Topology Optimization using the UNsmooth VARiational Topology OPtimization (UNVARTOP) method: an educational implementation in Matlab

EDUCATIONAL ARTICLE

Daniel Yago^{1,2} · **Juan Cante**^{1,2} · **Oriol Lloberas-Valls**^{2,3} · **Javier Oliver**^{2,3}

Abstract This paper presents an efficient and comprehensive MATLAB code to solve two-dimensional structural topology optimization problems, including minimum mean compliance, compliant mechanism synthesis and multi-load compliance problems. The Unsmooth Variational Topology Optimization (UNVARTOP) method, developed by Oliver et al. [22], is used in the topology optimization code, based on the finite element method (FEM), to compute the sensitivity and update the topology. The paper also includes instructions to improve the *bisection algorithm*, modify the computation of the Lagrangian multiplier by using an Augmented Lagrangian to impose the constraint, implement heat conduction problems and extend the code to three-dimensional topology optimization problems. The code, intended for students and newcomers in topology optimization, is included as an appendix (Appendix A) and it can be downloaded from https://github.com/DanielYago together with supplementary material.

✉ J. Oliver
E-mail: oliver@cimne.upc.edu

1 Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa (ESEIAAT)
Technical University of Catalonia (UPC/Barcelona Tech), Campus Terrassa UPC, c/ Colom 11, 08222 Terrassa, Spain

2 Centre Internacional de Mètodes Numèrics en Enginyeria (CIMNE)
Campus Nord UPC, Mòdul C-1 101, c/ Jordi Girona 1-3, 08034 Barcelona, Spain

3 E.T.S d'Enginyers de Camins, Canals i Ports de Barcelona (ETSECCPB)
Technical University of Catalonia (UPC/Barcelona Tech), Campus Nord UPC, Mòdul C-1, c/ Jordi Girona 1-3, 08034 Barcelona, Spain

Keywords Structural Topology optimization · Relaxed Topological Derivative · Compliance · Compliant Mechanism · Education · MATLAB code

1 Introduction

The dissemination of the Matlab code, included in this paper, is intended for education purposes, in order to provide students and those new to the field with the theoretical basis for topology optimization of structural problems as well as to familiarize a wider audience with the new technique. This article is inspired by similar ones (e.g. [27] and [5]) which presented a Matlab implementation and possible extensions of other topology optimization approaches for structural problems.

A wide variety of topology optimization approaches and the corresponding Matlab implementations can be found in the literature, including the *Solid Isotropic Material with Penalization* (SIMP) method ([6; 7] and [27]), the *Bidirectional Evolutionary Structural Optimization* (BESO) method ([39; 42] and [45]), the *Level-set* method using a shape derivative ([2; 3; 34] and [10; 35]), the *Topology Derivative* method ([29; 21] and [30]) and the *Phase-field* approaches ([31; 34; 41] and [24]), among others. Along years, researchers have adapted or combined some features of these techniques to propose alternative approaches. Nevertheless, some limitations remain in any of them.

The *Unsmooth Variational Topology Optimization* approach, first developed by Oliver et al. [22], appears to be an alternative to other well-established approaches due to the mathematical simplicity and robustness of the present method. So far, the UNVARTOP approach has been applied in a wide range of linear applications, including static structural [22] and steady-state thermal applications [40], considering the volume constraint as a single constraint equation, with promising results.

The domain, in the present approach, is implicitly represented through a 0-level-set function [23], using the so-called *discrimination function* ψ, to define a discrete *characteristic function*, χ, at each point of the domain, $\mathbf{x}$. This variable, used as design variable, is related to the *discrimination function* with the Heaviside function by $\chi(\mathbf{x}) = \mathcal{H}(\psi(\mathbf{x}))$, defining, thus, a *black-and-white design*, i.e. a binary configuration with two domains: a void and a material domain. This definition is in contrast to that used by density-based methods, such as SIMP method, where the relative density, ρ_e, in each element is used as design variable see Bendsøe and Sigmund [7]. In addition, this change in the design variable, typically from *Level-set methods*, allows smooth representation of the topology (void and material domains) and the corresponding boundary using the 0-level iso-surface of the *discrimination function*.

The *black-and-white design* is relaxed via the *ersatz material approach* to a bi-material setting, where the void material is replaced with a soft material, as proposed by Allaire et al. [1]. Despite this relaxation, the discrete nature of the *characteristic function* is maintained. However, this is not true for density-based methods, which have to be relaxed via a power-law interpolation function to intermediate values (i.e. between void and solid), leading thus to the SIMP method, in order to avoid the ill-conditioning of the topology optimization problem obtaining then blurry interfaces with semi-dense elements, as stated in Sigmund and Petersson [28].

The aim of a topology optimization must be defined by means of a cost function, which will be minimized. For each specific cost function, a sensitivity evaluating the variation of it to topological perturbations must be derived. This derivation may be mathematically challenging for some topology optimization approaches. For example, the *Topology Derivative* method requires heavy analytical derivation methods, dependent on the type of the topology optimization problem and the considered material in the optimization [14; 21]. However, in the current method, a consistent *relaxed topological derivative* is formulated within the *ersatz material approach*, and evaluated as a directional derivative of the cost function. Additionally, it can be interpreted as an approximation of the exact topological derivative, used in *Topology Derivative* method, resulting in a simpler and less time-consuming derivation.

Apart from the problem setting and the cost function, the procedure of updating the design variable is a crucial feature of each approach. Most of the topology optimization methods, that use a *level-set* function to define the topology layout at each iteration, update the design variable via a Hamilton-Jacobi equation using an appropriate velocity at boundaries (in terms of the precomputed sensitivity) [2; 34]. Despite using an equivalent level-set function (*discrimination function*), the topology is not updated neither via a Hamilton-Jacobi [37; 3; 41] nor a Reaction-Diffusion [24] equations, but it is updated via the solution of a *fixed-point, non-linear, closed-form algebraic system.* The fulfillment of the volume constraint is ensured within the closed-form solution by means of a *Lagrange multiplier*, similar to the one used with Optimality Criteria (OC) in SIMP methods, computed through an efficient bisection algorithm.

Almost every technique require some kind of filtering in order to avoid or at least mitigate the inherent ill-posedness of the topology optimization problem [28]. Through this filtering, the lack of mesh-independency is overcome. Density-based methods resort to density or sensitivity filtering, extensively used in density-based approaches. Nevertheless, alternative filters have been formulated in the last two decades. For instance, projection methods [15] or a Helmholtz filter [16] are also used for this purpose. This last filter, so called the *Laplacian regularization* [25; 33] is applied to the *discrimination function* to control the filament width. A similar approach is used by Yamada et al. [41] to control the complexity of the optimal design.

Finally, the last key feature is related with the volume constraint and how the requested volume percentage is achieved. An incremental time-advancing scheme is adopted in the present methodology for the volume percentage, as a control parameter, obtaining, then, intermediate converged, optimal topologies. The optimization procedure starts from a domain fully filled with stiff material. Then, the topology optimization for a given small volume percentage is performed, obtaining a converged, optimal topology. Subsequently, the volume percentage (*pseudo-time* in the algorithm) is increased and the new optimal topology is found. This procedure is repeated until the desired volume fraction is achieved, similar to the Pareto frontier-optimal tracing approach proposed by Suresh [30]. Although this implementation is not unique of the current approach, it differs from SIMP and Level-set based methods, since they directly seek the optimal topology for the requested volume fraction. Similar iterative schemes can be found in ESO/BESO approaches, where the volume fraction is incremented at each iteration until the final volume is achieved. However, optimal conditions are not fulfilled at these intermediate volumes.

Thanks to this set of features, the methodology proposed in this manuscript presents a lower computational cost, around 5 times, when it is compared with other methods, e.g. a Level-set method with the RTD, while obtaining very similar results, as reported in Oliver et al. [22] and Yago et al. [40]. In addition, intermediate converged optimal topologies are obtained for different volume values at no additional computational cost, allowing further decisions once the topology optimization optimization has finalized.

The remainder of the paper is organized as follows. The unsmooth variational topology optimization approach is briefly described in section 2 along with the particularities for minimum mean compliance, multi-load compliance and compliant mechanisms problems. In section 3, the code implementation of the present methodology, provided in Appendix A, is discussed in detail. Several numerical examples are addressed in section 4 to show the potential in the three optimization problems. Additionally, in section 5, possible extensions and enhancements of the code are discussed. Finally, section 6 concludes with some final remarks.

2 Problem formulation

2.1 Unsmooth variational topology optimization

Let us define a fixed rectangular design domain, $\Omega \subset \mathbb{R}^2$, composed by two smooth subdomains, Ω^+ and Ω^-, as depicted in Figure 1. These two domains, made respectively of solid and void materials, are defined via the nonsmooth *characteristic function*, $\chi(\mathbf{x}) : \Omega \to \{0,1\}$, as

$$\begin{cases} \Omega^+ := \{\mathbf{x} \in \Omega \;/\; \chi(\mathbf{x}) = 1\} \\ \Omega^- := \{\mathbf{x} \in \Omega \;/\; \chi(\mathbf{x}) = 0\} \end{cases} . \tag{1}$$

The topology layout can also be implicitly represented by the smooth *discrimination function*, $\psi(\mathbf{x}) : \Omega \to \mathbb{R}$, $\psi \in H^1(\Omega)$, (see Figure 2) defined as

$$\begin{cases} \Omega^+ := \{\mathbf{x} \in \Omega \;/\; \psi(\mathbf{x}) > 0\} \\ \Omega^- := \{\mathbf{x} \in \Omega \;/\; \psi(\mathbf{x}) < 0\} \end{cases} . \tag{2}$$

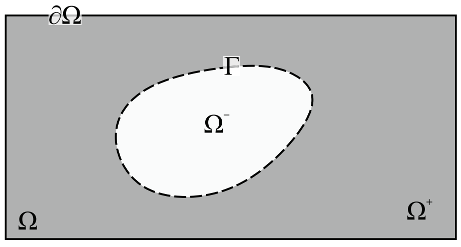


Fig. 1 Representation of the fixed design domain Ω.

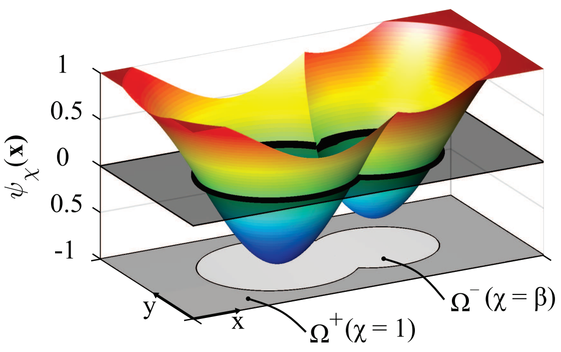


Fig. 2 Topology representation in terms of the *discrimination function*, ψ.

In addition, the *characteristic function*, $\chi_\psi(\mathbf{x}) : \Omega \to \{0,1\}$, can be expressed in terms of the *discrimination function* by

$$\chi_\psi(\mathbf{x}) = \mathcal{H}(\psi(\mathbf{x}))\,, \tag{3}$$

where $\mathcal{H}(\cdot)$ stands for the Heaviside function evaluated at $(\cdot)$. The *characteristic function*, used as the design variable, is now relaxed to $\chi_\psi(\mathbf{x}) : \Omega \to \{\beta, 1\}$, where the void material is replaced with a soft material with low stiffness (ersatz material approach), with β being the *relaxation factor*.

The topology optimization goal is to minimize a cost function $\mathcal{J}(\chi)$ subjected to one constraint, typically the volume, and governed by the state equations. The mathematical formulation of the corresponding topology optimization problem can be expressed as

$$\left[\begin{array}{lll} \displaystyle\min_{\chi\in\mathscr{U}_{ad}} & \displaystyle\mathcal{J}(\chi) \equiv \int_\Omega j(\chi,\mathbf{x})\, d\Omega & (a) \\ \text{subject to:} & & \\ & \displaystyle\mathcal{C}(\chi) \equiv \int_\Omega c(\chi,\mathbf{x})\, d\Omega = 0 & (b) \\ \text{governed by:} & & \\ & \textit{Equilibrium equation} & (c) \end{array}\right. , \tag{4}$$

where $\mathscr{U}_{ad}$ stands for the set of admissible solutions for χ and $\mathcal{C}(\chi)$ represents the constraint functional (e.g. the volume constraint).

Following Oliver et al. [22], the *Relaxed Topological Derivative* (RTD) evaluated as

$$\frac{\delta\mathcal{J}(\chi)}{\delta\chi}(\hat{\mathbf{x}}) = \left[\frac{\partial j(\chi,\mathbf{x})}{\partial\chi}\right]_{\mathbf{x}=\hat{\mathbf{x}}} \Delta\chi(\hat{\mathbf{x}})\,, \tag{5}$$

measures the sensitivity of the functional (4)-a when a material exchange is made at point $\hat{\mathbf{x}}$. The term $\Delta\chi(\hat{\mathbf{x}})$, denoted as the *exchange function*, corresponds to the signed variation of $\chi(\hat{\mathbf{x}})$, due to that material exchange, i.e.

$$\Delta\chi(\hat{\mathbf{x}}) = \begin{cases} -(1-\beta) < 0 & \textit{for } \hat{\mathbf{x}} \in \Omega^+ \\ (1-\beta) > 0 & \textit{for } \hat{\mathbf{x}} \in \Omega^- \end{cases} . \tag{6}$$

Mimicking equation (5), the RTD of the volume constraint ((4)-b) is computed as

$$\frac{\delta\mathcal{C}(\chi,t)}{\delta\chi}(\hat{\mathbf{x}}) = \left[\frac{\partial c(\chi,\mathbf{x})}{\partial\chi}\right]_{\mathbf{x}=\hat{\mathbf{x}}} \Delta\chi(\hat{\mathbf{x}}) = \frac{1}{|\Omega|}\operatorname{sgn}(\Delta\chi(\hat{\mathbf{x}})) , \tag{7}$$

where $\mathcal{C}(\chi,t) := t - \frac{|\Omega^-|(\chi)}{|\Omega|} = 0$ and $|\Omega^-|(\chi) = \int_\Omega \frac{1-\chi}{1-\beta}\, d\Omega$. Additionally, the term $t \in [0,T]$ corresponds to the *pseudo-time* parameter, given by the user, used in the *pseudo-time-advancing strategy*. Notice that the parameter T stands for the *pseudo-time* corresponding to the final volume.

The *Lagrangian function* of the optimization problem (4) can be expressed as

$$\mathcal{L}(\chi) = \mathcal{J}(\chi) + \lambda\mathcal{C}(\chi,t) , \tag{8}$$

where the *constraint equation*, $\mathcal{C}$, multiplied with a Lagrange multiplier, λ, is added to the original cost function $\mathcal{J}$. The value of λ is such that the volume constraint is fulfilled.

