Exercises with 'An 88/99 line topology optimization code written in Matlab' *

Ole Sigmund, Niels Aage and Casper S. Andreassen
Department of Solid Mechanics, Building 404

Jakob S. Jensen
Department of Electrical Engineering, Building 352

Mathias Stolpe
Department of Wind Energy, DTU Risø Campus

Technical University of Denmark
DK-2800 Lyngby, Denmark

June 29, 2015

1 Introduction

The following exercises are based on the paper *A 99 line topology optimization code written in Matlab* Sigmund (2001). Both the source code and a preprint of the paper can be downloaded from `www.topopt.dtu.dk`. The exercises require some basic knowledge of mechanics, finite element analysis and optimization theory. For students with advanced Matlab and Finite Element experience, the more efficient but less transparent 88 line code (Andreassen et al. 2011) may also be used as a basis for the exercises.

You should solve at least problems 1-6 during the course. If you have previous experience or you are fast you may skip problems 1-3 and solve some of the more complex problems 7 through 14 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 `www.topopt.dtu.dk/PETSc`. We recommend that you only choose these exercises if you have prior experience with programming languages such as C/C++ or Fortran.

The solutions must be presented in a poster session (consisting of a maximum of 16 A4 pages pinned on a poster board) and a copy of the poster and important source code(s) must be handed over to Ole Sigmund on Tuesday July 7th.

The accompanying paper (Sigmund 2001) (see also the appendix in Bendsøe and Sigmund (2004)) gives a lot of hints on how to solve problems 1 through 6.

From the file sharing pages on CampusNet (`www.cn.dtu.dk`) you can download the relevant Matlab subroutines and other relevant material during the course.

2 Getting started

- Form a group of preferably 2 students and login to the DTU system with your student account. Try to make sure that at least one group member has a mechanical engineer-

*Intended for the DCAMM-course: *Topology Optimization - Theory, Methods and Applications* held at DTU, Lyngby, Denmark, 1 – 7 July, 2015.

ing background and 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: `www.topopt.dtu.dk`
- Choose “Applets and software” and “Matlab program”
- Download the Matlab code `top.m` (or `top88.m`) to your home directory "My Documents"
- Start up Matlab
- Run the default MBB-example by writing `top(60,20,0.5,3.0,1.1)` in the Matlab command line
- Start experimenting with the code
- Keep an original version of the Matlab code and start editing new versions
- Extensions to the Matlab code such as a few lines which create a displacement picture, the equations for the strain-displacement matrix and a more efficient sparse assembly strategy are given in appendices A, B and C.

3 Recommended problems

The downloaded Matlab code solves the problem of optimizing the material distribution in the MBB-beam such that its compliance is minimized. A number of extensions and changes in the algorithm can be thought of.

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

Use the Matlab code to investigate the influence of the 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

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?

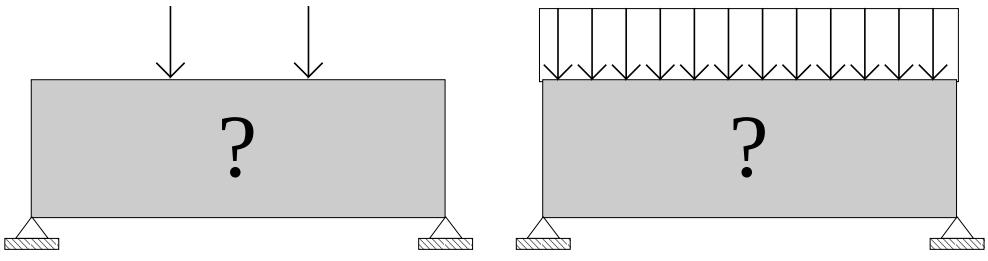

Figure 1: Topology optimization of a "bridge" structure. Left: two (simultaneous) point loads and right: distributed load.

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).

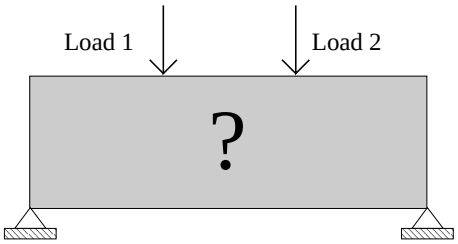

Figure 2: Topology optimization of a bridge structure with two load cases.

