Exercises with 'An 88(/99) line topology optimization code written in MATLAB' *

Ole Sigmund, Federico Ferrari, Niels Aage, Rasmus E. Christiansen,
Andreas H. Frederiksen and Casper S. Andreasen
Department of Civil and Mechanical Engineering, Solid Mechanics, Building 404

Technical University of Denmark
DK-2800 Lyngby, Denmark

June 10, 2025

1 Introduction

The following exercises are mainly based on the paper *Efficient topology optimization in MATLAB using 88 lines of code* (Andreassen et al. 2011). Both the source code and a preprint of the paper can be downloaded from topopt.dtu.dk. The exercises require some basic knowledge of mechanics, Finite Element analysis and optimization theory. For groups with less MATLAB and Finite Element experience, the less efficient[1] but more transparent 99 line code (Sigmund 2001) may also be used as a basis for the exercises.

You should solve at least problems 1-8 during the course. If you have previous experience you may skip problems 1-3 and instead move on to solve some of the more complex problems 9 through 18 in section 4. For the interested students there is also the possibility to work with large scale topology optimization using the C++ TopOpt_in_PETSc framework, see topopt.dtu.dk/PETSc. This will however require that you have significant prior experience with programming languages such as C/C++ or Fortran.

Your solutions must be presented on a poster at the presentations of exercise work session. A copy of the poster must be handed over to Ole Sigmund after the session on Tuesday June 27th. The evaluation procedure will be explained in detail at the lectures. We recommend that you document your findings by saving different versions of your programs and store key figures in a slide show already from the beginning.

The papers (Andreassen et al. 2011, Sigmund 2001) (see also the appendix in Bendsøe and Sigmund (2004)) give a lot of hints on how to solve problems 1 through 8.

From the file sharing pages on the course page on DTU-Learn (learn.inside.dtu.dk) you can download relevant MATLAB subroutines and other material during the course.

*Intended for the DCAMM-course: *Topology Optimization - Theory, Methods and Applications* held at DTU, Lyngby, Denmark, June 11th – June 17th, 2025.

[1] A speed quick-fix will be provided

2 Getting started

- Form a group of (preferably 2) students and login to the DTU system with your student/guest account. Try to make sure that at least one group member has a mechanical engineering background and at least one has MATLAB programming experience. Also try to hook up with somebody you do not already know - networking is important!
- Go to the TopOpt web-page: topopt.dtu.dk
- Click "Apps/software" and nagivate to "MATLAB® codes for minimum compliance problems" and click "Optimized 88-line Matlab code"
- Read the text on the page and download the MATLAB code `top88.m` (or `top.m`) to your home directory "My Documents"
- Start MATLAB and set the path to "My Documents"
- Run the default MBB-example by writing `top88(300,100,0.5,3.0,1.1,1)` in the MATLAB command line
- Save the original version of the MATLAB code and start editing new versions
- Experiment with the code
- Extensions to the MATLAB code (a few lines of code) which create a displacement picture; the equations for the strain-displacement matrix; and a more efficient sparse assembly strategy (for the 99-line code); are given in appendices A, B and C. You can also download these from the course webpage on `https://learn.inside.dtu.dk/` along with a tool that plots nodes, forces and boundary conditions.

3 Recommended problems

By default, the downloaded MATLAB code solves the problem of optimizing the material distribution in the MBB-beam such that its compliance is minimized.

Problem 1: Test the influence of discretization, filter type, filter size and penalization power

Use the MATLAB code to investigate the influence of the filter type (sensitivity filter in 99 line code; sensitivity and density filters in 88 line code), filter size `rmin`, the penalization power `penal` and the discretization (`nelx*nely`) on the topology of the MBB-beam (default example). Discuss the results.

Problem 2: Implement other boundary conditions / loadings

Change boundary and support conditions in order to solve the optimization problems sketched in Figure 1. Does the direction of the forces change the design?

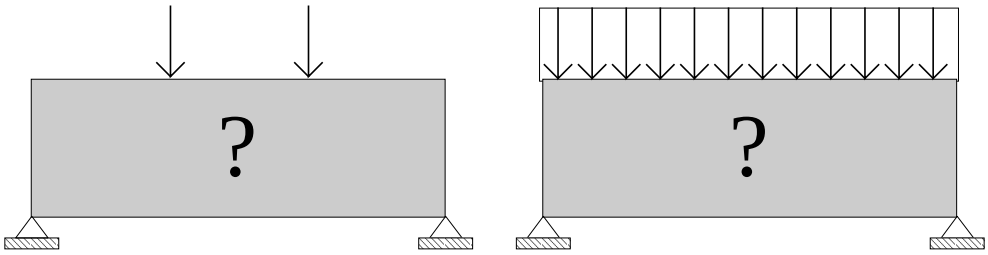

Figure 1: Topology optimization of a "bridge" structure. Left: two (simultaneous) point loads and right: distributed load.

Problem 3: Implement multiple load cases

Extend the algorithm such that it can solve the two-load case problem shown in Figure 2. Discuss the difference in topology compared to the one-load case solution from Figure 1(left).

