

An efficient 3D topology optimization code written in Matlab

Kai Liu · Andrés Tovar

Received: 17 October 2013 / Revised: 19 March 2014 / Accepted: 22 April 2014 / Published online: 25 June 2014
© Springer-Verlag Berlin Heidelberg 2014

Abstract This paper presents an efficient and compact MATLAB code to solve three-dimensional topology optimization problems. The 169 lines comprising this code include finite element analysis, sensitivity analysis, density filter, optimality criterion optimizer, and display of results. The basic code solves minimum compliance problems. A systematic approach is presented to easily modify the definition of supports and external loads. The paper also includes instructions to define multiple load cases, active and passive elements, continuation strategy, synthesis of compliant mechanisms, and heat conduction problems, as well as the theoretical and numerical elements to implement general non-linear programming strategies such as SQP and MMA. The code is intended for students and newcomers in the topology optimization. The complete code is provided in Appendix C and it can be downloaded from <http://top3dapp.com>.

Keywords Topology optimization · MATLAB · Compliance · Compliant mechanism · Heat conduction · Non-linear programming

1 Introduction

Topology optimization is a computational material distribution method for synthesizing structures without any pre-conceived shape. This freedom provides topology optimization

with the ability to find innovative, high-performance structural layouts, which has attracted the interest of applied mathematicians and engineering designers. From the work of Lucien Schmit in the 1960s (Schmit 1960)—who recognized the potential of combining optimization methods with finite-element analysis for structural design—and the seminal paper by Bendsøe and Kikuchi (1988), there have been more than eleven thousand journal publications in this area (Compendex list as of September 2013), several reference books (Hassani and Hinton 1998; Bendsøe and Sigmund 2003; Christensen and Klarbring 2009), and a number of readily available educational computer tools for MATLAB and other platforms. Some examples of such tools include the topology optimization program by Liu et al. (2005) for Femlab, the shape optimization program by Allaire and Pantz (2006) for FreeFem++, the open source topology optimization program ToPy by Hunter (2009) for Python, and the 99-line program for Michell-like truss structures by Sokół (2011) for Mathematica.

For MATLAB, Sigmund (2001) introduced the 99-line program for two-dimensional topology optimization. This program uses stiffness matrix assembly and filtering via nested loops, which makes the code readable and well-organized but also makes it slow when solving larger problems. Andreassen et al. (2011) presented the 88-line program with improved assembly and filtering strategies. When compared to the 99-line code in a benchmark problem with 7500 elements, the 88-line code is two orders of magnitude faster. From the same research group, Aage et al. (2013) introduced TopOpt, the first topology optimization App for hand-held devices.

Also for MATLAB, Wang et al. (2004) introduced the 199-line program TOPLSM making use of the level-set method. Challis (2010) also used the level-set method but with discrete variables in a 129-line program. Suresh (2010)

K. Liu (✉) · A. Tovar
Department of Mechanical Engineering,
Indiana University-Purdue University Indianapolis,
Indianapolis, IN 46202, USA
e-mail: kailiu@iupui.edu

presented a 199-line program `ParetoOptimalTracing` that traces the Pareto front for different volume fractions using topological sensitivities. More recently, Talischi et al. (2012a, b) introduced `PolyMesher` and `PolyTop` for density-based topology optimization using polygonal finite elements. The use of polygonal elements makes these programs suitable for arbitrary non-Cartesian design domains in two dimensions.

One of the few contributions to three-dimensional MATLAB programs is presented by Zhou and Wang (2005). This code, referred to as the 177-line program, is a successor to the 99-line program by Sigmund (2001) that inherits and amplifies the same drawbacks. Our paper presents a 169-line program referred to as `top3d` that incorporates efficient strategies for three-dimensional topology optimization. This program can be effectively used in personal computers to generate structures of substantial size. This paper explains the use of `top3d` in minimum compliance, compliant mechanism, and heat conduction topology optimization problems.

The rest of this paper is organized as follows. Section 2 briefly reviews theoretical aspects in topology optimization with focus on the density-based approach. Section 3 introduces 3D finite element analysis and its numerical implementation. Section 4 presents the formulation of three typical topology optimization problems, namely, minimum compliance, compliant mechanism, and heat conduction. Section 5 discusses the optimization methods and their implementation in the code. Section 6 shows the numerical implementation procedures and results of three different topology optimization problems, several extensions of the `top3d` code, and multiple alternative implementations. Finally, Section 7, offers some closing thoughts. The `top3d` code is provided in Appendix C and can also be downloaded for free from the website: <http://top3dapp.com>.

2 Theoretical background

2.1 Problem definition and ill-posedness

A topology optimization problem can be defined as a binary programming problem in which the objective is to find the distribution of material in a prescribed area or volume referred to as the *design domain*. A classical formulation, referred to as the *binary compliance problem*, is to find the “black and white” layout (i.e., solids and voids) that minimizes the work done by external forces (or compliance) subject to a volume constraint.

The binary compliance problem is known to be ill-posed (Kohn and Strang 1986a, b, c). In particular, it is possible to obtain a non-convergent sequence of feasible black-and-white designs that monotonically reduce the structure’s

compliance. As an illustration, assume that a design has one single hole. Then, it is possible to find an improved solution with the same mass and lower compliance when this hole is replaced by two smaller holes. Improved solutions can be successively found by increasing the number of holes and reducing their size. The design will progress towards a *chattering* design within infinite number of holes of infinitesimal size. That makes the compliance problem unbounded and, therefore, ill-posed.

One alternative to make the compliance problem well-posed is to control the perimeter of the structure (Haber and Jog 1996; Jog 2002). This method effectively avoids chattering configurations, but its implementation is not free of complications. It has been reported that the addition of a perimeter constraint creates fluctuations during the iterative optimization process so internal loops need to be incorporated (Duysinx 1997) Op. cit. (Bendsøe and Sigmund 2003). Also, small variations in the parameters of the algorithm lead to dramatic changes in the final layout (Jog 2002).

2.2 Homogenization method

Another alternative is to relax the binary condition and include intermediate material densities in the problem formulation. In this way, the chattering configurations become part of the problem statement by assuming a periodically perforated microstructure. The mechanical properties of the material are determined using the homogenization theory. This method is referred to as the *homogenization method* for topology optimization (Bendsøe 1995; Allaire 2001). The main drawback of this approach is that the optimal microstructure, which is required in the derivation of the relaxed problem, is not always known. This can be alleviated by restricting the method to a subclass of microstructures, possibly suboptimal but fully explicit. This approach, referred to as *partial relaxation*, has been utilized by many authors including Bendsøe and Kikuchi (1988), Allaire and Kohn (1993), Allaire et al. (2004), and references therein.

An additional problem with the homogenization methods is the manufacturability of the optimized structure. The “gray” areas found in the final designs contain microscopic length-scale holes that are difficult or impossible to fabricate. However, this problem can be mitigated with *penalization* strategies. One approach is to post-process the partially relaxed optimum and force the intermediate densities to take black or white values (Allaire et al. 1996). This *a posteriori* procedure results in binary designs, but it is purely numerical and mesh dependent. Other approach is to impose *a priori* restrictions on the microstructure that implicitly lead to black-and-white designs (Bendsøe 1995). Even though penalization methods have shown to be effective in avoiding or mitigating intermediate densities, they

revert the problem back to the original ill-posedness with respect to mesh refinement.

2.3 Density-based approach

An alternative that avoids the application of homogenization theory is to relax the binary problem using a continuous density value with no microstructure. In this method, referred to as the *density-based approach*, the material distribution problem is parametrized by the material density distribution. In a discretized design domain, the mechanical properties of the material element, i.e., the stiffness tensor, are determined using a power-law interpolation function between void and solid (Bendsøe 1989; Mlejnek 1992). The power law may *implicitly* penalize intermediate density values driving the structure towards a black-and-white configuration. This penalization procedure is usually referred to as the *Solid Isotropic Material with Penalization* (SIMP) method (Zhou and Rozvany 1991). The SIMP method does not solve the problem’s ill-posedness, but it is simpler than other penalization methods.

The SIMP method is based on a heuristic relation between (relative) element density x_i and element Young’s modulus E_i given by

$$E_i = E_i(x_i) = x_i^p E_0, \quad x_i \in]0, 1], \tag{1}$$

where E_0 is the elastic modulus of the solid material and p is the penalization power ($p > 1$). A modified SIMP approach is given by

$$E_i = E_i(x_i) = E_{\min} + x_i^p (E_0 - E_{\min}), \quad x_i \in [0, 1], \tag{2}$$

where E_{\min} is the elastic modulus of the void material, which is non-zero to avoid singularity of the finite element stiffness matrix. The modified SIMP approach, as (2), offers a number of advantages over the classical SIMP formulation, as shown in (1), including the independency between the minimum value of the material’s elastic modulus and the penalization power (Sigmund 2007).

However, topology optimization methods are likely to encounter numerical difficulties such as mesh-dependency, checkerboard patterns, and local minima (Bendsøe and Sigmund 2003). In order to mitigate such issues, researchers have proposed the use of regularization techniques (Sigmund and Peterson 1998). One of the most common approaches is the use of density filters (Bruns and Tortorelli 2001). A basic filter density function is defined as

$$\tilde{x}_i = \frac{\sum_{j \in N_i} H_{ij} v_j x_j}{\sum_{j \in N_i} H_{ij} v_j}, \tag{3}$$

where N_i is the neighborhood of an element x_i with volume v_i , and H_{ij} is a weight factor. The neighborhood is defined as

$$N_i = \{j : \text{dist}(i, j) \leq R\}, \tag{4}$$

where the operator $\text{dist}(i, j)$ is the distance between the center of element i and the center of element j , and R is the size of the neighborhood or filter size. The weight factor H_{ij} may be defined as a function of the distance between neighboring elements, for example

$$H_{ij} = R - \text{dist}(i, j), \tag{5}$$

where $j \in N_i$. The filtered density \tilde{x}_i defines a modified (physical) density field that is now incorporated in the topology optimization formulation and the SIMP model as

$$E_i(\tilde{x}_i) = E_{\min} + \tilde{x}_i^p (E_0 - E_{\min}), \quad \tilde{x}_i \in [0, 1]. \tag{6}$$

The regularized SIMP interpolation formula defined by (6) used in this work.

3 Finite element analysis

3.1 Equilibrium equation

Following the regularized SIMP method given by (6) and the generalized Hooke’s law, the three-dimensional constitutive matrix for an isotropic element i is interpolated from void to solid as

$$\mathbf{C}_i(\tilde{x}_i) = E_i(\tilde{x}_i) \mathbf{C}_i^0, \quad \tilde{x}_i \in [0, 1], \tag{7}$$

where \mathbf{C}_i^0 is the constitutive matrix with unit Young’s modulus. The unit constitutive matrix is given by

$$\mathbf{C}_i^0 = \frac{1}{(1 + \nu)(1 - 2\nu)} \times \begin{bmatrix} 1 - \nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1 - \nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1 - \nu & 0 & 0 & 0 \\ 0 & 0 & 0 & (1 - 2\nu)/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & (1 - 2\nu)/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & (1 - 2\nu)/2 \end{bmatrix}, \tag{8}$$

where ν is the Poisson’s ratio of the isotropic material. Using the finite element method, the elastic solid element stiffness matrix is the volume integral of the elements constitutive matrix $\mathbf{C}_i(\tilde{x}_i)$ and the strain–displacement matrix \mathbf{B} in the form of

$$\mathbf{k}_i(\tilde{x}_i) = \int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} \mathbf{B}^T \mathbf{C}_i(\tilde{x}_i) \mathbf{B} d\xi_1 d\xi_2 d\xi_3, \tag{9}$$

where ξ_e ($e = 1, \dots, 3$) are the natural coordinates as shown in Fig. 1, and the hexahedron coordinates of the corners are shown in Table 1. The strain–displacement matrix \mathbf{B} relates the strain $\boldsymbol{\epsilon}$ and the nodal displacement \mathbf{u} , $\boldsymbol{\epsilon} = \mathbf{B}\mathbf{u}$. Using the SIMP method, the element stiffness matrix is interpolated as

$$\mathbf{k}_i(\tilde{x}_i) = E_i(\tilde{x}_i) \mathbf{k}_i^0, \tag{10}$$

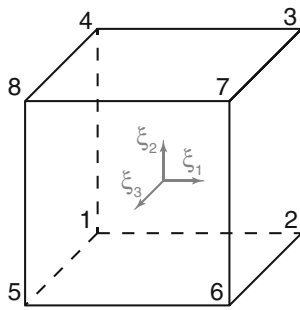


Fig. 1 The eight-node hexahedron and the natural coordinates ξ_1, ξ_2, ξ_3

where

$$\mathbf{k}_i^0 = \int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} \mathbf{B}^T \mathbf{C}^0 \mathbf{B} d\xi_1 d\xi_2 d\xi_3. \tag{11}$$

Replacing values in (11), the 24×24 element stiffness matrix \mathbf{k}_i^0 for an eight-node hexahedral element is

$$\mathbf{k}_i^0 = \frac{1}{(v+1)(1-2\nu)} \begin{bmatrix} \mathbf{k}_1 & \mathbf{k}_2 & \mathbf{k}_3 & \mathbf{k}_4 \\ \mathbf{k}_2^T & \mathbf{k}_5 & \mathbf{k}_6 & \mathbf{k}_4^T \\ \mathbf{k}_3^T & \mathbf{k}_6 & \mathbf{k}_5 & \mathbf{k}_2^T \\ \mathbf{k}_4 & \mathbf{k}_3 & \mathbf{k}_2 & \mathbf{k}_1 \end{bmatrix} \tag{12}$$

where \mathbf{k}_m ($m = 1, \dots, 6$) are 6×6 symmetric matrices (see Appendix A). One can also verify that \mathbf{k}_i^0 is positive definite. The global stiffness matrix \mathbf{K} is obtained by the assembly of element-level counterparts \mathbf{k}_i ,

$$\mathbf{K}(\tilde{\mathbf{x}}) = \mathcal{A}_{i=1}^n \mathbf{k}_i(\tilde{\mathbf{x}}_i) = \mathcal{A}_{i=1}^n E_i(\tilde{\mathbf{x}}_i) \mathbf{k}_i^0, \tag{13}$$

where n is the total number of elements. Using the global versions of the element stiffness matrices \mathbf{K}_i and \mathbf{K}_i^0 , (13) is expressed as

$$\mathbf{K}(\tilde{\mathbf{x}}) = \sum_{i=1}^n \mathbf{K}_i(\tilde{\mathbf{x}}_i) = \sum_{i=1}^n E_i(\tilde{\mathbf{x}}_i) \mathbf{K}_i^0. \tag{14}$$

Table 1 The eight-node hexahedral element with node numbering conventions

Node	ξ_1	ξ_2	ξ_3
1	-1	-1	-1
2	+1	-1	-1
3	+1	+1	-1
4	-1	+1	-1
5	-1	-1	+1
6	+1	-1	+1
7	+1	+1	+1
8	-1	+1	+1

where \mathbf{K}_i^0 is a constant matrix. Using the interpolation function defined in (6), one finally observes that

$$\mathbf{K}(\tilde{\mathbf{x}}) = \sum_{i=1}^n [E_{\min} + \tilde{x}_i^p (E_0 - E_{\min})] \mathbf{K}_i^0. \tag{15}$$

Finally, the nodal displacements vector $\mathbf{U}(\tilde{\mathbf{x}})$ is the solution of the equilibrium equation

$$\mathbf{K}(\tilde{\mathbf{x}}) \mathbf{U}(\tilde{\mathbf{x}}) = \mathbf{F}, \tag{16}$$

where \mathbf{F} is the vector of nodal forces and it is independent of the physical densities $\tilde{\mathbf{x}}$. For brevity of notation, we omitted the dependence of physical densities $\tilde{\mathbf{x}}$ on the design variables \mathbf{x} , $\tilde{\mathbf{x}} = \tilde{\mathbf{x}}(\mathbf{x})$.

