Educational article

Struct Multidisc Optim 21, 120–127 © Springer-Verlag 2001

A 99 line topology optimization code written in Matlab

O. Sigmund

Abstract The paper presents a compact Matlab implementation of a topology optimization code for compliance minimization of statically loaded structures. The total number of Matlab input lines is 99 including optimizer and Finite Element subroutine. The 99 lines are divided into 36 lines for the main program, 12 lines for the Optimality Criteria based optimizer, 16 lines for a mesh-independency filter and 35 lines for the finite element code. In fact, excluding comment lines and lines associated with output and finite element analysis, it is shown that only 49 Matlab input lines are required for solving a well-posed topology optimization problem. By adding three additional lines, the program can solve problems with multiple load cases. The code is intended for educational purposes. The complete Matlab code is given in the Appendix and can be down-loaded from the web-site `http://www.topopt.dtu.dk`.

Key words topology optimization, education, optimality criteria, world-wide web, Matlab code

1 Introduction

The Matlab code presented in this paper is intended for engineering education. Students and newcomers to the field of topology optimization can down-load the code from the web-page `http://www.topopt.dtu.dk`. The code may be used in courses in structural optimization where students may be assigned to do extensions such as multiple load-cases, alternative mesh-independency schemes, passive areas, etc. Another possibility is to use the program to develop students' intuition for optimal design. Advanced students may be asked to guess the optimal topology for given boundary condition and volume fraction and then the program shows the correct optimal topology for comparison.

In the literature, one can find a multitude of approaches for the solving of topology optimization problems. In the original paper Bendsøe and Kikuchi (1988) a so-called microstructure or homogenization based approach was used, based on studies of existence of solutions.

The homogenization based approach has been adopted in many papers but has the disadvantage that the determination and evaluation of optimal microstructures and their orientations is cumbersome if not unresolved (for noncompliance problems) and furthermore, the resulting structures cannot be built since no definite length-scale is associated with the microstructures. However, the homogenization approach to topology optimization is still important in the sense that it can provide bounds on the theoretical performance of structures.

An alternative approach to topology optimization is the so-called "power-law approach" or SIMP approach (Solid Isotropic Material with Penalization) (Bendsøe 1989; Zhou and Rozvany 1991; Mlejnek 1992). Here, material properties are assumed constant within each element used to discretize the design domain and the variables are the element relative densities. The material properties are modelled as the relative material density raised to some power times the material properties of solid material. This approach has been criticized since it was argued that no physical material exists with properties described by the power-law interpolation. However, a recent paper by Bendsøe and Sigmund (1999) proved that the power-law approach is physically permissible as long as simple conditions on the power are satisfied (e.g. $p \geq 3$ for Poisson's ratio equal to $\frac{1}{3}$). To ensure existence of solutions, the power-law approach must be combined with a perimeter constraint, a gradient constraint or with filtering techniques (see Sigmund and Petersson 1998, for an overview). The power-law approach to topology optimization has been applied to problems with multiple constraints, multiple physics and multiple materials.

Whereas the solution of the above mentioned approaches is based on mathematical programming techniques and continuous design variables, a number of papers have appeared on solving the topology optimization problem as an integer problem. Beckers (1999) success-

Received October 22, 1999

O. Sigmund

Department of Solid Mechanics, Building 404, Technical University of Denmark, DK-2800 Lyngby, Denmark
e-mail: `sigmund@fam.dtu.dk`

fully solved large-scale compliance minimization problems using a dual-approach but other approaches based on genetic algorithms or other semi-random approaches require thousands of function evaluations even for small number of elements and must be considered impractical.