Problem 4: 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 CampusNet together with the documentation for the Matlab MMA code. The program solves the following optimization problem

$$\left.\begin{array}{rcll} \min\limits_{\boldsymbol{x},\boldsymbol{y},z} & : & f_0(\boldsymbol{x}) + a_0\ z + \sum\limits_{i=1}^{m}(c_i y_i + \frac{1}{2} d_i y_i^2) & \\ \text{subject to} : & & f_i(\boldsymbol{x}) - a_i\ z - y_i \leq 0, \qquad i = 1,\ldots,m & \\ & : & x_j^{min} \leq x_j \leq x_j^{max}, \qquad j = 1,\ldots,n & \\ & : & y_i \geq 0, \qquad i = 1,\ldots,m & \\ & : & z \geq 0 & \end{array}\right\}, \tag{1}$$

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 ($f_i(\boldsymbol{x}) \leq 0$) 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

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

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 5: Mechanism synthesis

Extend the Matlab code to include compliant mechanism synthesis. Solve the inverter problem in Figure 3.

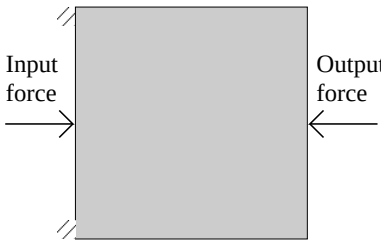

Figure 3: Synthesis of compliant inverter mechanism.

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

$$\left.\begin{aligned} \min_{\boldsymbol{x}} \; &: \; u_{out} \\ \text{subject to} : \; &V(\mathbf{x}) = \mathbf{v}^T \, \mathbf{x} \leq V^* \\ &: \; \mathbf{K}\mathbf{u} = \mathbf{f}_{in} \\ &: \; 0 < x_j^{min} \leq x_j \leq 1, \qquad j = 1, \ldots, n \end{aligned}\right\}, \tag{4}$$

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
- Check the correctness of the derivatives of objective function and constraints by the finite difference method for a number of different elements. Remember to turn of the checkerboard 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 movelimits, i.e. $xmin = max(0, x - move)$, etc.
- Start with Young's modulus E=100 and spring stiffnesses and input forces unity.

Problem 6: Optimization with time-harmonic loads

Extend your code to deal with the problem of undamped forced vibrations. With a time-harmonic load of angular frequency ω, the FE equation is changed from $\mathbf{Ku} = \mathbf{f}$ to:

$$\mathbf{Su} = \mathbf{f}, \tag{5}$$

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

$$\mathbf{S} = \mathbf{K} - \omega^2 \mathbf{M}, \tag{6}$$

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

Optimize the MBB-example (use a 3:1 length ratio and a fine mesh, eg. $nelx = 300$ and $nely = 100$) using the following objective function for "dynamic compliance":

$$c = |\mathbf{f}^T \mathbf{u}| \tag{7}$$

for increasing values of ω[1] (eg. starting with $\omega = 0$ and incrementing in steps of $\Delta\omega = 0.02$). Comment on the designs and compare the optimized designs to the design obtained by minimizing the static compliance. Show and comment on the frequency response curve (objective value c versus angular frequency ω) for each design in the range from $\omega = 0 - 0.20$.

Hints:

- Assemble the mass matrix $\mathbf{M}$ using the local mass matrices found in appendix B (use a total mass of unity).

[1] To get good results for higher frequencies ($\omega >\approx 0.03$ with the indicated settings) you will need to use a "proper" initial design (eg. a structure optimized for a lower frequency). For this purpose modify your code so that you can read in (and write out) design variables from a file.

- Use $E = 1$ and a volume fraction of 0.3.
- Introduce separate penalization factors for stiffness and mass interpolation. Discuss the physical interpretation of the interpolations and experiment with the choice of parameters.
- Base the sensitivity analysis on the formulas in Evgrafov (2015).
- Use MMA with externally fixed movelimits.
- Check your implementation by optimizing for $\omega = 0$.

