Noname manuscript No.
(will be inserted by the editor)

An efficient 3D topology optimization code written in Matlab

Kai Liu · Andrés Tovar

Received: date / Accepted: date

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. The code is intended for students and newcomers in the topology optimization. The complete code is provided in the Appendix and it can be downloaded from `http://engr.iupui.edu/~tovara/top3d`.

Keywords Topology optimization · MATLAB · Compliance · Compliant mechanism · Heat conduction

1 Introduction

Topology optimization is a computational material distribution method for synthesizing structures without any preconceived 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) 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

K. Liu, A. Tovar
Department of Mechanical Engineering
Indiana University-Purdue University Indianapolis
Indianapolis, IN 46202
E-mail: kailiu@iupui.edu

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 the state of the art in topology optimization. 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 method and its 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 B and can also be downloaded for free from the website: `http://engr.iupui.edu/~tovara/top3d`.

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 et al 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-possedness 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-possedness, 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

$$\mathrm{E}_i = \mathrm{E}_i(x_i) = x_i^p \mathrm{E}_0, \quad x_i \in]0,1], \tag{1}$$

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

$$\mathrm{E}_i = \mathrm{E}_i(x_i) = \mathrm{E}_{\min} + x_i^p(\mathrm{E}_0 - \mathrm{E}_{\min}), \quad x_i \in [0,1], \tag{2}$$

where $\mathrm{E}_{\min}$ 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 Eq. (2), offers a number of advantages over the classical SIMP formulation, as shown in Eq. (1), including the indecency between the minimum value of the material's elastic modulus and the penalization power (Sigmund 2007).

The density-based approach is also used to address other problems. However, it is likely to encounter numerical instabilities such as mesh-dependency, checkerboard patterns, and local minima (Christensen and Klarbring 2009). 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

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

where the operator $\mathrm{dist}(i,j)$ 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

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

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

$$\mathrm{E}_i(\tilde{x}_i) = \mathrm{E}_{\min} + \tilde{x}_i^p(\mathrm{E}_0 - \mathrm{E}_{\min}), \quad \tilde{x}_i \in [0,1]. \tag{6}$$

The regularized SIMP interpolation formula defined by Eq. (6) is used in the rest of this work.

The numerical implementation of density filter is described as follows:

In the program (Appendix B), the weight factor as in Eq. (5) is represented by matrix `H` and remains constant through the optimization procedure. The matrix `H` is obtained by six nested for loops (lines 37-59) and stores the distance between all element needed to define the element neighborhood N_i defined by Eq. (4).

Then the density filter defined by Eq. (3) can be performed as the following:

```
60   Hs = sum(H,2);
86   xPhys(:) = (H*x(:))./Hs;
```

The density filer can be applied efficiently by using the code shown above.

3 Finite element analysis

3.1 Equilibrium equation

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

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

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

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

where ξ_e $(e = 1, \ldots, 3)$ 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

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

where

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

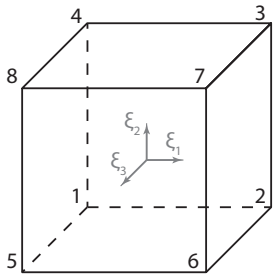


Fig. 1: The 8-node hexahedron and the natural coordinates ξ_1, ξ_2, ξ_3.

Table 1: The 8-node hexahedron 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	$+1$	$+1$

For an eight-node hexahedron element, the strain-displacement matrix $\mathbf{B}$ is defined by

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

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

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

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

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

where $\mathbf{k}_m$ $(m = 1, \ldots, 6)$ are 6×6 symmetric matrices. These matrices are described in 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 $\mathbf{k}_i$,

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

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

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

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

$$\mathbf{K}(\tilde{\mathbf{x}}) = \sum_{i=1}^{n}\left[\mathrm{E}_{\min} + \tilde{x}_i^p(\mathrm{E}_0 - \mathrm{E}_{\min})\right]\mathbf{K}_i^0. \tag{15}$$

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.

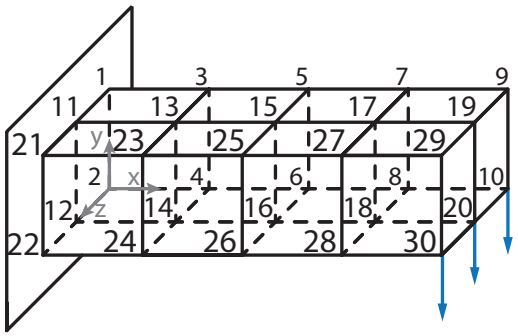


Fig. 2: Global node IDs in a prismatic structure composed of 8 elements.

Within each element, the eight nodes $N_1, \ldots, N_8$ 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 ($\texttt{NID}_i$) system in Fig. 2. Given the size of the volume ($\texttt{nelx} \times \texttt{nely} \times \texttt{nelz}$) 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

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

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.

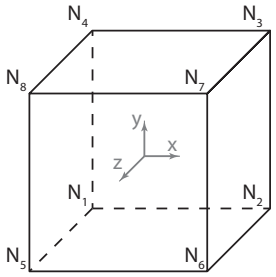


Fig. 3: Local node numbers within a cubic element.