Finally, applying the RTD to equation (8) and considering equations (5) and (7), the optimality condition of the *original topology optimization problem* can be written as

$$\begin{aligned} \frac{\delta\mathcal{L}(\chi,\lambda)}{\delta\chi}(\hat{\mathbf{x}}) &= \left(\frac{\partial j\,(\chi,\hat{\mathbf{x}})}{\partial\chi}\Delta\chi(\hat{\mathbf{x}}) + \lambda\operatorname{sgn}(\Delta\chi(\hat{\mathbf{x}}))\right) = \\ &= -\psi(\hat{\mathbf{x}},\chi) = -(\xi(\hat{\mathbf{x}},\chi) - \lambda) \quad \forall\hat{\mathbf{x}} \in \Omega , \end{aligned} \tag{9}$$

where $\psi(\hat{\mathbf{x}},\chi)$ corresponds to the *discrimination function* and $\xi(\hat{\mathbf{x}},\chi)$ is termed the *pseudo-energy* and must be computed for each optimization problem. Compared to other techniques, the *pseudo-energy* is first shifted[1] and normalized, yielding to the *modified energy density* defined as

$$\hat{\xi}(\hat{\mathbf{x}}) = \frac{\xi(\hat{\mathbf{x}}) - \chi(\hat{\mathbf{x}})\Delta_{shift}}{\Delta_{norm}} , \tag{10}$$

where Δ_{shift} and Δ_{norm} correspond to the shifting and normalization parameters defined at the first iteration as $\min(\xi_0, 0)$ and $\max(\operatorname{range}(\xi_0), \max(\xi_0))$, respectively. The resultant ψ, after replacing equation (10) into (9), is subsequently smoothed through a *Laplacian regularization* in order to mitigate mesh-dependency along with controlling the minimum filament's size. The *smooth discrimination function*, ψ_τ, corresponds to the solution of

$$\begin{cases} \psi_\tau - (\tau h_e)^2\Delta_{\mathbf{x}}\psi_\tau = \psi & \textit{in } \Omega \\ \nabla_{\mathbf{x}}\psi_\tau \cdot \mathbf{n} = 0 & \textit{on } \partial\Omega \end{cases} , \tag{11}$$

where, $\Delta_{\mathbf{x}}(\mathbf{x},\cdot)$ and $\nabla_{\mathbf{x}}(\mathbf{x},\cdot)$ are respectively the Laplacian and gradient operators, and $\mathbf{n}$ is the outwards normal to the boundary of the design domain, $\partial\Omega$. τ and h_e stand for the dimensionless *regularization parameter* and the typical size of the finite element mesh, respectively.

The topology layout, χ, is updated by means of the *Cutting&Bisection* algorithm, in which the value of λ, which enforces volume constraint (equation (4)-b), is computed. Then, a *closed-form solution* of the topology optimization problem (4) can be written as

$$\begin{cases} \psi(\hat{\mathbf{x}}) := \hat{\xi}(\hat{\mathbf{x}},\chi) - \lambda \\ \chi(\hat{\mathbf{x}}) = \mathcal{H}(\psi_\tau(\hat{\mathbf{x}})) & \textit{in } \Omega , \\ \mathcal{C}(\chi(\lambda),t) = 0 \end{cases} \tag{12}$$

where $\psi_\tau(\hat{\mathbf{x}})$ corresponds to the solution of equation (11) that must be applied at each iteration. Equation (12) constitutes a fundamental feature of the UNVARTOP method, as aforementioned in section 1. Nonetheless, the *Laplacian regularization* only affects the *modified energy density*, $\hat{\xi}(\hat{\mathbf{x}},\chi)$, since the term λ is constant, thus leading equation (12) to

$$\begin{cases} \psi_\tau(\hat{\mathbf{x}}) := \hat{\xi}_\tau(\hat{\mathbf{x}},\chi) - \lambda \\ \chi(\hat{\mathbf{x}}) = \mathcal{H}(\psi_\tau(\hat{\mathbf{x}})) & \textit{in } \Omega , \\ \mathcal{C}(\chi(\lambda),t) = 0 \end{cases} \tag{13}$$

where $\hat{\xi}_\tau$ is the solution of equation (11) for the *modified energy density*. Due to this modification, the computational cost of the bisection algorithm is significantly reduced.

For more details on the formulation, the reader is referred to Oliver et al. [22] and Yago et al. [40], where in-depth discussions are made on each subject.

2.2 State problem

The governing variational problem for linear elasticity, in terms of the displacement field ($\mathbf{u}_\chi$) and the virtual

[1]The shifting is applied in order to obtain positive *pseudo-energy*, ξ, in Ω at $t = 0$, thus, ensuring a converged topology for this time-step.

displacement field ($\mathbf{w}$), can be written as

$$\left[\begin{aligned} &\text{Find the displacement field } \boldsymbol{u}_\chi \in \mathcal{U}(\Omega) \text{ such that} \\ &\quad a(\mathbf{w},\mathbf{u}_\chi) = l(\mathbf{w}) \quad \forall \mathbf{w} \in \mathcal{V}(\Omega) \qquad (14) \\ &\text{where} \\ &\quad a(\mathbf{w},\mathbf{u}_\chi) = \int_\Omega \boldsymbol{\nabla}^S \mathbf{w}(\mathbf{x}) : \mathbb{C}_\chi(\mathbf{x}) : \boldsymbol{\nabla}^S \mathbf{u}_\chi(\mathbf{x})\, d\Omega\,, \qquad (15) \\ &\quad l(\mathbf{w}) = \int_{\partial_\sigma \Omega} \mathbf{w}(\mathbf{x}) \cdot \bar{\boldsymbol{\sigma}}(\mathbf{x})\, d\Gamma \\ &\qquad\quad + \int_\Omega \mathbf{w}(\mathbf{x}) \cdot \mathbf{b}_\chi(\mathbf{x})\, d\Omega\,, \qquad (16) \end{aligned}\right.$$

where $\mathbb{C}_\chi$ and $\mathbf{b}_\chi$ correspond to the fourth order elastic constitutive tensor and the volumetric force, respectively. In addition, $\bar{\boldsymbol{\sigma}}(\mathbf{x})$ stands for the boundary tractions applied on $\partial_\sigma \Omega \subset \partial\Omega$, while the term $\boldsymbol{\nabla}^S(\cdot)$ corresponds to the symmetrical gradient of $(\cdot)$. Finally, the set of admissible displacement fields, $\mathcal{U}(\Omega)$, is defined as $\mathcal{U}(\Omega) := \left\{\mathbf{u}(\mathbf{x}) \;/\; \mathbf{u} \in H^1(\Omega),\; \mathbf{u} = \bar{\mathbf{u}} \text{ on } \partial_u\Omega\right\}$, while the space of admissible virtual displacement fields is given by $\mathcal{V}(\Omega) := \left\{\mathbf{w}(\mathbf{x}) \;/\; \mathbf{w} \in H^1(\Omega),\; \mathbf{w} = 0 \text{ on } \partial_u\Omega\right\}$.

The constitutive tensor[2], $\mathbb{C}_\chi$, and the volumetric force, $\mathbf{b}_\chi$, depend on the topology. Thus, they are mathematically defined in terms of the *characteristic function* as follows

$$\begin{cases} \mathbb{C}_\chi(\mathbf{x}) = \chi_k^{m_k}(\mathbf{x})\overline{\mathbb{C}}(\mathbf{x})\,; \quad m_k > 1 & (17) \\ \mathbf{b}_\chi(\mathbf{x}) = \chi_b^{m_b}(\mathbf{x})\overline{\mathbf{b}}(\mathbf{x})\,; \quad m_b > 1 & (18) \end{cases}$$

where $m_{(\cdot)}$ stands for the *exponential factor* of property $(\cdot)$. The lower limit of the *relaxed characteristic function*, χ_β, is defined through the *contrast factor*, $\alpha_{(\cdot)}$, and $m_{(\cdot)}$ by $\beta_{(\cdot)} = \alpha_{(\cdot)}^{1/m_{(\cdot)}}$. Both $\overline{\mathbb{C}}$ and $\overline{\mathbf{b}}$ denote the corresponding nominal property of the stiff material.

Assuming plane-stress condition, the constitutive tensor $\overline{\mathbb{C}}$ is given by

$$\overline{\mathbb{C}}^{Pstress} = \frac{E}{1-\nu^2}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \dfrac{1-\nu}{2} \end{bmatrix}, \qquad (19)$$

with E representing the Young's modulus of the *stiff material* and ν, the Poisson's ratio of the isotropic material.

2.3 Finite element discretization

The state equation (14) is now discretized using the standard finite element method [44; 26]. The displacement field and its gradient are approximated as follows

$$\mathbf{u}_\chi(\mathbf{x}) \equiv \mathbf{N}_u(\mathbf{x})\hat{\boldsymbol{u}}_\chi \qquad (20)$$

$$\boldsymbol{\nabla}^S u_\chi(\mathbf{x}) \equiv \mathbf{B}(\mathbf{x})\hat{\boldsymbol{u}}_\chi \qquad (21)$$

where $\mathbf{N}_u(\mathbf{x})$ and $\mathbf{B}(\mathbf{x})$ stand for the displacement, shape function matrix and the strain-displacement matrix, respectively, and $\hat{\boldsymbol{u}}_\chi$ corresponds to the nodal displacement vector.

Introducing equations (17)-(18) and (20)-(21) into equations (14)-(16), the resultant state equation reads

$$\mathbb{K}_\chi \hat{\boldsymbol{u}}_\chi = \mathbf{f} \qquad (22)$$

with

$$\begin{cases} \mathbb{K}_\chi = \displaystyle\int_\Omega \mathbf{B}^{\mathrm{T}}(\mathbf{x})\; \mathbb{C}_\chi(\mathbf{x})\; \mathbf{B}(\mathbf{x})\, d\Omega \\ \mathbf{f} = \displaystyle\int_{\partial_\sigma\Omega} \mathbf{N_u}^{\mathrm{T}}(\mathbf{x})\bar{\boldsymbol{\sigma}}(\mathbf{x})\, d\Gamma \\ \qquad + \displaystyle\int_\Omega \mathbf{N_u}^{\mathrm{T}}(\mathbf{x})\mathbf{b}_\chi(\mathbf{x})\, d\Omega \end{cases}, \qquad (23)$$

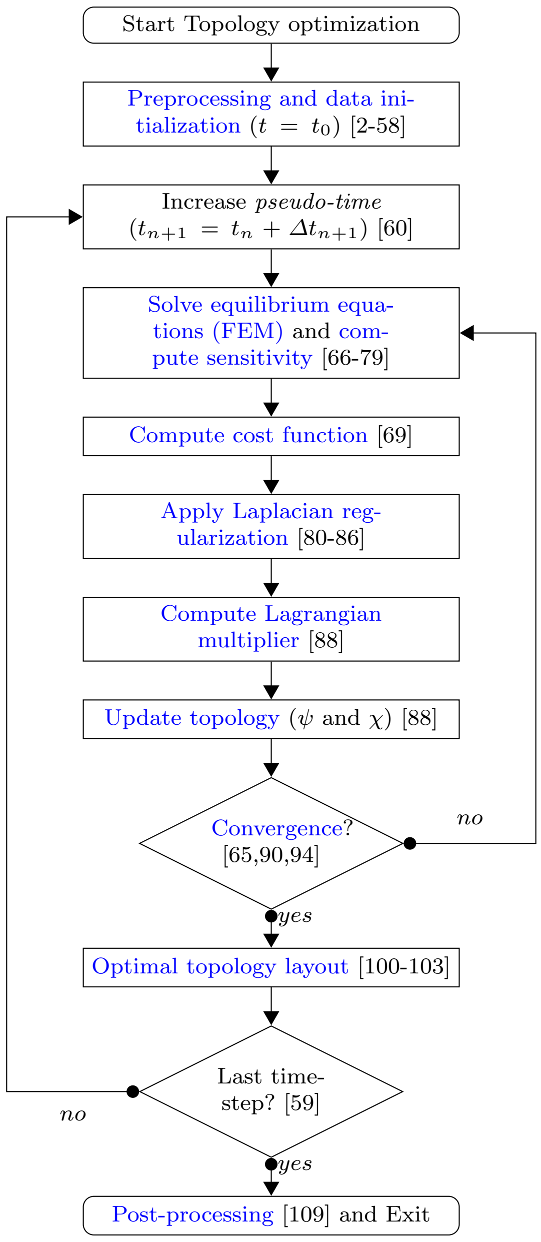
where $\mathbb{K}_\chi$ and $\mathbf{f}$ stand for the stiffness matrix and the external forces vector, respectively. The element stiffness matrix and the volumetric term of the force vector are numerically integrated inside each element, Ω_e, employing several quadrature points. Subsequently, these terms are assembled to obtain the global stiffness matrix and force vector.

2.4 Algorithm

The flowchart of the algorithm used to obtain the optimal topology layouts in terms of the *characteristic function*, χ, is illustrated in Figure 3.

The algorithm is based on a two-steps procedure: 1) data initialization and FE analysis pre-processing, e.g. mesh generation, creation of figures, computation of element FE matrices, assembly of Laplacian regularization matrix, along others, and 2) a topology optimization loop over time-steps. For each step, the state equation (22) is solved to obtain the displacement vector, and the corresponding sensitivities are computed (equations (5) and (7)), obtaining then the *pseudo-energy*, ξ, dependent on each topology optimization problem defined in subsequent sections, and the corresponding *modified energy density*, $\hat{\xi}$ (equation (10)). The cost function is then computed via equation (4)-a using the previously computed displacement vector. Then, the *Laplacian regularization* is applied to $\hat{\xi}$ (equation (11)) while the Lagrange multiplier is obtained by means of a *bisection algorithm* (equation (13)), thus obtaining the new optimal topology (in terms of the *discrimination*

[2]The constitutive tensor is governed by Hooke's law, i.e. $\boldsymbol{\sigma} = \mathbb{C}\boldsymbol{\varepsilon}$, with $\boldsymbol{\varepsilon}$ being the strain tensor ($\boldsymbol{\varepsilon} = \boldsymbol{\nabla}^S u_\chi(\mathbf{x})$).



Start Topology optimization
Preprocessing and data initialization ($t = t_0$) [2-58]
Increase *pseudo-time* ($t_{n+1} = t_n + \Delta t_{n+1}$) [60]
Solve equilibrium equations (FEM) and compute sensitivity [66-79]
Compute cost function [69]
Apply Laplacian regularization [80-86]
Compute Lagrangian multiplier [88]
Update topology (ψ and χ) [88]
Convergence? [65,90,94]
no
yes
Optimal topology layout [100-103]
Last time-step? [59]
no
yes
Post-processing [109] and Exit

Fig. 3 The flowchart for the unsmooth variational topology optimization algorithm with the corresponding code lines in brackets.

function ψ and the corresponding *characteristic function* χ). If tolerances are fulfilled[3], the topology is considered as converged and then the *pseudo-time*, t, is increased. Otherwise, an iteration is carried out with the new topology.

[3]The L2-norm of the *characteristic function* and the L∞-norm of the Lagrange multiplier are checked.

2.5 Mean compliance

The main goal of the minimum mean compliance problems is to seek the *optimal topology layout*, in terms of the *characteristic function*, χ, that maximizes the global stiffness of the structure given specific boundary conditions. That is, the external work produced by applied forces is minimized. The objective function is written as

$$\mathcal{J}(\mathbf{u}_\chi) \equiv l(\mathbf{u}_\chi) \equiv a_\chi(\mathbf{u}_\chi, \mathbf{u}_\chi) \equiv \\ \equiv 2\int_\Omega \frac{1}{2}\boldsymbol{\nabla}^S\mathbf{u}_\chi : \mathbb{C}_\chi : \boldsymbol{\nabla}^S\mathbf{u}_\chi \, d\Omega = 2\int_\Omega \mathcal{U}_\chi \, d\Omega \,, \tag{24}$$

where $\mathcal{U}_\chi$ can be identified as the *actual strain energy density* ($\mathcal{U}_\chi = \frac{1}{2}\boldsymbol{\nabla}^S\mathbf{u}_\chi : \mathbb{C}_\chi : \boldsymbol{\nabla}^S\mathbf{u}_\chi$), and $a_\chi(\mathbf{u}_\chi, \mathbf{u}_\chi)$ and $l(\mathbf{u}_\chi)$ are the bilinear forms of the elastic problem (14) for $\mathbf{w} = \mathbf{u}_\chi$.