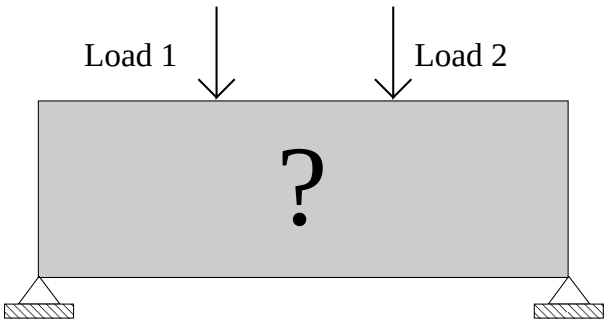

Figure 2: Topology optimization of a bridge structure with two load cases.

Problem 4: Implement passive elements

In some design cases, specific areas may be required to take the minimum density value, e.g. a hole for a pipe, or the maximum density value, e.g. the deck of a bridge. Add the necessary lines to the program such that the problem shown in Figure 3 can be solved. What is the difference in compliance compared to the solution to Problem 2(left)?

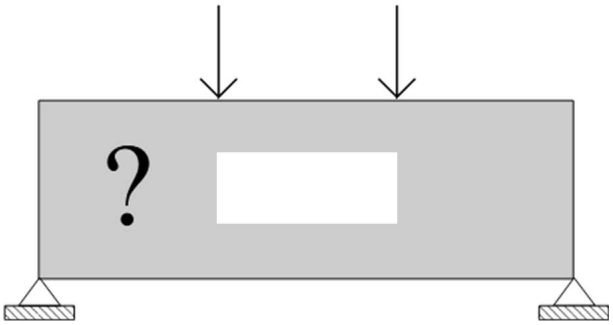

Figure 3: Topology optimization of a bridge structure with two loads and a centered area prescribed to be void.

Problem 5: Method of Moving Asymptotes (MMA)

Krister Svanberg from KTH in Stockholm Sweden has written an optimization routine called Method of Moving Asymptotes (MMA) (Svanberg 1987). Rewrite the topology optimization code such that it calls the MMA MATLAB routine instead of the Optimality Criteria Solver. Use the developed software to optimize the default MBB-beam and possibly other examples. How does it compare to the OC solver?

The MMA routines mmasub.m and subsolv.m can be downloaded from the file sharing pages on the course page on Learn together with the documentation for the MATLAB MMA code. The program solves the following optimization problem

$$\left.\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{y},z} \;:\; & f_0(\boldsymbol{x}) + a_0\, z + \sum_{i=1}^{m}(c_i y_i + \frac{1}{2} d_i y_i^2) \\ \text{subject to}: \; & f_i(\boldsymbol{x}) - a_i\, z - y_i \le 0, \qquad i = 1,\ldots,m \\ : \; & x_j^{min} \le x_j \le x_j^{max}, \qquad j = 1,\ldots,n \\ : \; & y_i \ge 0, \qquad i = 1,\ldots,m \\ : \; & z \ge 0 \end{aligned}\right\}, \tag{1}$$

where m and n are number of constraints and number of design variables respectively, $\mathbf{x}$ is the vector of design variables, $\mathbf{y}$ and z are positive optimization variables, f_0 is the objective function, $f_1,\ldots,f_m$ are the constraint functions ($f_i(\boldsymbol{x}) \le 0$) and a_i, c_i and d_i are positive constants which can be used to determine the type of optimization problem.

If we want to solve the simpler problem

$$\left.\begin{aligned} \min_{\boldsymbol{x}} \;:\; & f_0(\boldsymbol{x}) \\ \text{subject to}: \; & f_i(\boldsymbol{x}) \le 0, \qquad i = 1,\ldots,m \\ : \; & x_j^{min} \le x_j \le x_j^{max}, \qquad j = 1,\ldots,n \end{aligned}\right\}, \tag{2}$$

we must make sure that the optimization variables $\mathbf{y}$ and z from the original optimization problem (1) are equal to zero at optimum. This is obtained if we select the constants to the following values (suggested by Krister Svanberg)

$$a_0 = 1,\; a_i = 0,\; c_i = 1000,\; d = 0. \tag{3}$$

The call of the MMA routine requires the determination of sensitivities `df0dx`, `df0dx2`, `dfdx`, `dfdx2` corresponding to the derivatives $\frac{\partial f_0}{\partial x_j}$, $\frac{\partial^2 f_0}{\partial x_j^2}$, $\frac{\partial f_i}{\partial x_j}$ and $\frac{\partial^2 f_i}{\partial x_j^2}$, respectively. Since second

derivatives are difficult to determine in topology optimization problems, we simply set them to zero.