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 `nele` $\times$ 24 containing the node IDs for each element. For the volume in

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)	$\texttt{NID}_1^{\dagger}$	$3*\texttt{NID}_1-2$	$3*\texttt{NID}_1-1$	$3*\texttt{NID}_1$
N_2	(x_1+1, y_1, z_1)	$\texttt{NID}_2=\texttt{NID}_1+(\texttt{nely}+1)$	$3*\texttt{NID}_2-2$	$3*\texttt{NID}_2-1$	$3*\texttt{NID}_2$
N_3	(x_1+1, y_1+1, z_1)	$\texttt{NID}_3=\texttt{NID}_1+\texttt{nely}$	$3*\texttt{NID}_3-2$	$3*\texttt{NID}_3-1$	$3*\texttt{NID}_3$
N_4	(x_1, y_1+1, z_1)	$\texttt{NID}_4=\texttt{NID}_1-1$	$3*\texttt{NID}_4-2$	$3*\texttt{NID}_4-1$	$3*\texttt{NID}_4$
N_5	(x_1, y_1, z_1+1)	$\texttt{NID}_5=\texttt{NID}_1+\texttt{NID}_z^{\ddagger}$	$3*\texttt{NID}_5-2$	$3*\texttt{NID}_5-1$	$3*\texttt{NID}_5$
N_6	(x_1+1, y_1, z_1+1)	$\texttt{NID}_6=\texttt{NID}_2+\texttt{NID}_z$	$3*\texttt{NID}_6-2$	$3*\texttt{NID}_6-1$	$3*\texttt{NID}_6$
N_7	(x_1+1, y_1+1, z_1+1)	$\texttt{NID}_7=\texttt{NID}_3+\texttt{NID}_z$	$3*\texttt{NID}_7-2$	$3*\texttt{NID}_7-1$	$3*\texttt{NID}_7$
N_8	(x_1, y_1+1, z_1+1)	$\texttt{NID}_8=\texttt{NID}_4+\texttt{NID}_z$	$3*\texttt{NID}_8-2$	$3*\texttt{NID}_8-1$	$3*\texttt{NID}_8$

† $\texttt{NID}_1 = z_1*(\texttt{nelx}+1)*(\texttt{nely}+1)+x_1*(\texttt{nely}+1)+(\texttt{nely}+1-y_1)$.
‡ $\texttt{NID}_z = (\texttt{nelx}+1)*(\texttt{nely}+1)$.

Fig. 2, `nelx` $=4$, `nely` $=1$, and `nelz` $=2$, which results in

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

The element connectivity matrix `edofMat` is used to assemble the global stiffness matrix `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 in Appendix B). Matrices `iK` (size 24 `nele` $\times 24$) and `jK` (size `nele` $\times 24^2$), reshaped as column vectors, contain the rows and columns identifying the $24\times 24\times$ `nele` DOFs in the structure. The three-dimensional array `(xPhys)` (size `nely` $\times$ `nelx` $\times$ `nelz`) corresponds to the physical densities. The matrix `sK` (size $24^2\times$ `nele`) contains all element stiffness matrices. The assembly procedure of the (sparse) symmetric) global stiffness matrix `K` (line 71) avoids the use of nested `for` loops.

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

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

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, Appendix B). 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 $\mathbf{x}$ 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

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

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

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

where the physical densities $\tilde{\mathbf{x}} = \tilde{\mathbf{x}}(\mathbf{x})$ are defined by Eq. (3), n is the number of elements used to discretize the design domain, $\mathbf{v} = [v_1, \ldots, v_n]^{\mathrm{T}}$ 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 Eq. (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 Eq. (21) and

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

The derivative of Eq. (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 Eq. (15),

$$\begin{aligned} \frac{\partial \mathbf{K}(\tilde{\mathbf{x}})}{\partial \tilde{x}_i} &= \frac{\partial}{\partial \tilde{x}_i} \sum_{i=1}^{n} \left[\mathrm{E}_{\min} + \tilde{x}_i^p \left(\mathrm{E}_0 - \mathrm{E}_{\min} \right) \right] \mathbf{K}_i^0 \\ &= p \tilde{x}_i^{p-1} \left(\mathrm{E}_0 - \mathrm{E}_{\min} \right) \mathbf{K}_i^0. \end{aligned} \tag{26}$$

Using Eqs. (25) and (26), Eq. (23) results in

$$\frac{\partial c(\tilde{\mathbf{x}})}{\partial \tilde{x}_i} = -\mathbf{U}(\tilde{\mathbf{x}})^{\mathrm{T}} \left[p \tilde{x}_i^{p-1} (\mathrm{E}_0 - \mathrm{E}_{\min}) \mathbf{K}_i^0 \right] \mathbf{U}(\tilde{\mathbf{x}}). \tag{27}$$

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

$$\frac{\partial c(\tilde{\mathbf{x}})}{\partial \tilde{x}_i} = -\mathbf{u}_i(\tilde{\mathbf{x}})^{\mathrm{T}} \left[p \tilde{x}_i^{p-1} (\mathrm{E}_0 - \mathrm{E}_{\min}) \mathbf{k}_i^0 \right] \mathbf{u}_i(\tilde{\mathbf{x}}). \tag{28}$$