3.2 Numerical implementation

Consider the discretized prismatic structure in Fig. 2 composed of eight eight-noded cubic elements. The nodes identified with a number (node ID) ordered column-wise up-to-bottom, left-to-right, and back-to-front. The position of each node is defined with respect to Cartesian coordinate system with origin at the left-bottom-back corner.

Within each element, the eight nodes N_1, \dots, N_8 are ordered in counter-clockwise direction as shown in Fig. 3. Note that the “local” node number (N_i) does not follow the same rule as the “global” node ID (NID_i) system in Fig. 2. Given the size of the volume ($nel_x \times nel_y \times nel_z$) and the global coordinates of node N_1 (x_1, y_1, z_1), one can identify the global node coordinates and node IDs of the other seven nodes in that element by the mapping the relationships as summarized in Table 2.

Each node in the structure has three degrees of freedom (DOFs) corresponding to linear displacements in x - y - z directions (one element has 24 DOFs). The degrees of freedom are organized in the nodal displacement vector \mathbf{U} as

$$\mathbf{U} = [U_{1x}, U_{1y}, U_{1z}, \dots, U_{8 \times nz}]^T,$$

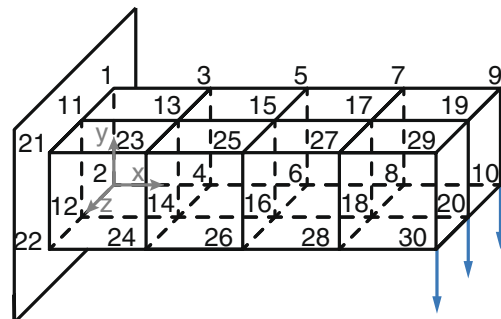


Fig. 2 Global node IDs in a prismatic structure composed of 8 elements

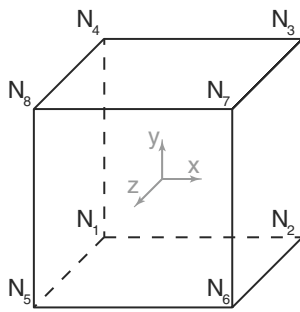


Fig. 3 Local node numbers within a cubic element

where n is the number of elements in the structure. The location of the DOFs in \mathbf{U} , and consequently \mathbf{K} and \mathbf{F} , can be determined from the node ID as shown in Table 2.

The node IDs for each element are organized in a connectivity matrix edofMat with following MATLAB lines:

```

20 nele = nelx*nely*nelz;
26 nodegrd = reshape(1:(nely+1)*(nelx+1),
nely+1,nelx+1);
27 nodeids = reshape(nodegrd(1:end-1,1:end
-1),nely*nelx,1);
28 nodeidz = 0:(nely+1)*(nelx+1):(nelz-1)*
(nely+1)*(nelx+1);
29 nodeids = repmat(nodeids, size(nodeidz))+
repmat(nodeidz, size(nodeids));
30 edofVec = 3*nodeids(:)+1;
31 edofMat = repmat(edofVec,1,24)+ ...
32 repmat([0 1 2 3*nely + [3 4 5 0 1 2]
-3 -2 -1 ...
33 3*(nely+1)*(nelx+1)+[0 1 2 3*nely +
[3 4 5 0 1 2] -3 -2 -1]],nele,1);
    
```

where nele is the total number of elements, nodegrd contains the node ID of the first grid of nodes in the x - y plane (for $z = 0$), the column vector edofVec contains the node IDs of the first node at each element, and the connectivity matrix edofMat of size $\text{nele} \times 24$ containing

the node IDs for each element. For the volume in Fig. 2, $\text{nelx} = 4$, $\text{nely} = 1$, and $\text{nelz} = 2$, which results in

$$\text{edofMat} = \begin{bmatrix} 4 & 5 & 6 & \dots & 31 & 32 & 33 \\ 10 & 11 & 12 & \dots & 37 & 38 & 39 \\ 16 & 17 & 18 & \dots & 43 & 44 & 45 \\ 22 & 23 & 24 & \dots & 49 & 50 & 51 \\ 34 & 35 & 36 & \dots & 61 & 62 & 63 \\ 40 & 41 & 42 & \dots & 67 & 68 & 69 \\ 46 & 47 & 48 & \dots & 73 & 74 & 75 \\ 52 & 53 & 54 & \dots & 79 & 80 & 81 \end{bmatrix} \begin{array}{l} \leftarrow \text{Element 1} \\ \leftarrow \text{Element 2} \\ \leftarrow \text{Element 3} \\ \leftarrow \text{Element 4} \\ \leftarrow \text{Element 5} \\ \leftarrow \text{Element 6} \\ \leftarrow \text{Element 7} \\ \leftarrow \text{Element 8} \end{array}$$

The element connectivity matrix edofMat is used to assemble the global stiffness matrix \mathbf{K} as follows:

```

25 KE = lk_H8(nu);
34 iK = kron(edofMat, ones(24,1))';
35 jK = kron(edofMat, ones(1,24))';
70 sK = KE(:)*(Emin+xPhys(:)).^penal*(E0-
Emin);
71 K = sparse(iK(:),jK(:),sK(:)); K = (K+K
')/2;
    
```

The element stiffness matrix KE (size 24×24) is obtained from the `lk_H8` subroutine (lines 99-146). Matrices iK (size $24 \text{ nele} \times 24$) and jK (size $\text{nele} \times 24^2$), reshaped as column vectors, contain the rows and columns identifying the $24 \times 24 \times \text{nele}$ DOFs in the structure. The three-dimensional array xPhys (size $\text{nely} \times \text{nelx} \times \text{nelz}$) corresponds to the physical densities. The matrix sK (size $24^2 \times \text{nele}$) contains all element stiffness matrices. The assembly procedure of the (sparse symmetric) global stiffness matrix \mathbf{K} (line 71) avoids the use of nested `for` loops.

Finally, the nodal displacement vector \mathbf{U} is obtained from the solution of the equilibrium equation (16) by pre-multiplying the inverse of the stiffness matrix \mathbf{K} and the vector of nodal forces \mathbf{F} ,

```

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

Table 2 Illustration of relationships between node number, node coordinates, node ID and node DOFs

Node Number	Node coordinates	Node ID	Node Degree of Freedoms		
			x	y	z
N_1	(x_1, y_1, z_1)	NID_1^\dagger	$3 * \text{NID}_1 - 2$	$3 * \text{NID}_1 - 1$	$3 * \text{NID}_1$
N_2	$(x_1 + 1, y_1, z_1)$	$\text{NID}_2 = \text{NID}_1 + (\text{nely} + 1)$	$3 * \text{NID}_2 - 2$	$3 * \text{NID}_2 - 1$	$3 * \text{NID}_2$
N_3	$(x_1 + 1, y_1 + 1, z_1)$	$\text{NID}_3 = \text{NID}_1 + \text{nely}$	$3 * \text{NID}_3 - 2$	$3 * \text{NID}_3 - 1$	$3 * \text{NID}_3$
N_4	$(x_1, y_1 + 1, z_1)$	$\text{NID}_4 = \text{NID}_1 - 1$	$3 * \text{NID}_4 - 2$	$3 * \text{NID}_4 - 1$	$3 * \text{NID}_4$
N_5	$(x_1, y_1, z_1 + 1)$	$\text{NID}_5 = \text{NID}_1 + \text{NID}_z^\ddagger$	$3 * \text{NID}_5 - 2$	$3 * \text{NID}_5 - 1$	$3 * \text{NID}_5$
N_6	$(x_1 + 1, y_1, z_1 + 1)$	$\text{NID}_6 = \text{NID}_2 + \text{NID}_z$	$3 * \text{NID}_6 - 2$	$3 * \text{NID}_6 - 1$	$3 * \text{NID}_6$
N_7	$(x_1 + 1, y_1 + 1, z_1 + 1)$	$\text{NID}_7 = \text{NID}_3 + \text{NID}_z$	$3 * \text{NID}_7 - 2$	$3 * \text{NID}_7 - 1$	$3 * \text{NID}_7$
N_8	$(x_1, y_1 + 1, z_1 + 1)$	$\text{NID}_8 = \text{NID}_4 + \text{NID}_z$	$3 * \text{NID}_8 - 2$	$3 * \text{NID}_8 - 1$	$3 * \text{NID}_8$

$^\dagger \text{NID}_1 = z_1 * (\text{nelx} + 1) * (\text{nely} + 1) + x_1 * (\text{nely} + 1) + (\text{nely} + 1 - y_1)$

$^\ddagger \text{NID}_z = (\text{nelx} + 1) * (\text{nely} + 1)$

where the indices `freedofs` indicate the unconstrained DOFs. For the cantilevered structure in Fig. 2, the constrained DOFs

```

16 [jf, kf] = meshgrid(1:nely+1, 1:nelz+1);
17 fixednid = (kf-1)*(nely+1)*(nelx+1)+jf;
18 fixeddof = [3*fixednid(:); 3*fixednid(:)
             -1; 3*fixednid(:)-2];
    
```

where `jf`, and `kf` are the coordinate of the fixed nodes, `fixednid` are the node IDs, and `fixeddof` are the location of the DOFs. The free DOFs, are then defined as

```

21 ndof = 3*(nelx+1)*(nely+1)*(nelz+1);
24 freedofs = setdiff(1:ndof, fixeddof);
    
```

where `ndof` is the total number of DOFs. By default, the code constraints the left face of the prismatic structure and assigns a vertical load to the structure’s free-lower edge as depicted in Fig. 2. The user can define different load and support DOFs by changing the corresponding node coordinates (lines 12 and 16). Several examples are presented in Section 6.

4 Optimization problem formulation

Three representative topology optimization problems are described in this section, namely: minimum compliance, compliant mechanism synthesis, and heat conduction.

4.1 Minimum compliance

The objective of the minimum compliance problem is to find the material density distribution $\tilde{\mathbf{x}}$ that minimizes the structure’s deformation under the prescribed support and loading condition. The structure’s compliance, which provides a global measure of deformation, is defined as

$$c(\tilde{\mathbf{x}}) = \mathbf{F}^T \mathbf{U}(\tilde{\mathbf{x}}), \tag{17}$$

where \mathbf{F} is the vector of nodal forces and $\mathbf{U}(\tilde{\mathbf{x}})$ is the vector of nodal displacements. Incorporating a volume constraint, the minimum compliance optimization problem is

$$\begin{aligned}
 &\text{find } \mathbf{x} = [x_1, x_2, \dots, x_e, \dots, x_n]^T \\
 &\text{minimize } c(\tilde{\mathbf{x}}) = \mathbf{F}^T \mathbf{U}(\tilde{\mathbf{x}}) \\
 &\text{subject to } v(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}}^T \mathbf{v} - \bar{v} \leq 0 \\
 &\quad \mathbf{x} \in \mathcal{X}, \quad \mathcal{X} = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{0} \leq \mathbf{x} \leq \mathbf{1}\},
 \end{aligned} \tag{18}$$

where the physical densities $\tilde{\mathbf{x}} = \tilde{\mathbf{x}}(\tilde{\mathbf{x}})$ are defined by (3), n is the number of elements used to discretize the design domain, $\mathbf{v} = [v_1, \dots, v_n]^T$ is a vector of element volume, and \bar{v} is the prescribed volume limit of the design domain. The nodal force vector \mathbf{F} is independent of the design variables and the nodal displacement vector $\mathbf{U}(\tilde{\mathbf{x}})$ is the solution of $\mathbf{K}(\tilde{\mathbf{x}})\mathbf{U}(\tilde{\mathbf{x}}) = \mathbf{F}$.

The derivative of the volume constraint $v(\tilde{\mathbf{x}})$ in (18) with respect to the design variable x_e is given

$$\frac{\partial v(\tilde{\mathbf{x}})}{\partial x_e} = \sum_{i \in N_e} \frac{\partial v(\tilde{\mathbf{x}})}{\partial \tilde{x}_i} \frac{\partial \tilde{x}_i}{\partial x_e} \tag{19}$$

where

$$\frac{\partial v(\tilde{\mathbf{x}})}{\partial \tilde{x}_i} = v_i \tag{20}$$

and

$$\frac{\partial \tilde{x}_i}{\partial x_e} = \frac{H_{ie} v_e}{\sum_{j \in N_i} H_{ij} v_j}. \tag{21}$$

The code uses a mesh with equally sized cubic elements of unit volume, then $v_i = v_j = v_e = 1$.

The derivative of the compliance is

$$\frac{\partial c(\tilde{\mathbf{x}})}{\partial x_e} = \sum_{i \in N_e} \frac{\partial c(\tilde{\mathbf{x}})}{\partial \tilde{x}_i} \frac{\partial \tilde{x}_i}{\partial x_e} \tag{22}$$

where $\partial \tilde{x}_i / \partial x_e$ is given by (21) and

$$\begin{aligned}
 \frac{\partial c(\tilde{\mathbf{x}})}{\partial \tilde{x}_i} &= \mathbf{F}^T \frac{\partial \mathbf{U}(\tilde{\mathbf{x}})}{\partial \tilde{x}_i} \\
 &= \mathbf{U}(\tilde{\mathbf{x}})^T \mathbf{K}(\tilde{\mathbf{x}}) \frac{\partial \mathbf{U}(\tilde{\mathbf{x}})}{\partial \tilde{x}_i}.
 \end{aligned} \tag{23}$$

The derivative of (16) with respect to \tilde{x}_i is

$$\frac{\partial \mathbf{K}(\tilde{\mathbf{x}})}{\partial \tilde{x}_i} \mathbf{U}(\tilde{\mathbf{x}}) + \mathbf{K}(\tilde{\mathbf{x}}) \frac{\partial \mathbf{U}(\tilde{\mathbf{x}})}{\partial \tilde{x}_i} = \mathbf{0}, \tag{24}$$

which yields

$$\frac{\partial \mathbf{U}(\tilde{\mathbf{x}})}{\partial \tilde{x}_i} = -\mathbf{K}(\tilde{\mathbf{x}})^{-1} \frac{\partial \mathbf{K}(\tilde{\mathbf{x}})}{\partial \tilde{x}_i} \mathbf{U}(\tilde{\mathbf{x}}). \tag{25}$$

Using (15),

$$\begin{aligned}
 \frac{\partial \mathbf{K}(\tilde{\mathbf{x}})}{\partial \tilde{x}_i} &= \frac{\partial}{\partial \tilde{x}_i} \sum_{i=1}^n [E_{\min} + \tilde{x}_i^p (E_0 - E_{\min})] \mathbf{K}_i^0 \\
 &= p \tilde{x}_i^{p-1} (E_0 - E_{\min}) \mathbf{K}_i^0.
 \end{aligned} \tag{26}$$

Using (25) and (26), (23) results in

$$\frac{\partial c(\tilde{\mathbf{x}})}{\partial \tilde{x}_i} = -\mathbf{U}(\tilde{\mathbf{x}})^T [p \tilde{x}_i^{p-1} (E_0 - E_{\min}) \mathbf{K}_i^0] \mathbf{U}(\tilde{\mathbf{x}}). \tag{27}$$

Since \mathbf{K}_i^0 is the global version of an element matrix, (27) may be transformed from the global level to the element level, obtaining

$$\frac{\partial c(\tilde{\mathbf{x}})}{\partial \tilde{x}_i} = -\mathbf{u}_i(\tilde{\mathbf{x}})^T [p \tilde{x}_i^{p-1} (E_0 - E_{\min}) \mathbf{k}_i^0] \mathbf{u}_i(\tilde{\mathbf{x}}). \tag{28}$$

where \mathbf{u}_i is the element vector of nodal displacements. Since \mathbf{k}_i^0 is positive definite, $\partial c(\tilde{\mathbf{x}}) / \partial \tilde{x}_i < 0$.

The numerical implementation of minimum compliance problem can be done as the following:

```

74 ce = reshape(sum((U(edofMat)*KE).*U(
edofMat),2),[nely,nelx,nelz]);
75 c = sum(sum(sum((Emin+xPhys.^penal*(E0-
Emin)).*ce)));
76 dc = -penal*(E0-Emin)*xPhys.^(penal-1).*
ce;
77 dv = ones(nely,nelx,nelz);
79 dc(:) = H*(dc(:)./Hs);
80 dv(:) = H*(dv(:)./Hs);
    
```

The objective function in (18) is calculated in Line 75. The sensitivities of the objective function and volume fraction constraint with respect to the physical density are given by lines 76-77. Finally, the chain rule as stated in (22) is deployed in lines 79-80.

4.2 Compliant mechanism synthesis

A compliant mechanism is a morphing structure that undergoes elastic deformation to transform force, displacement, or energy (Bruns and Tortorelli 2001). A typical goal for a compliant mechanism design is to maximize the output port displacement. The optimization problem is

$$\begin{aligned}
 & \text{find } \tilde{\mathbf{x}} = [x_1, x_2, \dots, x_e, \dots, x_n]^T \\
 & \text{minimize } c(\tilde{\mathbf{x}}) = -u_{\text{out}}(\tilde{\mathbf{x}}) = -\mathbf{L}^T \mathbf{U}(\tilde{\mathbf{x}}) \\
 & \text{subject to } v(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}}^T \mathbf{v} - \bar{v} \leq 0 \\
 & \mathbf{x} \in \mathcal{X}, \quad \mathcal{X} = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{0} \leq \mathbf{x} \leq \mathbf{1}\},
 \end{aligned} \tag{29}$$

where \mathbf{L} is a unit length vector with zeros at all degrees of freedom except at the output point where it is one, and $\mathbf{U}(\tilde{\mathbf{x}}) = \mathbf{K}(\tilde{\mathbf{x}})^{-1} \mathbf{F}$.

To obtain the sensitivity of the new cost function $c(\tilde{\mathbf{x}})$ in (29), let us define a global adjoint vector $\mathbf{U}_d(\tilde{\mathbf{x}})$ from the solution of the adjoint problem

$$\mathbf{K}(\tilde{\mathbf{x}}) \mathbf{U}_d(\tilde{\mathbf{x}}) = -\mathbf{L}. \tag{30}$$

Using (30) in (29), the objective function can be expressed as

$$c(\tilde{\mathbf{x}}) = \mathbf{U}_d(\tilde{\mathbf{x}})^T \mathbf{K}(\tilde{\mathbf{x}}) \mathbf{U}(\tilde{\mathbf{x}}). \tag{31}$$

The derivative of $c(\tilde{\mathbf{x}})$ with respect to the design variable x_e is again obtained by the chain rule,

$$\frac{\partial c(\tilde{\mathbf{x}})}{\partial x_e} = \sum_{i \in N_e} \frac{\partial c(\tilde{\mathbf{x}})}{\partial \tilde{x}_i} \frac{\partial \tilde{x}_i}{\partial x_e},$$

where $\partial \tilde{x}_i / \partial x_e$ is described by (21), and $\partial c(\tilde{\mathbf{x}}) / \partial \tilde{x}_i$ can be obtained using direct differentiation. The use of the interpolation given by (6) yields an expression similar to the one obtained in (28),

$$\frac{\partial c(\tilde{\mathbf{x}})}{\partial \tilde{x}_i} = \mathbf{u}_{di}(\tilde{\mathbf{x}})^T \left[p \tilde{x}_i^{p-1} (E_0 - E_{\min}) \mathbf{k}_i^0 \right] \mathbf{u}_i(\tilde{\mathbf{x}}). \tag{32}$$

where $\mathbf{u}_{di}(\mathbf{k}_i^0)$ is the part of the adjoint vector associated with element i . In this case, $\partial c(\mathbf{k}_i^0) / \partial \tilde{x}_i$ may be positive or negative.

The numerical implementation of the objective function (31) and sensitivity (32) are

```

74a Ud = U(:,2);
74b U = U(:,1);
74 ce = reshape(sum((U(edofMat)*KE).*Ud(
edofMat),2),[nely,nelx,nelz]);
75 c = U(dout,1);
76 dc = penal*(E0-Emin)*xPhys.^(penal-1).*
ce;
77 dv = ones(nely,nelx,nelz);
    
```

Vector \mathbf{U}_d (Line 74a) is the dummy load displacement field and vector \mathbf{U} (line 74b) is the input load displacement. The codes for the implementation of chain rule are not shown above since they are same as lines 79-80.

4.3 Heat conduction

Heat in physics is defined as energy transferred between a system and its surrounding. The direct microscopic exchange of kinetic energy of particles through the boundary between two systems is called diffusion or heat conduction. When a body is at a different temperature from its surrounding, heat flows so that the body and the surroundings reach the same temperature. This condition is known as thermal equilibrium. The equilibrium condition for heat transfer in finite element formulation is described by

$$\mathbf{K}(\mathbf{k}_i^0) \mathbf{U}(\mathbf{k}_i^0) = \mathbf{F},$$

where $\mathbf{U}(\mathbf{k}_i^0)$ now donates the finite element global nodal temperature vector, \mathbf{F} donates the global thermal load vector, and $\mathbf{K}(\mathbf{k}_i^0)$ donates the global thermal conductivity matrix. For a material with isotropic properties, conductivity is the same in all directions.

The optimization problem for heat conduction is

$$\begin{aligned}
 & \text{find } \mathbf{k}_i^0 = [x_1, x_2, \dots, x_e, \dots, x_n]^T \\
 & \text{minimize } c(\mathbf{k}_i^0) = \mathbf{F}^T \mathbf{U}(\mathbf{k}_i^0) \\
 & \text{subject to } v(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}}^T \mathbf{v} - \bar{v} \leq 0 \\
 & \mathbf{x} \in \mathcal{X}, \quad \mathcal{X} = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{0} \leq \mathbf{x} \leq \mathbf{1}\},
 \end{aligned} \tag{33}$$

where $\mathbf{U}(\tilde{\mathbf{x}}) = \mathbf{K}(\tilde{\mathbf{x}})^{-1} \mathbf{F}$, and $\mathbf{K}(\tilde{\mathbf{x}})$ is obtained by the assembly of element thermal conductivity matrices $\mathbf{k}_i(\tilde{x}_i)$. Following the interpolation function in (6), the element conductivity matrix is expressed as

$$\mathbf{k}_i(\tilde{x}_i) = [k_{\min} + (k_0 - k_{\min}) \tilde{x}_i^p] \mathbf{k}_i^0, \tag{34}$$

where k_{\min} and k_0 represent the limits of the material's thermal conductivity coefficient and \mathbf{k}_i^0 donates the element conductivity matrix. Note that (34) may be considered as the distribution of two material phases: a good thermal conduction (k_0) and the other a poor conductor (k_{\min}).

The sensitivity analysis of the cost function in (33) is given by

$$\frac{\partial c(\tilde{\mathbf{x}})}{\partial x_e} = \sum_{i \in N_e} \frac{\partial c(\tilde{\mathbf{x}})}{\partial \tilde{x}_i} \frac{\partial \tilde{x}_i}{\partial x_e},$$

where $\partial \tilde{x}_i / \partial x_e$ is described by (21) and

$$\frac{\partial c(\tilde{\mathbf{x}})}{\partial \tilde{x}_i} = -\mathbf{u}_i(\tilde{\mathbf{x}})^T \left[(k_0 - k_{\min}) p \tilde{x}_i^{p-1} \mathbf{k}_i^0 \right] \mathbf{u}_i(\tilde{\mathbf{x}}). \tag{35}$$

The numerical implementation only requires an optional change in the material property name:

```

74 ce = reshape( sum( (U(edofMat)*KE) .* U(
    edofMat), 2), [nely, nelx, nelz] );
75 c = sum( sum( sum( (kmin+(k0-kmin)*xPhys.^
    penal) .* ce) ) );
76 dc = -penal*(k0-kmin)*xPhys.^(penal-1) .*
    ce;
77 dv = ones( nely, nelx, nelz );
    
```

where k_0 and k_{\min} are the limits of the material’s thermal conductivity. The chain rule is applied same as before.

5 Optimization algorithms

Non-linear programming (NLP) problems, such as minimum compliance (18), compliant mechanism (29), and heat conduction (33), can be addressed using sequential convex approximations such as sequential quadratic programming (SQP) (Wilson 1963) and the method of moving asymptotes (MMA) (Svanberg 1987). The premise of these methods is that, given a current design $\mathbf{x}^{(k)}$, the NLP algorithm is able to find a convex approximation of the original NLP problem from which an improved design $\mathbf{x}^{(k+1)}$ can be derived. The nature of the problem’s approximation is determined by the type of algorithm, e.g., quadratic programming (QP) or MMA. An special case of the latter approach, which is historically older than SQP and MMA, is the optimality criterion (OC) method. This method still find its place in topology optimization due to its numerical simplicity and numerical efficiency (Christensen and Klarbring 2009). The following sections presents the implementation of the SQP, MMA, and OC methods to the solution of the minimum compliance topology optimization problems presented in this paper.

5.1 Sequential quadratic programming

A QP problem has a quadratic objective function and linear constraints (Nocedal and Wright 2006). Given the current design $\mathbf{x}^{(k)}$ and all corresponding active constraints, the QP

approximation of the minimum compliance problem in (18) can be expressed as

$$\begin{aligned} & \text{find } \mathbf{d} \\ & \text{minimize } \frac{1}{2} \mathbf{d}^T \nabla^2 c^{(k)} \mathbf{d} + \nabla c^{(k)T} \mathbf{d} \\ & \text{subject to } \mathbf{A} \mathbf{d} = \mathbf{0}, \end{aligned} \tag{36}$$

where $c^{(k)}$ is the value of the objective function evaluated at $\mathbf{x}^{(k)}$ and $\mathbf{d} = \mathbf{x} - \mathbf{x}^{(k)}$, and \mathbf{A} is the matrix of active constraints. The optimality and feasibility conditions of (36) yield

$$\begin{aligned} \nabla^2 c^{(k)} \mathbf{d} + \mathbf{A}^T \boldsymbol{\lambda} &= -\nabla c^{(k)} \\ \mathbf{A} \mathbf{d} &= \mathbf{0} \end{aligned} \tag{37}$$

This SQP approach, referred to as the *active set* algorithm (Nocedal and Wright 2006), allows to determine a step $\tilde{\mathbf{d}}^{(k)}$ from the solution of the of system of linear equations in (37) expressed as

$$\begin{bmatrix} \nabla^2 c^{(k)} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{d}^{(k)} \\ \boldsymbol{\lambda}^{(k)} \end{bmatrix} = \begin{bmatrix} -\nabla c^{(k)} \\ \mathbf{0} \end{bmatrix}. \tag{38}$$

The updated design is given by

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)} \mathbf{d}^{(k)}, \tag{39}$$

where the step size parameter $\alpha^{(k)}$ is determined by a line search procedure. The Hessian $\nabla^2 c^{(k)}$ can be numerically approximated but, for the problems considered in this paper, one can determined the closed-form expression (see Appendix B), which is given by

$$\frac{\partial^2 c}{\partial \tilde{x}_i \partial \tilde{x}_j} = \begin{cases} 0, & i \neq j, \\ 2 \left[p \tilde{x}_i^{p-1} (E_0 - E_{\min}) \right]^2 & i = j, \\ \left[E_{\min} + \tilde{x}_i^p (E_0 - E_{\min}) \right]^{-1} \mathbf{u}_i^T \mathbf{k}_i^0 \mathbf{u}_i, & \end{cases} \tag{40}$$

This line-search, active-set SQP algorithm is summarized in Algorithm 1.

Algorithm 1 SQP Algorithm

- Choose an initial feasible design $\mathbf{x}^{(0)}$; set $k \leftarrow 0$;
 - while** (convergence criteria are not met) **do**
 - Evaluate $c^{(k)}, \nabla c^{(k)}$ (28), $\nabla^2 c^{(k)}$ (40);
 - Identify active constraint matrix \mathbf{A} in (36);
 - Solve for $\mathbf{d}^{(k)}$ (38);
 - Find appropriate step size $\alpha^{(k)}$;
 - Set $\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \alpha^{(k)} \mathbf{d}^{(k)}$;
 - Set $k \leftarrow k + 1$
 - end while**
-

The implementation of (40) in the program can be done in just two lines, since the term $\mathbf{u}_i^T + \mathbf{k}_i^0 + \mathbf{u}_i$ has already been calculated, namely matrix $\mathbf{c}e$ in line 74:

```

1 Hc = 2*(penal*(E0-Emin)*xPhys.^ (penal-1)
  ).^2 ./ (E0 + (E0-Emin)*xPhys.^ penal) .*
  ce;
2 Hc(:) = H*(Hc(:) ./ Hs);
    
```

Finally, MATLAB has built-in constrained NLP solver `fmincon`. The implementation of using `fmincon` as an optimizer in our `top3d` program is quite easy, but need some reconstructions of the program (one needs to divide program into different subfunctions, e.g., objective function, constraint function, Hessian function). To further assist on the implementation of an SQP strategy, the reader can find a step-by-step tutorial on our website <http://top3dapp.com>.

5.2 Method of moving asymptotes

The MMA algorithm (Svanberg 1987) was proposed to adjust the curvature of the convex linearization (CONLIN) method introduced by Fleury (1989). Give the current design $\mathbf{x}^{(k)}$ the MMA approximation of the minimum compliance problem in (18) yields to the following linear programming problem:

$$\begin{aligned}
 & \text{find } \mathbf{x} \\
 & \text{minimize} \quad - \sum_{i=1}^n \left[\frac{(x_i^{(k)} - L_i^{(k)})^2}{x_i - L_i^{(k)}} \frac{\partial c}{\partial x_i} (\tilde{\mathbf{x}}^{(k)}) \right] \\
 & \text{subject to} \quad \tilde{\mathbf{x}}^T \mathbf{v} - \bar{v} \leq 0 \\
 & \quad \mathbf{x} \in \mathcal{X}^{(k)},
 \end{aligned} \tag{41}$$

where

$$\mathcal{X}^{(k)} = \{ \mathbf{x} \in \mathcal{X} \mid 0.9L_i^{(k)} + 0.1x_i^{(k)} \leq x_i \leq 0.9U_i^{(k)} + 0.1x_i^{(k)}, i = 1, \dots, n \}. \tag{42}$$

The lower and upper asymptotes $L_i^{(k)}$ and $U_i^{(k)}$ are iteratively updated to mitigate oscillation or improve convergence rate. The heuristic rule proposed by Svanberg (1987) is as follows: For $k = 1$ and $k = 2$,

$$\begin{aligned}
 U_i^{(k)} + L_i^{(k)} &= 2x_i^{(k)}, \\
 U_i^{(k)} - L_i^{(k)} &= 1.
 \end{aligned} \tag{43}$$

For $k \geq 3$,

$$\begin{aligned}
 U_i^{(k)} + L_i^{(k)} &= 2x_i^{(k)}, \\
 U_i^{(k)} - L_i^{(k)} &= \gamma_i^{(k)},
 \end{aligned} \tag{44}$$

where

$$\gamma_i^{(k)} = \begin{cases} 0.7 \begin{pmatrix} x_i^{(k)} - x_i^{(k-1)} \\ x_i^{(k)} - x_i^{(k-1)} \end{pmatrix} \begin{pmatrix} x_i^{(k-1)} - x_i^{(k-2)} \\ x_i^{(k-1)} - x_i^{(k-2)} \end{pmatrix} < 0 \\ 1.2 \begin{pmatrix} x_i^{(k)} - x_i^{(k-1)} \\ x_i^{(k)} - x_i^{(k-1)} \end{pmatrix} \begin{pmatrix} x_i^{(k-1)} - x_i^{(k-2)} \\ x_i^{(k-1)} - x_i^{(k-2)} \end{pmatrix} > 0 \\ 1 \begin{pmatrix} x_i^{(k)} - x_i^{(k-1)} \\ x_i^{(k)} - x_i^{(k-1)} \end{pmatrix} \begin{pmatrix} x_i^{(k-1)} - x_i^{(k-2)} \\ x_i^{(k-1)} - x_i^{(k-2)} \end{pmatrix} = 0 \end{cases} \tag{45}$$

Note from (45) that the signs of three successive iterations are stored. If the signs are opposite, meaning x_i oscillates, the two asymptotes are brought closer to $x_i^{(k)}$ to have a more conservative MMA approximation. On the other hand, if the signs are same, the two asymptotes are extended away from $x_i^{(k)}$ in order to speed up the convergence. The MMA algorithm is explained in Algorithm 2.

Algorithm 2 MMA Algorithm

```

Choose an initial feasible design  $\mathbf{x}^{(0)}$ ; set  $k \leftarrow 0$ ;
while (convergence criteria are not met) do
  if  $k = 1$  or  $k = 2$  then
    Update  $L_i^k$ , and  $U_i^k$  using (43);
  else
    Update  $L_i^k$ , and  $U_i^k$  using (44) and (45);
  end if
  Calculate derivate (28);
  Solve the MMA subproblem (41) to obtain  $\tilde{\mathbf{x}}^{(k+1)}$ ;
  Set  $\mathbf{x}^{(k-2)} \leftarrow \mathbf{x}^{(k-1)}$ ,  $\mathbf{x}^{(k-1)} \leftarrow \mathbf{x}^{(k)}$ ,  $\mathbf{x}^{(k)} \leftarrow \tilde{\mathbf{x}}^{(k+1)}$ ;
  Set  $k \leftarrow k + 1$ ;
end while
    
```

The MMA algorithm is available for MATLAB (`mmasub`). The reader may obtain a copy by contacting Prof. Krister Svanberg (<http://www.math.kth.se/~krille/Welcome.html>) from KTH in Stockholm Sweden. Although `mmasub` has total of 29 input and output variables, its implementation for `top3d` is straightforward. The details can be found at <http://top3dapp.com>.

5.3 Optimality criteria

A classical approach to structural optimization problems is the Optimality Criteria (OC) method. The OC method is historically older than sequential approximation methods such as Sequential Linear Programming (SLP) or SQP. The OC method is formulated on the grounds that if constraint $\mathbf{0} \leq \mathbf{x} \leq \mathbf{1}$ is inactive, then convergence is achieved when the KKT condition

$$\frac{\partial c(\tilde{\mathbf{x}})}{\partial x_e} + \lambda \frac{\partial v(\tilde{\mathbf{x}})}{\partial x_e} = 0, \tag{46}$$

is satisfied for $k = 1, \dots, n$, where λ is the Lagrange multiplier associated with the constraint $v(\tilde{\mathbf{x}})$. This optimality condition can be expressed as $B_e = 1$, where

$$B_e = -\frac{\partial c(\tilde{\mathbf{x}})}{\partial x_e} \left(\lambda \frac{\partial v(\tilde{\mathbf{x}})}{\partial x_e} \right)^{-1}. \quad (47)$$

The code implements the OC updating scheme proposed by (Bendsøe 1995) to update design variables:

$$x_e^{\text{new}} = \begin{cases} \max(0, x_e - m), & \text{if } x_e B_e^\eta \leq \max(0, x_e - m), \\ \min(1, x_e + m), & \text{if } x_e B_e^\eta \geq \min(1, x_e + m), \\ x_e B_e^\eta, & \text{otherwise,} \end{cases} \quad (48)$$

where m is a positive move-limit, and η is a numerical damping coefficient. The choice of $m = 0.2$ and $\eta = 0.5$ is recommended for minimum compliance problems (Bendsøe 1995; Sigmund 2001). For compliant mechanisms, $\eta = 0.3$ improves the convergence of the algorithm. The only unknown in (48) is the value of the Lagrange multiplier λ , which satisfies that

$$v(\tilde{\mathbf{x}}(\mathbf{x}^{\text{new}}(\lambda))) = 0. \quad (49)$$

Numerically, λ is found by a root-finding algorithm such as the bisection method. Finally, the termination criteria are satisfied when a maximum number of iterations is reached or

$$\|\mathbf{x}^{\text{new}} - \mathbf{x}\|_\infty \leq \epsilon, \quad (50)$$

where the tolerance ϵ is a relatively small value, for example $\epsilon = 0.01$.

The numerical implementation begins with the initialization of design and physical variables,

```
62 x = repmat(volfrac, [nely, nelx, nelz]);
63 xPhys = x;
```

where `volfrac` represents the volume fraction limit. Initially, the physical densities are assigned a constant uniform value, which is iteratively updated following the OC updating scheme (Algorithm 3).

6 Numerical examples

The code is executed MATLAB with the following command:

```
top3d(nelx, nely, nelz, volfrac, penal, rmin)
```

where `nelx`, `nely`, and `nelz` are number of elements along x , y , and z directions, `volfrac` is the volume fraction limit (\bar{v}), `penal` is the penalization power (p), and `rmin` is filter size (R). User-defined variables are set between lines 3 and 18. These variables determine the material model, termination criteria, loads, and supports. The following examples demonstrate the application of the

Algorithm 3 OC Algorithm

Choose an initial design $\mathbf{x}^{(k)}$; set $k \leftarrow 0$;
while (convergence criteria are not met) **do**
 FE-analysis using (16) to obtain the corresponding nodal displacement $\mathbf{U}^{(k)}$;
 Compute objective function, e.g., compliance c , output displacement u_{out} ;
 Sensitivity analysis by using the equations as discussed in Section 4;
 Apply filter techniques, e.g. (3) in Section 2.3 or any other filters like those discussed in Section 6.1.4
 Update design variables using (48) to obtain $\mathbf{x}^{(k+1)}$;
 Set $\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)}$; Set $k \leftarrow k + 1$;
end while

code to minimum compliance problems, and its extension to compliant mechanism synthesis and heat condition.

6.1 Minimum compliance

By default, the code solves a minimum compliance problem for the cantilevered beam in Fig. 4. The prismatic design domain is fully constrained in one end and a unit distributed vertical load is applied downwards on the lower free edge. Figure 4 shows the topology optimization results for solving minimum compliance problem with the following MATLAB input lines:

```
top3d(60, 20, 4, 0.3, 3, 1.5)
```

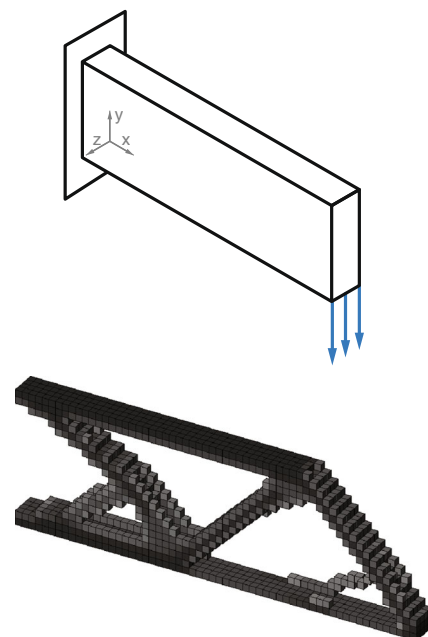


Fig. 4 Topology optimization of 3D cantilever beam. *Top* Initial design domain, *bottom* topology optimized beam

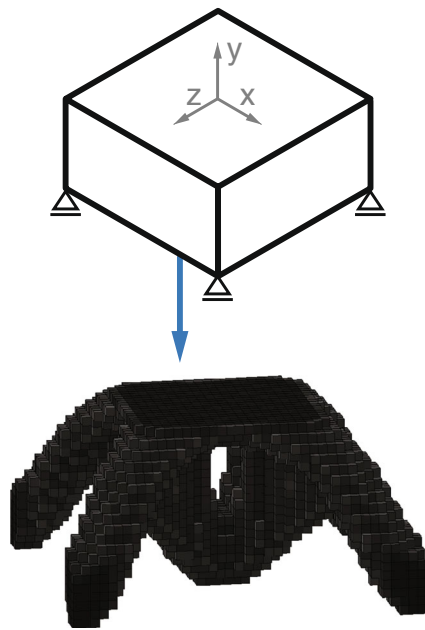


Fig. 5 Topology optimization of 3D wheel. *Top* Initial design domain, *bottom* topology optimized result

6.1.1 Boundary conditions

The boundary conditions and loading conditions are defined in lines 12-18. Since the node coordinates and node numbers are automatically mapped by the program, defining different boundary conditions is very simple. To solve a 3D wheel problem as shown in Fig. 5, which is constrained by planar joint on the corners with a downward point load in the center of the bottom, the following changes need to be made:

Firstly, changing loading conditions

```

12 il = nelx/2; j1 = 0 ; kl = nelz/2;
13 loadnid = kl*(nelx+1)*(nely+1)+il*(nely
14   +1)+(nely+1-j1);
14 load dof = 3*loadnid (:) - 1;

```

Secondly, defining the corresponding boundary conditions

```

16 iif = [0 0 nelx nelx]; jf = [0 0 0 0];
17 kf = [0 nelz 0 nelz];
17 fixednid = kf*(nelx+1)*(nely+1)+iif*(
18   nely+1)+(nely+1-jf);
18 fixeddof = [3*fixednid (:); 3*fixednid (:)
19   -1; 3*fixednid (:) -2];

```

then the problem can be promoted by line:

```
top3d(40,20,40,0.2,3.0,1.5)
```

6.1.2 Multiple load cases

In order to solve a multiple load cases problem, as shown in Fig. 6, a few changes need to be made. First, the loading conditions (line 12) are changed correspondingly:

```

12 il = [nelx nelx]; j1 = [0 nely]; kl = [
13   nelz/2 nelz/2];

```

Also the force vector (line 22) and displacement vector (line 23) become more than one column:

```

22 F = sparse(load dof, [1 2], [-1 1], ndof, 2);
23 U = zeros(ndof, 2);

```

The objective function is now the sum of different load cases

$$c(\tilde{\mathbf{x}}) = \sum_{l=1}^M c_l(\tilde{\mathbf{x}}) = \sum_{l=1}^M \mathbf{F}_l^T \mathbf{U}_l(\tilde{\mathbf{x}}) \tag{51}$$

where M is the number of load cases.

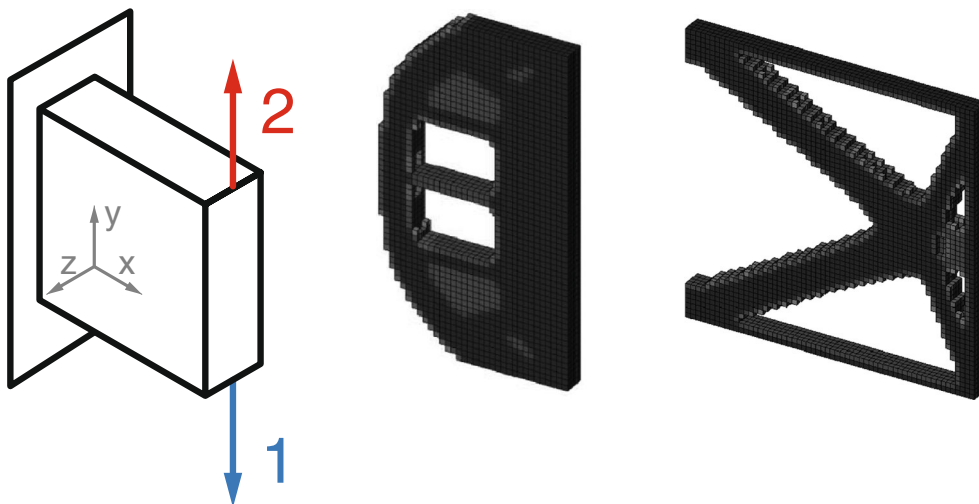


Fig. 6 Topology optimization of cantilever beam with multiple load cases. *Left* Initial design domain, *middle* topology optimized beam with one load case, and *right* topology optimized beam with two load cases

Then lines 74–76 are substituted with lines

```

74a c = 0.;
74b dc = zeros(nely, nelx, nelz);
74c for i = 1:size(F,2)
74d     Ue = U(:, i);
74     ce = reshape(sum((Ue(edofMat)*KE).*
Ue(edofMat),2),[nely, nelx, nelz]);
75     c = c + sum(sum(sum((Emin+xPhys.^
penal*(E0-Emin)).*ce)));
76     dc = dc - penal*(E0-Emin)*xPhys.^(
penal-1).*ce;
76a end

```

This example is promoted by the line

```
top3d(60,60,4,0.4,3.0,1.5)
```

6.1.3 Active and passive elements

In some designs, some elements may be desired to be solid or void. A $nely \times nelx \times nelz$ matrix with ones at elements desired to be solid, and a $nely \times nelx \times nelz$ matrix with zeros at elements desired to be void can be added to the program. To solve the problem as shown in Fig. 7, the passive elements need to be defined first by adding the following lines after line 62:

```

62a for ely = 1:nely
62b     for elx = 1:nelx
62c         if sqrt((ely-nely/2.)^2+(elx-
nelx/3.)^2) < nely/3.
62d             passive(ely, elx) = 1;
62e         else
62f             passive(ely, elx) = 0;
62g         end
62h     end
62i end
62j passive = repmat(passive,[1,1,nelz]);
62k x(find(passive)) = 0;

```

In addition, one line is added in the OC subroutine under line 85:

```
85a xnew(find(passive)) = 0;
```

The optimized beam shown in Fig. 7 is promoted by the line

```
top3d(60,20,4,0.3,3.0,1.5)
```

6.1.4 Alternative filters

In the topology optimization, filters are introduced to avoid numerical instabilities. Different filtering techniques may result different discreteness of the final solutions, and sometimes may even contribute to different topologies. In addition to density filter, in the literatures there are bunch of different filtering schemes. For example, sensitivity filter (Sigmund 1994, 1997), morphology based black and white

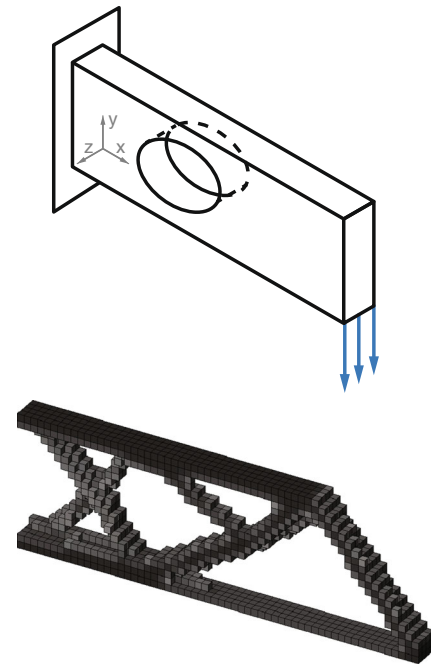


Fig. 7 Topology optimization of cantilever beam with passive design domain. *Top* Initial design domain, *bottom* topology optimized beam

filters (Sigmund 2007), filtering technique using MATLAB built-in function `conv2` (Andreassen et al. 2011), filtering based on Helmholtz type differential equations (Andreassen et al. 2011), Heaviside filter (Guest et al. 2004, 2011), and gray scale filter (Groenwold and Etman 2009). All the filters pursue a simple goal to achieve black-and-white structures. Two of them are chosen, which stand for classic and better performance, as well as easy implementation.

Sensitivity filter Sigmund (1994, 1997) introduced the sensitivity filter. The working principle is to replace the real sensitivities by the filtered sensitivities. In addition, the sensitivity filter is implemented in the 99-line code as the default filtering scheme. It modifies the element sensitivity during every iteration by the following

$$\frac{\widehat{\partial c(\mathbf{x})}}{\partial x_i} = \frac{1}{\max(\gamma, x_i) \sum_{j \in N_i} H_{ij}} \sum_{j \in N_i} H_{ij} x_j \frac{\partial c(\mathbf{x})}{\partial x_j}.$$

where $\gamma (= 10^{-3})$ is a small number in order to avoid division by zero.

The implementation of the sensitivity filter can be achieved by adding and changing a few lines.

Change line 2 by adding one input variable `ft` (`ft = 1` for density filter, `ft = 2` for sensitivity filter)

```
2 function top3dft(nelx, nely, nelz, volfrac,
penal, rmin, ft)
```

Adding the sensitivity filter to the program, by changing lines 79–80

```

79a if ft == 1
79b   dc(:) = H*(dc(:) ./ Hs);
79c   dv(:) = H*(dv(:) ./ Hs);
80a elseif ft == 2
80b   dc(:) = H*(x(:) .* dc(:)) ./ Hs ./ max(1e
      -3, x(:));
80c end

```

Changing the design variable update strategy (line 86) in the optimal search procedure

```

86a if ft == 1
86b   xPhys(:) = (H*xnew(:)) ./ Hs;
86c elseif ft == 2
86d   xPhys = xnew;
86e end

```

Gray scale filter A simple non-linear gray-scale filter or intermediate density filter has been proposed by Groenwold and Etman (2009) to further achieve black-and-white topologies. The implementation of the gray scale filter is by changing the OC update scheme as the following

$$x_i^{new} = \begin{cases} \max(0, x_i - m), & \text{if } x_i B_i^\eta \leq \max(0, x_i - m) \\ \min(1, x_i + m), & \text{if } x_i B_i^\eta \geq \min(1, x_i + m) \\ (x_i B_i^\eta)^q, & \text{otherwise} \end{cases} \quad (52)$$

The standard OC updating method is a special case of (52) with $q = 1$. A typical value of q for the SIMP-based topology optimization is $q = 2$.

The implementation of the gray scale filter to the code can be done as follows:

Adding one input variable q to the program (line 2)

```

2 function top3dgsf(nelx, nely, nelz, volfrac,
  penal, q, rmin)

```

Change the OC updating method (line 85) to

```

85 xnew = max(0, max(x-move, min(1, min(x+move
  , (x .* sqrt(-dc ./ dv / lmid)).^q)))));

```

The factor q should be increased gradually by adding one line after line 68

```

68a if loop <= 15, q = 1; else q = min(qmax
  , 1.01*q); end

```

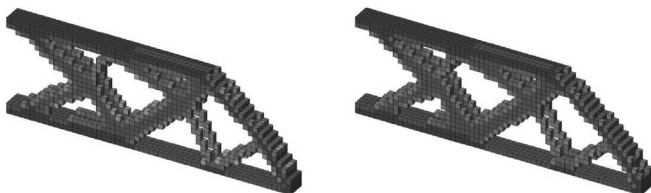


Table 3 Time usage of finite element analysis time for different solvers

Mesh size	Direct solver	Iterative solver
30 × 10 × 2	0.018 sec	0.129 sec
60 × 20 × 4	0.325 sec	0.751 sec
150 × 50 × 10	74.474 sec	22.445 sec

Figure 8 demonstrates the optimized beams applying different filtering techniques. As can be seen from final results, both sensitivity filter, density filter and gray scale filter suppress checkerboard patterns. The gray scale filter combines with the sensitivity filter provides the most black-and-white solution.

6.1.5 Iterative solver

If the finite element mesh size becomes large, the traditional direct solver (line 72) used to address the finite element analysis is suffered by longer solving time and some other issues. However, iterative solver (Hestenes and Stiefel 1952; Augarde et al. 2006) can solve large-scale problems efficiently. To this end, line 72 is replaced by a built-in MATLAB function `pcg`, called preconditioned conjugate gradients method, as shown in the following

```

72a tolit = 1e-8;
72b maxit = 8000;
72c M = diag(diag(K(freedofs, freedofs)));
72   U(freedofs, :) = pcg(K(freedofs, freedofs)
  ), F(freedofs, :), tolit, maxit, M);

```

Direct solver is a special case by setting the preconditioner (line 72c) to $M = \text{inv}(K)$.

Table 3 gives the comparison of two different finite element analysis solvers. As shown in the table, a speed up factor of 30.81 has been measured when solving large scale problem. Hence, the iterative solver is more suitable for large-scale problems, and vice versa.

Some examples include a cantilever beam, the Messerschmitt-Bölkow-Blohm (MBB) beam and L-shape problems (Table 4) are solved by using iterative solver

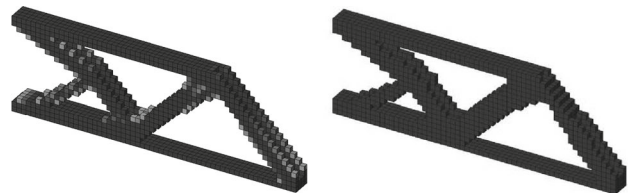
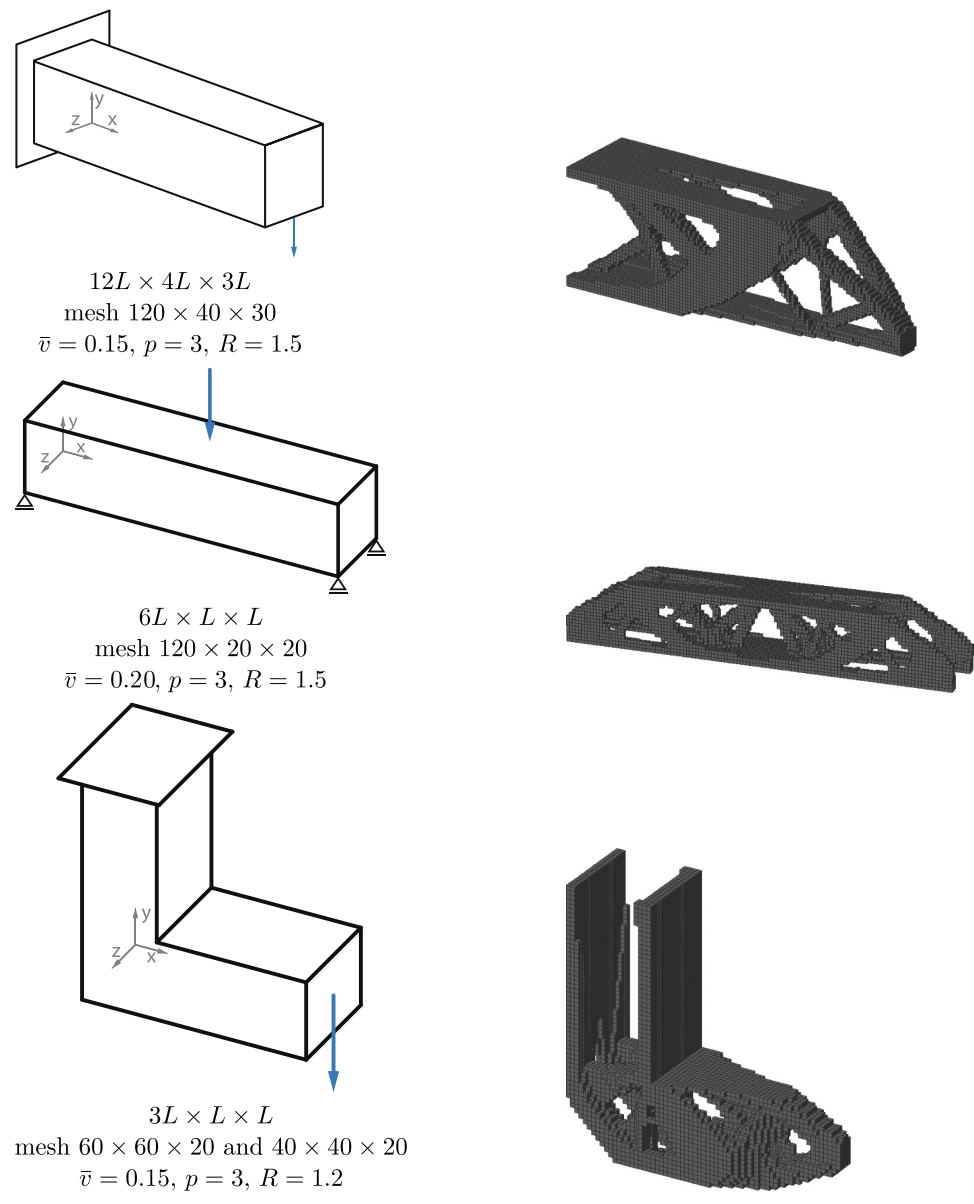


Fig. 8 Topology optimized design used a mesh with 30 × 10 × 2 elements. *Left* optimized design using density filter, *middle left* optimized design using density filter, *middle right* optimized design using density

filter and gray scale filter, and *right* optimized design using sensitivity filter and gray scale filter

Table 4 Three-dimensional examples: Cantilever, MBB, and L-shape problem. Left: Initial design domains, right: topology optimized results



and applying gray scale filter. The underlined triangle represents a three-dimensional planar joint.

6.1.6 Continuation strategy

Convexity is a very preferable property since every local minima is also the global minima, and what the program is solving for is the global minima. Unfortunately, the use of SIMP method to achieve binary solution will destroy the convexity of the optimization problem. For such problems, it is possible that for different starting points the program converges to totally different local minima. In order to penalize intermediate densities and mitigate the premature convergence to one of the multiple local minima when solving the non-convex problem, one could

perform a continuation step. As previously presented by Groenwold and Etman (2010), the continuation step is given as

$$p^k = \begin{cases} 1 & k \leq 20, \\ \min\{p^{\max}, 1.02p^{k-1}\} & k > 20, \end{cases} \quad (53)$$

where k is the iteration number, and p^{\max} is the maximum penalization power.

Though this methodology is not proven to converge to the global optimum, it regularizes the algorithm and allows the comparison of different optimization strategies.

Implementing the continuation strategy is done by adding a single line after line 68:

```
68a  if loop <= 20, penal = 1; else penal =
      min(pmax, 1.02*penal); end
```

6.2 Compliant mechanism synthesis

A compliant mechanism problem involves loading cases: input loading case and dummy loading case. The code also needs to implement a new objective function and its corresponding sensitivity analysis. To demonstrate this implementation, let us consider a three-dimensional force inverter problem as shown in Fig. 9. With an input load defined in the positive direction, the design goal is to maximize the negative horizontal output displacement. Both the top face and the side force are imposed with symmetric constraints; i.e., nodes can only move within the plane.

The new loading conditions as well as input and output points are defined as follows:

```

11 % USER-DEFINED LOAD DOFs
11a % [Input Dummy]
12 il = [0 nelx]; j1 = [nely nely]; kl = [0
    0];
13 loadnid = kl*(nelx+1)*(nely+1)+il*(nely
    +1)+(nely+1-j1);
14 loaddof = 3*loadnid(:) - 2;
14a din = loaddof(1); dout = loaddof(2);
22 F = sparse(loaddof,[1 2],[1 -1],ndof,2);

```

and the boundary conditions are defined as below:

```

15 % USER-DEFINED SUPPORT FIXED DOFs
15a % Top face
15b [iif, kf] = meshgrid(0:nelx,0:nelz);
15c fixednid = kf*(nelx+1)*(nely+1)+iif*(
    nely+1)+1;
15d fixeddof_t = 3*fixednid(:) - 1;
16 % Side face
16a [iif, jf] = meshgrid(0:nelx,0:nely);
16b fixednid = iif*(nely+1)+(nely+1-jf);
16c fixeddof_s = 3*fixednid(:);
17 % Two pins
17a iif = [0 0]; jf = [0 1]; kf = [nelz nelz
    ];
17b fixednid = kf*(nelx+1)*(nely+1)+iif*(
    nely+1)+(nely+1-jf);
17c fixeddof_p = [3*fixednid(:); 3*fixednid
    (:)-1; 3*fixednid(:)-2];
18 % Fixed DOFs
18a fixeddof = union(fixeddof_t, fixeddof_s);
18b fixeddof = union(fixeddof, fixeddof_p);
18c fixeddof = sort(fixeddof);
23a % Displacement vector
23 U = zeros(ndof,2);
24 freedofs = setdiff(1:ndof, fixeddof);

```

The external springs with stiffness 0.1 are added at input and output points after line 71.

```

71a K(din, din) = K(din, din) + 0.1;
71b K(dout, dout) = K(dout, dout) + 0.1;

```

The expressions of the objective function (31) and sensitivity (32) are modified in lines 74-76.

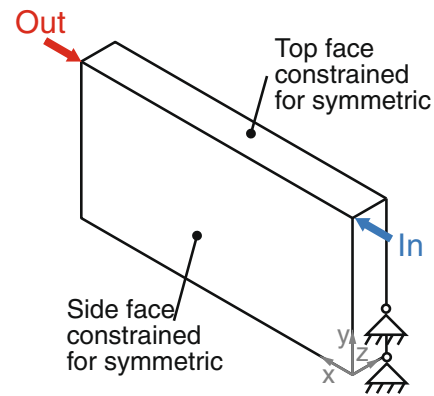


Fig. 9 Design domain of 3D force inverter problem

```

74a Ud = U(:,2);
74b U = U(:,1);
74 ce = reshape(sum((U(edofMat))*KE).*Ud(
    edofMat),2),[nely, nelx, nelz]);
75 c = U(dout,1);
76 dc = penal*(E0-Emin)*xPhys.^(penal-1).*
    ce;

```

The convergence criteria for the bi-sectioning algorithm (lines 82-83) is improved by the following lines:

```

82 l1 = 0; l2 = 1e9; move = 0.1;
83 while (l2-l1)/(l1+l2) > 1e-4 && l2 > 1e
    -40

```

To improve the convergence stability, the damping factor of OC-method changes from 0.5 to 0.3 and also takes the positive sensitivities into account, then line 85 is changed to:

```

85 xnew = max(0, max(x-move, min(1, min(x+move
    , x.*(max(1e-10, -dc./dv/lmid)).^0.3))));

```

The final design shown in Fig. 10 is promoted by the line in the MATLAB:

```
top3d(40,20,5,0.30,3.0,1.5)
```

6.3 Heat conduction

The implementation of heat conduction problems is not more complex than the one for compliant mechanism synthesis since the number of DOF per node is one rather

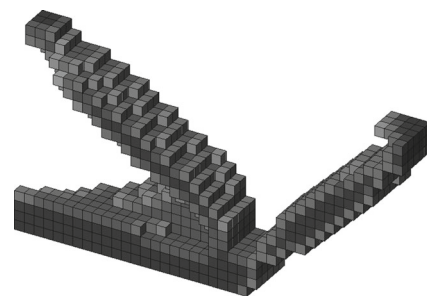


Fig. 10 Topology optimized force inverter

than three. Following the implementation of heat conduction problems in two dimensions (Bendsøe and Sigmund 2003), the implementation for three dimension problems is suggested in the following steps.

First, the elastic material properties (lines 8-10) are changed to the thermal conductivities of materials

```
8 k0 = 1; % Good thermal conductivity
9 kmin = 1e-3; % Poor thermal conductivity
```

Furthermore, the boundary conditions for the heat conduction problem, i.e., a rectangular plate with a heat sink on the middle of top face and all nodes are given a thermal load as shown in Fig. 11, are changed corresponding (lines 10-18).

```
10 % USER-DEFINED SUPPORT FIXED DOFS
11 i1 = nelx/2-nelx/20:nelx/2+nelx/20; j1 =
    nely; k1 = 0:nelz;
12 fixedxy = i1*(nely+1)+(nely+1-j1);
13 fixednid = repmat(fixedxy', size(k1)) +
    ...
14 repmat(k1*(nelx+1)*(nely+1), size(
    fixedxy, 2), 1);
15 fixeddof = reshape(fixednid, [], 1);
```

Also, since there is only one DOF per node in heat conduction problems, some variables need to change correspondingly, such as `ndof`, `edofMat`.

Change the total number of DOFs and the force vector in lines 21–22

```
21 ndof = (nelx+1)*(nely+1)*(nelz+1);
22 F = sparse(1:ndof, 1, -0.01, ndof, 1);
```

The element conductivity matrix is called in line 25 by

```
25 KE = lk_H8(k0);
```

and it is defined in lines 99–145

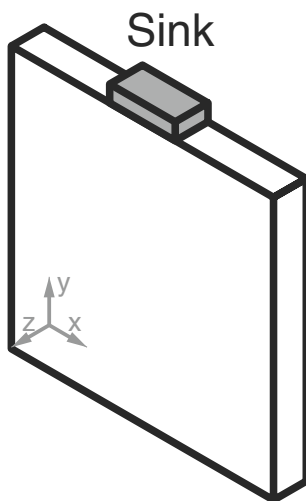


Fig. 11 Initial design domain of heat conduction problem

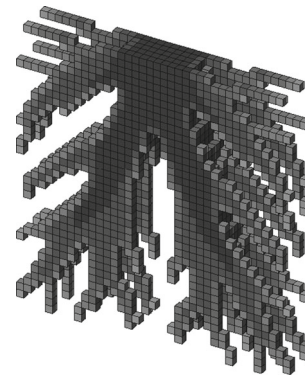


Fig. 12 Resulting topology of heat conduction problem

```
99 function [KE] = lk_H8(k)
100 A1 = 4*eye(2); A2 = -eye(2);
101 A3 = fliplr(A2); A4 = -ones(2);
102 KE1 = [A1 A2; A2 A1];
103 KE2 = [A3 A4; A4 A3];
104 KE = 1/12 * k * [KE1 KE2; KE2 KE1];
105 end
```

The finite element connectivity matrix `edofMat` is changed in lines 30–35

```
30 edofVec = nodeids(:)+1;
31 edofMat = repmat(edofVec, 1, 8)+ ...
32 repmat([0 nely + [1 0] -1 ...
33 (nely+1)*(nelx+1)+[0 nely + [1 0]
    -1]], nele, 1);
34 iK = reshape(kron(edofMat, ones(8,1))
    ', 8*8*nele, 1);
35 jK = reshape(kron(edofMat, ones(1,8))
    ', 8*8*nele, 1);
```

The global conductivity matrix is assembled in a different way, hence line 70 need to change as

```
70 sK = reshape(KE(:)*(kmin+(1-kmin)*xPhys
    (:).^penal), 8*8*nele, 1);
```

The evolution of the objective function and analysis of the sensitivity are given in lines 75–76

```
75 c = sum(sum(sum((kmin+(1-kmin)*xPhys.^
    penal).*ce)));
76 dc = -penal*(1-kmin)*xPhys.^(penal-1).*
    ce;
```

The optimized topology is derived as shown in Fig. 12 by promoting the following line in the MATLAB:

```
top3d(40, 40, 5, 0.30, 3.0, 1.4)
```

7 Conclusions

This paper presents MATLAB the analytical elements and the numerical implementation of an academic three-dimensional structural topology optimization algorithm referred to as `top3d`. In this topology optimization algorithm, the problem formulation follows a density-based

approach with a modified SIMP interpolation for physical densities. The finite element formulation makes use of eight-node hexahedral elements for which a closed-form expression of the element stiffness matrix is derived and numerically implemented. The hexahedral finite elements are used to uniformly discretize a prismatic design domain and solve three related topology optimization problems: minimum compliance, compliant mechanism, and heat conduction problems. For each problem, this paper includes the analytical derivation of the sensitivity coefficients used by three gradient-based optimization algorithms: SQP, MMA, and OC, which is implemented by default. For the implementation of SQP, this paper derives an analytic expression for the second order derivative.

The use of `top3d` is demonstrated through several numerical examples. These examples include problems with a variety of boundary conditions, multiple load cases, active and passive elements, filters, and continuation strategies to mitigate convergence to a local minimum. The architecture of the code allows the user to map node coordinates of node degrees-of-freedom boundary conditions. In addition, the paper provides a strategy to handle large models with the use of an iterative solver. For large-scale finite-element models, the iterative solver is about 30 times faster than the traditional direct solver. While this implementation is limited to linear topology optimization problems with a linear constraint, it provides a clear perspective of the analytical and numerical effort involved in addressing three-dimensional structural topology optimization problems. Finally, additional academic resources such the use of MMA and SQP are available at <http://top3dapp.com>.

Appendix A: Symbolic expression of \mathbf{k}^0

This appendix presents the analytical results of the element stiffness matrix \mathbf{k}^0 as discussed in Section 3. This symbolic expression of \mathbf{k}^0 is also been used by the program subroutine `lk_H8`.

Recall that, for an eight-node hexahedral element, the strain-displacement matrix \mathbf{B} is defined by

$$\mathbf{B} = \begin{bmatrix} \frac{\partial n_1(\xi_e)}{\partial \xi_1} & 0 & 0 & \dots & \frac{\partial n_q(\xi_e)}{\partial \xi_1} & 0 & 0 \\ 0 & \frac{\partial n_1(\xi_e)}{\partial \xi_2} & 0 & \dots & 0 & \frac{\partial n_q(\xi_e)}{\partial \xi_2} & \\ 0 & 0 & \frac{\partial n_1(\xi_e)}{\partial \xi_3} & \dots & 0 & 0 & \frac{\partial n_q(\xi_e)}{\partial \xi_3} \\ \frac{\partial n_1(\xi_e)}{\partial \xi_2} & \frac{\partial n_1(\xi_e)}{\partial \xi_1} & 0 & \dots & \frac{\partial n_q(\xi_e)}{\partial \xi_2} & \frac{\partial n_q(\xi_e)}{\partial \xi_1} & 0 \\ 0 & \frac{\partial n_1(\xi_e)}{\partial \xi_3} & \frac{\partial n_1(\xi_e)}{\partial \xi_2} & \dots & 0 & \frac{\partial n_q(\xi_e)}{\partial \xi_3} & \frac{\partial n_q(\xi_e)}{\partial \xi_2} \\ \frac{\partial n_1(\xi_e)}{\partial \xi_3} & 0 & \frac{\partial n_1(\xi_e)}{\partial \xi_1} & \dots & \frac{\partial n_q(\xi_e)}{\partial \xi_3} & 0 & \frac{\partial n_q(\xi_e)}{\partial \xi_1} \end{bmatrix},$$

for $e = 1, \dots, 3$ and $q = 1, \dots, 8$. The corresponding shape functions n_q in a natural coordinate systems ξ_e are defined by

$$n_q(\xi_e) = \frac{1}{8} \begin{Bmatrix} (1 - \xi_1)(1 - \xi_2)(1 - \xi_3) \\ (1 + \xi_1)(1 - \xi_2)(1 - \xi_3) \\ (1 + \xi_1)(1 + \xi_2)(1 - \xi_3) \\ (1 - \xi_1)(1 + \xi_2)(1 - \xi_3) \\ (1 - \xi_1)(1 - \xi_2)(1 + \xi_3) \\ (1 + \xi_1)(1 - \xi_2)(1 + \xi_3) \\ (1 + \xi_1)(1 + \xi_2)(1 + \xi_3) \\ (1 - \xi_1)(1 + \xi_2)(1 + \xi_3) \end{Bmatrix}.$$

Substituting values to (11), the 24×24 element stiffness matrix \mathbf{k}_i^0 for an eight-node hexahedral element can be expressed as

$$\mathbf{k}_i^0 = \frac{1}{(\nu + 1)(1 - 2\nu)} \begin{bmatrix} \mathbf{k}_1 & \mathbf{k}_2 & \mathbf{k}_3 & \mathbf{k}_4 \\ \mathbf{k}_2^T & \mathbf{k}_5 & \mathbf{k}_6 & \mathbf{k}_4^T \\ \mathbf{k}_3^T & \mathbf{k}_6 & \mathbf{k}_5^T & \mathbf{k}_2^T \\ \mathbf{k}_4 & \mathbf{k}_3 & \mathbf{k}_2 & \mathbf{k}_1^T \end{bmatrix},$$

where

$$\mathbf{k}_1 = \begin{bmatrix} k_1 & k_2 & k_2 & k_3 & k_5 & k_5 \\ k_2 & k_1 & k_2 & k_4 & k_6 & k_7 \\ k_2 & k_2 & k_1 & k_4 & k_7 & k_6 \\ k_3 & k_4 & k_4 & k_1 & k_8 & k_8 \\ k_5 & k_6 & k_7 & k_8 & k_1 & k_2 \\ k_5 & k_7 & k_6 & k_8 & k_2 & k_1 \end{bmatrix},$$

$$\mathbf{k}_2 = \begin{bmatrix} k_9 & k_8 & k_{12} & k_6 & k_4 & k_7 \\ k_8 & k_9 & k_{12} & k_5 & k_3 & k_5 \\ k_{10} & k_{10} & k_{13} & k_7 & k_4 & k_6 \\ k_6 & k_5 & k_{11} & k_9 & k_2 & k_{10} \\ k_4 & k_3 & k_5 & k_2 & k_9 & k_{12} \\ k_{11} & k_4 & k_6 & k_{12} & k_{10} & k_{13} \end{bmatrix},$$

$$\mathbf{k}_3 = \begin{bmatrix} k_6 & k_7 & k_4 & k_9 & k_{12} & k_8 \\ k_7 & k_6 & k_4 & k_{10} & k_{13} & k_{10} \\ k_5 & k_5 & k_3 & k_8 & k_{12} & k_9 \\ k_9 & k_{10} & k_2 & k_6 & k_{11} & k_5 \\ k_{12} & k_{13} & k_{10} & k_{11} & k_6 & k_4 \\ k_2 & k_{12} & k_9 & k_4 & k_5 & k_3 \end{bmatrix},$$

$$\mathbf{k}_4 = \begin{bmatrix} k_{14} & k_{11} & k_{11} & k_{13} & k_{10} & k_{10} \\ k_{11} & k_{14} & k_{11} & k_{12} & k_9 & k_8 \\ k_{11} & k_{11} & k_{14} & k_{12} & k_8 & k_9 \\ k_{13} & k_{12} & k_{12} & k_{14} & k_7 & k_7 \\ k_{10} & k_9 & k_8 & k_7 & k_{14} & k_{11} \\ k_{10} & k_8 & k_9 & k_7 & k_{11} & k_{14} \end{bmatrix},$$

$$\mathbf{k}_5 = \begin{bmatrix} k_1 & k_2 & k_8 & k_3 & k_5 & k_4 \\ k_2 & k_1 & k_8 & k_4 & k_6 & k_{11} \\ k_8 & k_8 & k_1 & k_5 & k_{11} & k_6 \\ k_3 & k_4 & k_5 & k_1 & k_8 & k_2 \\ k_5 & k_6 & k_{11} & k_8 & k_1 & k_8 \\ k_4 & k_{11} & k_6 & k_2 & k_8 & k_1 \end{bmatrix},$$

$$\mathbf{k}_6 = \begin{bmatrix} k_{14} & k_{11} & k_7 & k_{13} & k_{10} & k_{12} \\ k_{11} & k_{14} & k_7 & k_{12} & k_9 & k_2 \\ k_7 & k_7 & k_{14} & k_{10} & k_2 & k_9 \\ k_{13} & k_{12} & k_{10} & k_{14} & k_7 & k_{11} \\ k_{10} & k_9 & k_2 & k_7 & k_{14} & k_7 \\ k_{12} & k_2 & k_9 & k_{11} & k_7 & k_{14} \end{bmatrix},$$

and

$$\begin{aligned} k_1 &= -(6\nu - 4)/9, \\ k_2 &= 1/12, \\ k_3 &= -1/9, \\ k_4 &= -(4\nu - 1)/12, \\ k_5 &= (4\nu - 1)/12, \\ k_6 &= 1/18, \\ k_7 &= 1/24, \\ k_8 &= -1/12, \\ k_9 &= (6\nu - 5)/36, \\ k_{10} &= -(4\nu - 1)/24, \\ k_{11} &= -1/24, \\ k_{12} &= (4\nu - 1)/24, \\ k_{13} &= (3\nu - 1)/18, \\ k_{14} &= (3\nu - 2)/18. \end{aligned}$$

As can be seen from above, the 64×64 entries in the element stiffness matrix can be represented by fourteen components (but not independent!).

Appendix B: Derivation of the Hessian Matrix

In this appendix, we will discuss the derivation of the second order derivative of the objective function. Structure compliance will be used as the objective function, the expression for other objective functions can be derived similarly.

Recall that the first order derivative of the compliance is given by

$$\frac{\partial c}{\partial \tilde{x}_i} = -\mathbf{u}_i^T \left(\frac{\partial \mathbf{k}_i}{\partial \tilde{x}_i} \right) \mathbf{u}_i,$$

where by applying the modified SIMP method (6), we have

$$\frac{\partial \mathbf{k}_i}{\partial \tilde{x}_i} = p \tilde{x}_i^{p-1} (E_0 - E_{\min}) \mathbf{k}_i^0. \tag{B1}$$

Note that for brevity of notation, we omitted the dependence on $\tilde{\mathbf{x}}$ in this appendix.

A second differentiation of the compliance yields

$$\begin{aligned} \frac{\partial^2 c}{\partial \tilde{x}_i \partial \tilde{x}_j} &= \frac{\partial}{\partial \tilde{x}_j} \left[-\mathbf{u}_i^T \frac{\partial \mathbf{k}_i}{\partial \tilde{x}_i} \mathbf{u}_i \right] \\ &= -\frac{\partial \mathbf{u}_i^T}{\partial \tilde{x}_j} \left(\frac{\partial \mathbf{k}_i}{\partial \tilde{x}_i} \right) \mathbf{u}_i - \mathbf{u}_i^T \left(\frac{\partial \mathbf{k}_i}{\partial \tilde{x}_i \partial \tilde{x}_j} \right) \mathbf{u}_i \\ &\quad - \mathbf{u}_i^T \left(\frac{\partial \mathbf{k}_i}{\partial \tilde{x}_i} \right) \frac{\partial \mathbf{u}_i}{\partial \tilde{x}_j} \end{aligned} \tag{B2}$$

From (B1), the middle term of (B2) goes to zero. In order to get the expression for $\partial \mathbf{u}_i / \partial \tilde{x}_j$ in the first and third terms, let us rewriting (16) as

$$\mathbf{k}_i \mathbf{u}_i = \mathbf{f}_i.$$

Now if we differentiate both sides with respect to \tilde{x}_j , we get

$$\frac{\partial \mathbf{k}_i}{\partial \tilde{x}_j} \mathbf{u}_i + \mathbf{k}_i \frac{\partial \mathbf{u}_i}{\partial \tilde{x}_j} = \mathbf{0},$$

which yields

$$\frac{\partial \mathbf{u}_i}{\partial \tilde{x}_j} = -\mathbf{k}_i^{-1} \left(\frac{\partial \mathbf{k}_i}{\partial \tilde{x}_j} \right) \mathbf{u}_i. \tag{B3}$$

Substitute (B3) into (B1), we have

$$\begin{aligned} \frac{\partial^2 c}{\partial \tilde{x}_i \partial \tilde{x}_j} &= - \left[-\mathbf{k}_i^{-1} \left(\frac{\partial \mathbf{k}_i}{\partial \tilde{x}_j} \right) \mathbf{u}_i \right]^T \frac{\partial \mathbf{k}_i}{\partial \tilde{x}_i} \mathbf{u}_i - \mathbf{u}_i^T \frac{\partial \mathbf{k}_i}{\partial \tilde{x}_i} \left[-\mathbf{k}_i^{-1} \left(\frac{\partial \mathbf{k}_i}{\partial \tilde{x}_j} \right) \mathbf{u}_i \right], \\ &= 2 \mathbf{u}_i^T \left(\frac{\partial \mathbf{k}_i}{\partial \tilde{x}_j} \right) \mathbf{k}_i^{-1} \left(\frac{\partial \mathbf{k}_i}{\partial \tilde{x}_i} \right) \mathbf{u}_i, \end{aligned}$$

where the last equality holds since

$$\begin{aligned} \mathbf{u}_i^T \left(\frac{\partial \mathbf{k}_i}{\partial \tilde{x}_j} \right) \mathbf{k}_i^{-1} \left(\frac{\partial \mathbf{k}_i}{\partial \tilde{x}_i} \right) \tilde{\mathbf{u}}_i &= \left\{ \left[\mathbf{u}_i^T \left(\frac{\partial \mathbf{k}_i}{\partial \tilde{x}_j} \right) \mathbf{k}_i^{-1} \right] \left[\left(\frac{\partial \mathbf{k}_i}{\partial \tilde{x}_i} \right) \mathbf{u}_i \right] \right\}^T \\ &= \left[\left(\frac{\partial \mathbf{k}_i}{\partial \tilde{x}_i} \right) \mathbf{u}_i \right]^T \left[\mathbf{u}_i^T \left(\frac{\partial \mathbf{k}_i}{\partial \tilde{x}_j} \right) \mathbf{k}_i^{-1} \right]^T \\ &= \mathbf{u}_i^T \left(\frac{\partial \mathbf{k}_i}{\partial \tilde{x}_i} \right) \mathbf{k}_i^{-1} \left(\frac{\partial \mathbf{k}_i}{\partial \tilde{x}_j} \right) \mathbf{u}_i. \end{aligned} \tag{B4}$$

As discussed earlier, when $i \neq j$, $\partial \mathbf{k}_i / \partial \tilde{x}_j = 0$. When $i = j$, substituting (B1), (10) and (6) into (B4), we have

$$\begin{aligned} \frac{\partial^2 c}{\partial \tilde{x}_i^2} &= 2 \mathbf{u}_i^T \left[p \tilde{x}_i^{p-1} (E_0 - E_{\min}) \mathbf{k}_i^0 \right] [E_{\min} + \\ &\quad \tilde{x}_i^p (E_0 - E_{\min}) \mathbf{k}_i^0]^{-1} \left[p \tilde{x}_i^{p-1} (E_0 - E_{\min}) \mathbf{k}_i^0 \right] \mathbf{u}_i \\ &= 2 \left[p \tilde{x}_i^{p-1} (E_0 - E_{\min}) \right]^2 [E_{\min} + \tilde{x}_i^p (E_0 - E_{\min})]^{-1} \\ &\quad \mathbf{u}_i^T \mathbf{k}_i^0 \mathbf{u}_i. \end{aligned}$$

Therefore, the Hessian of the structural compliance is given by:

$$\frac{\partial^2 c}{\partial \tilde{x}_i \partial \tilde{x}_j} = \begin{cases} 0, & i \neq j, \\ 2 \left[p \tilde{x}_i^{p-1} (E_0 - E_{\min}) \right]^2 [E_{\min} + \tilde{x}_i^p (E_0 - E_{\min})]^{-1} \tilde{\mathbf{u}}_i^T \mathbf{k}_i^0 \mathbf{u}_i, & i = j. \end{cases}$$

Appendix C: MATLAB Program top3d

```

1  % A 169 LINE 3D TOPOLOGY OPTIMIZATION CODE BY LIU AND TOVAR (JUL 2013)
2  function top3d(nelx , nely , nelz , volfrac , penal , rmin)
3  % USER-DEFINED LOOP PARAMETERS
4  maxloop = 200;      % Maximum number of iterations
5  tolx = 0.01;       % Termination criterion
6  displayflag = 1;   % Display structure flag
7  % USER-DEFINED MATERIAL PROPERTIES
8  E0 = 1;            % Young's modulus of solid material
9  Emin = 1e-9;       % Young's modulus of void-like material
10 nu = 0.3;          % Poisson's ratio
11 % USER-DEFINED LOAD DOFs
12 il = nelx; j1 = 0; k1 = 0:nelz;                                % Coordinates
13 loadnid = k1*(nelx+1)*(nely+1)+il*(nely+1)+(nely+1-j1); % Node IDs
14 loaddof = 3*loadnid(:) - 1;                                    % DOFs
15 % USER-DEFINED SUPPORT FIXED DOFs
16 [jf , kf] = meshgrid(1:nely+1,1:nelz+1);                      % Coordinates
17 fixednid = (kf-1)*(nely+1)*(nelx+1)+jf;                       % Node IDs
18 fixeddof = [3*fixednid(:); 3*fixednid(:)-1; 3*fixednid(:)-2]; % DOFs
19 % PREPARE FINITE ELEMENT ANALYSIS
20 nele = nelx*nely*nelz;
21 ndof = 3*(nelx+1)*(nely+1)*(nelz+1);
22 F = sparse(loaddof,1,-1,ndof,1);
23 U = zeros(ndof,1);
24 freedofs = setdiff(1:ndof, fixeddof);
25 KE = lk_H8(nu);
26 nodegrd = reshape(1:(nely+1)*(nelx+1), nely+1, nelx+1);
27 nodeids = reshape(nodegrd(1:end-1,1:end-1), nely*nelx, 1);
28 nodeidz = 0:(nely+1)*(nelx+1):(nelz-1)*(nely+1)*(nelx+1);
29 nodeids = repmat(nodeids, size(nodeidz))+repmat(nodeidz, size(nodeids));
30 edofVec = 3*nodeids(:)+1;
31 edofMat = repmat(edofVec,1,24)+ ...
32     repmat([0 1 2 3*nely + [3 4 5 0 1 2] -3 -2 -1 ...
33     3*(nely+1)*(nelx+1)+[0 1 2 3*nely + [3 4 5 0 1 2] -3 -2 -1]],nele,1);
34 iK = kron(edofMat, ones(24,1))';
35 jK = kron(edofMat, ones(1,24))';
36 % PREPARE FILTER
37 iH = ones(nele*(2*(ceil(rmin)-1)+1)^2,1);
38 jH = ones(size(iH));
39 sH = zeros(size(iH));
40 k = 0;
41 for k1 = 1:nelz
42     for i1 = 1:nelx
43         for j1 = 1:nely
44             e1 = (k1-1)*nelx*nely + (i1-1)*nely+j1;
45             for k2 = max(k1-(ceil(rmin)-1),1):min(k1+(ceil(rmin)-1), nelz)
46                 for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1), nelx)
47                     for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-1), nely)
48                         e2 = (k2-1)*nelx*nely + (i2-1)*nely+j2;
49                         k = k+1;
50                         iH(k) = e1;
51                         jH(k) = e2;
52                         sH(k) = max(0, rmin-sqrt((i1-i2)^2+(j1-j2)^2+(k1-k2)^2));
53                     end
54                 end
55             end
56         end
57     end
58 end
59 H = sparse(iH, jH, sH);
60 Hs = sum(H,2);
61 % INITIALIZE ITERATION
62 x = repmat(volfrac, [nely, nelx, nelz]);
63 xPhys = x;
64 loop = 0;
65 change = 1;
66 % START ITERATION
67 while change > tolx && loop < maxloop

```