The MMA call is

```
function [xmma,ymma,zmma,lam,xsi,eta,mu,zet,s,low,upp] = ...
    mmasub(m,n,iter,xval,xmin,xmax,xold1,xold2, ...
    f0val,df0dx,df0dx2,fval,dfdx,dfdx2,low,upp,a0,a,c,d);
```

Hints:

- *Check the MMA documentation or the* `mmasub.m` *file for explanations and definitions of the MMA variables*
- *Watch out for the difference between column and row vectors*
- *Remember to normalize constraints and objective function, i.e. instead of* $V(\mathbf{x}) - V^* \leq 0$ *use* $V(\mathbf{x})/V^* - 1 \leq 0$
- *In order to convert a MATLAB matrix to a MATLAB vector you may make use of the MATLAB command* `reshape` *(type* `help reshape` *in the MATLAB prompt to get help on* `reshape`)

Problem 6: Implementing a sensitivity check

When performing gradient-based optimization it is essential to ensure high-quality gradient information. To check that a gradient calculation is performed correctly, one may compare it to a *finite-difference* approximation of the gradient. In this exercise the code is to be extended to include a *finite-difference* (sensitivity) check.

A first order finite-difference approximation of the gradients can be obtained by perturbing a single design variable by a small amount e.g. $\delta_p = 0.01$ and computing the corresponding objective $\Phi_{p,e}$ i.e.

$$\frac{\mathrm{d}\Phi}{\mathrm{d}x_e} \approx \frac{\Phi_{p,e} - \Phi}{\delta_p} \tag{4}$$

To perform the sensitivity check the finite difference approximation in (4) is to be compared with the gradient computed using the adjoint method. Note that the absolute error may be small even if the gradient computation is wrong. A convergence analysis will reveal this. Thus, it is your task to compute the finite-difference approximation for a vanishing series of perturbations, e.g. $\delta_p = 10^s$ for $s \in \{-1, -2, ..., -6\}$ and compute the relative error to the gradient obtained using the adjoint method. Show a plot of the relative error for the gradient of a single design variable vs. the perturbation size, which should display a slope of 1 in a log-log plot (rel.err. vs. δ_p), until machine precision is reached.

Hints:

- *The finite difference check (and the MMA method) can be implemented as a switch in the code. By introducing a variable to select the method e.g.* `method="fdcheck"` *one can select the updating mechanism*:

```
switch method
  case "oc"
    % put here the OC update
    ...
```

```
  case "mma"
    % put here the MMA update
    ...
  case "fdcheck"
    % Finite difference check
    ...
  otherwise
    error("Method not implemented");
end
```

- *The original loop can be used to drive the perturbation towards zero e.g.* `delta_p=power(10,-loop)`
- *In the first loop you should compute the unperturbed objective value and save the initial design, the objective value and the sensitivity vector for use in the subsequent loops.*
- *Remember to reset the design variables to the original values between each new perturbation* (`delta_p` *value*).

Problem 7: Mechanism synthesis

Extend the MATLAB code to include compliant mechanism synthesis. Solve the inverter problem in Figure 4.

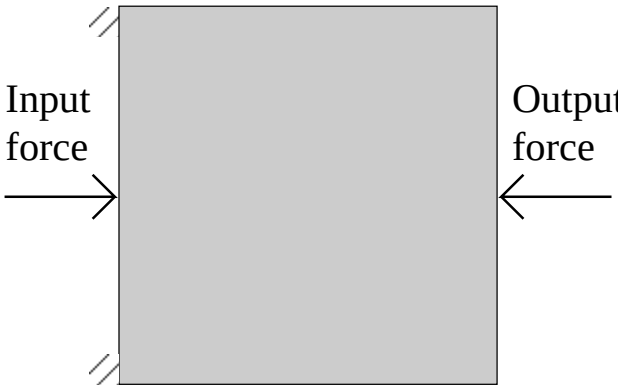

Figure 4: Synthesis of compliant inverter mechanism.

The simplest compliant mechanism design formulation is one with springs at both input and output ports. The input spring stiffness together with the input force constitute a model of so-called strain-based actuators, such as piezo-electric or shape memory alloy (SMA) actuators. The input force is given by the actuator blocking force and the spring stiffness is given by the actuator stiffness or the ratio between the blocking force and the unloaded actuator displacement. The resulting optimization problem is formulated as

$$\left.\begin{aligned} \min_{\boldsymbol{x}} \;&:\; u_{out} \\ \text{subject to}:\; &V(\mathbf{x}) = \mathbf{v}^T\,\mathbf{x} \leq V^* \\ &:\; \mathbf{K}\mathbf{u} = \mathbf{f}_{in} \\ &:\; 0 < x_j^{min} \leq x_j \leq 1, \qquad j = 1,\ldots,n \end{aligned}\right\}, \tag{5}$$

A slightly more complicated and less physical formulation is the one from Sigmund (1997), where the input spring/force model is substituted with an input displacement constraint $u_{in} \leq u_{in}^*$. One can here use the same input force as for the double spring model above but the input spring is removed from the model. As initial value for the input displacement constraint one can use the input displacement obtained after the optimization of the double spring problem. The inclusion of an additional constraint is a good starting exercise when learning to solve topology optimization problems with multiple constraints.