Apart from above mentioned approaches, which all solve well defined problems (e.g. minimization of compliance) a number of heuristic or intuition based approaches have been shown to decrease compliance or other objective functions. Among these methods are so-called evolutionary design methods (see e.g. Xie and Steven 1997; Baumgartner *et al.* 1992). Apart from being very easy to understand and implement (at least for the compliance minimization case), the main motivation for the evolutionary approaches seems to be that mathematically based or continuous variable approaches "involve some complex calculus operations and mathematical programming" (citation from Li *et al.* 1999) and they contain "mathematical methods of some complexity" (citation from Zhao *et al.* 1998) whereas the evolutionary approach "takes advantage of powerful computing technology and intuitive concepts of evolution processes in nature" (citation from Li *et al.* 1999). Two things can be argued against this. First, the evolutionary approaches become complicated themselves, once more complex objectives than compliance minimization are considered and second, as shown in this paper, the "mathematically based" approaches for compliance minimization are simple to implement as well and are computationally equally efficient. Furthermore, mathematical programming based methods can easily be extended to other non-compliance objectives such as non-self-adjoint and multiphysics problems and to problems with multiple constraints (e.g. Sigmund 1999). Extensions of the evolutionary approach to such cases seem more questionable.

The complete Matlab code is given in the Appendix. The remainder of the paper consists of definition and discussion of the optimization problem (Sect. 2), comments about the Matlab implementation (Sect. 3) followed by a discussion of extensions (Sect. 4) and a conclusion (Sect. 5).

2 The topology optimization problem

A number of simplifications are introduced to simplify the Matlab code. First, the design domain is assumed to be rectangular and discretized by square finite elements. In this way, the numbering of elements and nodes is simple (column by column starting in the upper left corner) and the aspect ratio of the structure is given by the ratio of elements in the horizontal (`nelx`) and the vertical direction (`nely`).[1]

[1] Names in type-writer style refer to Matlab variable names that differ from the obvious (see the Matlab code in the Appendix)

A topology optimization problem based on the power-law approach, where the objective is to minimize compliance can be written as

$$\left.\begin{aligned} \min_{\mathbf{x}}: \quad & c(\mathbf{x}) = \mathbf{U}^T\mathbf{K}\mathbf{U} = \sum_{e=1}^{N}(x_e)^p\ \mathbf{u}_e^T\ \mathbf{k}_0\ \mathbf{u}_e \\ \text{subject to}: \quad & \frac{V(\mathbf{x})}{V_0} = f \\ : \quad & \mathbf{K}\mathbf{U} = \mathbf{F} \\ : \quad & \mathbf{0} < \mathbf{x}_{\min} \le \mathbf{x} \le \mathbf{1} \end{aligned}\right\}, \tag{1}$$

where $\mathbf{U}$ and $\mathbf{F}$ are the global displacement and force vectors, respectively, $\mathbf{K}$ is the global stiffness matrix, $\mathbf{u}_e$ and $\mathbf{k}_e$ are the element displacement vector and stiffness matrix, respectively, $\mathbf{x}$ is the vector of design variables, $\mathbf{x}_{\min}$ is a vector of minimum relative densities (non-zero to avoid singularity), N $(= \texttt{nelx} \times \texttt{nely})$ is the number of elements used to discretize the design domain, p is the penalization power (typically $p = 3$), $V(\mathbf{x})$ and V_0 is the material volume and design domain volume, respectively and f (`volfrac`) is the prescribed volume fraction.

The optimization problem (1) could be solved using several different approaches such as Optimality Criteria (OC) methods, Sequential Linear Programming (SLP) methods or the Method of Moving Asymptotes (MMA by Svanberg 1987) and others. For simplicity, we will here use a standard OC-method.

Following Bendsøe (1995) a heuristic updating scheme for the design variables can be formulated as

$$x_e^{\text{new}} = \begin{cases} \max(x_{\min}, x_e - m) \\ \quad \text{if} \quad x_e B_e^{\eta} \le \max(x_{\min}, x_e - m)\,, \\ x_e B_e^{\eta} \\ \quad \text{if} \quad \max(x_{\min}, x_e - m) < x_e B_e^{\eta} < \min(1, x_e + m)\,, \\ \min(1, x_e + m) \\ \quad \text{if} \quad \min(1, x_e + m) \le x_e B_e^{\eta}\,, \end{cases} \tag{2}$$

where m (`move`) is a positive move-limit, η $(= 1/2)$ is a numerical damping coefficient and B_e is found from the optimality condition as

$$B_e = \frac{-\dfrac{\partial c}{\partial x_e}}{\lambda \dfrac{\partial V}{\partial x_e}}, \tag{3}$$

where λ is a Lagrangian multiplier that can be found by a bi-sectioning algorithm.

