ nargout > 1 & \% \text{fun called with two output arguments} \\
\text{...} \\
gc &= \ldots & \% \text{Gradient of the function evaluated at } \phi \\
\end{align*}
\]

Also define the matrices and vectors that are needed for the weight and feasibility constraints; use here \(\bar{\phi} = 0\), \(\bar{\phi} = +\infty\), and \(\gamma = 1\) so you can compare with the exact solution (3). Use \(\phi_0 = \text{ones}(\text{nint}, 1)\) as initial guess.

Numerically solve problem (P) with \texttt{fmincon}. To specify that analytical derivatives should be used, specify \'(GradObj', 'on'\) when setting the variable option.

For each of the optimizations you run, record carefully the number of function evaluations and, separately, the number of gradient evaluations. An easy way to obtain these numbers is to update within the function \texttt{fun} two counters, defined as \textit{global variables}. See the Help in Matlab (for instance by typing \texttt{help global} at the Matlab prompt) for the syntax of global variables. Also note the information provided in structure \texttt{output} returned by \texttt{fmincon}.

Start with a crude discretization, say \(\text{nint} = 15\) and refine a few times, up to at least \(\text{nint} = 100\). Plot the optimal thickness distribution together with the exact solution (3) and record the optimal compliance. For the finer discretization, it may be necessary to sharpen the default convergence tolerance by specifying \'(To1Fun', \(\text{tol}\), where \(\text{tol}\) is a small positive number; good values are from \(\text{tol} = 1e-6\) and a few orders-of-magnitude smaller.

2. Now assume that you do not have access to \texttt{grad.m}. Implement a Matlab function \texttt{gradCS} that returns a gradient approximated by the "complex step", that is, finite differences in the imaginary direction. Run the same optimizations as in Part 1, replacing \texttt{grad} with \texttt{gradCS}, compare the results and the number of function evaluations.

3. Even easier than using the complex step is to use the built-in finite-difference gradient approximations in \texttt{fmincon}. Repeat the computations in Part 1 and 2 but with the option \'(GradObj', 'off'\). Then \texttt{fmincon} will only use function evaluation.

4. Now assume that we do not accept too thin or thick beams. Solve problem (P) with \(\phi = 0.6\) and \(\bar{\phi} = 1.2\). Solve the problem for a crude and for a fine (at least \(n_{el} = 100\)) discretization using analytical gradients, complex-stepped gradients, and the build-in finite difference gradients.

5. \textit{Challenge problem, non-mandatory}: Derive the exact solution (3) for \(\phi = 0\), \(\bar{\phi} = +\infty\), and \(\gamma = 1\).