Hints:

- *The input displacement is found as* $u_{in} = \mathbf{u}^T\mathbf{l}_{in}$ *where* $\mathbf{l}_{in}$ *is a unit vector which has the value one at the input degree of freedom and is zero for all other degrees of freedom.*
- *The output displacement is found as* $u_{out} = \mathbf{u}^T\mathbf{l}_{out}$ *where* $\mathbf{l}_{out}$ *is a unit vector which has the value one at the output degree of freedom and is zero for all other degrees of freedom.*
- *Use the adjoint method to determine sensitivities*
- *The input and output springs are added to the analysis by adding the respective spring stiffnesses at the positions of the input and output degrees of freedom in the global stiffness matrix, respectively.*
- *Make use of symmetry*
- *Add passive domains*
- *Check the correctness of the derivatives of objective function and constraints by the finite difference method for a number of different elements. When using a sensitivity filter, remember to turn off the filter during this test.*
- *In order to stabilize convergence you may change the values of* `asyincr` *and* `asydecr` *in the* `mmasub.m` *routine to 1.07 and 0.65, respectively (strategy for moving of asymptotes). You may also introduce external fixed move limits, i.e.* $xmin = max(0, x - move)$*, etc.*
- *Start with Young's modulus E=100 and spring stiffnesses and input forces unity.*

Problem 8: Optimization with time-harmonic loads

Extend your code to be able to optimize to control and manipulate undamped forced vibrations. Assuming a time-harmonic load of angular frequency ω, the FE equation system is changed from $\mathbf{Ku} = \mathbf{f}$ to:

$$\mathbf{Su} = \mathbf{f}, \tag{6}$$

where $\mathbf{f}$ now represents the amplitude of the load and $\mathbf{u}$ the amplitude of vibration. $\mathbf{S}$ is the system matrix (or "dynamic stiffness" matrix) defined as

$$\mathbf{S} = \mathbf{K} - \omega^2\mathbf{M}, \tag{7}$$

where $\mathbf{M}$ is the global mass matrix.

Optimize the MBB-example for minimum dynamic compliance using the objective $|\mathbf{u}^T\mathbf{f}|$ for a specific excitation frequency ω. The necessary expressions for the sensitivity calculations can be found in Jensen (2017). Complete the following steps:

1. Use an MBB-beam optimized for static compliance as the initial design.
2. Plot the frequency response (objective values vs frequency) of this design on a log y-scale.

3. Choose the optimization frequency, ω_t, below the first resonance frequency, ω_1, (e.g. $\omega_t = 0.9\ \omega_1$) and optimize the MMB-beam for dynamic compliance at ω_t.

4. Compute the frequency response of the optimized structure and compare it to the one obtained for the initial structure.

5. Try also an optimization frequency closer to or above the first resonance frequency - comment on your observations.

Hints:

- *Assemble the mass matrix* $\mathbf{M}$ *using the local mass matrix found in appendix B.*
- *Use a linear interpolation for the mass.*
- *Use unit mass density of the material: rho0=1 and a minimum value in the void given as: rhomin=1e-8. Change the minimum stiffness to Emin=1e-4.*
- *Use MMA (with movelimits) as optimizer.*
- *Use the density filter (note: the 99 line code does not have the density filter implemented by default).*

4 Voluntary MATLAB problems

Problem 9: Min-Max formulation of multiple load cases

Minimize the maximum compliance of a three load case problem. This problem can be solved by fiddling with the constants in the MMA optimizer.

An optimization problem looking like

$$\left.\begin{array}{rcll} \min\limits_{\boldsymbol{x}} & : & \max\limits_{k=1,.....,p} \{|h_k(\boldsymbol{x})|\} & \\ \text{subject to} & : & g_i(\boldsymbol{x}) \leq 0, \qquad i = 1, \ldots,\ q & \\ & : & x_j^{min} \leq x_j \leq x_j^{max}, \qquad j = 1, \ldots, n & \end{array}\right\}, \tag{8}$$

can be re-written to

$$\left.\begin{array}{rcll} \min\limits_{\boldsymbol{x},\ z} & : & z & \\ \text{subject to} & : & h_i - z \leq 0, \qquad i = 1, \ldots,\ p & \\ & : & -h_i - z \leq 0, \qquad i = 1, \ldots,\ p & \\ & : & g_i(\boldsymbol{x}) \leq 0, \qquad i = 1, \ldots,\ q & \\ & : & x_j^{min} \leq x_j \leq x_j^{max}, \qquad j = 1, \ldots, n & \end{array}\right\}, \tag{9}$$

which may be solved by MMA using the constants

$$\begin{aligned} m &= 2p+q \\ f_0(\mathbf{x}) &= 0 \\ f_i(\mathbf{x}) &= h_i(\mathbf{x}), \qquad i=1,\ldots,\ p \\ f_{p+i}(\mathbf{x}) &= -h_i(\mathbf{x}), \qquad i=1,\ldots,\ p \\ f_{2p+i}(\mathbf{x}) &= g_i(\mathbf{x}), \qquad i=1,\ldots,\ q \\ a_0 &= 1 \\ a_i &= 1, \qquad i=1,\ldots,2p \\ a_{2p+i} &= 0, \qquad i=1,\ldots,q \\ d_i &= 0, \qquad i=1,\ldots,m \\ c_i &= 1000, \qquad i=1,\ldots,m \end{aligned} \tag{10}$$

If not minimizing the absolute value $|h_i|$ but just h_i the second set of inequalities in (9) can be left out.