The sensitivity of the objective function is found as

$$\frac{\partial c}{\partial x_e} = -p(x_e)^{p-1}\ \mathbf{u}_e^T\ \mathbf{k}_0\ \mathbf{u}_e\,. \tag{4}$$

For more details on the derivation and implementation of the optimality criteria method, the reader is referred to the literature (e.g. Bendsøe 1995).

In order to ensure existence of solutions to the topology optimization problem (1), some sort of restriction on the resulting design must be introduced (see Sigmund and Petersson 1998, for an overview). Here we use a filtering technique (Sigmund 1994, 1997). It must be emphasized that this filter has not yet been proven to ensure existence of solutions, but numerous applications by the author have proven the the filter produces mesh-independent designs in practice.

The mesh-independency filter works by modifying the element sensitivities as follows:

$$\widehat{\frac{\partial c}{\partial x_e}} = \frac{1}{x_e \sum_{f=1}^{N} \hat{H}_f} \sum_{f=1}^{N} \hat{H}_f \, x_f \, \frac{\partial c}{\partial x_f} \,. \tag{5}$$

The convolution operator (weight factor) $\hat{H}_f$ is written as

$$\hat{H}_f = r_{\min} - \mathrm{dist}(e, f) \,,$$

$$\{f \in N \mid \mathrm{dist}(e, f) \le r_{\min}\}, \quad e = 1, \ldots, N \,, \tag{6}$$

where the operator $\mathrm{dist}(e, f)$ is defined as the distance between centre of element e and centre of element f. The convolution operator $\hat{H}_f$ is zero outside the filter area. The convolution operator decays linearly with the distance from element f. Instead of the original sensitivities (4), the modified sensitivities (5) are used in the Optimality Criteria update (3).

3 Matlab implementation

The Matlab code (see the Appendix), is built up as a standard topology optimization code. The main program is called from the Matlab prompt by the line

```
top(nelx,nely,volfrac,penal,rmin)
```

where `nelx` and `nely` are the number of elements in the horizontal and vertical directions, respectively, `volfrac` is the volume fraction, `penal` is the penalization power and `rmin` is the filter size (divided by element size). Other variables as well as boundary conditions are defined in the Matlab code itself and can be edited if needed. For each iteration in the topology optimization loop, the code generates a picture of the current density distribution. Figure 1 shows the resulting density distribution obtained by the code given in the Appendix called with the input line

```
top(60,20,0.5,3.0,1.5)
```

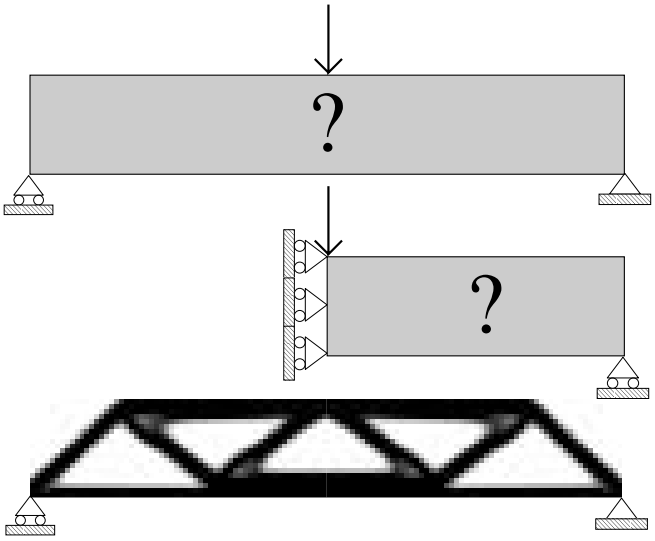

Fig. 1 Topology optimization of the MBB-beam. Top: full design domain, middle: half design domain with symmetry boundary conditions and bottom: resulting topology optimized beam (both halves)

The default boundary conditions correspond to half of the "MBB-beam" (Fig. 1). The load is applied vertically in the upper left corner and there is symmetric boundary conditions along the left edge and the structure is supported horizontally in the lower right corner.