4 Voluntary Matlab problems

Problem 7: 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

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

can be re-written to

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

which may be solved by MMA using the constants

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

Problem 8: 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 7) and solving for the three geometry cases in each iteration.

Problem 9: 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).

We will utilize an interior-point algorithm implemented as a part of Matlab's Optimization Toolbox, namely as a part of a general-purpose non-linear programming solver `fmincon`. Type `help fmincon` or `doc fmincon` to learn more about a multitude of various options and parameters of `fmincon` routine.

Start with a template program `top88_fmincon.m` downloadable from CampusNet, 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 two programs on some benchmark examples (e.g., coming from Problems 1–3).

Whereas computing individual second order derivatives of compliance is an expensive operation, one may relatively inexpensively compute Hessian-vector products: it comes to solving only one additional linear problem. Namely, if we denote with $\mathbf{z}$ the product between the Hessian of the compliance and a vector $\mathbf{y}$:

$$\mathbf{z} := [\nabla^2_{\mathbf{xx}}(\mathbf{f}^{\mathrm{T}}\mathbf{u})] \cdot \mathbf{y},$$

then the ith component of $\mathbf{z}$ equals to

$$\begin{aligned} z_i = \sum_j \mathbf{f}^{\mathrm{T}} \frac{\partial^2 \mathbf{u}}{\partial x_i \partial x_j} y_j = & - y_i \mathbf{u}^{\mathrm{T}} \frac{\partial^2 \mathbf{K}}{\partial x_i^2}(\mathbf{x})\mathbf{u} \\ & + 2\left[\frac{\partial \mathbf{K}}{\partial x_i}(\mathbf{x})\mathbf{u}\right]^{\mathrm{T}} \Bigg\{ \underbrace{\mathbf{K}^{-1}(\mathbf{x})\left[\sum_j y_j \frac{\partial \mathbf{K}}{\partial x_j}(\mathbf{x})\mathbf{u}\right]}_{\text{adjoint problem, depends on } \mathbf{y}} \Bigg\}, \end{aligned} \tag{11}$$

where we used the fact that the mixed second order derivatives $\partial^2\mathbf{K}/\partial x_i \partial x_j = 0$ for $i \neq j$ because of our choice of the finite element discretization.

Start with a template program `top88_fmincon_hessmult.m` downloadable from CampusNet, which is very similar to `top88_fmincon.m`.

Fill in the code lines marked with comments `%FIXIT`, as before, taking into account that some of them are now related to the expression (11). Compare the performance of your the new program on the same benchmark examples.

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 `'DerivativeChecking'` 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`.
- When comparing different methods, pay attention to the total number of optimization iterations, total runtime, and the final value of the objective function found.

Problem 10: SAND optimization formulation

In this exercise you will solve maximum stiffness sizing optimization problems using an alternative technique which is often refereed to as Simultaneous Analysis and Design (SAND). In the SAND approach the variables in the optimal design problem are both the design variables $\mathbf{x}$ and the state variables $\mathbf{u}$, i.e. the displacements. In the implementation the fixed displacements have been removed. In the SAND approach the equilibrium equations are also explicitly included as nonlinear equality constraints in the problem formulation. The main advantage of the SAND approach is that the equilibrium equations do not need to be solved at each iteration, they must only be satisfied once the optimal design is approached and this is enforced by the optimization method. Another advantage is the simplified sensitivity analysis. The main disadvantage is the increased problem size, both in terms of number of variables and the number of constraints.

In this exercise we focus on the Variable Thickness Sheet (VTS) problem. The design variables in this problem represent the thickness of a 2D design domain. This problem is obtained if there is no penalization of gray material, i.e. if $p = 1$ in the SIMP model, and if no sensitivity or density based filters are used. The single-load VTS problem is stated as

$$\left.\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{u}} \;:\; & c(\mathbf{x},\mathbf{u}) = \mathbf{f}^T\mathbf{u} \\ \text{subject to}:\; & \mathbf{K}(\mathbf{x})\mathbf{u} - \mathbf{f} = 0 \\ :\; & \mathbf{v}^T\,\mathbf{x} - V^* \leq 0 \\ :\; & 0 < x_j^{min} \leq x_j \leq 1, \qquad j = 1,\ldots,n \end{aligned}\right\} \tag{12}$$

where the stiffness matrix $\mathbf{K}(\mathbf{x})$, for this exercise, is assumed to be linear in the design variables, i.e.