```

68     loop = loop+1;
69     % FE-ANALYSIS
70     sK = KE(:)*(Emin+xPhys(:)')^penal*(E0-Emin);
71     K = sparse(iK(:),jK(:),sK(:)); K = (K+K')/2;
72     U(freedofs,:) = K(freedofs,freedofs)\F(freedofs,:);
73     % OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
74     ce = reshape(sum((U(edofMat)*KE).*U(edofMat)),2,[nely,nelx,nelz]);
75     c = sum(sum(sum((Emin+xPhys.^penal*(E0-Emin)).*ce)));
76     dc = -penal*(E0-Emin)*xPhys.^(penal-1).*ce;
77     dv = ones(nely,nelx,nelz);
78     % FILTERING AND MODIFICATION OF SENSITIVITIES
79     dc(:) = H*(dc(:)./Hs);
80     dv(:) = H*(dv(:)./Hs);
81     % OPTIMALITY CRITERIA UPDATE
82     l1 = 0; l2 = 1e9; move = 0.2;
83     while (l2-l1)/(l1+l2) > 1e-3
84         lmid = 0.5*(l2+l1);
85         xnew = max(0,max(x-move,min(1,min(x+move,x.*sqrt(-dc./dv/lmid))));
86         xPhys(:) = (H*xnew(:))./Hs;
87         if sum(xPhys(:)) > volfrac*nele, l1 = lmid; else l2 = lmid; end
88     end
89     change = max(abs(xnew(:)-x(:)));
90     x = xnew;
91     % PRINT RESULTS
92     fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f ch.:%7.3f\n',loop,c,mean(xPhys(:)),change);
93     % PLOT DENSITIES
94     if displayflag, clf; display_3D(xPhys); end
95 end
96 clf; display_3D(xPhys);
97 end
98 % ===== AUXILIARY FUNCTIONS =====
99 % GENERATE ELEMENT STIFFNESS MATRIX
100 function [KE] = lk_H8(nu)
101 A = [32 6 -8 6 -6 4 3 -6 -10 3 -3 -4 -8;
102     -48 0 0 -24 24 0 0 0 12 -12 0 12 12 12];
103 k = 1/72*A'*[1; nu];
104 % GENERATE SIX SUB-MATRICES AND THEN GET KE MATRIX
105 K1 = [k(1) k(2) k(2) k(3) k(5) k(5);
106     k(2) k(1) k(2) k(4) k(6) k(7);
107     k(2) k(2) k(1) k(4) k(7) k(6);
108     k(3) k(4) k(4) k(1) k(8) k(8);
109     k(5) k(6) k(7) k(8) k(1) k(2);
110     k(5) k(7) k(6) k(8) k(2) k(1)];
111 K2 = [k(9) k(8) k(12) k(6) k(4) k(7);
112     k(8) k(9) k(12) k(5) k(3) k(5);
113     k(10) k(10) k(13) k(7) k(4) k(6);
114     k(6) k(5) k(11) k(9) k(2) k(10);
115     k(4) k(3) k(5) k(2) k(9) k(12);
116     k(11) k(4) k(6) k(12) k(10) k(13)];
117 K3 = [k(6) k(7) k(4) k(9) k(12) k(8);
118     k(7) k(6) k(4) k(10) k(13) k(10);
119     k(5) k(5) k(3) k(8) k(12) k(9);
120     k(9) k(10) k(2) k(6) k(11) k(5);
121     k(12) k(13) k(10) k(11) k(6) k(4);
122     k(2) k(12) k(9) k(4) k(5) k(3)];
123 K4 = [k(14) k(11) k(11) k(13) k(10) k(10);
124     k(11) k(14) k(11) k(12) k(9) k(8);
125     k(11) k(11) k(14) k(12) k(8) k(9);
126     k(13) k(12) k(12) k(14) k(7) k(7);
127     k(10) k(9) k(8) k(7) k(14) k(11);
128     k(10) k(8) k(9) k(7) k(11) k(14)];
129 K5 = [k(1) k(2) k(8) k(3) k(5) k(4);
130     k(2) k(1) k(8) k(4) k(6) k(11);
131     k(8) k(8) k(1) k(5) k(11) k(6);
132     k(3) k(4) k(5) k(1) k(8) k(2);
133     k(5) k(6) k(11) k(8) k(1) k(8);
134     k(4) k(11) k(6) k(2) k(8) k(1)];
135 K6 = [k(14) k(11) k(7) k(13) k(10) k(12);
136     k(11) k(14) k(7) k(12) k(9) k(2);
137     k(7) k(7) k(14) k(10) k(2) k(9);
138     k(13) k(12) k(10) k(14) k(7) k(11)];

```

```

139     k(10) k(9)  k(2)  k(7)  k(14) k(7);
140     k(12) k(2)  k(9)  k(11) k(7)  k(14)];
141     KE = 1/((nu+1)*(1-2*nu))*...
142     [ K1 K2 K3 K4;
143     K2' K5 K6 K3';
144     K3' K6 K5' K2';
145     K4 K3 K2 K1'];
146     end
147     % DISPLAY 3D TOPOLOGY (ISO-VIEW)
148     function display_3D(rho)
149     [nely, nelx, nelz] = size(rho);
150     hx = 1; hy = 1; hz = 1; % User-defined unit element size
151     face = [1 2 3 4; 2 6 7 3; 4 3 7 8; 1 5 8 4; 1 2 6 5; 5 6 7 8];
152     set(gcf, 'Name', 'ISO display', 'NumberTitle', 'off');
153     for k = 1:nelz
154         z = (k-1)*hz;
155         for i = 1:nelx
156             x = (i-1)*hx;
157             for j = 1:nely
158                 y = nely*hy - (j-1)*hy;
159                 if (rho(j,i,k) > 0.5) % User-defined display density threshold
160                     vert = [x y z; x y-hx z; x+hx y-hx z; x+hx y z; x y z+hx; x y-hx z+hx; x+hx y-hx z
161                             +hx; x+hx y z+hx];
162                     vert(:, [2 3]) = vert(:, [3 2]); vert(:, 2, :) = -vert(:, 2, :);
163                     patch('Faces', face, 'Vertices', vert, 'FaceColor', [0.2+0.8*(1-rho(j,i,k))
164                                 , 0.2+0.8*(1-rho(j,i,k)), 0.2+0.8*(1-rho(j,i,k))]);
165                     hold on;
166                 end
167             end
168         end
169     end
170     end
171     axis equal; axis tight; axis off; box on; view([30,30]); pause(1e-6);
172     end
173     % =====
174     % == This code was written by K Liu and A Tovar, Dept. of Mechanical ==
175     % == Engineering, Indiana University-Purdue University Indianapolis, ==
176     % == Indiana, United States of America ==
177     % ==
178     % == Please send your suggestions and comments to: kailiu@iupui.edu ==
179     % ==
180     % ==
181     % == The code as well as step-by-step tutorials can be found from the ==
182     % == website: http://top3dapp.com ==
183     % ==
184     % ==
185     % == Disclaimer: ==
186     % == The authors reserves all rights for the program. ==
187     % == The code may be distributed and used for educational purposes. ==
188     % == The authors do not guarantee that the code is free from errors, and ==
189     % == they shall not be liable in any event caused by the use of the code.==

```

References

- Aage N, Nobel-Jørgensen M, Andreasen CS, Sigmund O (2013) Interactive topology optimization on hand-held devices. *Struct Multidiscip Optim* 47(1):1–6
- Allaire G (2001) Shape optimization by the homogenization method. Springer, New York
- Allaire G, Kohn R (1993) Optimal design for minimum weight and compliance in plane-stress using extremal microstructures. *Eur J Mech* 12(6):839–878
- Allaire G, Pantz O (2006) Structural optimization with freefem++. *Struct Multidiscip Optim* 32(3):173–181
- Allaire G, Belhachmi Z, Jouve F (1996) The homogenization method for topology and shape optimization. single and multiple loads case. *Eur J Finite Elem* 5:649–672
- Allaire G, Jouve F, Toader AM (2004) Structural optimization using sensitivity analysis and a level-set method. *J Comput Phys* 194(1):363–393
- Andreassen E, Clausen A, Schevenels M, Lazarov BS, Sigmund O (2011) Efficient topology optimization in matlab using 88 lines of code. *Struct Multidiscip Optim* 43(1):1–16
- Augarde C, Ramage A, Staudacher J (2006) An element-based displacement preconditioner for linear elasticity problems. *Comput Struct* 84(31–32):2306–2315
- Bendsøe MP (1989) Optimal shape design as a material distribution problem. *Struct Multidiscip Optim* 1(4):193–202
- Bendsøe MP (1995) Optimization of structural topology shape and material. Springer, New York
- Bendsøe MP, Kikuchi N (1988) Generating optimal topologies in structural design using a homogenization method. *Comput Methods Appl Mech Eng* 71(2):197–224

- Bendsøe MP, Sigmund O (2003) Topology optimization: theory, method and applications. Springer
- Bruns TE, Tortorelli DA (2001) Topology optimization of non-linear elastic structures and compliant mechanisms. *Comput Methods Appl Mech Eng* 190(26–27):3443–3459
- Challis VJ (2010) A discrete level-set topology optimization code written in matlab. *Struct Multidiscip Optim* 41(3):453–464
- Christensen PW, Klarbring A (2009) An introduction to structural optimization. Springer
- Duysinx P (1997) Layout optimization: A mathematical programming approach, DCAMM report. Technical report, Danish Center of Applied Mathematics and Mechanics, Technical University of Denmark, DK-2800 Lyngby
- Fléury C (1989) CONLIN: an efficient dual optimizer based on convex approximation concepts. *Struct Optim* 1(2):81–89
- Groenwold AA, Etman LFP (2009) A simple heuristic for gray-scale suppression in optimality criterion-based topology optimization. *Struct Multidiscip Optim* 39(2):217–225
- Groenwold AA, Etman LFP (2010) A quadratic approximation for structural topology optimization. *Int J Numer Methods Eng* 82(4):505–524
- Guest JK, Prevost JH, Belytschko T (2004) Achieving minimum length scale in topology optimization using nodal design variables and projection functions. *Int J Numer Methods Eng* 61:238–254
- Guest JK, Asadpoure A, Ha SH (2011) Eliminating beta-continuation from heaviside projection and density filter algorithms. *Struct Multidiscip Optim* 44:443–453
- Haber R, Jog C (1996) A new approach to variable-topology shape design using a constraint on perimeter. *Struct Optim* 11(1):1–12
- Hassani B, Hinton E (1998) Homogenization and structural topology optimization: theory, practice and software. Springer
- Hestenes MR, Stiefel E (1952) Methods of conjugate gradients for solving linear systems. *J Res Natl Bur Stand* 49(6)
- Hunter W (2009) Predominantly solid-void three-dimensional topology optimisation using open source software. Master's thesis, University of Stellenbosch
- Jog C (2002) Topology design of structures using a dual algorithm and a constraint on the perimeter. *Int J Numer Methods Eng* 54(7):1007–1019
- Kohn R, Strang G (1986a) Optimal design and relaxation of variational problems (part I). *Commun Pure Applied Math* 39(1):113–137
- Kohn R, Strang G (1986b) Optimal design and relaxation of variational problems (part II). *Commun Pure Appl Math* 39(2):139–182
- Kohn R, Strang G (1986c) Optimal design and relaxation of variational problems (part III). *Commun Pure Appl Math* 39(3):353–377
- Liu Z, Korvink JG, Huang I (2005) Structure topology optimization: fully coupled level set method via FEMLAB. *Struct Multidiscip Optim* 6(29):407–417
- Mlejnek H (1992) Some aspects of the genesis of structures. *Struct Optim* 5(1–2):64–69
- Nocedal J, Wright S (2006) Numerical optimization, 2nd edn. Springer
- Schmit LA (1960) Structural design by systematic synthesis. In: 2nd ASCE conference of electrical compounds. Pittsburgh, pp 139–149
- Sigmund O (1994) Design of material structures using topology optimization, PhD thesis, Technical University of Denmark
- Sigmund O (1997) On the design of compliant mechanisms using topology optimization. *Mech Struct Mach* 25(4):495–526
- Sigmund O (2001) A 99 line topology optimization code written in matlab. *Struct Multidiscip Optim* 21(2):120–127
- Sigmund O (2007) Morphology-based black and white filters for topology optimization. *Struct Multidiscip Optim* 33:401–424
- Sigmund O, Peterson J (1998) Numerical instabilities in topology optimization: a survey on procedures dealing with checkerboards, mesh-dependencies and local minima. *Struct Optim* 16:68–75
- Sokół T (2011) A 99 line code for discretized michell truss optimization written in mathematica. *Struct Multidiscip Optim* 43(2):181–190
- Suresh K (2010) A 199-line matlab code for pareto-optimal tracing in topology optimization. *Struct Multidiscip Optim* 42:665–679
- Svanberg K (1987) The method of moving asymptotes—a new method for structural optimization. *Int J Numer Methods Eng* 24:359–373
- Talischi C, Paulino GH, Pereira A, Menezes IFM (2012a) Polymesher: a general-purpose mesh generator for polygonal elements written in matlab. *Struct Multidiscip Optim* 45:309–328
- Talischi C, Paulino GH, Pereira A, Menezes IFM (2012b) Polytop: a matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes. *Struct Multidiscip Optim* 45:329–357
- Wang MY, Chen S, Xia Q (2004) Structural topology optimization with the level set method. <http://www2.acae.cuhk.edu.hk/cmdl/download.htm>
- Wilson RB (1963) A simplicial method for convex programming, PhD thesis, Harvard University
- Zhou M, Rozvany G (1991) The COC algorithm, part II: topological, geometrical and generalized shape optimization. *Comp Meth Appl Mech Eng* 89:309–336
- Zhou S, Wang MY (2005) 3d structural topology optimization with the simp method. <http://www2.acae.cuhk.edu.hk/cmdl/download.htm>