Important details of the Matlab code are discussed in the following subsections.

3.1 Main program (lines 1–36)

The main program (lines 1–36) starts by distributing the material evenly in the design domain (line 4). After some other initializations, the main loop starts with a call to the Finite Element subroutine (line 12) which returns the displacement vector `U`. Since the element stiffness matrix for solid material is the same for all elements, the element stiffness matrix subroutine is called only once (line 14). Following this, a loop over all elements (lines 16–24) determines objective function and sensitivities (4). The variables `n1` and `n2` denote upper left and right element node numbers in global node numbers and are used to extract the element displacement vector `Ue` from the global displacement vector `U`. The sensitivity analysis is followed by a call to the mesh-independency filter (line 26) and the Optimality Criteria optimizer (line 28). The current compliance as well as other parameters are printed by lines 30–33 and the resulting density distribution is plotted (line 35). The main loop is terminated if the change in design variables (`change` determined in line 30) is less than 1 percent[2]. Otherwise above steps are repeated.

[2] this is a rather "sloppy" convergence criterion and could be decreased if needed

3.2
Optimality criteria based optimizer (lines 37–48)

The updated design variables are found by the optimizer (lines 37–48). Knowing that the material volume (`sum(sum(xnew))`) is a monotonously decreasing function of the Lagrange multiplier (`lag`), the value of the Lagrangian multiplier that satisfies the volume constraint can be found by a bi-sectioning algorithm (lines 40-48). The bi-sectioning algorithm is initialized by guessing a lower `l1` and an upper `l2` bound for the Lagrangian multiplier (line 39). The interval which bounds the Lagrangian multiplier is repeatedly halved until its size is less than the convergence criteria (line 40).

3.3
Mesh-independency filtering (lines 49–64)

Lines 49–64 represent the Matlab implementation of (5). Note that not all elements in the design domain are searched in order to find the elements that lie within the radius `rmin` but only those within a square with side lengths two times `round(rmin)` around the considered element. By selecting `rmin` less than one in the call of the routine, the filtered sensitivities will be equal to the original sensitivities making the filter inactive.

3.4
Finite element code (lines 65–99)

The finite element code is written in lines 65–99. Note that the solver makes use of the sparse option in Matlab. The global stiffness matrix is formed by a loop over all elements (lines 70–77). As was the case in the main program, variables `n1` and `n2` denote upper left and right element node numbers in global node numbers and are used to insert the element stiffness matrix at the right places in the global stiffness matrix.

As mentioned before, both nodes and elements are numbered column wise from left to right. Furthermore, each node has two degrees of freedom (horizontal and vertical), thus the command `F(2,1)=-1.` (line 79) applies a vertical unit force force in the upper left corner.

Supports are implemented by eliminating fixed degrees of freedom from the linear equations. Matlab can do this very elegantly with the line

```
84 U(freedofs,:) = K(freedofs,freedofs) \
                   F(freedofs,:);
```

where `freedofs` indicate the degrees of freedom which are unconstrained. Mostly, it is easier to define the degrees of freedom that are fixed (`fixeddofs`) thereafter the `freedofs` are found automatically using the Matlab operator `setdiff` which finds the free degrees of freedoms as the difference between all degrees of freedom and the fixed degrees of freedom (line 82).