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

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

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

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

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

$$\frac{\partial c(\tilde{\mathbf{x}})}{\partial \tilde{x}_i} = \mathbf{u}_{di}(\tilde{\mathbf{x}})^{\mathrm{T}} \left[p\tilde{x}_i^{p-1}(\mathrm{E}_0 - \mathrm{E}_{\min})\mathbf{k}_i^0 \right] \mathbf{u}_i(\tilde{\mathbf{x}}). \tag{32}$$

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

The numerical implementation of the objective function Eq. (31) and sensitivity Eq. (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 `Ud` (Line 74a) is the dummy load displacement field and vector `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

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

where $\mathbf{U}(\tilde{\mathbf{x}})$ now donates the finite element global nodal temperature vector, $\mathbf{F}$ donates the global thermal load vector, and $\mathbf{K}(\tilde{\mathbf{x}})$ 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

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

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 Eq. (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 Eq. (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 Eq. (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 Eq. (21) and

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

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 `k0` and `kmin` are the limits of the material's thermal conductivity. The chain rule is applied same as before.

5 Optimization algorithm

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 sequential quadratic programming (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

$$\frac{\partial c(\tilde{\mathbf{x}})}{\partial x_e} + \lambda \frac{\partial v(\tilde{\mathbf{x}})}{\partial x_e} = 0, \tag{36}$$

is satisfied for $k = 1, \ldots, n$, 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

$$B_e = -\frac{\partial c(\tilde{\mathbf{x}})}{\partial x_e} \left(\lambda \frac{\partial v(\tilde{\mathbf{x}})}{\partial x_e} \right)^{-1}. \tag{37}$$

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

$$x_e^{\text{new}} = \begin{cases} \max(0,\ x_e - m), & \text{if } x_e B_e^{\eta} \leqslant \max(0,\ x_e - m), \\ \min(1,\ x_e + m), & \text{if } x_e B_e^{\eta} \geqslant \min(1,\ x_e - m), \\ x_e B_e^{\eta}, & \text{otherwise,} \end{cases} \tag{38}$$

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 Eq. (38) is the value of the Lagrange multiplier λ, which satisfies that

$$v(\tilde{\mathbf{x}}(\mathbf{x}^{\text{new}}(\lambda))) = 0. \tag{39}$$

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

$$||\mathbf{x}^{\text{new}} - \mathbf{x}||_{\infty} \leq \epsilon, \tag{40}$$

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 equal to the design variables since the design variables over the design domain are uniform. The OC update incorporating the bisection method is

```
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
```

where `lmid` is the value of the Lagrange multiplier. The complete optimization algorithm is

```
64 loop = 0;
65 change = 1;
66 % START ITERATION
67 while change > tolx && loop < maxloop
68 loop = loop + 1;
69 % FE-ANALYSIS
70-72 (...)
73 % OBJECTIVE FUNCTION AND SENSITIVITY
     ANALYSIS
74-77 (...)
78 % FILTERING AND MODIFICATION OF
     SENSITIVITIES
79-80 (...)
81 % OPTIMALITY CRITERIA UPDATE
82-88 (...)
89 change = max(abs(xnew(:)-x(:)));
90 x = xnew;
95 end
```

where the loop counter (`loop`) is initially zero and the change in design variables (`change`) is intially one. The values for the maximum number of iterations (`maxloop`) and tolerance (`tolx`) and are set by the user in lines 4 and 5, respectively. Visualization of results is in lines 91-94 and 146-169.

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 parameters, loads, and supports. The following examples demonstrate the application of the 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)
```

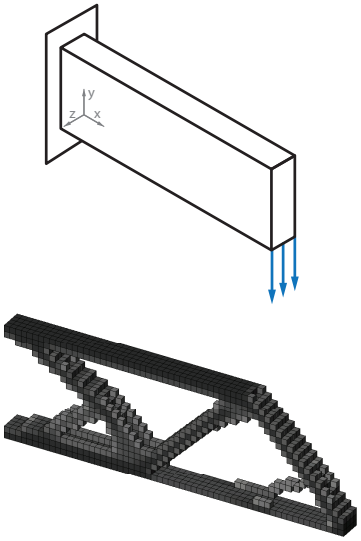


Fig. 4: Topology optimization of 3D cantilever beam. Top: Initial design domain, bottom: topology optimized beam.

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; jl = 0 ; kl = nelz/2;
13  loadnid = kl*(nelx+1)*(nely+1)+il*(nely
    +1)+(nely+1-jl);
14  loaddof = 3*loadnid(:) - 1;
```

Secondly, defining the corresponding boundary conditions

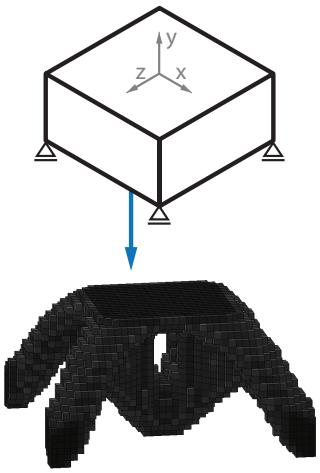


Fig. 5: Topology optimization of 3D wheel. Top: Initial design domain, bottom: topology optimized result.