$$\mathbf{K}(\mathbf{x}) = \sum_j x_j \mathbf{K}_j.$$

The VTS problem (12) has one linear inequality constraint (the volume constraint), $2n$ bound constraints, and one equality constraint per degree of freedom. Since MMA is not capable of handling equality constraints, we suggest that you solve this problem using the primal-dual interior point method implemented in Matlab's optimization routine `fmincon`. This requires that the Optimization Toolbox is available and that the version of Matlab is recent. Interior-point methods are second order methods and hence need second order sensitivity information. The interior-point method in `fmincon` needs three user defined functions, one function to compute the objective function and its gradient, one function to compute the constraints and the Jacobian of the constraint functions, and one function to compute a Hessian of a certain Lagrange function (cf. below).

Additional information regarding the functions `fmincon` and `optimset` can be found by typing `doc fmincon` or `doc optimset` at the Matlab command prompt.

- The objective function $c(\mathbf{x}, \mathbf{u}) = \mathbf{f}^T\mathbf{u}$ is linear in the state variable $\mathbf{u}$. Show that the gradient of $c(\mathbf{x}, \mathbf{u})$ is given by the column vector

$$\nabla c(\mathbf{x}, \mathbf{u}) = \begin{pmatrix} \mathbf{0} \\ \mathbf{f} \end{pmatrix}.$$

- Show that the Jacobian of the constraints is the sparse matrix

$$\begin{pmatrix} \mathbf{F}(\mathbf{u}) & \mathbf{K}(\mathbf{x}) \\ \mathbf{v}^T & \mathbf{0}^T \end{pmatrix},$$

  where $\mathbf{F}(\mathbf{u})$ is the matrix

$$\begin{pmatrix} \mathbf{K}_1\mathbf{u} & \mathbf{K}_2\mathbf{u} & \cdots & \mathbf{K}_n\mathbf{u} \end{pmatrix}.$$

- Show that the Hessian (with respect to the primal variables $(\mathbf{x}, \mathbf{u})$) of the Lagrange function

$$L(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}, \nu, \boldsymbol{\sigma}_+, \boldsymbol{\sigma}_-) = \mathbf{f}^T\mathbf{u} + \boldsymbol{\lambda}^T(\mathbf{K}(\mathbf{x})\mathbf{u} - \mathbf{f}) + \nu(\mathbf{v}^T\,\mathbf{x} - V^*) + \boldsymbol{\sigma}_+^T(\mathbf{x} - \mathbf{1}) + \boldsymbol{\sigma}_-^T(\mathbf{x}^{min} - \mathbf{x})$$

  becomes

$$\nabla^2_{\mathbf{xu}} L = \begin{pmatrix} \nabla^2_{xx} L & \nabla^2_{xu} L \\ \nabla^2_{ux} L & \nabla^2_{uu} L \end{pmatrix} = \begin{pmatrix} \mathbf{0} & \mathbf{F}^T(\boldsymbol{\lambda}) \\ \mathbf{F}(\boldsymbol{\lambda}) & \mathbf{0} \end{pmatrix}.$$

  The Lagrange multipliers $\boldsymbol{\lambda}$, ν, $\boldsymbol{\sigma}_+$, and $\boldsymbol{\sigma}_-$ are computed by `fmincon` and supplied to the relevant function.

- Download the template file `top_sand_vts.m` from CampusNet. The functions in this file are based on the 88-line topology optimization code which is described in Andreassen et al. (2011). The 88-line code is similar to the 99-line code but the Matlab implementation is significantly more efficient. The function `top_sand_vts` initializes the problem (defines the design domain, the supports, and the external load) and sets appropriate optimization parameters and calls the interior-point method in `fmincon`. The three functions `fdf`, `cdc`, and `d2L` are subsequently called by `fmincon`.

- Complete the function `fdf` to evaluate the objective function and the gradient of the objective function.