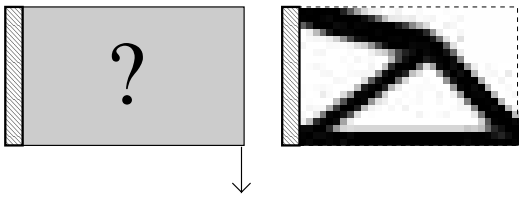

Fig. 2 Topology optimization of a cantilever beam. Left: design domain and right: topology optimized beam

The element stiffness matrix is calculated in lines 86–99. The 8 by 8 matrix for a square bi-linear 4-node element was determined analytically using a symbolic manipulation software. The Young's modulus `E` and the Poisson's ratio `nu` can be altered in lines 88 and 89.

4
Extensions

The Matlab code given in the Appendix solves the problem of optimizing the material distribution in the MBB-beam (Fig. 1) such that its compliance is minimized. A number of extensions and changes in the algorithm can be thought of, a few of which are mentioned in the following.

4.1
Other boundary conditions

It is very simple to change boundary conditions and support conditions in order to solve other optimization problems. In order to solve the short cantilever example shown in Fig. 2, only lines 79 and 80 must be changed to

```
79 F(2*(nelx+1)*(nely+1),1) = -1;
80 fixeddofs   = [1:2*(nely+1)];
```

With these changes, the input line for the case shown in Fig. 2 is

```
top(32,20,0.4,3.0,1.2)
```

4.2
Multiple load cases

It is also very simple to extend the algorithm to account for multiple load cases. In fact, this can be done by adding only three additional lines and making minor changes to another 4 lines.

In the case of two load cases, force and displacement vectors must be defined as two-column vectors which means that line 69 is changed to

```
69 F = sparse(2*(nely+1)*(nelx+1),2);
   U = sparse(2*(nely+1)*(nelx+1),2);
```

The objective function is now the sum of two compliances, i.e.

$$c(\mathbf{x}) = \sum_{i=1}^{2} \mathbf{U}_i^T \mathbf{K} \mathbf{U}_i \tag{7}$$

thus lines 20–22 are substituted with the lines

```
19b   dc(ely,elx) = 0.;
19c   for i = 1:2
20     Ue = U([2*n1-1;2*n1; 2*n2-1;2*n2;
            2*n2+1;2*n2+2;2*n1+1;2*n1+2],i);
21     c = c + x(ely,elx)^penal*Ue'*KE*Ue;
22     dc(ely,elx) = dc(ely,elx) -
          penal*x(ely,elx)^(penal-1)*Ue'*KE*Ue;
22b   end
```

To solve the two-load problem indicated in Fig. 3, a unit upward load in the top-right corner is added to line 79, which then becomes

```
79 F(2*(nelx+1)*(nely+1),1) = -1.;
             F(2*(nelx)*(nely+1)+2,2) = 1.;
```

The input line for Fig. 3 is

```
top(30,30,0.4,3.0,1.2).
```

4.3 Passive elements

In some cases, some of the elements may be required to take the minimum density value (e.g. a hole for a pipe).

An `nely` × `nelx` array `passive` with zeros at elements free to change and ones at elements fixed to be zero can be defined in the main program and transferred to the OC subroutine (adding `passive` to the call in lines 28 and 38). The added line

```
42b   xnew(find(passive)) = 0.001;
```

in the OC subroutine looks for passive elements and sets their density equal to the minimum density (0.001).

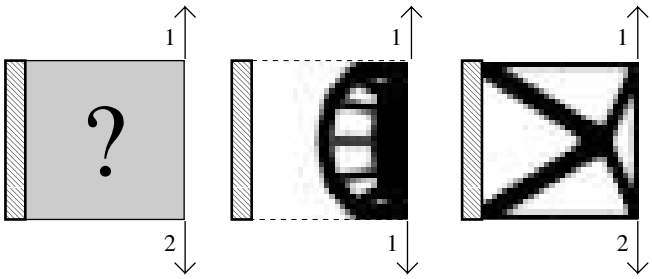

Fig. 3 Topology optimization of a cantilever beam with two load-cases. Left: design domain, middle: topology optimized beam using one load case and right: topology optimized beam using two load cases