Considering equations (22) and (24), the corresponding finite element discretization counterpart of problem (4) reads

$$\left[\begin{array}{ll} \displaystyle\min_{\chi \in \mathcal{U}_{ad}} \; \mathcal{J}^{(h_e)}(\mathbf{u}_\chi(t)) \equiv \mathbf{f}^{\mathrm{T}}\hat{\boldsymbol{u}}_\chi(t) & (a) \\ \text{subject to:} & \\ \quad \mathcal{C}(\chi, t) := t - \dfrac{|\Omega^-|(\chi)}{|\Omega|} = 0\,; \quad t \in [0,1] & (b) \\ \text{governed by:} & \\ \quad \mathbb{K}_\chi \hat{\boldsymbol{u}}_\chi = \mathbf{f} & (c) \end{array}\right. , \tag{25}$$

where $\mathbf{f}^{\mathrm{T}}\hat{\boldsymbol{u}}_\chi$ denotes the structural compliance.

According to Oliver et al. [22], the *relaxed topological derivative* with respect to $\chi(\mathbf{x})$, using the adjoint method[4], is defined as

$$\frac{\delta \mathcal{J}^{(h_e)}(\chi)}{\delta \chi}(\hat{\mathbf{x}}) = \left[2\frac{\delta \mathbf{f}_\chi^{\mathrm{T}}}{\delta \chi}(\mathbf{x})\hat{\boldsymbol{u}}_\chi - \hat{\boldsymbol{u}}_\chi^{\mathrm{T}} \frac{\delta \mathbb{K}_\chi}{\delta \chi}(\mathbf{x})\hat{\boldsymbol{u}}_\chi \right]_{\mathbf{x}=\hat{\mathbf{x}}} . \tag{26}$$

Assuming that no volumetric forces are applied on the domain and substituting the definition of the relaxed topological derivative of each term (5), equation (26) can be expressed as

$$\frac{\delta \mathcal{J}^{(h_e)}(\mathbf{u}_\chi)}{\delta \chi}(\hat{\mathbf{x}}) = -2m_k \left(\chi_k(\hat{\mathbf{x}})\right)^{m_k - 1} \overline{\mathcal{U}}(\hat{\mathbf{x}}) \Delta\chi_k(\hat{\mathbf{x}}) \,, \tag{27}$$

[4]The adjoint method is used to avoid explicitly compute the sensitivities of the displacements. The minimum mean compliance problem is self-adjoint.

where the *nominal energy density*, $\overline{\mathcal{U}}(\hat{\mathbf{x}})$, is given by

$$\overline{\mathcal{U}}(\hat{\mathbf{x}}) = \frac{1}{2}\left(\boldsymbol{\nabla}^S\mathbf{u}_\chi : \overline{\mathbb{C}} : \boldsymbol{\nabla}^S\mathbf{u}_\chi\right)(\hat{\mathbf{x}}) \geq 0\,. \tag{28}$$

Finally, comparing equation (27) with equation (9), the *pseudo-energy*, $\xi(\hat{\mathbf{x}},\chi)$, of topology problem (25) reads

$$\xi(\hat{\mathbf{x}},\chi) = 2m_k\left(\chi_k(\hat{\mathbf{x}})\right)^{m_k-1}\overline{\mathcal{U}}(\hat{\mathbf{x}})\Delta\chi_k(\hat{\mathbf{x}})\,, \tag{29}$$

which must be then modified as detailed in equation (10). Discretizing the terms in equation (29), and after some mathematical manipulations, it can be numerically computed as

$$\xi(\hat{\mathbf{x}},\chi) = \gamma_1\hat{\mathbf{u}}(\hat{\mathbf{x}})^{\mathrm{T}}\left[\mathbf{B}(\hat{\mathbf{x}})^{\mathrm{T}}\overline{\mathbb{C}}\mathbf{B}(\hat{\mathbf{x}})\right]\hat{\mathbf{u}}(\hat{\mathbf{x}})\,, \tag{30}$$

with $\gamma_1 = 2m_k\left(\chi_k(\hat{\mathbf{x}})\right)^{m_k-1}\Delta\chi_k(\hat{\mathbf{x}})$.

2.6 Multi-load mean compliance

Multi-load compliance problems are considered a specific case of minimum compliance problems (see section 2.5), in which a set of elastic problems with different loading conditions are solved independently. The objective function (24) is replaced with the weighted average sum of all the cases, i.e.

$$\begin{aligned}\mathcal{J}(\mathbf{u}_\chi) &\equiv \sum_{i=1}^{n_l} l\left(\mathbf{u}_\chi^{(i)}\right) \equiv \\ &\equiv \sum_{i=1}^{n_l}\int_\Omega \boldsymbol{\nabla}^S\mathbf{u}_\chi^{(i)} : \mathbb{C}_\chi : \boldsymbol{\nabla}^S\mathbf{u}_\chi^{(i)}\,d\Omega = \\ &= \sum_{i=1}^{n_l} 2\int_\Omega \mathcal{U}_\chi^{(i)}\,d\Omega\,,\end{aligned} \tag{31}$$

where n_l stands for the number of loading states and $\mathcal{U}_\chi^{(i)}$ corresponds to the *actual energy density* of the i-th loading state. Then, according to this new definition, equation (25) is rewritten as

$$\left[\begin{array}{ll}\min\limits_{\chi\in\mathscr{U}_{ad}} \mathcal{J}^{(h_e)}(\mathbf{u}_\chi(t)) \equiv \sum\limits_{i=1}^{n_l}\mathbf{f}^{(i)\mathrm{T}}\hat{\boldsymbol{u}}_\chi^{(i)}(t) & (a)\\ \text{subject to:} & \\ \quad\mathcal{C}(\chi,t) := t - \dfrac{|\Omega^-|(\chi)}{|\Omega|} = 0\,;\quad t\in[0,1] & (b)\\ \text{governed by:} & \\ \quad\mathbb{K}_\chi\hat{\boldsymbol{u}}_\chi^{(i)} = \mathbf{f}^{(i)}\quad \forall i\in[1,n_l] & (c)\end{array}\right. . \tag{32}$$

Equations (26) to (29) are consequently modified to account multiple loading cases, leading to

$$\xi(\hat{\mathbf{x}},\chi) = \gamma_1\sum_{i=1}^{n_l}\hat{\mathbf{u}}^{(i)}(\hat{\mathbf{x}})^{\mathrm{T}}\left[\mathbf{B}(\hat{\mathbf{x}})^{\mathrm{T}}\overline{\mathbb{C}}\mathbf{B}(\hat{\mathbf{x}})\right]\hat{\mathbf{u}}^{(i)}(\hat{\mathbf{x}})\,. \tag{33}$$

Bear in mind that the optimal topology layout will considerably differ from the single minimum compliance problem with all the loads applied at the same time. Multi-load optimization problems are employed to find a trade-off between optimal topologies for each loading state.

2.7 Compliant mechanisms

Compliant mechanisms are flexible structures that transfer an action (force or displacement) at the *input port* to the *output port*, obtaining a desired force or displacement at that port. The objective function, $\mathcal{J}$, can be expressed in terms of the displacement at the output port, when maximum displacement is sought, as

$$\mathcal{J}(\mathbf{u}_\chi) \equiv \mathbf{1}^{\mathrm{T}}\hat{\mathbf{u}}_\chi\,, \tag{34}$$

where $\mathbf{1}$ represents a dummy constant force vector applied only on the *output port* at the desired direction. Additional springs, denoted by K_{in} and K_{out}, must be considered in the *input* and *output ports*, respectively.

In the context of finite element discretization, like in equation (25), the *topology optimization problem* (4) can be expressed as

$$\left[\begin{array}{ll}\min\limits_{\chi\in\mathscr{U}_{ad}} \mathcal{J}^{(h_e)}(\mathbf{u}_\chi(t)) \equiv -\mathbf{1}^{\mathrm{T}}\hat{\boldsymbol{u}}_\chi(t) & (a)\\ \text{subject to:} & \\ \quad\mathcal{C}(\chi,t) := t - \dfrac{|\Omega^-|(\chi)}{|\Omega|} = 0\,;\quad t\in[0,1] & (b)\\ \text{governed by:} & \\ \quad\mathbb{K}_\chi\hat{\boldsymbol{u}}_\chi = \mathbf{f} & (c)\end{array}\right. , \tag{35}$$

where the cost function (34) has been defined as a minimization problem by changing its sign.

Contrary to the problem of minimal compliance (section 2.5), the compliant mechanism problem is not self-adjoint. Thus, an *auxiliary state problem* must be solved in addition to the *original state problem* (22). Both systems present the same stiffness matrix $\mathbb{K}_\chi$ but different actions and solutions $\hat{\boldsymbol{u}}_\chi^{(1)}$ and $\hat{\boldsymbol{u}}_\chi^{(2)}$, respectively, defined as

$$\left\{\begin{array}{ll}\mathbb{K}_\chi\,\hat{\boldsymbol{u}}_\chi^{(1)} = \mathbf{f}^{(1)} & \text{(system I)}\\ \mathbb{K}_\chi\,\hat{\boldsymbol{u}}_\chi^{(2)} = \mathbf{1} & \text{(system II)}\end{array}\right. \tag{36}$$

Following Oliver et al. [22], the *relaxed topological derivative* of the optimization problem (35), once the *adjoint state equation* (36) has been substituted in, can be expressed as

$$\frac{\delta\mathcal{J}^{(h_e)}(\chi)}{\delta\chi}(\hat{\mathbf{x}}) = \left[\hat{\boldsymbol{u}}_\chi^{(2)\ \mathrm{T}}\frac{\delta\mathbb{K}_\chi}{\delta\chi}(\mathbf{x})\hat{\boldsymbol{u}}_\chi^{(1)} - \hat{\boldsymbol{u}}_\chi^{(2)\ \mathrm{T}}\frac{\delta\mathbf{f}_\chi^{(1)}}{\delta\chi}(\mathbf{x})\right]_{\mathbf{x}=\hat{\mathbf{x}}}.$$

(37)

As proceeded in section 2.5, equation (37) can be simplified and expressed in terms of a *pseudo-energy density*, yielding to

$$\frac{\delta \overline{\mathcal{J}}^{(h_e)}(\mathbf{u}_\chi)}{\delta \chi}(\hat{\mathbf{x}}) = 2m_k \left(\chi_k(\hat{\mathbf{x}})\right)^{m_k - 1} \overline{\mathcal{U}}_{1-2}(\hat{\mathbf{x}}) \Delta\chi_k(\hat{\mathbf{x}}) \,, \tag{38}$$

when volumetric forces are neglected. The corresponding *nominal pseudo-energy density* can be determined as

$$\overline{\mathcal{U}}_{1-2}(\hat{\mathbf{x}}) = \frac{1}{2}\left(\boldsymbol{\nabla}^S \mathbf{u}_\chi^{(2)} : \overline{\mathbb{C}} : \boldsymbol{\nabla}^S \mathbf{u}_\chi^{(1)}\right)(\hat{\mathbf{x}}) \,. \tag{39}$$

Finally, mimicking equation (30), the *pseudo-energy*, $\xi(\hat{\mathbf{x}}, \chi)$, can be obtained as

$$\xi(\hat{\mathbf{x}}, \chi) = -\gamma_1 \hat{\mathbf{u}}^{(2)}(\hat{\mathbf{x}})^{\mathrm{T}} \left[\mathbf{B}(\hat{\mathbf{x}})^{\mathrm{T}} \overline{\mathbb{C}} \mathbf{B}(\hat{\mathbf{x}})\right] \hat{\mathbf{u}}^{(1)}(\hat{\mathbf{x}}) \,. \tag{40}$$

3 MATLAB implementation

The user can run the code from the Matlab prompt with the following Matlab call

```
UNVARTOP_2D_compliance (nelx,nely,nsteps,
   Vol0,Vol,k,tau)
```

where `nelx` and `nely` stand for the number of quadrilateral elements in the horizontal and vertical directions, respectively.[5] The following four parameters define the time evolution of the optimization procedure, being `nsteps` the number of increments to get from the initial void volume (`Vol0`) to the final void volume (`Vol`), and parameter `k` defines the curvature of the exponential function, in case this type of time-advancing sequence is preferred. For an equally-spaced pseudo-time advance, set `k` to 0. The remaining input variable, `tau`, rules the minimum filament's width of the optimal design. Other variables related with the topology optimization algorithm and the numerical example (geometry and boundary conditions) are defined inside the function (see Appendix A), and can be modified if needed.

For instance, the code can be called with the input line

```
UNVARTOP_2D_compliance
   (100,50,10,0,0.5,0,0.5)
```

[5]The design domains are assumed to be rectangular domains discretized with quadrilateral unit square finite elements.

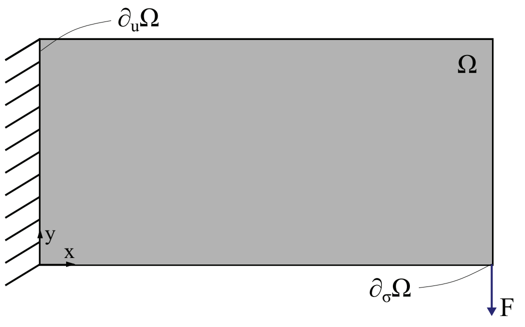


Fig. 4 Cantilever beam: topology optimization domain and boundary conditions.

for the default example, which corresponds to a cantilever beam with a vertical load applied on the bottom-right corner of Ω, and the displacements are prescribed on the left side of it, as illustrated in Figure 4. The algorithm generates two output figures, the first one displays the optimal topology for each iteration, and the second one shows the evolution of the cost function J_χ and the void volume, $|\Omega^-|$ along the time-steps, as depicted in Figure 5. At the end, a graphical user inter-

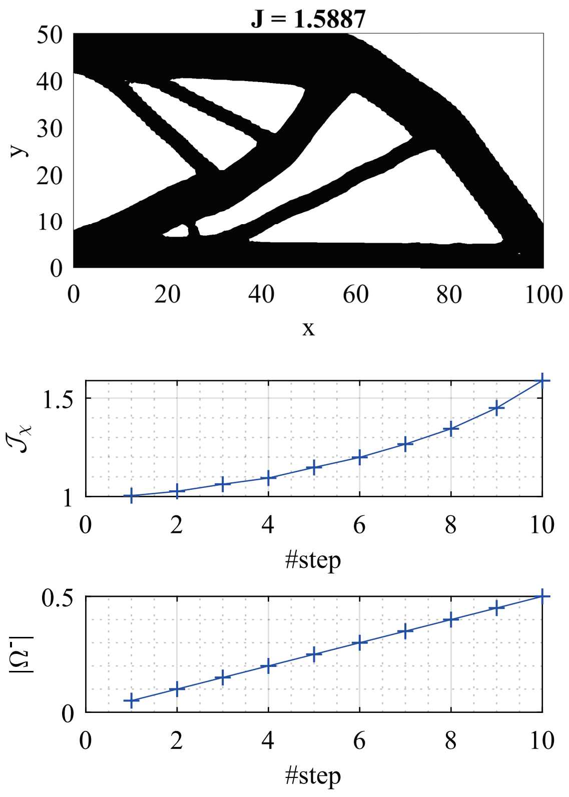


Fig. 5 Cantilever beam: topology optimization results.

face (GUI) with topology evolution, animated in Online Resource 1, is shown.

Relevant details of the Matlab code are explained in the following subsections for the minimum mean compliance problem (section 2.5), referring to the code in Appendix A, along with the required modifications to solve the topology optimization problems defined in sections 2.6 and 2.7.