Note that z is always strictly positive. This means that h_k has to be augmented to $h_k + c$ if h_k takes negative values, where c is a sufficiently large positive constant.

Problem 10: Robust topology optimization

Implement the robust design formulation Sigmund (2009), Wang et al. (2011) and test it on the force inverter problem from course Problem 5. The 88-line code already has the Heaviside projection filter built-in so the main challenge consists in implementing the min-max formulation (like in course Problem 9) and solving for the three geometry cases in each iteration.

Problem 11: Optimizing under uncertainty - Stochastic Gradient

For some applications, fabrication or operational uncertainties are significant and should be taken into account in the structural design process. To do so, one may employ a stochastic method. In this exercise, we consider the method proposed and educational code provided in A. Uihlein (2025).

Consider again the example from Problem 2 (Fig. 1 (right)). Instead of the fixed distributed load, we now assume a point load placed at a random position $x_{load}(\psi)$ along the top boundary. The position is drawn from a truncated normal distribution, $\psi \sim \mathcal{N}(\frac{1}{2}, 0.15)$, with 0 indicating the leftmost and 1 the rightmost position. We now desire to minimize the expected compliance of the structure under random loading, denoted $\mathbb{E}_\psi[c(\rho,\psi)]$. Use the provided `topS140_load` code (utilizing the MATLAB function `fsparse`) and scripts `samples_DCAMM` and `script_DCAMM` to answer the following questions:

1. Familiarize yourself with the `topS140_load` code by executing the script `script_DCAMM` and observe the result.

2. For the optimized structure found by solving Problem 2 with a fixed distributed load, evaluate the expected compliance of this structure for the random loading described above.

3. Adjust the `topS140_load` code to consider the current problem settings (boundary conditions, forcing and stochastic data generation according to the normal distribution) and use the code to optimize a structure for the expected compliance. Compare the obtained expected compliance to the one computed in 2. above.

4. Adjust the `topS140_load` code to plot the nearest neighbor model in each iteration (see lecture slides) and solve the problem again, observing how the nearest neighbor model changes during the optimization.

   - *Hint*: The integration points, `y`, versus the compliance, `ComH`, at these points, `csw`.

5. Compare two different sampling strategies of the random field, by solving the problem again with uniform random sampling of the load position. How does the optimization process and results differ with the truncated normal-distribution-based sampling. (Note: both sampling strategies are already implemented in `topS140_load` and are set through `'type'`)

6. Solve the "exact" problem deterministically using a large number of load positions (multiple load cases) along the top boundary to compute the expected compliance in each design iteration. Comment on:

   (a) The performance differences between the stochastic and the "exact" approach.

   (b) The feasibility of the "exact" approach for higher resolution and/or 3D problems.

7. **Optional:** By setting `maxsmpl=1` and `type='distribution'`, the provided code acts like a combination of OCM with standard SG. Compare this setting to the others used above.

8. **Challenge:** Explore if better ways exists for generating the sampling sequence.

Problem 12: Mechanisms with multiple outputs

Design an "elevator mechanism" as shown in Figure 5 (the platform must remain horizontal during elevation) or a gripping mechanism with parallel moving jaws.

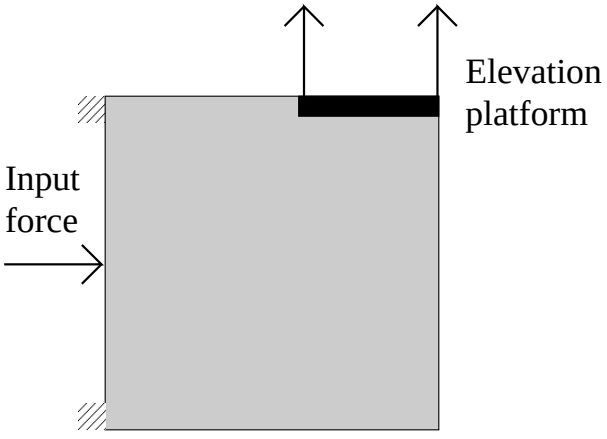

Figure 5: Mechanism synthesis for an "elevator mechanism". The platform must remain horizontal during elevation.