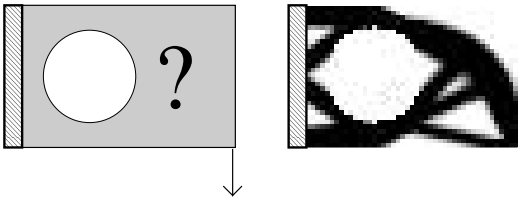

Fig. 4 Topology optimization of a cantilever beam with a fixed hole. Left: design domain and right: topology optimized beam

Figure 4 shows the resulting structure obtained with the input

```
top(45,30,0.5,3.0,1.5),
```

when the following 10 lines were added to the main program (after line 4) in order to find passive elements within a circle with radius `nely/3.` and center (`nely/2.`,`nelx/3.`)

```
for ely = 1:nely
  for elx = 1:nelx
    if sqrt((ely-nely/2.)^2+(elx-nelx/3.)^2) <
                                        nely/3.
      passive(ely,elx) = 1;
      x(ely,elx) = 0.001;
    else
      passive(ely,elx) = 0;
    end
  end
end
```

4.4 Alternative optimizer

Admittedly, the optimality criteria based optimizer implemented here is only good for a single constraint and it is based on a heuristic fixed point type updating scheme. In order to install a better optimizer, one can obtain (free

of charge for academic purposes) the Matlab version of the MMA-algorithm (Svanberg 1987) from Krister Svanberg, KTH, Sweden. The MMA code is called with the following input line

```
mmasub(INPUT-variables, ... ,
       OUTPUT-variables)
```

where the total number of input/output variables is 20, including objective function, constraints, old and new densities, etc. Implementing the MMA-optimizer is fairly simple, but requires the definition of several auxiliary variables. However, it allows for the solving of more complex design problems with more than one constraint. The Matlab optimizer will solve the standard topology optimization problem using less iterations at the cost of a slightly increased CPU-time per iteration.

4.5 Other extensions

Extensions to three dimensions should be straight forward whereas more complex problems such as compliant mechanism design (Sigmund 1997) requires the implementation of the MMA optimizer and the definition of extra constraints. The simplicity of the Matlab commands allow for easy extensions of the graphical output, interactive input etc.

5 Conclusions

This paper has presented a very simple implementation of a mathematical programming base topology optimization algorithm. The code is implemented using only 99 Matlab input lines and includes optimizer, mesh-independency filtering and Finite Element code.

The Matlab code can be down-loaded from the web-page `http://www.topopt.dtu.dk` and is intended for educational purposes. The code can easily be extended to include multi load problems and the definition of passive areas.

Running the code in Matlab is rather slow compared to a Fortran implementation of the same code which can be tested at the web-site `http://www.topopt.dtu.dk`. However, an add-on package to Matlab (MATLAB Compiler) allows for the generation of more efficient C-code that can be optimized for run-time (this option, however, has not been tested by the author). It should be noted that speed can be gained by modifying the Matlab code itself, however the speed is gained on the cost of simplicity of the program. The modification is suggested by Andreas Rietz from Linköping University who uses sparsity options in the assembly of the global stiffness matrix. The reader may down-load his code at the web-page: `http://www.mekanik.ikp.liu.se/andridiv/matlab/theory.html`.

The code was intentionally kept compact in order to keep the total number of lines below 100. If users of the code should find ways to further compactify or simplify the code, the author would be happy to receive suggested modifications that can be implemented in the public domain code (the author's e-mail address is `sigmund@fam.dtu.dk`).

Since its first publication on the World Wide Web in October 1999, the Matlab code has been down-loaded more than 500 times by different users (as of August 2000). Among other positive feedbacks, several professors reported that they have used the code in courses on structural optimization and have let their students implement alternative boundary conditions and multiple load cases.

Acknowledgements This work was supported by the Danish Technical Research Council through the THOR/Talent-programme: Design of MicroElectroMechanical Systems (MEMS). The author would also like to thank Thomas Buhl, Technical University of Denmark, for his inputs to an earlier version of the code.