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

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

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

The objective function is now the sum of different load cases

$$c(\tilde{\mathbf{x}}) = \sum_{l=1}^{M} c_l(\tilde{\mathbf{x}}) = \sum_{l=1}^{M} \mathbf{F}_l^{\mathrm{T}} \mathbf{U}_l(\tilde{\mathbf{x}}) \tag{41}$$

where M is the number of 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);
```

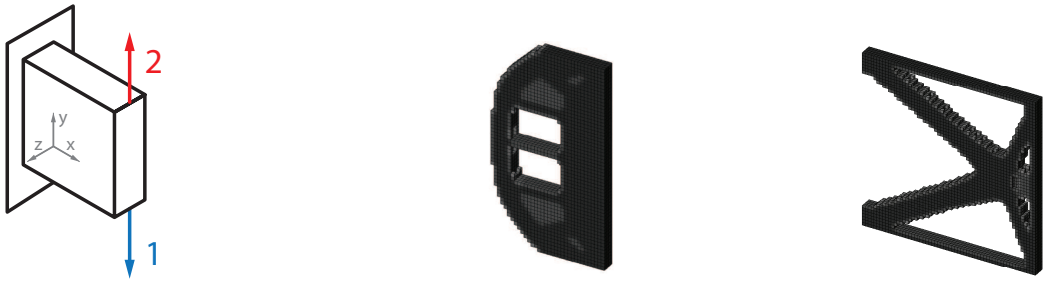


Fig. 6: Topology optimization of cantilever beam with multiple load cases. Left: Initial design domain, middle: topology optimized beam with one load case, and right: topology optimized beam with two load cases.

```
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`×`nelx`×`nelz` matrix with ones at elements desired to be solid, and a `nely` × `nelx` × `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)
```

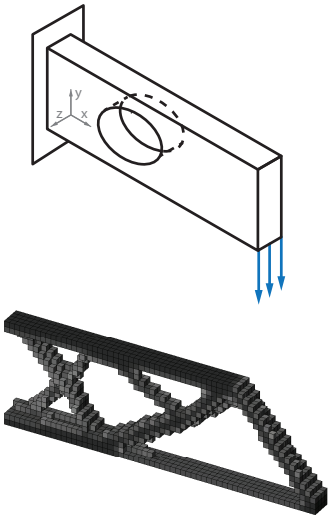


Fig. 7: Topology optimization of cantilever beam with passive design domain. Top: Initial design domain, bottom: topology optimized beam.

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 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

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

where γ $(= 10^{-3})$ 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

$$x_i^{\text{new}} = \begin{cases} \max(0,\ x_i - m), & \text{if } x_i B_i^{\eta} \leqslant \max(0,\ x_i - m) \\ \min(1,\ x_i + m), & \text{if } x_i B_i^{\eta} \geqslant \min(1,\ x_i - m) \\ (x_i B_i^{\eta})^q, & \text{otherwise} \end{cases} \tag{42}$$

The standard OC updating method is a special case of Eq. (42) 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
```

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 = 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 Messerchimitt-Bölkow-Blohm (MBB) beam and L-shape problems (Table. 4) are solved by using iterative solver 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

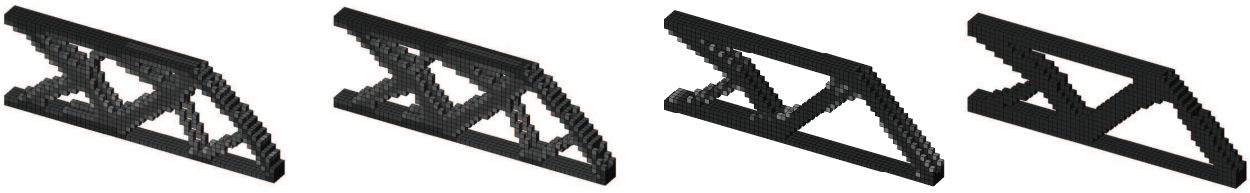


Fig. 8: Topology optimized design used a mesh with $30 \times 10 \times 2$ elements. Left: optimized design using density filter, middle ledt: optimized design using density filter, middle right: optimized design using density filter and gray scale filter, and right: optimized design using sensitivity filter and gray scale filter.

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

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

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

$$p^k = \begin{cases} 1 & k \leqslant 20, \\ \min\{p^{\max}, 1.02p^{k-1}\} & k > 20, \end{cases} \tag{43}$$

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 (based on the code in Appendix B), 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.

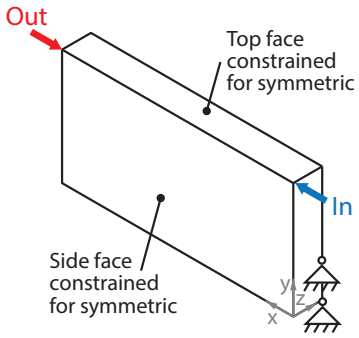


Fig. 9: Design domain of 3D force inverter problem.