3.1 Parameter definition: lines 2-4

Table 1 shows the list of variables and fields required by the program and used along it, excluding the variables already defined in the previous section. These parameters can be grouped in three blocks: all the parameters of the first block are related to the physical problem and the finite element used in the FEM analysis, the next three parameters conform the second block, which define the threshold iterations of the algorithm, and the last one defines a structure of optional parameters to choose which graphics are displayed and which solver is used to solve the *Laplacian regularization.*

3.2 Geometry definition: lines 5-9

The design domain, as aforementioned, is assumed to be rectangular and discretized with square elements. The FE mesh is defined via the coordinates and connectivities arrays, named `coord` and `connect` in the code. A coarse example mesh of the default example, see Figure 4, is illustrated in Figure 6, consisting of 15 nodes and 8 elements, numbered in column-wise (top to bottom) from left to right. The position of each node is defined respect to Cartesian coordinate system with origin at the left-bottom corner.

The `coord` matrix is generated using Matlab's `meshgrid` function and then, the obtained `X,Y` matrices are reshaped into the coordinates matrix, which dimensions are `[n x n_dim]`, i.e.

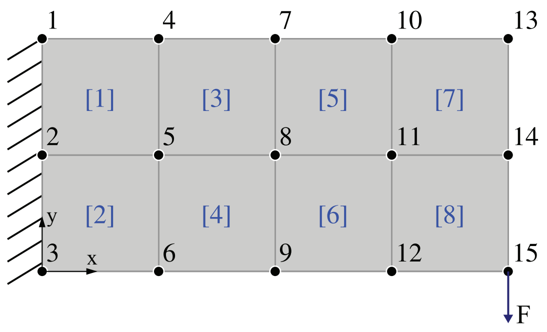


Fig. 6 Cantilever beam: mesh discretization.

Table 1 List of fields used in the code.

Variable	Value	Definition
`n_dim`	2	number of dimensions of the problem
`n_unkn`	2	number of unknown per node
`n_nodes`	4	number of nodes per element (e.g. 4 nodes for the quadrilateral element)
`n_gauss`	{1, 4}	total number of quadrature points of the quadrilateral element
`n`	`(nelx+1)*(nely+1)`	total number of nodes
`h_e`	1	element's size
`alpha0`	1e-3	Prescribed value of ψ for active/passive nodes
`iter_max_step`	20	maximum number of in-step iterations
`iter_min_step`	4	minimum number of in-step iterations
`iter_max`	500	maximum number of iterations
`opt.Plot_top_iso`	{true, false}	Boolean variable to plot the topology along iterations
`opt.Plot_vol_step`	{true, false}	Boolean variable to plot the evolution of the volume along iterations
`opt.EdgeColor`	{'none', RGB-color}	RGB color of the sides of the quadrilateral elements
`opt.Solver_Lap`	{'direct', 'iterative'}	method to solve the Laplacian regularization

$$\text{coord} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & \dots & 4 & 4 & 4 \\ 2 & 1 & 0 & 2 & 1 & \dots & 2 & 1 & 0 \end{bmatrix}^{\mathrm{T}} . \tag{41}$$

The connectivity matrix, `connect`, is constructed following the same procedure for computing the degree of freedom connectivity matrix, `edofMat`, described by Andreassen et al. [5]. First, matrix `nodenrs` is created with node IDs in a `(nely+1)x(nelx+1)` matrix in line 7, mimicking the numbering in Figure 6. Next, the left-bottom node ID of each element is stored in `nodeVec` vector, by using matrix `nodenrs`. For the given example, this variables are defined as follows:

$$\text{nodenrs} = \begin{bmatrix} 1 & 4 & 7 & 10 & 13 \\ 2 & 5 & 8 & 11 & 14 \\ 3 & 6 & 9 & 12 & 15 \end{bmatrix} \rightarrow \\ \rightarrow \text{nodeVec} = [2, 3, 5, 6, 8, 9, 11, 12]^{\mathrm{T}} . \tag{42}$$

Finally, thanks to the repetitive structure of the grid, the connectivity table, `connect`, can be constructed using only `nodeVec` and numbering within an element in anticlockwise order starting from the left-bottom node, which reads as

$$\text{connect} = \begin{bmatrix} 2 & 5 & 4 & 1 \\ 3 & 6 & 5 & 2 \\ \vdots & \vdots & \vdots & \vdots \\ 11 & 14 & 13 & 10 \\ 12 & 15 & 14 & 11 \end{bmatrix} . \qquad (43)$$

3.3 Load and boundary definition: lines 10-17

Lines 11-17 define the boundary conditions for the displacement and force field. First, the force vector, `F`, and the displacement vector, `U`, are initialized in lines 11 and 12, respectively. Next, line 13 assigns the imposed force to the force vector, which corresponds to a downwards force applied at the bottom right corner, as illustrated in Figure 4, with a small value to limit the maximum displacement of the structure. The next line defines the prescribed degrees of freedom, and stores them in `fixed_dofs`.

Parameters `active_node` and `passive_node` of line 15 are used to force some nodes to be included in the stiff (Ω^+) and soft (Ω^-) material domains, respectively. It is done via the modification of the *discrimination function*, `psi`, as in line 54 for the initialization of the *discrimination function* or in the *bisection algorithm* (line 149), by imposing the value `alpha0` or `-alpha0`.

Finally, the list with free degrees of freedom is generated and stored in `free_dofs` (line 16), and the displacement of `fixed_dofs` are prescribed to the corresponding value, e.g. 0.

3.4 Material definition: lines 18-19

The material used for the analysis is defined in terms of the Young's modulus `E0`, of the stiff phase (material domain) and the Poisson's ratio `nu`, ν (see section 2.2). In addition, and as a specific parameter of the algorithm, the coefficient `m` is defined and prescribed to `m=5` for the minimum mean compliance problem. This coefficient in conjunction with the *contrast factor*, `alpha`, is used to compute the corresponding *relaxation factor*, `beta`. Notice that a noticeably small *contrast factor* can be imposed for compliance problem.

3.5 Animation preparation: lines 20-23

Lines 21-23 initialize the vectors `psi_vec`, `chi_vec` and `U_vec` to 0, which correspond respectively to the *discrimination function*, the *characteristic function* and the displacement vector. This vectors are used to store the corresponding variables at the convergence of each time-step (line 103), and are later called by the `Topology_evolution` GUI.

3.6 Finite element analysis preprocessing: lines 24-40

As already mentioned, the regularity in the mesh is highly exploited when computing the global stiffness matrix, `K`, to reduce the computational time inside the optimization loop. For that reason, only two element stiffness matrices are required, one for the mixed elements[6] and another for the other elements. The first one, is computed with a central quadrature point `posgp1` , while the second one requires at least 4 quadrature points to be correctly integrated, `posgp4`. The weights of each point are stored in `W1` and `W4`, respectively. This information is computed by evoking `gauss_points` function (lines 111-114) with the total number of point inside the quadrilateral element, as will be later explained.

Next, the nominal constitutive tensor `DE` for `E0` and `nu`, assuming plane-stress (equation (19)), is computed in line 27, by calling `D_matrix_stress` function. The element stiffness matrix, evaluated as

$$K_e = \int_{\Omega_e} \mathbf{B}^{\mathrm{T}} \, \overline{\mathbb{C}} \, \mathbf{B} \, d\Omega = \sum_{i=1}^{n_{gauss}} w_i |J_i| \mathbf{B}_i^{\mathrm{T}} \, \overline{\mathbb{C}}_i \, \mathbf{B}_i \, , \qquad (44)$$

is computed in lines 28-34 for solid and void elements, `KE`. The equivalent nominal matrix, for bisected elements, `KE_cut` is computed in lines 35-37. The strain-displacement matrix $\mathbf{B}$, defined in lines 119-124 (`B_matrix`), is evaluated in each gauss point along with the corresponding determinant of the Jacobian. Moreover, the product $\mathbf{B}^{\mathrm{T}} \overline{\mathbb{C}} \mathbf{B}$ for the i-th gauss point is stored in `KE_i` and `K_cut`, respectively.

Finally, the connectivity table of DOFs, `edofMat`, is generated in line 38 using built-in `kron` and `repmat` functions. Each row represents the degrees of freedom of a

[6] The elements bisected by the zero-level of the *discrimination function* are sub-integrated with a single quadrature point according a three-field ($\boldsymbol{\varepsilon}$-$\boldsymbol{\sigma}$-$\hat{\mathbf{u}}$) mixed element. Further details can be found in [22].

different element, e.g.

$$\text{edofMat} = \begin{bmatrix} 3 & 4 & 9 & 10 & 7 & 8 & 1 & 2 \\ 5 & 6 & 11 & 12 & 9 & 10 & 3 & 4 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 21 & 22 & 27 & 28 & 25 & 26 & 19 & 20 \\ 23 & 24 & 29 & 30 & 27 & 28 & 21 & 22 \end{bmatrix}. \tag{45}$$

This matrix is now used to compute the indices `iK` and `jK` used to generate the global stiffness matrix as a sparse matrix from the triplets `iK`, `jK` and `sK`, as will be explained later.

3.6.1 Gauss points, Shape function and Cartesian derivatives: lines 110-125

The bilinear quadrilateral element is used in the FE analysis, which consists of four nodes. Its numerical implementation can be found in the literature [44; 26].

This element is correctly integrated when 4 quadrature points are employed. The position and weights are computed in `gauss_points` function (lines 111-114), where the gauss quadrature points in one direction (parent dimension) are extended to two dimensions, using Matlab's `meshgrid` function. `posgp` defines the position $[\xi, \eta]$ in the parent square element, where each column represent a different point. The weight values are stored in `W` as a row vector.

The shape matrix, $\mathbf{N}$, (size `n_nodes x n_gauss`) is computed in lines 116-117 inside `N_matrix` function, as explained in [44; 26].

Last, the shape derivatives (size `n_dim x n_nodes`), the Jacobian matrix J (size `n_dim x n_dim`) and the Cartesian derivatives (size `n_dim x n_nodes`) are obtained in `B_matrix` for a given `gauss_points` (lines 119-124), assuming a square unit element. Finally, the strain-displacement matrix $\mathbf{B}$ for the case of interest is computed (size `3 x n_nodes*n_unkn`).

3.6.2 Element Stiffness matrix: lines 125-135

The constitutive tensor of each element depends on the material properties, which are common to all the elements, and the *characteristic function*. Due to this regularity, the nominal constitutive tensor, $\overline{\mathbb{C}}$, is computed only once for the stiff material properties, in lines 126 and 127, and the corresponding stiffness matrix is later multiplied by the term χ_β^m, which depends on each element. The *relaxed characteristic function* is calculated in lines 129-131, inside `interp_property` function, as follows

$$coeff = \begin{cases} (\chi + (1-\chi)\beta)^m & \text{for stiffness} \\ m\,(\chi + (1-\chi)\beta)^{m-1}\,(1-\beta) & \text{for sensitivity} \end{cases}$$

with $\chi \in [0,1]$.

The global stiffness matrix is assembled at each iteration inside the optimization loop using Matlab's `sparse` function to addition the components with same i-th (`iK`) and j-th (`jK`) degree of freedom, calling `assembly_stiff_mat`. Its definition is written in lines 133-135, where the third component (`sK`) for the `sparse` function is computed. Each column of the `sK` matrix corresponds to the stiffness matrix of element `e`. It is worth emphasizing that the bisected elements must be multiplied by `KE_cut`.

3.7 Laplacian regularization preparation: lines 41-52

Mimicking the preprocessing procedure of the global stiffness matrix (see section 3.6.2), the lhs matrix of equation (11) can be computed just once (lines 42-48), since it does not depend on the topology but on the mesh, which is regular. Thus, the terms $\nabla\mathbf{N}^{\mathrm{T}}\nabla\mathbf{N}$ and $\mathbf{N}^{\mathrm{T}}\mathbf{N}$, which correspond to `KE_Lap` and `ME_Lap` defined in lines 42 and 43, are analytically computed and defined as

$$KE_{Lap} = \int_\Omega \mathbf{B}^{\mathrm{T}}\,\mathbf{B}\,d\Omega \rightarrow KE_{Lap} = \frac{1}{6}\begin{bmatrix} 4 & -1 & -2 & -1 \\ -1 & 4 & -1 & -2 \\ -2 & -1 & 4 & -1 \\ -1 & -2 & -1 & 4 \end{bmatrix}$$

$$ME_{Lap} = \int_\Omega \mathbf{N}^{\mathrm{T}}\,\mathbf{N}\,d\Omega \rightarrow ME_{Lap} = \frac{1}{36}\begin{bmatrix} 4 & 2 & 1 & 2 \\ 2 & 4 & 2 & 1 \\ 1 & 2 & 4 & 2 \\ 2 & 1 & 2 & 4 \end{bmatrix}.$$

Next, combining both matrices and the *regularization parameter* τ, the lhs matrix is generated and saved in `KE_Lap` (line 44). Lines 45 to 47 define the triplets `i_KF`, `j_KF` and `s_KF`, which are then used to obtain the sparse matrix `K_Lap` in line 48.

Depending on `opt.solver_Lap`, the Laplacian regularization will be solved using a direct or an iterative method. This procedure can be sped up by computing the Cholesky factorization of the lhs (`chol(K_Lap, 'lower')`) if the direct method is chosen, or computing the incomplete Cholesky factorization (`ichol(K_Lap, opts)`, with `opts = struct('type','ict','droptol',1e-3, 'diagcomp',0.1)`) in case an iterative algorithm is desired. It will be later used as a preconditioner.

The rhs must be computed at each iteration, since it depends on the *discrimination function*, `psi`, as detailed in section 3.8.3. Nevertheless, the resolution procedure of equation (11) can be prepared by computing both the shape matrix, `N_T`, of size `n_nodes x n_gauss` [7], and the indexes of the element nodes `i_xi` (reshaping the

[7]The matrix is transposed with respect to the common one.

connectivity matrix into a column vector). The assembly is carried out in lines 83 and 85 evoking `accumarray` function.

3.8 Main program: lines 53-107

The main optimization procedure starts by initializing the topology via the *discrimination function* to `alpha0` , constant to all the nodes, except for those listed in `passive_node`. Next, the *characteristic function* is obtained via `compute_volume` function. Line 57 is used to initialize several vectors, which will accumulate the convergence variables (cost function, volume and lambda), and other essential variables. The initial topology is displayed in the next line by means of `plot_isosurface`.

The optimization starts in line 59, where the loop over time-steps is defined. As explained in section 2.4, the *reference pseudo-time* is iteratively increased following a linear or exponential expression, which definition is written through lines 186-188, and for each time-step the optimization loop is repeated until convergence is achieved. The optimization loop (lines 65-99) consists of five parts: finite element analysis, sensitivity computation, Laplacian regularization, topology update and convergence check.

Finally, at each iteration, the topology is plotted (line 92), the intermediate results are printed in display (line 96) and the iteration counters are increased (line 97).

3.8.1 Finite element code: lines 66-69

The global stiffness matrix, `K`, is assembled inside `assembly_stiff_mat` function using the `sparse` function, where `sK` is computed considering the corresponding *relaxed characteristic function* for each element. Next, in line 68, the equilibrium equation (22) is solved using a direct method. The displacements are stored in `U`. Next, the cost function, `J`, normalized with the one of the first iteration (`J_ref`), can be obtained at the current topology layout.

3.8.2 Sensitivity computation: lines 70-79

According to equation (9), the *energy density* is defined as the partial derivative of the cost objective's kernel multiplied by the exchange function, $\Delta\chi$. The *energy density* is computed in two parts, in the first one (lines 72 to 75) the sensitivity of non-bisected elements is obtained for the 4 quadrature points, while the sensitivity for the mixed elements is calculated in the second part (lines 76 to 78).