References

Baumgartner, A., Harzheim, L.; Mattheck, C. 1992: SKO (Soft Kill Option): The biological way to find an optimum structure topology. *Int. J. Fatigue* **14**, 387–393

Beckers, M. 1999: Topology optimization using a dual method with discrete variables. *Struct. Optim.* **17**, 14–24

Bendsøe, M.P. 1989: Optimal shape design as a material distribution problem. *Struct. Optim.* **1**, 193–202

Bendsøe, M.P. 1995: *Optimization of structural topology, shape and material*. Berlin, Heidelberg, New York: Springer

Bendsøe, M.P.; Kikuchi, N. 1988: Generating optimal topologies in optimal design using a homogenization method. *Comp. Meth. Appl. Mech. Engrg.* **71**, 197–224

Bendsøe, M.P.; Sigmund, O. 1999: Material interpolations in topology optimization. *Arch. Appl. Mech.* **69**, 635–654

Li, Q.; Steven, G.P.; Xie, Y. M. 1999: On equivalence between stress criterion and stiffness criterion in evolutionary structural optimization. *Struct. Optim.* **18**, 67–73

Mlejnek, H.P. 1992: Some aspects of the genesis of structures. *Struct. Optim.* **5**, 64–69

Sigmund, O. 1994: *Design of material structures using topology optimization*. Ph.D. Thesis, Department of Solid Mechanics, Technical University of Denmark

Sigmund, O. 1997: On the design of compliant mechanisms using topology optimization. *Mech. Struct. Mach.* **25**, 495–526

Sigmund, O. 1999: Topology optimization of multi-physics, multi-material structures. *Proc. WCSMO-3* (held in Buffalo, NY)

Sigmund, O.; Petersson, J. 1998: Numerical instabilities in topology optimization: a survey on procedures dealing with checkerboards, mesh-dependencies and local minima. *Struct. Optim.* **16**, 68–75

Svanberg, K. 1987: The method of moving asymptotes – a new method for structural optimization. *Int. J. Numer. Meth. Engrg.* **24**, 359–373

Xie, Y.M.; Steven, G.P. 1997: *Evolutionary structural optimization*. Berlin, Heidelberg, New York: Springer

Zhao, C.; Hornby, P.; Steven, G.P.; Xie, Y.M. 1998: A generalized evolutionary method for numerical topology optimization of structures under static loading conditions. *Struct. Optim.* **15**, 251–260

Zhou, M.; Rozvany, G.I.N. 1991: The COC algorithm, part II: Topological, geometry and generalized shape optimization. *Comp. Meth. Appl. Mech. Engrng.* **89**, 197–224