- Complete the function `cdc` to evaluate the constraints and the Jacobian of the constraints.

- Complete the function `d2L` to evaluate the Hessian of the Lagrangian.

- Run the same examples as in Problems 1, 2, and 5 (with the difference that no penalization or filter are included). Solve the problems with different discretizations, e.g. solve the MBB-like problem with the following discretizations 30×10, 60×20, 120×40, and 240×80. What happens to the number of iterations as the size of the problem increases?

- Solve the same problems with MMA (set the penalization to one and set the filter radius to a very small value). Compare the quality of the solutions. Compare the number of iterations.

- Generalize the code to handle multiple loads. Solve the same problems as in Problem 3 (again without any penalization or filters).

Problem 11: Mechanisms with multiple outputs

Design an "elevator mechanism" as shown in Figure 4 (the platform must remain horizontal during elevation) or a gripping mechanism with parallel moving jaws.

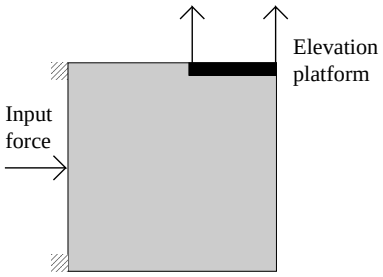

Figure 4: Mechanism synthesis for an "elevator mechanism". The platform must remain horizontal during elevation.

Problem 12: Three dimensions

How many lines of Matlab code does it take to optimize the topology of a three-dimensional MBB-beam? Implement the code and run some examples.

Problem 13: Alternative measures of dynamic compliance

Optimize the MBB-example for harmonic loads using alternative formulations for the dynamic compliance: 1) dissipated energy in the structure and 2) total energy level in the structure. Compare the optimized structures to the ones obtained in exercise 7.

Problem 14: Others

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

References

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.

Evgrafov, A.: 2015, Sensitivity analysis of discrete(-ized) systems cheat sheet, *Report*, Department of Mathematics, Technical University of Denmark.

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.

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;
  for ely = 1:nely
    for elx = 1:nelx
      n1 = (nely+1)*(elx-1)+ely;
      n2 = (nely+1)* elx   +ely;
      Ue = 0.005*U([2*n1-1;2*n1; 2*n2-1;2*n2; 2*n2+1;2*n2+2; 2*n1+1;2*n1+2],1);
      ly = ely-1; lx = elx-1;
      xx = [Ue(1,1)+lx Ue(3,1)+lx+1 Ue(5,1)+lx+1 Ue(7,1)+lx  ]';
      yy = [-Ue(2,1)-ly -Ue(4,1)-ly   -Ue(6,1)-ly-1 -Ue(8,1)-ly-1]';
      patch(xx,yy,-x(ely,elx))
    end
  end
  drawnow; clf;
```

Note that the factor 0.005 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 CampusNet (`www.cn.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,df0dx2,fval,dfdx,dfdx2,low,upp,a0,a,c,d);
%
%    Written in May 1999 by
%    Krister Svanberg <krille@math.kth.se>
%    Department of Mathematics
```

```
%    SE-10044 Stockholm, Sweden.
%
%    Modified ("spdiags" instead of "diag") April 2002
%
%
%    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.
% df0dx2 = Column vector with the non-mixed second derivatives of the
%          objective function f_0 with respect to the variables x_j,
%          calculated at xval. df0dx2(j) = the second derivative
%          of f_0 with respect to x_j (twice).
%          Important note: If second derivatives are not available,
%          simply let df0dx2 = 0*df0dx.
%  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.
%  dfdx2 = (m x n)-matrix with the non-mixed second derivatives of the
%          constraint functions f_i with respect to the variables x_j,
%          calculated at xval. dfdx2(i,j) = the second derivative
%          of f_i with respect to x_j (twice).
%          Important note: If second derivatives are not available,
%          simply let dfdx2 = 0*dfdx.
%  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.
%
 epsimin = sqrt(m+n)*10^(-9);
 feps = 0.000001;
 asyinit = 0.5;
 asyincr = 1.2;
 asydecr = 0.7;
```