The element sensitivity, as detailed in section 2.5 for the minimum compliance problem, is computed as $m\chi_\beta^{m-1}\,\mathbf{u}_e^{\mathrm{T}} K_{e,i}\mathbf{u}_e\,(1-\beta)$ for the e-th element and the i-th gauss point (see equation (30)). However, for the bisected elements, the element stiffness matrix $K_{e,i}$ is replaced by `K_cut`, and the resultant value is copied to the four gauss points.

At the first iteration, the parameters `xi_shift` and `xi_norm` are defined as

```
xi_shift = min(0,min(Energy(:)))
xi_norm = max([range(Energy(:));Energy(:)])
```

and will be used to obtain the *modified energy density*, $\hat{\xi}(\mathbf{x})$, described in equation (10).

3.8.3 Laplacian regularization: lines 80-86

As aforementioned, instead of applying the *Laplacian regularization* (11) to the resultant *discrimination function*, at each iteration of the *bisection algorithm* and since it does not affect constant fields such as λ, the *Laplacian regularization* is only implemented for $\hat{\xi}$. The corresponding system is defined as

$$\begin{cases} \xi_\tau - (\tau h_e)^2 \Delta_{\mathbf{x}}\xi_\tau = \hat{\xi} & \textit{in } \Omega \\ \nabla_{\mathbf{x}}\xi_\tau \cdot \mathbf{n} = 0 & \textit{on } \partial\Omega \end{cases}, \tag{46}$$

where $\hat{\xi}$ and ξ_τ stand for the *modified unfiltered energy density* and the *smooth energy density*, respectively. As commented in section 2.1, the lhs has been precomputed (see section 3.7) and the rhs is now computed based on the *modified energy density* (field on gauss points). FE discretization of the rhs leads to

$$rhs = \int_\Omega \mathbf{N}^{\mathrm{T}}\hat{\xi}(\mathbf{x})\,d\Omega\,, \tag{47}$$

which can be rewritten in matrix form, defined in line 81, as

```
xi_int = N_T*(Energy-xi_shift*chi)/xi_norm;
```

The nodal contribution of this integral is later constructed by means of the built-in `accumarray` Matlab function.

The system of linear equations (46), as mentioned, can be solved using the Cholesky factorization and a direct solver (line 83) or an iterative solver (e.g. `minres` solver) applying the incomplete Cholesky factorization as the preconditioner of the system, as described in section 3.7.

It is worth to mention that for low number of elements, as it is the case of this paper since it is for academic purposes, the *Laplacian regularization* may generate boundary waves for thin filaments, as displayed in some figures. This undesirable effect should vanish if finer meshes are used.

3.8.4 Update of χ and ψ: line 88

The topology layout, satisfying the constraint equation, is obtained by means of a *bisection algorithm* (solution of equation (13)) called in line 88. The `find_volume` functions computes the Lagrange multiplier λ (`lambda`), the new *discrimination function* ψ (`psi`) and the corresponding *characteristic function* χ (`chi`).

Bisection algorithm: lines 137-152 The *bisection algorithm* consists of a search for a suitable bracket, and the subsequent root finding. The left and right extremes of the interval are easily defined by the minimum and maximum value of the energy density field, and stored as `l1` and `l2`, respectively. The corresponding constraint values are saved as `c1` and `c2`. Lines 139-141 tests the last λ as a trial extreme of the interval, by means of `compute_volume_lambda`, to reduce the number of iterations. The bisection loop is written in lines 142 to 146, where the root of the constraint equation is estimated as the midpoint of the bracketing interval (line 143).

At each iteration of the bisection, given a *density function* `xi` and a trial `lambda`, the *discrimination function* is obtained at line 149. The active and passive nodes are considered by modifying the `psi` function, as aforementioned. The void volume ratio `vol` and the *characteristic function* `chi` are obtained from `compute_volume` in line 150. Next, the constraint equation is evaluated in line 151, and the extremes of the interval are updated accordingly. This procedure is repeated until the void volume is within 10^{-4} of the reference time.

Volume computation: lines 153-161 The computation of the volume is done by means of an integration with 36 quadrature points. This methodology differs for simplicity of the implementation from the one used in Oliver et al. [22], where a modified marching squares was employed.

The position and weights of the 36 quadrature points are assigned as defined in [26]. First, line 158 determines which elements are bisected by the internal boundary through the nodal value of the *discrimination function*. In case they all have the same sign, the boundary will not cross throughout the element. Then, the element nodal ψ, `psi_n`, is evaluated in the quadrature points for the bisected elements and saved as `psi_x`. The *characteristic function* is obtained as the dot product of `W` and `phi_x>0`. Finally, the void volume ratio is computed in line 161.

3.8.5 Convergence check: lines 90 and 94

Lines 90 and 94 compute the convergence tolerances of the algorithm, along with the constraint tolerance `Tol_constr`. The convergence is checked inside the while condition at line 65, and it only converges when the number of in-step iterations (`iter_step`) is in between `iter_min_step` and `iter_max_step`, and the three following conditions are satisfied: the L2-norm of the *characteristic function* is less than 0.1, the relative difference of the Lagrange multiplier with respect the previous one is less than 0.1, and the volume magnitude is within 10^{-4} of the desired *pseudo-time*, `t_ref`.

The optimization terminates if the maximum number of iterations, `iter_max`, or the maximum number of in-step iterations are achieved, showing a warning message in command window.

3.9 Iso-surface plot: lines 92 (162-172)

The `plot_isosurface` function shows the optimal topology via the *discrimination function* `psi` in a black-and-white design, as seen in Figure 5 (top view). The behavior of this function depends on the iteration, i.e. the first time it is called, a figure is generated and its handle saved as `fig_handle`. In addition, the topology is represented using built-in `patch` function, the handle of which is stored as `obj_handle`, by means of the coordinates matrix, connectivity matrix and the nodal *discrimination function*. However, only `psi` field is updated in the other iterations using `set(obj_handle,'FaceVertexCData',psi)
;`.

3.10 Cost function and volume vs. step plot: line 102 (173-184)

The definition of `plot_volume_iter` function is similar to that of `plot_isosurface` function. At iteration 1, it creates the figure, and two axes using `subplot` function. The cost function evolution is illustrated in the top subplot, while the volume evolution is displayed at the bottom. At other iterations, the lines are updated using `set` function, with the updated `J_vec` and `vol_vec` vectors, respectively.

3.11 Topology evolution GUI: lines 108-109

Once the Topology Optimization problem has been solved, the results can be graphically post-processed by means of a graphical user interface (GUI), where the topology and displacement fields are displayed for the set of time-steps. It is created by the following function call:

```
Topology_evolution(coord,connect,[Vol0,
    vol_vec],psi_vec,chi_vec,U_vec);
```

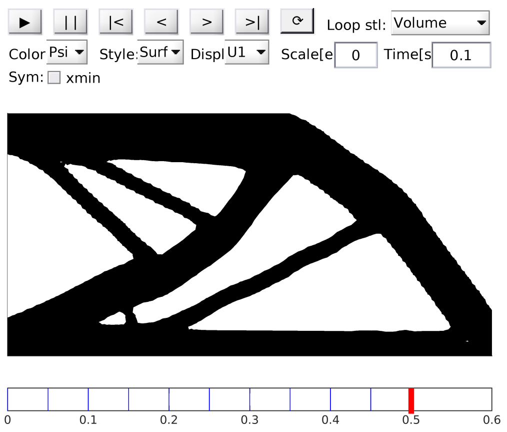


Fig. 7 GUI's design.

where `vol_vec` corresponds to the set of *pseudo-time* values for which the topology has been optimized. Then, `psi_vec` and `chi_vec` correspond respectively to the *discrimination function* (nodal scalar field) and the *characteristic function* (element scalar field), each column corresponding to a different time-step. Similarly, `U_vec` correspond to the displacement field, where each column and layer of the array represent a different loading condition and a different time-step, respectively.

The interface allows to select the field to display (psi, chi or the norm of the displacement for any load condition) and the style of the representation (surface only, wireframe only and surface plus wireframe). The user can also choose the scale factor and the displacement field to deform the mesh as it can be observed in Figure 7.

The set of push-buttons on the top-left area controls the animation of the topology along the *pseudo-time*, the time between time-steps can be modified in the *dt* text edit field. The last button corresponds to a toggle-button, which animates indefinitely the topology until it is clicked. Depending on the chosen loop style option, the topology is animated along the time-steps (Volume) or along the scale factor for a given time-step (Scale linear and Scale sine).

The possibility to mirror/symmetrize the topology is the last relevant feature of this figure. A set of check-boxes allow to symmetrize the mesh and its properties on any of the sides of the domain.

3.12 Multi-load mean compliance: code modification

According to section 2.6, the program can be easily adapted to optimize multi-load problems, as shown in Figure 8. Then, the cost function as well as the sensitivity are evaluated as weighted averages of each individual optimization problem.

First, the loads and boundary conditions are changed to include the second loading state[8], defined in the second column of `F`:

```
11   F = sparse(n_unkn*n,2);
12   U = zeros(n_unkn*n,2);
13-1 F(n_unkn*find(coord(:,2)==nely & coord(:,1)
         ==nelx),1) = 0.01*nelx;
13-2 F(n_unkn*find(coord(:,2)==0 & coord(:,1)==
         nelx),2) = -0.01*nelx;
```

Furthermore, an additional column is added to `U_vec` by replacing line 23 with

```
23   U_vec = zeros(n_unkn*size(coord,1),size(U,2)
         ,nsteps+1);
```

Next, the sensitivity computation must be adapted to include multiple loading states, via a `for` loop. Then, lines 73-75 are substituted with

```
73-1  for i_load=1:size(F,2)
73-2   u_e = reshape(U(edofMat(id,:)',i_load),
        n_nodes*n_unkn,[]); w_e = u_e;
74     for i=1:n_gauss; Energy(i,id) = Energy(i,
        id) + sum(w_e.*(KE_i(:,:,i)*u_e),1); end
75    end; Energy(:,id) = int_chi.*Energy(:,id);
```

[8] More than one additional loading state can be considered.

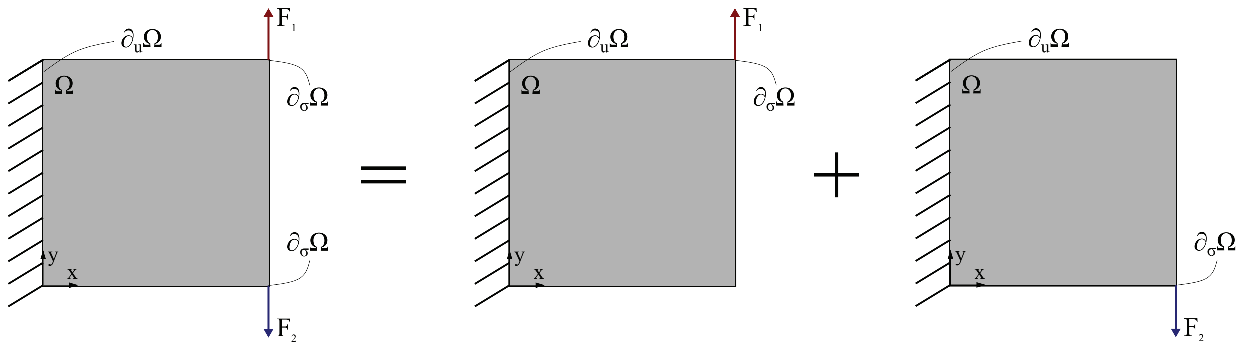


Fig. 8 Multi-load beam: topology optimization domain and boundary conditions.

and equivalently, lines 77 and 78 are replaced by

```
77-1  for i_load=1:size(F,2)
77-2   u_e = reshape(U(edofMat(id,:)',i_load),
        n_nodes*n_unkn,[]); w_e = u_e;
78-1   Energy(:,id) = Energy(:,id) + repmat(sum(
        w_e.*(K_cut*u_e),1),n_gauss,1);
78-2  end; Energy(:,id) = int_chi.*Energy(:,id);
```

This example can be simulated by the following line

```
UNVARTOP_2D_multiload
 (50,50,11,0,0.55,0,0.5)
```

The resultant optimal topology, at $t_{ref} = 0.55$, is displayed in Figure 9, while the topology evolution is shown in Online Resource 2. It can be observed in Figure 10 how much the topology differs from the single loading condition, when the two loads of Figure 8 are applied at the same time.

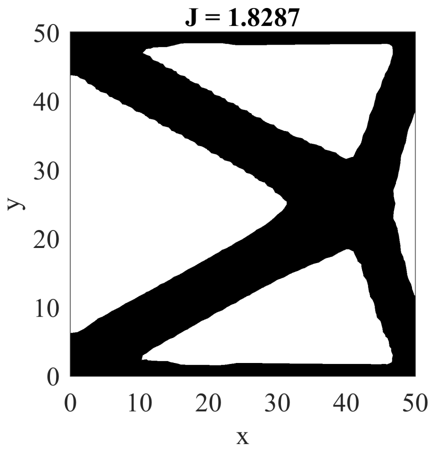


Fig. 9 Multi-load beam: optimal topology layout.

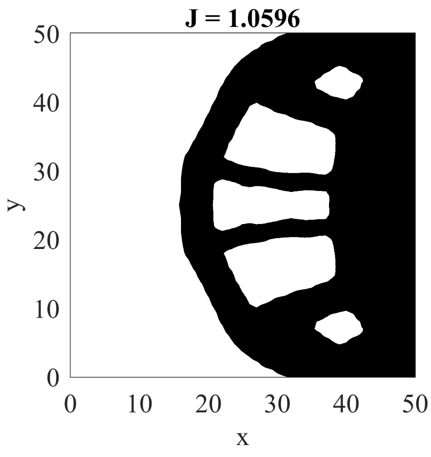


Fig. 10 Multi-load beam: optimal topology layout when loads are applied at the same time.

3.13 Compliant mechanisms: code modification

Mimicking the previous section, the base code in Appendix A also requires some modifications in order to optimize compliant mechanisms, as depicted in Figure 11. A second loading state must be solved to compute the *adjoint state* $\mathbf{w}$, which is later used in the sensitivity computation. This second state is loaded with a dummy constant load applied in the *output nodes* in the same direction as the desired displacement. Then, the loads and boundary conditions are modified to

```
11   F = sparse(n_unkn*n,2);
12   U = zeros(n_unkn*n,2);
13-1 F(n_unkn*find(coord(:,2)>=0.9*nely & coord
        (:,1)==0)-1,1) =    0.0001*nelx;
13-2 F(n_unkn*find(coord(:,2)>=0.9*nely & coord
        (:,1)==nelx)-1,2) = -0.0001*nelx;
14-1 fixed_dofs = [reshape(n_unkn*find(coord(:,2)
        ==nely),1,[]),...
14-2  reshape(n_unkn*find(coord(:,1)==0 & coord
        (:,2)<=0.1*nely)+(-n_unkn+1:0),1,[])];
15-1 active_node = find(coord(:,2)>0.9*nely&(
        coord(:,1)<0.05*nelx|coord(:,1)>0.95*
        nelx));
15-2 passive_node = [];
```

Notice that the force is applied along a segment, and not only in a single node. Furthermore, only half of the design is computed thanks to the symmetry of the design and some nodes surrounding the *input* and *output ports* are forced to remain as stiff material.

The properties of the material (line 19) should be also changed to `m=3` and `alpha=1e-2`. This adjustment increases convergence.