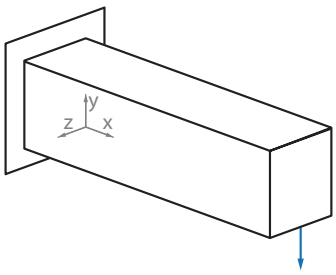
The new loading conditions as well as input and output points are defined as follows:

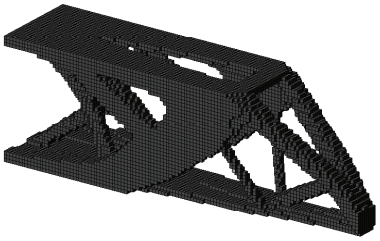
```
11  % USER-DEFINED LOAD DOFs
11a % [Input Dummy]
12  il = [0 nelx]; jl = [nely nely]; kl = [0
     0];
13  loadnid = kl*(nelx+1)*(nely+1)+il*(nely
    +1)+(nely+1-jl);
14  loaddof = 3*loadnid(:) - 2;
14a din = loaddof(1); dout = loaddof(2);
22  F = sparse(loaddof,[1 2],[1 -1],ndof,2);
```

and the boundary conditions are defined as below:

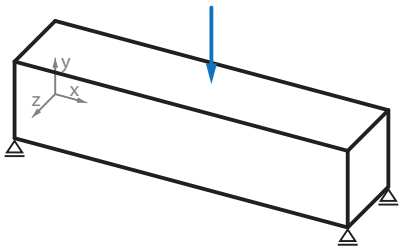
```
15  % USER-DEFINED SUPPORT FIXED DOFs
15a % Top face
15b [iif,kf] = meshgrid(0:nelx,0:nelz);
15c fixednid = kf*(nelx+1)*(nely+1)+iif*(
    nely+1)+1;
15d fixeddof_t = 3*fixednid(:)-1;
16  % Side face
16a [iif,jf] = meshgrid(0:nelx,0:nely);
16b fixednid = iif*(nely+1)+(nely+1-jf);
16c fixeddof_s = 3*fixednid(:);
17  % Two pins
```

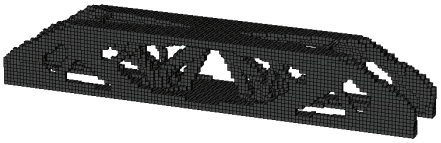
Table 4: Three-dimensional examples: Cantilever, MBB, and L-shape problem. Left: Initial design domains, right: topology optimized results..



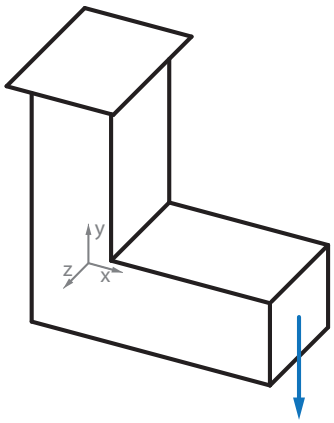


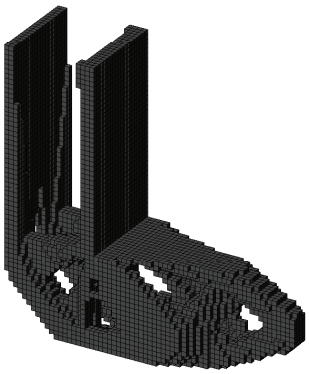
$12L \times 4L \times 3L$
mesh $120 \times 40 \times 30$
$\bar{v} = 0.15$, $p = 3$, $R = 1.5$





$6L \times L \times L$
mesh $120 \times 20 \times 20$
$\bar{v} = 0.2$, $p = 3$, $R = 1.5$





$3L \times L \times L$
mesh $60 \times 20 \times 20$ and $40 \times 40 \times 20$
$\bar{v} = 0.15$, $p = 3$, $R = 1.2$

```
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 Eq. (31) and sensitivity Eq. (32) are modified in lines 74-76.

```
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)
```

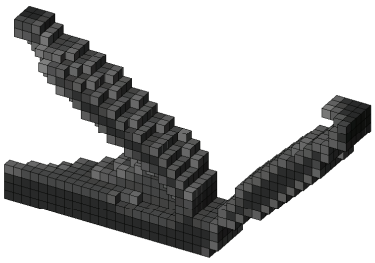


Fig. 10: Topology optimized force inverter.

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 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 condition problem, i.e., a rectangular plate with a heat sink

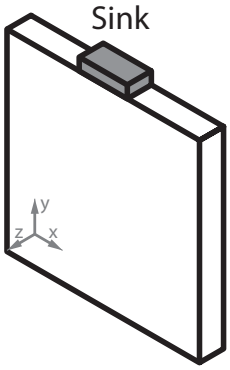


Fig. 11: Initial design domain of heat conduction problem.

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  il = nelx/2-nelx/20:nelx/2+nelx/20; jl =
     nely; kl = 0:nelz;
12  fixedxy = il*(nely+1)+(nely+1-jl);
13  fixednid = repmat(fixedxy',size(kl)) +
    ...
14      repmat(kl*(nelx+1)*(nely+1),size(
    fixedxy,2),1);
15  fixeddof = reshape(fixednid,[],1);
```

Also, since there is only one DOF per node in heat condition 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

```
 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 evulation 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)
```

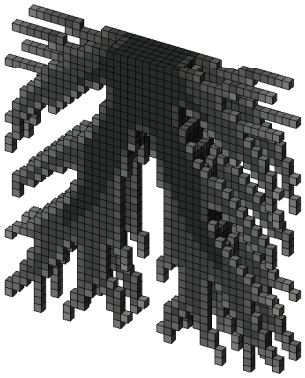


Fig. 12: Resulting topology of heat conduction problem.