Problem 13: Stress constraints

Implement stress constraints for a problem of your choice. Use the general formulation:

$$\begin{aligned} \min_{\boldsymbol{\rho}} &: \Phi(\boldsymbol{\rho}) \\ \text{s.t.} &: \mathbf{K}(\boldsymbol{\rho})\mathbf{u} = \mathbf{f} \\ &: \sum_{e=1}^{N} v_e \rho_e = \mathbf{v}^T \boldsymbol{\rho} \leqslant V^* \\ &: ||\boldsymbol{\sigma}_{VM}(\boldsymbol{\rho})||_P \leqslant \sigma_y^* \\ &: \mathbf{0} < \boldsymbol{\rho}_{\mathbf{min}} \leqslant \boldsymbol{\rho} \leqslant \mathbf{1}, \end{aligned} \tag{11}$$

where the global Von-Mises stress measure is defined as

$$||\boldsymbol{\sigma}_{VM}(\boldsymbol{\rho})||_P = \left(\sum_{e=1}^{N} v_e \sigma_{VM,e}^P \right)^{\frac{1}{P}} \tag{12}$$

An implementation for a mechanism design problem (including sensitivity analysis and appropriate density interpolation schemes) can be found in Leon et al. (2015).

Problem 14: Infill design

Implement a local volume constraint for infill design using the general formulation:

$$\begin{aligned} \min_{\boldsymbol{\rho}} &: \Phi(\hat{\boldsymbol{\rho}}) \\ \text{s.t.} &: \mathbf{K}(\hat{\boldsymbol{\rho}})\mathbf{u} = \mathbf{f} \\ &: \sum_{e=1}^{N} v_e \hat{\rho}_e = \mathbf{v}^T \hat{\boldsymbol{\rho}} \leqslant V^* \\ &: ||\bar{\boldsymbol{\rho}}(\hat{\boldsymbol{\rho}}) - \alpha|| \leqslant \epsilon \\ &: \mathbf{0} < \boldsymbol{\rho}_{\mathbf{min}} \leqslant \boldsymbol{\rho} \leqslant \mathbf{1}, \end{aligned} \tag{13}$$

where $\hat{\rho}$ is the standard (density or filtered physical density) and $\bar{\rho}$ denotes a larger (density) filtering with radius yielding microstructure lengthscale with prescribed local density α.

See more details in Wu et al. (2017).

Problem 15: Interior point method

In Problem 4 you have written a program that calls an optimization routine (MMA in our case) to perform one design update. Most off-the-shelf optimization packages operate in a different way. The software user is required to provide a number of callback functions, which evaluate the value and the gradient of an objective function, values and gradients (Jacobian matrix) of the non-linear constraints, if any are present. Further, some solver may utilize second order derivatives (Hessian matrix) of the objective function and non-linear constraints in order to achieve faster convergence (Newton-like methods), whereas other algorithms may approximate second order derivatives based on the gradient information evaluated at previous iterations (quasi-Newton approaches).

There are at least two possible optimizers that can be compared with MMA:

- `fmincon` which is a part of MATLAB's Optimization Toolbox[2]

[2]Type `help fmincon` or `doc fmincon` to learn more about a multitude of various options and parameters of `fmincon` routine.

- `Ipopt` Interior Point Optimizer (pronounced "Eye-Pea-Opt") is an open source software package for large-scale nonlinear optimization.[3]

Start with a template program `top88_fmincon.m` downloadable from `learn.inside.dtu.dk`, which is simply a re-arranged 88-line topology optimization code written in MATLAB.

Fill in the code lines marked with comments `%FIXIT`, this should give you a compliance minimization code similar to the one in Problem 5, but which is based on the interior-point optimization algorithm with approximate second order information.

Compare the performance of the different optimizers on some benchmark examples (e.g., coming from Problems 1–5). When comparing, pay attention to the number of FE analyzes performed.

Hints:

- *When debugging the script* `top88_fmincon.m` *you may wish to turn on the automatic verification of user supplied derivatives by setting the option* `'Derivative Checking'` *to* `'on'`.
- *You may study how the number of gradient vectors stored by the L-BFGS algorithm affects the total number of optimization iterations and the total time of the solution by adjusting the number after the option* `'lbfgs'`. *More vectors mean potentially better approximation of the Hessian, which comes at some computational cost. However, "too many vectors" mean that the Hessian is approximated using the gradients evaluated far away from the current optimization point, which may in fact be disadvantageous.*
- *By default,* `fmincon` *solves the optimization problem to a very high precision,* $1 \cdot 10^{-6}$. *This precision may be adjusted by setting the optimization parameters* `TolX` , `TolFun` , *and in the presence of non-linear constraints,* `TolCon` .

Problem 16: Three dimensions

Download the 3D code `top3D125.m` based on the `top99neo.m` code (`https://www.topopt.mek.dtu.dk/Apps-and-software/New-99-line-topology-optimization-code-written-in-MATLAB` - by Ferrrari and Sigmund) and implement some of the exercises in this code.

Problem 17: Alternative measures of dynamic compliance

Optimize the MBB-example for time-harmonic loads using an alternative choice of dynamic compliance: 1) dissipated energy in the structure or 2) total energy level in the structure. Compare the optimized structures to the ones obtained in Problem 8.

Problem 18: Others

Look in Bendsøe and Sigmund (2004) and get inspired to solve some other problems like thermal loads, conduction, self-weight etc.