To ensure fast convergence, external springs must be included in the *input* and *output ports* at the same degrees of freedom as the applied forces. These degrees are obtained by means of the following lines:

```
id_in  = find(F(:,1)); id_in  = sub2ind(
    n_unkn*(nely+1)*(nelx+1)*[1 1],id_in,
    id_in);
id_out = find(F(:,2)); id_out = sub2ind(
    n_unkn*(nely+1)*(nelx+1)*[1 1],id_out,
    id_out);
```

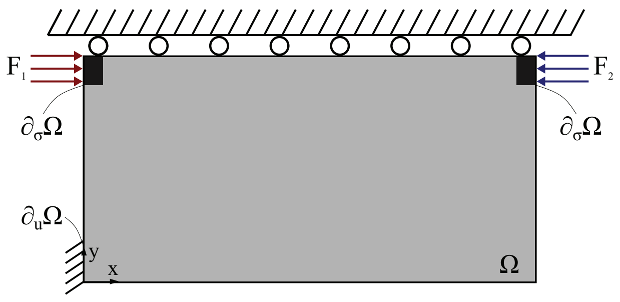


Fig. 11 Inverter (compliant mechanism): topology optimization domain and boundary conditions.

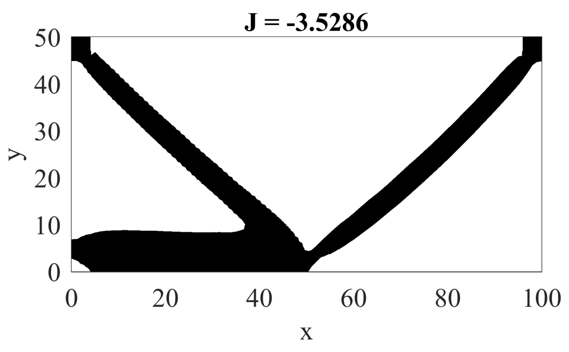


Fig. 12 Inverter (compliant mechanism): optimal topology layout.

which must be inserted between lines 17 and 18. These two lists are used inside `assembly_stiff_mat`, thus its call has to be replaced by

```
67   [K] = assmebly_stiff_mat (chi,KE,KE_cut,
       beta,m,iK,jK,n_unkn,nelx,nely,id_in,
       id_out);
```

as well as its definition at line 133

```
133 function [K] = assmebly_stiff_mat (chi,KE,
       KE_cut,beta,m,iK,jK,n_unkn,nelx,nely,
       id_in,id_out)
```

The external springs, using `id_in` and `id_out`, are added to the global stiffness matrix after line 135:

```
K(id_in)  = K(id_in)  + 0.002;
K(id_out) = K(id_out) + 0.002;
```

The prescribed value for the springs must be adjusted for each individual example.

The cost function must be also replaced by the corresponding work at the *output port*, since the cost function is defined as the maximization of the output displacement. It is implemented by the following line:

```
69   if iter == 1; U_vec(:,:,1)=U; J_ref = -abs
       (F(:,2)'*U(:,1)); end; J = F(:,2)'*U
       (:,1)/J_ref;
```

As in section 3.12, `U_vec` must be substituted by

```
23 U_vec = zeros(n_unkn*size(coord,1),2,nsteps
       +1);
```

Finally, the displacements of the *adjoint system*, used in the calculation of the sensitivity, must be replaced by the corresponding displacements of the second system. Thus, these lines are now defined as

```
73-1   u_e = reshape(U(edofMat(id,:)',1),n_nodes*
         n_unkn,[]);
73-2   w_e = -reshape(U(edofMat(id,:)',2),n_nodes
         *n_unkn,[]);
```

and

```
77-1   u_e = reshape(U(edofMat(id,:)',1),n_nodes*
         n_unkn,[]);
77-2   w_e = -reshape(U(edofMat(id,:)',2),n_nodes
         *n_unkn,[]);
```

The optimal topology, for the given boundary conditions, illustrated in Figure 12 can be performed with

```
UNVARTOP_2D_complmechanism
 (100,50,10,0,0.8,-2,0.5)
```

The resultant compliant mechanism is animated in Online Resource 3.

4 Numerical examples

The following numerical examples exhibit the potential of the unsmooth variational topology optimization technique in 2D problems. Unless otherwise stated, the parameters and material properties are left as the default examples, for each of the three optimization problems described in this work. The design domain, the function call and the boundary conditions for each example are defined in Table 2.

4.1 Cantilever beam

A variation of the initial examples is now performed. In this case, the load is not applied at the bottom-right corner but in the middle of the right side of the domain, as depicted in the first row of Table 2. Dirichlet conditions are not modified, i.e. the displacements are prescribed on the left boundary of the domain. The optimal topology layout, for the last time-step, is illustrated in Figure 13, with the values from Table 2. That is, the interval of interest $[0, 0.65]$ is discretized with 12

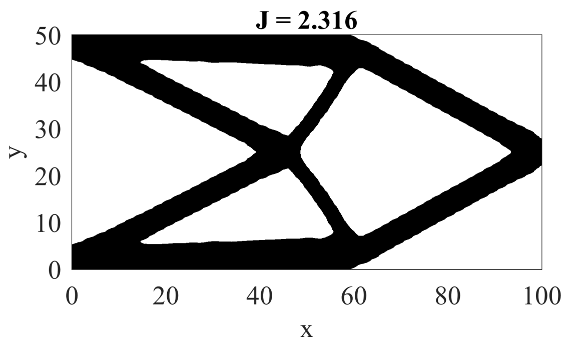
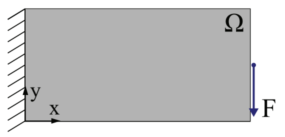
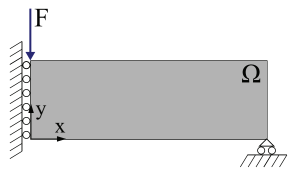
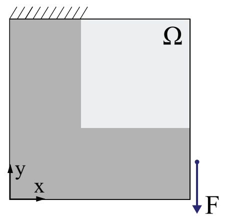
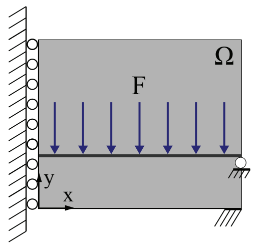
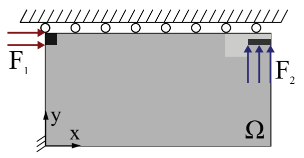
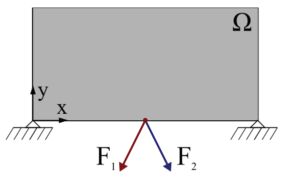


Fig. 13 Cantilever beam (load applied at the middle): optimal topology layout.

Table 2 List of examples.

Domain	Matlab's call	Boundary conditions
	`UNVARTOP_2D_compliance(100, 50, 12, 0, 0.65, 0, 0.5)`	`F(n_unkn*find(coord(:,2)==round(0.5*nely) & coord(:,1)==nelx),1) = -0.01*nelx; fixed_dofs = reshape(n_unkn*find(coord(:,1)==0)+(-n_unkn+1:0),1,[]); active_node = []; passive_node = [];`
	`UNVARTOP_2D_compliance(150, 50, 10, 0, 0.6, 0, 1)`	`F(n_unkn*find(coord(:,2)==nely & coord(:,1)==0),1) = -0.01*nelx; fixed_dofs = [reshape(n_unkn*find(coord(:,1)==0)-1,1,[]),... reshape(n_unkn*find(coord(:,1)==nelx & coord(:,2)==0),1,[])]; active_node = []; passive_node = [];`
	`UNVARTOP_2D_compliance(100, 100, 12, 0.36, 0.75, 0, 0.5)`	`F(n_unkn*find(coord(:,2)==round(0.2*nely) & coord(:,1)==nelx),1) = -0.01*nelx; fixed_dofs = reshape(n_unkn*find(coord(:,1)<=0.4*nelx & coord(:,2)==nely)+(-n_unkn+1:0),1,[]); active_node = []; passive_node = find(coord(:,1)>ceil(nelx*0.4) & coord(:,2)>ceil(nely*0.4));`
	`UNVARTOP_2D_compliance(240, 200, 32, 0, 0.775, 0, 0.5)`	`F(n_unkn*find(coord(:,2)==floor(nely*1.6/5)),1) = -0.01*nelx; fixed_dofs = [reshape(n_unkn*find(coord(:,1)==0)-1,1,[]),... reshape(n_unkn*find(coord(:,1)>=5.75/6*nelx & coord(:,2)==0)+(-n_unkn+1:0),1,[]),... reshape(n_unkn*find(coord(:,1)==nelx & coord(:,2)==floor(nely*1.5/5)),1,[])]; active_node = find(coord(:,2)>=nely*1.5/5 & coord(:,2)<=nely*1.6/5); passive_node = [];`
	`UNVARTOP_2D_complmechanism(150, 75, 14, 0, 0.85, -2, 0.5)`	`F(n_unkn*find(coord(:,2)>=0.9*nely & coord(:,1)==0)-1,1) = 0.0001*nelx; F(n_unkn*find(coord(:,2)==round(0.9*nely) & coord(:,1)>=0.9*nelx),2) = 0.0001*nelx; fixed_dofs = [reshape(n_unkn*find(coord(:,2)==nely),1,[]), reshape(n_unkn*find(coord(:,1)==0 & coord(:,2)<=0.1*nely)+(-n_unkn+1:0),1,[])]; active_node = [find(coord(:,2)>0.9*nely&coord(:,1)<0.05*nelx); find(coord(:,2)>0.9*nely&coord(:,2)<=0.95*nely&coord(:,1)>=0.9*nelx)]; passive_node = [find(coord(:,1)>0.8*nelx & coord(:,1)<0.9*nelx & coord(:,2)>0.8*nely); find(coord(:,1)>=0.9*nelx & coord(:,2)>0.95*nely)];`

Domain	Matlab's call	Boundary conditions

	`UNVARTOP_2D_multiload (200, 100, 24, 0, 0.6, 0, 0.5)`	`F(n_unkn*find(coord(:,2)==0 & coord(:,1)== round(nelx/2))-1,1) = -0.01*nelx;`
		`F(n_unkn*find(coord(:,2)==0 & coord(:,1)== round(nelx/2)),1) = -2*0.01*nelx;`
		`F(n_unkn*find(coord(:,2)==0 & coord(:,1)== round(nelx/2))-1,2) = 0.01*nelx;`
		`F(n_unkn*find(coord(:,2)==0 & coord(:,1)== round(nelx/2)),2) = -2*0.01*nelx;`
		`fixed_dofs = reshape(n_unkn*find((coord (:,1)==0 & coord(:,2)==0) | (coord (:,1)==nelx & coord(:,2)==0))+(-n_unkn +1:0),1,[]);`
		`active_node = []; passive_node = [];`

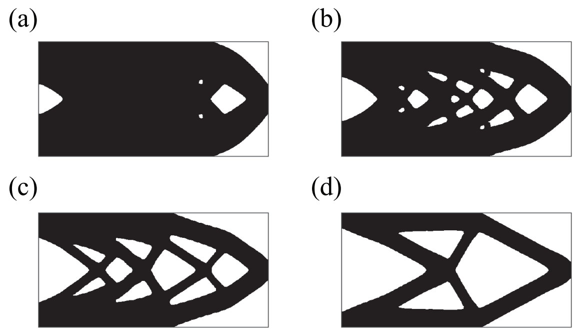


Fig. 14 Cantilever beam: topology evolution. (a) optimal topology at $t_{ref} = 0.11$, (b) optimal topology at $t_{ref} = 0.22$, (c) optimal topology at $t_{ref} = 0.38$ and (d) optimal topology at $t_{ref} = 0.54$.

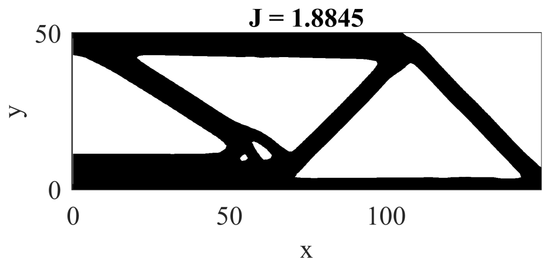


Fig. 15 MBB beam: optimal topology layout.

equally spaced steps and the regularization parameter is prescribed to $\tau = 0.5$. In addition, the topology evolution, shown in the animation (Online Resource 4), is displayed in Figure 14 for time-steps 2, 4, 7, and 10. Similar results obtained with other optimization techniques can be found in [30; 8; 43; 11].

4.2 Messerschmitt-Bölkow-Blohm (MBB) beam

Half of the MBB-beam, with an aspect ratio of 3:1, is optimized in the second example. Symmetry is assumed on the left side of the domain and the vertical displacement at the bottom-right corner is constrained, as observed in Table 2, and only a point-wise load is applied at the top-left corner. The last requested $t_{ref} = 0.6$ is achieved in 10 time steps. Figure 15 depicts the resulting optimal topology layout using the provided code, adapted with the corresponding boundary conditions (see second row of Table 2). Additionally, Online Resource 5 displays the animation of optimal topologies for the given time-steps. The results are comparable to those presented by [27; 5; 32; 43; 11], among others.

4.3 L-Shape structure

The L-Shape structure, shown in Table 2, represents a simplified version of a hook. The domain has a prescribed void zone in the top right area, defined by $x_i \geq 0.4$ and $y_i \geq 0.4$. All the nodes contained in this area are listed in `passive_node`. A single vertical load is applied on the right side of the domain at $y = 0.2$.[9] The nodes on the top-left boundary ($y = 1$ and $x < 0.4$) are fixed. The optimal configuration, shown in Figure 16, is obtained by the inputs described in Table 2. As in previous examples, the topology evolution is animated in Online Resource 6. Similar optimal designs are obtained by Biyikli and To [8] and Liu and Tovar [18], for 2D and 3D problems, respectively.

[9] All measures are relative to the dimensions of the domain.

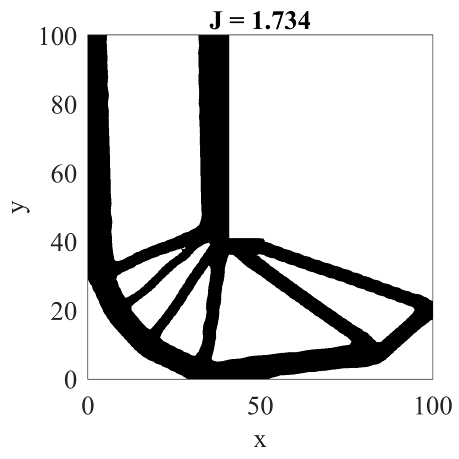


Fig. 16 L-shape structure: optimal topology layout.

4.4 Bridge

The fourth numerical example in Table 2 corresponds to a bridge, which domain is given by 12x5 rectangle. However, only half of it is optimized thanks to the central symmetry. Then, the horizontal displacement on the left side of the domain is prescribed to 0. In addition, the domain is supported by a small segment on the bottom-right corner of it and the vertical displacement is prescribed at the right side of the road. A distributed vertical downside load is applied on the road, which does not change throughout the optimization procedure (i.e. it can not be removed since all its nodes are included in `active_node` list). The corresponding boundary conditions of this problem are listed in Table 2.

The optimal topology, at $t_{ref} = 0.775$, is displayed in Figure 17, along with the corresponding animation in Online Resource 7. The topology in Figure 17 is closely similar to that obtained by Feijoo et al. [13] and Liang and Steven [17]. Furthermore, the design can be compared with the solution of a multi-load problem done by [19].