7 Conclusions

This paper presents a 169-line MATLAB topology optimization code in three dimensions, called `top3d`. It is published for the same purpose as others: to give students and beginners in the field of topology optimization an easy-to-use introductory tool. The implementation of the code is very simple. The code is provided in Appendix B and can be downloaded from `http://engr.iupui.edu/~tovara/top3d`.

The difference between the original 177-line code (Zhou and Wang 2005) and this `top3d` code is the computational cost. As listed in Table 3, the top3d code is faster by a factor of 30.81. This speed up was mainly achieved by avoiding nested `for` loops, and providing a symbolic expression of the element stiffness matrix.

Additionally, the code can be extended to different filter techniques (e.g., sensitivity filter, gray scale filter) or an iterative solver in order to solve large-scale problems. Moreover, a continuation strategy can be applied to the code to ensure the global minima.

To solve different optimization problems, users must provide the corresponding force and support DOFs. In both the 177-line code and the 88-line code, finding the DOFs corresponding to the boundary conditions is a difficult process. In this presented `top3d` code the boundary conditions are defined in an evolutionary way. Thanks to mapping relationships between the node coordinates and the node numbers, users only need to know the node coordinates of the boundary conditions. Once the node coordinates are provided, its corresponding node numbers and DOFs are generated automatically. Hence, changing boundary conditions becomes an effortless process. Therefore, the code can be easily and quickly adopted for other boundary conditions, and multiple load cases. Moreover, without an heavy investment in programming, this program is capable of solving problems with required active/passive elements, or even solving different topology optimization problems.

The code was intended to be as compact as possible. If users of the code can find ways to further simplify or accelerate the code, the authors would welcome suggestion for modifications that can be incorporated into the public code. The authors can be reached at kailiu@iupui.edu, tovara@iupui.edu.

References

Aage N, Nobel-Jørgensen M, Andreasen CS, Sigmund O (2013) Interactive topology optimization on hand-held devices. Structural and Multidisciplinary Optimization 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. European Journal of Mechanics 12(6):839–878

Allaire G, Pantz O (2006) Structural optimization with freefem++. Structural and Multidisciplinary Optimization 32(3):173–181

Allaire G, Belhachmi Z, Jouve F (1996) The homogenization method for topology and shape optimization. single and multiple loads case. European Journal of Finite Elements 5:649–672

Allaire G, Jouve F, Toader AM (2004) Structural optimization using sensitivity analysis and a level-set method. Journal of Computational Physics 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. Structural and Multidisciplinary Optimization 43(1):1–16

Augarde C, Ramage A, Staudacher J (2006) An element-based displacement preconditioner for linear elasticity problems. Computers and Structures 84(31-32):2306–2315

Bendsøe MP (1989) Optimal shape design as a material distribution problem. Structural and Multidisciplinary Optimization 1(4):193–202

Bendsøe MP (1995) Optimization of structural topology, shape and material. New York: Springer

Bendsøe MP, Kikuchi N (1988) Generating optimal topologies in structural design using a homogenization method. Computer Methods in Applied Mechanics and Engineering 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. Computer Methods in Applied Mechanics and Engineering 190(26-27):3443–3459

Challis VJ (2010) A discrete level-set topology optimization code written in matlab. Structural and Multidisciplinary Optimization 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. Tech. rep., Danish Center of Applied Mathematics and Mechanics, Technical University of Denmark, DK-2800 Lyngby

Groenwold AA, Etman LFP (2009) A simple heuristic for gray-scale suppression in optimality criterion-based topology optimization. Structural and Multidisciplinary Optimization 39(2):217–225

Groenwold AA, Etman LFP (2010) A quadratic approximation for structural topology optimization. International Journal for Numerical Methods in Engineering 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. International Journal for Numerical Methods in Engineering 61:238–254

Guest JK, Asadpoure A, Ha SH (2011) Elimiating beta-continuation from heaviside projection and density filter algorithms. Structural and Multidisciplinary Optimization 44:443–453

Haber R, Jog C, Bendsøe M (1996) A new approach to variable-topology shape design using a constraint on perimeter. Structural Optimization 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. Journal of Research of the National Bureau of Standards 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. International Journal for Numerical Methods in Engineering 54(7):1007–1019

Kohn R, Strang G (1986a) Optimal design and relaxation of variational problems (part I). Communications of Pure and Applied Mathematics 39(1):113–137

Kohn R, Strang G (1986b) Optimal design and relaxation of variational problems (part II). Communications of Pure and Applied Mathematics 39(2):139–182