[3]The Ipopt project homepage is `https://coin-or.github.io/Ipopt/`. A MATLAB interface can be found at `https://se.mathworks.com/matlabcentral/fileexchange/53040-ebertolazzi-mexipopt`

References

A. Uihlein, O. Sigmund, M. S.: 2025, A 140 line matlab code for topology optimization problems with probabilistic parameters, *https://arxiv.org/abs/2505.10421* .

Andreassen, E., Clausen, A., Schevenels, M., Lazarov, B. and Sigmund, O.: 2011, Efficient topology optimization in matlab using 88 lines of code, *Structural and Multidisciplinary Optimization* **43**, 1–6. MATLAB code available online at: www.topopt.dtu.dk.

Bendsøe, M. P. and Sigmund, O.: 2004, *Topology Optimization - Theory, Methods and Applications*, Springer Verlag, Berlin Heidelberg.

Jensen, J. S.: 2017, Adjoint sensitivity analysis for linear dynamic systems with time-harmonic excitation, *Report*, Department of Mechanical Engineering, Technical University of Denmark.

Leon, D. D., Alexandersen, J., Fonseca, J. and Sigmund, O.: 2015, Stress-constrained topology optimization for compliant mechanism design, *Struct Multidisc Optim* **52**, 929–943.

Sigmund, O.: 1997, On the design of compliant mechanisms using topology optimization, *Mechanics of Structures and Machines* **25**(4), 493–524.

Sigmund, O.: 2001, A 99 line topology optimization code written in MATLAB, *Structural and Multidisciplinary Optimization* **21**, 120–127. MATLAB code available online at: www.topopt.dtu.dk.

Sigmund, O.: 2009, Manufacturing tolerant topology optimization, *Acta Mechanica Sinica* **25**(2), 227–239.

Svanberg, K.: 1987, The Method of Moving Asymptotes - A new method for structural optimization, *International Journal for Numerical Methods in Engineering* **24**, 359–373.

Wang, F., Lazarov, B. and Sigmund, O.: 2011, On projection methods, convergence and robust formulations in topology optimization, *Structural and Multidisciplinary Optimization* **43**, 767–784.

Wu, J., Aage, N., Westermann, R. and Sigmund, O.: 2017, Infill optimization for additive manufacturing - approaching bone-like porous structures, *IEEE Transactions on Visualization and Computer Graphics* **in press**.

A Appendix: MATLAB extension for plotting displacements

To obtain plots with displacements exchange the line

```
colormap(gray); imagesc(-x); axis equal; axis tight; axis off;pause(1e-6);
```

with the lines

```
%  colormap(gray); imagesc(-x); axis equal; axis tight; axis off;pause(1e-6);
    colormap(gray); axis equal; axis tight; axis off;
    Faces = ((edofMat(:,[1 3 5 7])-1)/2)+1;
    [XGrid,YGrid]=meshgrid(0:nelx,nely:-1:0);
    Grid = [XGrid(:),YGrid(:)];
    Deform = [U(1:2:end,1),U(2:2:end,1)];
    patch('Faces',Faces,'Vertices',Grid+Deform*0.05,'FaceColor','Flat', ...
        'FaceVertexCData',1-x(:),'EdgeColor','none');
    drawnow; clf;
```

Note that the factor 0.05 is a scaling factor that may be freely chosen.

B Appendix: MATLAB mass and strain-displacement matrices for 4-node element

The strain displacement matrix ($\varepsilon = \mathbf{B}\mathbf{u}_\mathbf{e}$)

```
bmat = [-1/2  0    1/2  0   1/2 0   -1/2  0
         0   -1/2  0   -1/2 0   1/2 0     1/2
        -1/2 -1/2 -1/2  1/2 1/2 1/2 1/2  -1/2];
```

and the mass matrix (with total mass equal to unity)

```
m0 = [4/9 0 2/9 0 1/9 0 2/9 0
      0 4/9 0 2/9 0 1/9 0 2/9
      2/9 0 4/9 0 2/9 0 1/9 0
      0 2/9 0 4/9 0 2/9 0 1/9
      1/9 0 2/9 0 4/9 0 2/9 0
      0 1/9 0 2/9 0 4/9 0 2/9
      2/9 0 1/9 0 2/9 0 4/9 0
      0 2/9 0 1/9 0 2/9 0 4/9]/(4*nel);
```

and the constitutive matrix for plane stress

```
Emat = E/(1-nu^2)*[ 1 nu 0
                    nu 1 0
                    0  0 (1-nu)/2];
```

C Appendix: Fast MATLAB sparse assembly

The `top.m` code uses a sparse assembly strategy that is easy to read but highly inefficient for larger problem:

```
K=sparse(2*(nelx+1)*(nely+1), 2*(nelx+1)*(nely+1));
F=sparse(2*(nely+1)*(nelx+1),1);
U=zeros(2*(nely+1)*(nelx+1),1);
for elx = 1:nelx
  for ely = 1:nely
    n1 = (nely+1)*(elx-1)+ely;
    n2 = (nely+1)* elx   +ely;
    edof = [2*n1-1; 2*n1; 2*n2-1; 2*n2; 2*n2+1; 2*n2+2; 2*n1+1; 2*n1+2];
    K(edof,edof) = K(edof,edof) + x(ely,elx)^penal*KE;
  end
end
```