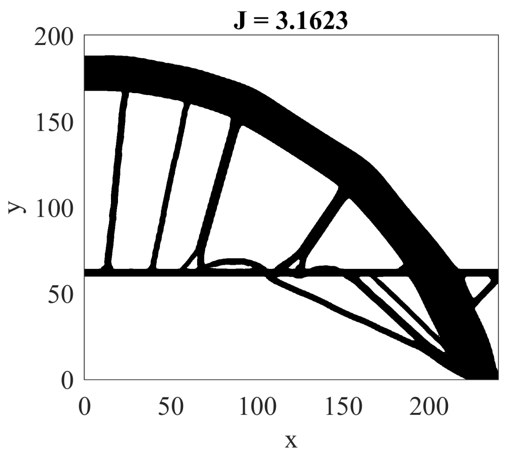


Fig. 17 Bridge: optimal topology layout.

4.5 Gripper mechanism

Let us now consider a compliant mechanism different from the one explained in section 3.13 and inspired by [22; 41]. The goal of this optimization is to maximize the compressive displacement at the *output port* (vertical displacement at the top-right side) when an horizontal force is applied at the *input port* (top-left side of the domain). The domain is supported by a small area in the bottom-left corner and symmetry is applied on the top side of Ω, as it can be observed in Table 2 (fifth row). A small area in the *output port* is set to soft material (i.e. included in `passive_node` list) in order to represent the gap in the jaws of the gripper. Furthermore, some stiff material areas are restricted in both ports, and the

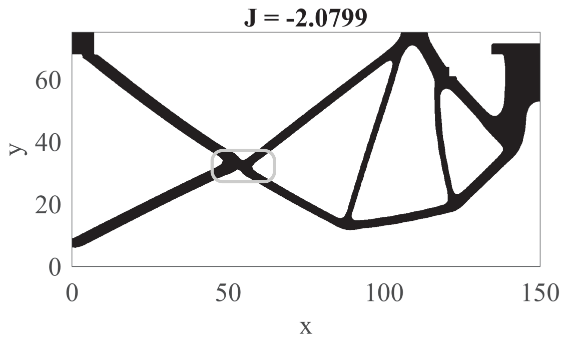


Fig. 18 Gripper (compliant mechanism): optimal topology layout. The central hinge is highlighted with a gray square.

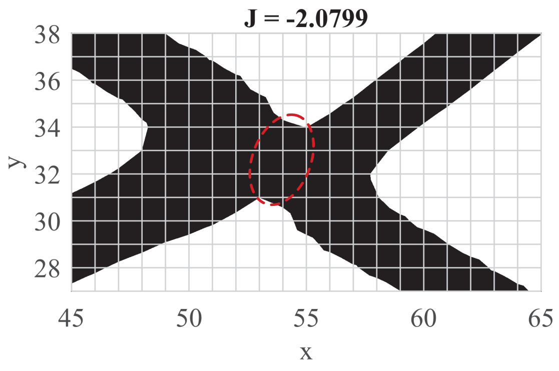


Fig. 19 Gripper (compliant mechanism): close-up view of the central hinge.

corresponding spring stiffness values must be replaced by 0.01.

The *pseudo-time* is updated following an exponential expression in 14 time steps. The given optimal topology of Figure 18 is obtained evoking `UNVARTOP_2D_complmechanism` function with the appropriate boundary conditions. A close-up view of the central hinge is illustrated in Figure 19, where the flexible (thin) material, circled in red, performs as a hinge. The compliant mechanism of Figure 18 is animated in Online Resource 8, where the displacements are updated following a sinus function.

4.6 Michell multi-load structure

The last numerical example corresponds to a multi-load mean compliance problem with two loading states. The 2x1 rectangular domain is supported by its two bottom corners and subjected to a pair of forces in the middle

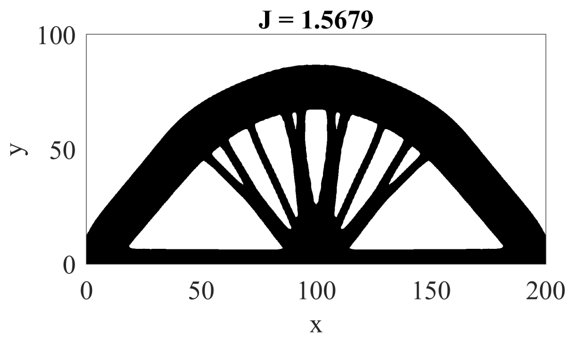


Fig. 20 Multi-load michell structure: optimal topology layout.

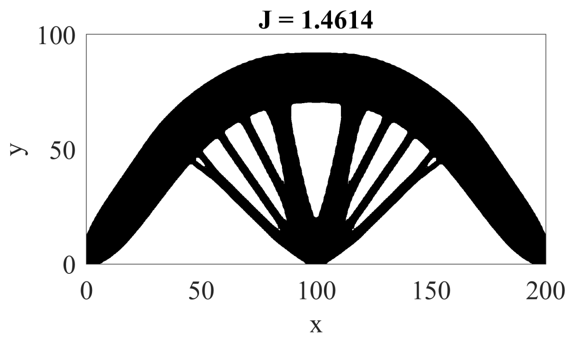


Fig. 21 Multi-load michell structure: optimal topology layout when loads are applied at the same time.

of the bottom side at an angle of 30° with respect to the vertical. The desired *pseudo-time* is prescribed to 0.6, which optimal topology is illustrated in Figure 20. The topology animation is given in Online Resource 9.

The topology layout, as already noted, deviates from the corresponding optimal topology when both loads are applied at the same time, as shown in Figure 21. In this setting, the bottom bars, which connect the supporting nodes with the central node, have been removed. The problem definition is based on [19]. Other variations can be found in [10; 32].

5 Extensions

5.1 Bisection algorithm

The bisection algorithm of the *cutting&bisection* algorithm, which estimates the solution as the midpoint of the bracketing interval (see section 3.8.4), can be easily improved by introducing either a *regula falsi method* [9] or a more sophisticated method, like the *Anderson-Björk with Illinois algorithm* [4]. These two mathematical techniques reduce the number of iterations required to find the root of the *constraint equation* ((25)-b), $\mathcal{C}$.

5.1.1 Regula falsi

In order to compute the test `lambda`[10] through the *regula falsi* approximation inside the *bisection algorithm* [9], line 143 must be replaced by

```
143 lambda = l1 - c1*(l2-l1)/(c2-c1);
```

where `l1` and `l2` stand for the left and right λ brackets, while `c1` and `c2` are respectively the corresponding constraint values. The linear interpolation with the endpoints of the bracketing interval is used to find the value of the root, i.e. the root is approximated as the intersecting point between the line joining the extremes and the x-axis. Next, the subinterval is updated by checking the sign of the constraint equation at `lambda`, as mentioned in section 3.8.4, until the tolerance is attained.

5.1.2 Anderson-Björck with Illinois algorithm

The *regula falsi* method usually converges faster than the the regular *bisection algorithm*. However, for some specific situations, it can show slower convergence. To avoid these numerical instabilities, the *Anderson-Björck algorithm with an Illinois algorithm* [4] is implemented. [11] Lines 143 and 152 are changed to:

[10]The *Lagrange multiplier* is denoted as `lambda` in the code.

[11]On one hand, the *Illinois* method [12] seeks to eliminate the ill-condition generated by permanently retaining one of the

```
143 lambda = l1 - c1*(l2-l1)/(c2-c1);
```

and

```
152-1 if c2*Tol_constr<=0; l1=l2; c1=c2; l2=
        lambda; c2=Tol_constr;
152-2 else; g=1-Tol_constr/c2; g=(g-0.5)*(g>0)
        +0.5; l2=lambda; c1=g*c1; c2=Tol_constr;
         end
```

With these changes, the number of iterations and the computational cost/time of the optimization procedure is reduced.

5.2 Plane-strain assumption

The corresponding constitutive tensor, $\overline{\mathbb{C}}$, for the plane-strain assumption

$$\overline{\mathbb{C}}^{Pstrain} = \frac{E}{(1-\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \dfrac{1-2\nu}{2} \end{bmatrix} \tag{48}$$

can be easily used by replacing the definition of the constitutive tensor of the plane-stress assumption, see section 2.2, in lines 126-127 with the following

```
126 function [DE] = D_matrix_strain(E,nu) %
        Planestrain
127 DE = E/((1+nu)*(1-2*nu))*[(1-nu) nu 0;nu (1-
        nu) 0;0 0 (1-2*nu)/2];
```

Line 27 must be also modified, to call the `D_matrix_strain` function, to

```
27 [DE] = D_matrix_stress(E0,nu);
```

5.3 Augmented Lagrangian to impose volume constraint

The constraint equation (25)-b, $\mathcal{C}$, can be also imposed through an Augmented Lagrangian method [20], which updates the lagrangian multiplier according the following definition

$$\lambda_{i+1} = \lambda_i + \rho \mathcal{C}_i \,, \tag{49}$$

to prescribed an equality constraint. The penalty value, ρ, can be either set to a constant value or increased along iterations, which improves convergence rate. Then, the penalty coefficient is updated as

$$\rho_{i+1} = \begin{cases} \min\left(1.02\rho_i, 100\rho_0\right) & \text{for} \quad |\mathcal{C}_{i+1} - \mathcal{C}_i| < 10^{-3} \\ \rho_i & \text{otherwise} \,, \end{cases} \tag{50}$$

where ρ_0 corresponds to the initial penalty value and i represents the i-th iteration. The values 1.02 and 100 can be modified at the user's discretion, and will highly depend on each specific numerical example. In this implementation, the Lagrangian equation (8) is defined as

$$\mathcal{L} = J + \lambda \mathcal{C} + \frac{1}{2}\rho \mathcal{C}^2 \,. \tag{51}$$

In order to impose the constraint with this methodology, a few changes need to be made to the original code of Appendix A. First, the initialization of constraint vector and the penalty value must be initialized by inserting

```
Tol_constr_vec = []; rho = rho0;
```

between lines 57 and 58, and the constraint must be computed before starting the optimization loop, just below line 64:

```
 Tol_constr = t_ref - vol;
```

and stored in the corresponding vector after line 95

```
  Tol_constr_vec = [Tol_constr_vec,abs(
    Tol_constr)];
```

The convergence criteria of line 65 must be also changed to include the constraint equation as an extra convergence condition by introducing `abs(Tol_constr)>1e-3`.

Second, the *bisection algorithm* (lines 137-146) and its function call in the optimization loop (line 88) must be replaced with the corresponding updating of λ and ρ (equations (49) and (50)), defined as

```
function [lambda,rho,chi,psi,vol,Tol_constr]
     = find_volume (iter,xi,connect,
    active_node,passive_node,t_ref,lambda,
    rho,rho0,alpha0,Tol_constr,
    Tol_constr_vec)
lambda = lambda + rho * Tol_constr;
psi = xi - lambda; psi(passive_node) = -
    alpha0; psi(active_node) = alpha0;
[vol,chi] = compute_volume (psi,connect);
Tol_constr = -(vol-t_ref);
if iter>=3; rho = min(0.02*rho*(abs(diff(
    Tol_constr_vec(end-1:end)))<1e-3) + rho
    ,100*rho0); end
```

and

```
88  [lambda,rho,chi_n,psi,vol,Tol_constr] =
      find_volume (iter,xi,connect,active_node
      ,passive_node,t_ref,lambda,rho,rho0,
      alpha0,Tol_constr,Tol_constr_vec);
```

end-points (always set to the left bracket in the code). This issue is fixed by multiplying the retained extreme point by $g = 0.5$. On the other hand, *Anderson-Björck algorithm* improves the *regula falsi approach* by combining linear interpolation (when the left bracket should be replaced) with parabolic interpolation (when the right bracket should be replaced). Furthermore, it includes an *Illinois-scheme* with $g = 1 - \frac{\mathcal{C}(\lambda)}{\mathcal{C}(\lambda_2)}$, when g is positive, or 0.5, otherwise.

Last, the cost function must be computed according equation (51), which takes into account the constraint equation. The additional terms are summed in one extra line under line 69:

```
J = J + nelx*nely*(lambda*Tol_constr + rho
  *Tol_constr^2)/J_ref*xi_norm;
```

5.4 Thermal problem

According to Yago et al. [40], the implementation of the thermal compliance problem is rather analogous to the structural mean compliance problem, detailed in section 2.5. In that case, the temperature, $\hat{\theta}$, is the only unknown per node (`n_unkn=1`) and the steady-state problem is used as the state equation. Therefore, the cost function (24) has to be replaced by

$$\begin{aligned}\mathcal{J}(\theta_\chi) &\equiv \frac{1}{2}l(\theta_\chi) = \frac{1}{2}a_\chi(\theta_\chi,\theta_\chi) = \\ &\equiv \frac{1}{2}\left(\int_\Omega \boldsymbol{\nabla}\theta_\chi \cdot \boldsymbol{\mathcal{K}}_\chi \cdot \boldsymbol{\nabla}\theta_\chi \, d\Omega\right) = \int_\Omega \mathcal{U}_\chi \, d\Omega \,,\end{aligned} \tag{52}$$

where $l(\theta_\chi)$ and $a_\chi(\theta_\chi,\theta_\chi)$ correspond to the bilinear forms of the thermal problem. Furthermore, $\boldsymbol{\nabla}\theta_\chi$ and $\boldsymbol{\mathcal{K}}_\chi$ represent the thermal gradient vector and the symmetric second order thermal conductivity tensor, respectively. Unlike the elastic material behavior used in section 2.2, the conductive material follows Fourier's law, i.e. the heat flux is proportional to the thermal gradient by $\boldsymbol{q}(\mathbf{x},\chi) = -\boldsymbol{\mathcal{K}}(\mathbf{x},\chi)\cdot\boldsymbol{\nabla}\theta_\chi(\mathbf{x})$.

The state equation (14) must be also substituted by

$$\left[\begin{aligned}&\text{Find the temperature field } \boldsymbol{\theta}_\chi \in \mathcal{U}(\Omega) \text{ such that}\\ &a(w,\theta_\chi) = l(w) \quad \forall w \in \mathcal{V}(\Omega) &(53)\\ &\text{where}\\ &a(w,\theta_\chi) = \int_\Omega \boldsymbol{\nabla}w(\mathbf{x})\cdot\boldsymbol{\mathcal{K}}_\chi(\mathbf{x})\cdot\boldsymbol{\nabla}\theta_\chi(\mathbf{x})\,d\Omega\,, &(54)\\ &l(w) = -\int_{\partial_q\Omega} w(\mathbf{x})\overline{q}(\mathbf{x})\,d\Gamma\\ &\qquad + \int_\Omega w(\mathbf{x})r_\chi(\mathbf{x})\,d\Omega\,, &(55)\end{aligned}\right.$$

where $\mathcal{U}(\Omega)$ and $\mathcal{V}(\Omega)$ stand for the corresponding set of admissible temperature fields and the corresponding space of admissible virtual temperature fields, respectively. $r(\mathbf{x},\chi)$ and $\overline{q}(\mathbf{x})$ correspond respectively to the heat source function and the prescribed heat flux on the boundaries of Ω.

After applying the RTD to equation (52), mimicking the procedure described in section 2.5, the resultant *pseudo-energy*, $\xi(\hat{\mathbf{x}})$, is expressed as

$$\xi(\hat{\mathbf{x}},\chi) = 2m_\kappa\left(\chi_\kappa(\hat{\mathbf{x}})\right)^{m_\kappa-1}\overline{\mathcal{U}}(\hat{\mathbf{x}})\Delta\chi_\kappa(\hat{\mathbf{x}})\,, \tag{56}$$

with

$$\overline{\mathcal{U}}(\hat{\mathbf{x}}) = \frac{1}{2}\left(\boldsymbol{\nabla}\theta_\chi\cdot\overline{\boldsymbol{\mathcal{K}}}\cdot\boldsymbol{\nabla}\theta_\chi\right)(\hat{\mathbf{x}}) \geq 0\,. \tag{57}$$

Several modification to the provided code, based on equation (52) to (56), are required in order to solve thermal problems. The most relevant ones are listed next: the number of unknowns per node must be set to 1, the gradient matrix, $\mathbf{B}$, must be adjusted to be equal to the Cartesian derivatives, the material property is now the conductivity value of the high conductive material instead of E and ν and the constitutive tensor $\boldsymbol{\mathcal{K}}$ is now defined as

$$\boldsymbol{\mathcal{K}} = \kappa\begin{bmatrix}K_{11} & K_{12}\\ K_{21} & K_{22}\end{bmatrix}\,. \tag{58}$$

In addition, boundary condition must be defined accordingly to the thermal problem.