6 Appendix – Matlab code

```
%%% A 99 LINE TOPOLOGY OPTIMIZATION CODE BY OLE SIGMUND, OCTOBER 1999 %%%
function top(nelx,nely,volfrac,penal,rmin);
% INITIALIZE
x(1:nely,1:nelx) = volfrac;
loop = 0;
change = 1.;
% START ITERATION
while change > 0.01
  loop = loop + 1;
  xold = x;
% FE-ANALYSIS
  [U]=FE(nelx,nely,x,penal);
% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
  [KE] = lk;
  c = 0.;
  for ely = 1:nely
    for elx = 1:nelx
      n1 = (nely+1)*(elx-1)+ely;
      n2 = (nely+1)* elx   +ely;
      Ue = U([2*n1-1;2*n1; 2*n2-1;2*n2; 2*n2+1;
           2*n2+2; 2*n1+1;2*n1+2],1);
      c = c + x(ely,elx)^penal*Ue'*KE*Ue;
      dc(ely,elx) = -penal*x(ely,elx)^(penal-1)*
                    Ue'*KE*Ue;
    end
  end
% FILTERING OF SENSITIVITIES
  [dc]   = check(nelx,nely,rmin,x,dc);
% DESIGN UPDATE BY THE OPTIMALITY CRITERIA METHOD
  [x]    = OC(nelx,nely,x,volfrac,dc);
% PRINT RESULTS
  change = max(max(abs(x-xold)));
  disp([' It.: ' sprintf('%4i',loop) ' Obj.: '
        sprintf('%10.4f',c) ...
        ' Vol.: ' sprintf('%6.3f',sum(sum(x))/
          (nelx*nely)) ...
         ' ch.: ' sprintf('%6.3f',change )])
% PLOT DENSITIES
  colormap(gray); imagesc(-x); axis equal; axis
  tight; axis off;pause(1e-6);
end
%%%%%%%%%% OPTIMALITY CRITERIA UPDATE %%%%%%%%%
function [xnew]=OC(nelx,nely,x,volfrac,dc)
l1 = 0; l2 = 100000; move = 0.2;
while (l2-l1 > 1e-4)
 lmid = 0.5*(l2+l1);
 xnew = max(0.001,max(x-move,min(1.,min(x+move,x.
        *sqrt(-dc./lmid)))));
 if sum(sum(xnew)) - volfrac*nelx*nely > 0;
   l1 = lmid;
 else
   l2 = lmid;
 end
end
%%%%%%%%%% MESH-INDEPENDENCY FILTER %%%%%%%%%%
function [dcn]=check(nelx,nely,rmin,x,dc)
dcn=zeros(nely,nelx);
for i = 1:nelx
  for j = 1:nely
    sum=0.0;
    for k = max(i-round(rmin),1):
          min(i+round(rmin),nelx)
      for l = max(j-round(rmin),1):
      min(j+round(rmin), nely)
        fac = rmin-sqrt((i-k)^2+(j-l)^2);
        sum = sum+max(0,fac);
        dcn(j,i) = dcn(j,i) + max(0,fac)*x(l,k)
        *dc(l,k);
      end
    end
    dcn(j,i) = dcn(j,i)/(x(j,i)*sum);
  end
end
%%%%%%%%%% FE-ANALYSIS %%%%%%%%%%%%
function [U]=FE(nelx,nely,x,penal)
[KE] = lk;
K = sparse(2*(nelx+1)*(nely+1), 2*(nelx+1)*
    (nely+1));
F = sparse(2*(nely+1)*(nelx+1),1); U =
    sparse(2*(nely+1)*(nelx+1),1);
for ely = 1:nely
  for elx = 1:nelx
    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
% DEFINE LOADS AND SUPPORTS (HALF MBB-BEAM)
F(2,1) = -1;
fixeddofs   = union([1:2:2*(nely+1)],
              [2*(nelx+1)*(nely+1)]);
alldofs     = [1:2*(nely+1)*(nelx+1)];
freedofs    = setdiff(alldofs,fixeddofs);
% SOLVING
```

```
84 U(freedofs,:) = K(freedofs,freedofs) \
                   F(freedofs,:);
85 U(fixeddofs,:)= 0;
86 %%%%%%%%%% ELEMENT STIFFNESS MATRIX %%%%%%%
87 function [KE]=lk
88 E = 1.;
89 nu = 0.3;
90 k=[ 1/2-nu/6  1/8+nu/8 -1/4-nu/12 -1/8+3*nu/8 ...
91    -1/4+nu/12 -1/8-nu/8  nu/6       1/8-3*nu/8];
92 KE = E/(1-nu^2)*
     [ k(1) k(2) k(3) k(4) k(5) k(6) k(7) k(8)
93     k(2) k(1) k(8) k(7) k(6) k(5) k(4) k(3)
94     k(3) k(8) k(1) k(6) k(7) k(4) k(5) k(2)
95     k(4) k(7) k(6) k(1) k(8) k(3) k(2) k(5)
96     k(5) k(6) k(7) k(8) k(1) k(2) k(3) k(4)
97     k(6) k(5) k(4) k(3) k(2) k(1) k(8) k(7)
98     k(7) k(4) k(5) k(2) k(3) k(8) k(1) k(6)
99     k(8) k(3) k(2) k(5) k(4) k(7) k(6) k(1)];
```