A dramatic speed-up can be obtained (for large problems) if the lines above are replaced by the following lines:

```
I=zeros(nelx*nely*64,1);
J=zeros(nelx*nely*64,1);
X=zeros(nelx*nely*64,1);
F=sparse(2*(nely+1)*(nelx+1),1);
U=zeros(2*(nely+1)*(nelx+1),1);
ntriplets=0;
for elx = 1:nelx
  for ely = 1:nely
    n1 = (nely+1)*(elx-1)+ely;
    n2 = (nely+1)* elx   +ely;
    edof = [2*n1-1 2*n1 2*n2-1 2*n2 2*n2+1 2*n2+2 2*n1+1 2*n1+2];
```

```
    xval = x(ely,elx)^penal;
    for krow = 1:8
        for kcol = 1:8
            ntriplets = ntriplets+1;
            I(ntriplets) = edof(krow);
            J(ntriplets) = edof(kcol);
            X(ntriplets) = xval*KE(krow,kcol);
        end
    end
  end
end
K=sparse(I,J,X,2*(nelx+1)*(nely+1),2*(nelx+1)*(nely+1));
```

D Appendix: MMA MATLAB documentation

You may download the MMA-code from the file sharing pages on Learn (`learn.inside.dtu.dk`).

```
%-----------------------------------------------------
%    This is the file mmasub.m
%
function [xmma,ymma,zmma,lam,xsi,eta,mu,zet,s,low,upp] = ...
mmasub(m,n,iter,xval,xmin,xmax,xold1,xold2, ...
f0val,df0dx,fval,dfdx,low,upp,a0,a,c,d);
%
%    Version September 2007 (and a small change August 2008)
%
%    Krister Svanberg <krille@math.kth.se>
%    Department of Mathematics, SE-10044 Stockholm, Sweden.
%
%    This function mmasub performs one MMA-iteration, aimed at
%    solving the nonlinear programming problem:
%
%      Minimize  f_0(x) + a_0*z + sum( c_i*y_i + 0.5*d_i*(y_i)^2 )
%    subject to  f_i(x) - a_i*z - y_i <= 0,  i = 1,...,m
%                xmin_j <= x_j <= xmax_j,    j = 1,...,n
%                z >= 0,   y_i >= 0,         i = 1,...,m
%*** INPUT:
%
%   m    = The number of general constraints.
%   n    = The number of variables x_j.
%  iter  = Current iteration number ( =1 the first time mmasub is called).
%  xval  = Column vector with the current values of the variables x_j.
%  xmin  = Column vector with the lower bounds for the variables x_j.
%  xmax  = Column vector with the upper bounds for the variables x_j.
%  xold1 = xval, one iteration ago (provided that iter>1).
%  xold2 = xval, two iterations ago (provided that iter>2).
%  f0val = The value of the objective function f_0 at xval.
%  df0dx = Column vector with the derivatives of the objective function
%          f_0 with respect to the variables x_j, calculated at xval.
%  fval  = Column vector with the values of the constraint functions f_i,
%          calculated at xval.
%  dfdx  = (m x n)-matrix with the derivatives of the constraint functions
%          f_i with respect to the variables x_j, calculated at xval.
%          dfdx(i,j) = the derivative of f_i with respect to x_j.
%  low   = Column vector with the lower asymptotes from the previous
%          iteration (provided that iter>1).
```

```
%  upp   = Column vector with the upper asymptotes from the previous
%          iteration (provided that iter>1).
%  a0    = The constants a_0 in the term a_0*z.
%  a     = Column vector with the constants a_i in the terms a_i*z.
%  c     = Column vector with the constants c_i in the terms c_i*y_i.
%  d     = Column vector with the constants d_i in the terms 0.5*d_i*(y_i)^2.
%
%*** OUTPUT:
%
%  xmma  = Column vector with the optimal values of the variables x_j
%          in the current MMA subproblem.
%  ymma  = Column vector with the optimal values of the variables y_i
%          in the current MMA subproblem.
%  zmma  = Scalar with the optimal value of the variable z
%          in the current MMA subproblem.
%  lam   = Lagrange multipliers for the m general MMA constraints.
%  xsi   = Lagrange multipliers for the n constraints alfa_j - x_j <= 0.
%  eta   = Lagrange multipliers for the n constraints x_j - beta_j <= 0.
%   mu   = Lagrange multipliers for the m constraints -y_i <= 0.
%  zet   = Lagrange multiplier for the single constraint -z <= 0.
%   s    = Slack variables for the m general MMA constraints.
%  low   = Column vector with the lower asymptotes, calculated and used
%          in the current MMA subproblem.
%  upp   = Column vector with the upper asymptotes, calculated and used
%          in the current MMA subproblem.
%
asyinit = 0.5;
asyincr = 1.2;
asydecr = 0.7;
.....
```