Kohn R, Strang G (1986c) Optimal design and relaxation of variational problems (part III). Communications of Pure and Applied Mathematics 39(3):353–377

Liu Z, Korvink JG, Huang I (2005) Structure topology optimization: fully coupled level set method via FEMLAB. Structural and Multidisciplinary Optimization 6(29):407–417

Mlejnek H (1992) Some aspects of the genesis of structures. Structural Optimization 5(1-2):64–69

Schmit LA (1960) Structural design by systematic synthesis. In: 2nd ASCE Conf. Electrical Compounds, Pittsburgh, PA, 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. Mechanics of Structures and Machines 25(4):495–526

Sigmund O (2001) A 99 line topology optimization code written in matlab. Structural and Multidisciplinary Optimization 21(2):120–127

Sigmund O (2007) Morphology-based black and white filters for topology optimization. Structural and Multidisciplinary Optimization 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. Structural and Multidisciplinary Optimization 43(2):181–190

Suresh K (2010) A 199-line matlab code for pareto-optimal tracing in topology optimization. Structural and Multidisciplinary Optimization 42:665–679

Talischi C, Paulino GH, Pereira A, Menezes IFM (2012a) Polymesher: a general-purpose mesh generator for polygonal elements written in matlab. Structural and Multidisciplinary Optimization 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. Structural and Multidisciplinary Optimization 45:329–357

Wang MY, Chen S, Xia Q (2004) Structural topology optimization with the level set method. URL `http://www2.acae.cuhk.edu.hk/~cmdl/download.htm`

Zhou M, Rozvany G (1991) The COC algorithm, part II: Topological, geometrical and generalized shape optimization. Comp Meth Appl Mech Engrg 89:309–336

Zhou S, Wang MY (2005) 3d structural topology optimization with the simp method. URL `http://www2.acae.cuhk.edu.hk/~cmdl/download.htm`

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 hexahedron element, the strain-displacement matrix $\mathbf{B}$ is defined by

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

for $e = 1, \ldots, 3$ and $q = 1, \ldots, 8$. 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 Eq. (11), the 64×64 element stiffness matrix $\mathbf{k}_i^0$ for an eight-node hexahedron element can be expressed as

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

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: MATLAB Program `top3d`

```
% A 169 LINE 3D TOPOLOGY OPITMIZATION CODE BY LIU AND TOVAR (JUL 2013)
function top3d(nelx,nely,nelz,volfrac,penal,rmin)
% USER-DEFINED LOOP PARAMETERS
maxloop = 200;    % Maximum number of iterations
tolx = 0.01;      % Termination criterion
displayflag = 1;  % Display structure flag
% USER-DEFINED MATERIAL PROPERTIES
E0 = 1;           % Young's modulus of solid material
Emin = 1e-9;      % Young's modulus of void-like material
nu = 0.3;         % Poisson's ratio
% USER-DEFINED LOAD DOFs
il = nelx; jl = 0; kl = 0:nelz;                          % Coordinates
loadnid = kl*(nelx+1)*(nely+1)+il*(nely+1)+(nely+1-jl); % Node IDs
loaddof = 3*loadnid(:) - 1;                              % DOFs
% USER-DEFINED SUPPORT FIXED DOFs
[jf,kf] = meshgrid(1:nely+1,1:nelz+1);                   % Coordinates
fixednid = (kf-1)*(nely+1)*(nelx+1)+jf;                  % Node IDs
fixeddof = [3*fixednid(:); 3*fixednid(:)-1; 3*fixednid(:)-2]; % DOFs
% PREPARE FINITE ELEMENT ANALYSIS
nele = nelx*nely*nelz;
ndof = 3*(nelx+1)*(nely+1)*(nelz+1);
F = sparse(loaddof,1,-1,ndof,1);
U = zeros(ndof,1);
freedofs = setdiff(1:ndof,fixeddof);
KE = lk_H8(nu);
nodegrd = reshape(1:(nely+1)*(nelx+1),nely+1,nelx+1);
nodeids = reshape(nodegrd(1:end-1,1:end-1),nely*nelx,1);
nodeidz = 0:(nely+1)*(nelx+1):(nelz-1)*(nely+1)*(nelx+1);
nodeids = repmat(nodeids,size(nodeidz))+repmat(nodeidz,size(nodeids));
edofVec = 3*nodeids(:)+1;
edofMat = repmat(edofVec,1,24)+ ...
    repmat([0 1 2 3*nely + [3 4 5 0 1 2] -3 -2 -1 ...
    3*(nely+1)*(nelx+1)+[0 1 2 3*nely + [3 4 5 0 1 2] -3 -2 -1]],nele,1);
iK = kron(edofMat,ones(24,1))';
jK = kron(edofMat,ones(1,24))';
% PREPARE FILTER
iH = ones(nele*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for k1 = 1:nelz
    for i1 = 1:nelx
        for j1 = 1:nely
            e1 = (k1-1)*nelx*nely + (i1-1)*nely+j1;
            for k2 = max(k1-(ceil(rmin)-1),1):min(k1+(ceil(rmin)-1),nelz)
                for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),nelx)
                    for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-1),nely)
                        e2 = (k2-1)*nelx*nely + (i2-1)*nely+j2;
                        k = k+1;
                        iH(k) = e1;
                        jH(k) = e2;
                        sH(k) = max(0,rmin-sqrt((i1-i2)^2+(j1-j2)^2+(k1-k2)^2));
                    end
                end
            end
        end
    end
end
H = sparse(iH,jH,sH);
Hs = sum(H,2);
% INITIALIZE ITERATION
x = repmat(volfrac,[nely,nelx,nelz]);
xPhys = x;
loop = 0;
change = 1;
% START ITERATION
while change > tolx && loop < maxloop
```

```
    loop = loop+1;
    % FE-ANALYSIS
    sK = KE(:)*(Emin+xPhys(:)'.^penal*(E0-Emin));
    K = sparse(iK(:),jK(:),sK(:)); K = (K+K')/2;
    U(freedofs,:) = K(freedofs,freedofs)\F(freedofs,:);
    % OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
    ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),[nely,nelx,nelz]);
    c = sum(sum(sum((Emin+xPhys.^penal*(E0-Emin)).*ce)));
    dc = -penal*(E0-Emin)*xPhys.^(penal-1).*ce;
    dv = ones(nely,nelx,nelz);
    % FILTERING AND MODIFICATION OF SENSITIVITIES
    dc(:) = H*(dc(:)./Hs);
    dv(:) = H*(dv(:)./Hs);
    % OPTIMALITY CRITERIA UPDATE
    l1 = 0; l2 = 1e9; move = 0.2;
    while (l2-l1)/(l1+l2) > 1e-3
        lmid = 0.5*(l2+l1);
        xnew = max(0,max(x-move,min(1,min(x+move,x.*sqrt(-dc./dv/lmid)))));
        xPhys(:) = (H*xnew(:))./Hs;
        if sum(xPhys(:)) > volfrac*nele, l1 = lmid; else l2 = lmid; end
    end
    change = max(abs(xnew(:)-x(:)));
    x = xnew;
    % PRINT RESULTS
    fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f ch.:%7.3f\n',loop,c,mean(xPhys(:)),change);
    % PLOT DENSITIES
    if displayflag, clf; display_3D(xPhys); end
end
clf; display_3D(xPhys);
end
% ========== AUXILIARY FUNCTIONS ==========
% GENERATE ELEMENT STIFFNESS MATRIX
function [KE] = lk_H8(nu)
A = [32 6 -8 6 -6 4 3 -6 -10 3 -3 -3 -4 -8;
    -48 0 0 -24 24 0 0 0 12 -12 0 12 12 12];
k = 1/72*A'*[1; nu];
% GENERATE SIX SUB-MATRICES AND THEN GET KE MATRIX
K1 = [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)];
K2 = [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)];
K3 = [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)];
K4 = [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)];
K5 = [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)];
K6 = [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)];
KE = 1/((nu+1)*(1-2*nu))*...
    [ K1  K2  K3  K4;
    K2'  K5  K6  K3';
    K3' K6  K5' K2';
    K4  K3  K2  K1'];
end
% DISPLAY 3D TOPOLOGY (ISO-VIEW)
function display_3D(rho)
[nely,nelx,nelz] = size(rho);
hx = 1; hy = 1; hz = 1;             % User-defined unit element size
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];
set(gcf,'Name','ISO display','NumberTitle','off');
for k = 1:nelz
    z = (k-1)*hz;
    for i = 1:nelx
        x = (i-1)*hx;
        for j = 1:nely
            y = nely*hy - (j-1)*hy;
            if (rho(j,i,k) > 0.5)  % User-defined display density threshold
                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
                    +hx;x+hx y z+hx];
                vert(:,[2 3]) = vert(:,[3 2]); vert(:,2,:) = -vert(:,2,:);
                patch('Faces',face,'Vertices',vert,'FaceColor',[0.2+0.8*(1-rho(j,i,k))
                    ,0.2+0.8*(1-rho(j,i,k)),0.2+0.8*(1-rho(j,i,k))]);
                hold on;
            end
        end
    end
end
axis equal; axis tight; axis off; box on; view([30,30]); pause(1e-6);
end
% ===================================================================
% === This code was written by K Liu and A Tovar, Dept. of Mechanical  ===
% === Engineering, Indiana University-Purdue University Indianapolis,  ===
% === Indiana, United States of America                                ===
% === ------------------------------------------------------------- ===
% === Please send your suggestions and comments to: kailiu@iupui.edu   ===
% === ------------------------------------------------------------- ===
% === The code is intended for educational purposes, and the details   ===
% === and extensions can be found in the paper:                        ===
% === '' ''                                                           ===
% === ------------------------------------------------------------- ===
% === The code as well as an uncorrected version of the paper can be   ===
% === downloaded from the website: http://engr.iupui.edu/~tovara/top3d ===
% === ------------------------------------------------------------- ===
% === Disclaimer:                                                     ===
% === The authors reserves all rights for the program.                ===
% === The code may be distributed and used for educational purposes.  ===
% === The authors do not guarantee that the code is free from errors, and =
% === they shall not be liable in any event caused by the use of the code.=
% ===================================================================
```