5.5 3D extension

The topology optimization code *UNVARTOP* can be readily extended to solve 3D problems. All the functions related to FE analysis must be rewritten, starting from the mesh, the shape matrices $\mathbf{N}$, the corresponding strain-displacement matrices $\mathbf{B}$ and the constitutive tensor $\mathbb{C}$. Therefore, element stiffness matrices should be recomputed, along with the stiffness and mass matrices for the Laplacian regularization. It is recommended to use an iterative solver (e.g. `minres` solver) to compute the displacements, as employed for the Laplacian regularization (see section 3.8.3), in order to reduce computational cost. Function `compute_volume` must be slightly adapted to hexahedral elements. In addition, functions `isosurface`, `isocaps` and `isonormals` must be used to represent the optimal topology.

It is important to notice that the algorithm inside the topology optimization does not require any modification.

6 Conclusions

This paper has presented the 2D implementation in Matlab of the unsmooth variational topology optimization approach, previously formulated for structural [22] and thermal [40] topology optimization problems. The paper described and implemented the approach for educational purposes while demonstrating its capabilities and maintaining high computational efficiency and readability of the code. Furthermore, the implementation preserves the finite element analysis of the domain, thus introducing students to the numerical analysis as well as the topology optimization field.

The numerical examples performed in this work illustrate the potential and effectiveness of the technique to tackle a large set of different problems with a volume constraint, e.g minimum mean compliance problems (section 2.5), multi-load mean compliance problems (section 2.6) and compliant mechanisms synthesis (section 2.7). The set of numerical examples include a variety of boundary conditions, active and passive nodes, number of time-steps, along others. Additionally, section 5.4 shows how to easily switch from the structural problem of minimum mean compliance to the thermal problem, where thermal compliance is minimized. Finally, section 5.5 provides some guidelines for the extension of the code to the resolution of 3D problems.

The topologies obtained for these examples are comparable to those shown by other researchers using more established techniques (e.g. SIMP method or Level-set method). In addition, smooth topology configuration have been obtained in all the benchmarks with a relatively small number of iterations. That is a feature to be highlighted against more conventional techniques based on elemental densities, such as SIMP method.

In conclusion, the dissemination of this code will provide newcomers in this field a better understanding in how this new topology optimization approach works as well as to encourage future research of this technique for miscellaneous applications.

The Matlab code, detailed in appendix A, along with some variations of it, can be downloaded from the author's GitHub repository https://github.com/DanielYago. Additional online resources, such as figures and animations, are also stored in the repository.

Acknowledgements This research has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Proof of Concept Grant agreement n 874481) through the project "Computational design and prototyping of acoustic metamaterials for target ambient noise reduction" (METACOUSTIC). The authors also acknowledge financial support from the Spanish Ministry of Economy and Competitiveness, through the research grant DPI2017-85521-P for the project "Computational design of Acoustic and Mechanical Metamaterials" (METAMAT) and through the "Severo Ochoa Programme for Centres of Excellence in R&D" (CEX2018-000797-S). D. Yago acknowledges the support received from the Spanish Ministry of Education through the FPU program for PhD grants.

Conflict of interest

The authors declare that they have no conflict of interest as regards this work.

Appendix A Matlab code

```
function [iter,J] = UNVARTOP_2D_compliance (nelx,nely,nsteps,Vol0,Vol,k,tau)
n_dim = 2; n_unkn = 2; n_nodes = 4; n_gauss = 4; n = (nelx+1)*(nely+1); h_e = 1; alpha0 = 1
    e-3;
iter_max_step = 20; iter_min_step = 4; iter_max = 500;
opt = struct('Plot_top_iso',1,'Plot_vol_step',1,'EdgeColor','none','solver_Lap','direct');
%% Vector for assembling matrices
[X,Y] = meshgrid(0:nelx,nely:-1:0); coord = [X(:),Y(:)]; clear X Y
nodenrs = reshape(1:n,1+nely,1+nelx);
nodeVec = reshape(nodenrs(1:end-1,1:end-1)+1,nelx*nely,1); clear nodenrs;
connect = nodeVec+[0 nely+[1 0] -1]; clear nodeVec;
%% Loads and boundary setting for Cantilever beam
F = sparse(n_unkn*n,1);
U = zeros(n_unkn*n,1);
F(n_unkn*find(coord(:,2)==0 & coord(:,1)==nelx),1) = -0.01*nelx;
fixed_dofs = reshape(n_unkn*find(coord(:,1)==0)+(-n_unkn+1:0),1,[]);
active_node = []; passive_node = [];
free_dofs = setdiff(1:(n_unkn*n),fixed_dofs);
U(fixed_dofs,:) = 0;
%% Parameter definition
m = 5; E0 = 1; alpha = 1e-6; beta = nthroot(alpha,m); nu = 0.3;
%% Prepare animation
psi_vec = zeros(size(coord,1),nsteps+1);
chi_vec = zeros(size(connect,1),nsteps+1);
U_vec = zeros(n_unkn*size(coord,1),1,nsteps+1);
%% Finite element analysis preparation
[posgp4,W4] = gauss_points(n_gauss);
[posgp1,W1] = gauss_points(1);
[DE] = D_matrix_stress(E0,nu);
```

```
KE = zeros(n_nodes*n_unkn,n_nodes*n_unkn);
KE_i = zeros(n_nodes*n_unkn,n_nodes*n_unkn,n_gauss);
for i=1:n_gauss
 [BE,Det_Jacobian] = B_matrix(posgp4(:,i),n_unkn,n_nodes);
 KE_i(:,:,i) = BE'*DE*BE;
 KE = KE + KE_i(:,:,i)*Det_Jacobian*W4(i);
end
[BE_cut,Det_Jacobian_cut] = B_matrix(posgp1,n_unkn,n_nodes);
K_cut = BE_cut'*DE*BE_cut;
KE_cut = K_cut*Det_Jacobian_cut*W1(1);
edofMat = kron(connect,n_unkn*ones(1,n_unkn)) + repmat(1-n_unkn:0,1,n_nodes);
iK = reshape(kron(edofMat,ones(n_nodes*n_unkn,1))',(n_nodes*n_unkn)^2*nelx*nely,1);
jK = reshape(kron(edofMat,ones(1,n_nodes*n_unkn))',(n_nodes*n_unkn)^2*nelx*nely,1);
%% Laplacian filter preparation
KE_Lap = 1/6* [ 4 -1 -2 -1;-1 4 -1 -2;-2 -1 4 -1;-1 -2 -1 4];
ME_Lap = 1/36*[ 4  2  1  2; 2 4  2  1; 1  2 4  2; 2  1  2 4];
KE_Lap = ME_Lap + (tau*h_e).^2*KE_Lap;
i_KF = reshape(kron(connect,ones(n_nodes,1))',n_nodes^2*nelx*nely,1);
j_KF = reshape(kron(connect,ones(1,n_nodes))',n_nodes^2*nelx*nely,1);
s_KF = reshape(KE_Lap(:)*ones(1,nelx*nely),n_nodes^2*nelx*nely,1); clear KE_Lap ME_Lap;
K_Lap = sparse(i_KF,j_KF,s_KF);
if strcmp(opt.solver_Lap,'direct'); LF = chol(K_Lap,'lower'); clear K_Lap i_KF j_KF s_KF;
else; LF = ichol(K_Lap, struct('type','ict','droptol',1e-3,'diagcomp',0.1)); clear i_KF
    j_KF s_KF; end
i_xi = reshape(connect',n_nodes*nelx*nely,1);
N_T = N_matrix(posgp4).*W4/4;
%% Loop over steps
psi = alpha0*ones(n,1); psi(passive_node) = -alpha0; psi(active_node) = alpha0; psi_vec
    (:,1)=psi;
[~,chi] = compute_volume (psi,connect); chi0_step = chi; chi_vec(:,1) = chi';
% Initialize variables
iter = 1; J_vec = []; vol_vec = []; lambda_vec = 0; lambda = 0; fhandle6 = [];
[fhandle2,ohandle2] = plot_isosurface([],[],0,psi,coord,connect,1,opt);
for i_step = 1:nsteps
 [t_ref] = set_reference_volume(i_step,Vol0,Vol,nsteps,k);
 % Main loop by steps
 Tol_chi = 1;
 Tol_lambda = 1;
 iter_step = 1;
 while (((Tol_chi>1e-1 || Tol_lambda>1e-1) && iter_step<iter_max_step) || iter_step<=
    iter_min_step)
  % FE-analysis
  [K] = assmebly_stiff_mat (chi,KE,KE_cut,beta,m,iK,jK,n_unkn,nelx,nely);
  U(free_dofs,:) = K(free_dofs,free_dofs) \ (F(free_dofs,:) - K(free_dofs,fixed_dofs)*U(
    fixed_dofs,:));
  if iter == 1; U_vec(:,:,1)=U; J_ref = full(abs(sum(sum(F.*U,1),2))); end; J = full(sum(
    sum(F.*U,1),2))/J_ref;
  % Calculate sensitivities
  Energy = zeros(n_gauss,nelx*nely);
  id = chi==1|chi==0; int_chi = interp_property (m,m-1,beta,chi(id));
  u_e = reshape(U(edofMat(id,:)',1),n_nodes*n_unkn,[]); w_e = u_e;
  for i=1:n_gauss; Energy(i,id) = sum(w_e.*(KE_i(:,:,i)*u_e),1); end
  Energy(:,id) = int_chi.*Energy(:,id);
  id = ~id; int_chi = interp_property (m,m-1,beta,chi(id));
  u_e = reshape(U(edofMat(id,:)',1),n_nodes*n_unkn,[]); w_e = u_e;
  Energy(:,id) = repmat(int_chi.*sum(w_e.*(K_cut*u_e),1),n_gauss,1);
  if iter == 1; xi_shift = min(0,min(Energy(:))); xi_norm = max(range(Energy(:)),max(Energy
    (:))); end
  % Apply Laplacian regularization
  xi_int = N_T*(Energy-xi_shift*chi)/xi_norm;
  if strcmp(opt.solver_Lap,'direct')
   xi = LF'\(LF\accumarray(i_xi,xi_int(:),[n 1]));
  else
   [xi,flag] = minres(K_Lap,accumarray(i_xi,xi_int(:),[n 1]),1e-6,500,LF,LF'); assert(flag
    == 0);
  end
  % Compute topology
  [lambda,chi_n,psi,vol] = find_volume (xi,connect,active_node,passive_node,t_ref,lambda,alpha0);
  lambda_vec = [lambda_vec,lambda];
  Tol_lambda = (lambda_vec(iter+1)-lambda_vec(iter))/lambda_vec(iter+1);
  % Plot topology
  [fhandle2,ohandle2] = plot_isosurface(fhandle2,ohandle2,iter,psi,coord,connect,J,opt);
  % Update variables
  Tol_chi = sqrt(sum((chi-chi_n).^2))/sqrt(sum(chi0_step.^2));
  chi = chi_n;
  fprintf(' Step:%5i It.:%5i Obj.:%11.4f Vol.:%7.3f \n',i_step,iter_step,J,vol);
  iter_step = iter_step+1; iter = iter+1;
  drawnow;
 end
 chi0_step = chi;
 if J<10
  [fhandle6,J_vec,vol_vec] = plot_volume_iter(fhandle6,i_step,J_vec,J,vol_vec,vol,opt.Plot_vol_step,6,'Cost function Step','#step','+-b');
  psi_vec(:,i_step+1)=psi; chi_vec(:,i_step+1)=chi'; U_vec(:,:,i_step+1)=U;
 end
 if iter_step >= iter_max_step; warning('VarTopOpt:Max_iter_step','Maximum number of in-step iterations achieved.'); break; end
 if iter > iter_max; warning('VarTopOpt:Max_iter','Maximum number of iterations achieved.'); break; end
end
%% Animation
Topology_evolution(coord,connect,[Vol0,vol_vec],psi_vec,chi_vec,U_vec);
%%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%%
function [posgp,W] = gauss_points (n_gauss)
if n_gauss==1; s = 0; w = 2; else; s = sqrt(3)/3*[-1 1]; w = [1 1]; end
[s,t] = meshgrid(s,s); posgp = [s(:) t(:)]';
W=w'*w; W=W(:)';
%%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%%
function [N] = N_matrix(posgp)
N = 0.25*(1+[-1 1 1 -1]'*posgp(1,:)).*(1+[-1 -1 1 1]'*posgp(2,:));
%%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%%
function [BE,Det_Jacobian,cart_deriv] = B_matrix(posgp,n_unkn,n_nodes)
dshape = 0.25*[-1 -1;1 -1;1 1;-1 1]'.* flip(1+[-1 -1;1 -1;1 1;-1 1]'.*posgp,1);
Jacobian_mat = dshape*[0 0;1 0;1 1;0 1];
Det_Jacobian = det(Jacobian_mat);
cart_deriv = Jacobian_mat\dshape;
BE = zeros(3,n_unkn*n_nodes); BE([1 3],1:n_unkn:end) = cart_deriv; BE([3 2],2:n_unkn:end) = cart_deriv;
%%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%%
function [DE] = D_matrix_stress(E,nu) %Planestress
DE = E/(1-nu^2)*[1 nu 0; nu 1 0;0 0 (1-nu)/2];
%%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%%
function [coeff] = interp_property (m,n,beta,chi)
coeff = chi + (1-chi).*beta;
coeff = double(m==n).*coeff.^m + double(m~=n).*m*coeff.^n*(1-beta);
%%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%%
function [K] = assmebly_stiff_mat (chi,KE,KE_cut,beta,m,iK,jK,n_unkn,nelx,nely)
sK = interp_property(m,m,beta,chi).*KE(:); sK(:,chi~=1&chi~=0) = interp_property(m,m,beta,chi(chi~=1&chi~=0)).*KE_cut(:);
K = sparse(iK,jK,sK,n_unkn*(1+nelx)*(1+nely),n_unkn*(1+nelx)*(1+nely)); K = (K+K')/2; clear sK;
%%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%%
function [lambda,chi,psi,vol,Tol_constr] = find_volume (xi,connect,active_node,passive_node,t_ref,lambda,alpha0)
l1 = min(xi); c1 = t_ref; l2 = max(xi); c2 = t_ref-1; Tol_constr = 1; iter=1;
if lambda>l1 && lambda<l2
 [chi,psi,vol,l1,l2,c1,c2,Tol_constr] = compute_volume_lambda(xi,connect,active_node,passive_node,t_ref,lambda,l1,l2,c1,c2,alpha0);
end
while (abs(Tol_constr)>1e-4) && iter<1000
 lambda = 0.5*(l1+l2);
 [chi,psi,vol,l1,l2,c1,c2,Tol_constr] = compute_volume_lambda(xi,connect,active_node,passive_node,t_ref,lambda,l1,l2,c1,c2,alpha0);
 iter = iter + 1;
end
%%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%%
function [chi,psi,vol,l1,l2,c1,c2,Tol_constr] = compute_volume_lambda(xi,connect,active_node,passive_node,t_ref,lambda,l1,l2,c1,c2,alpha0)
psi = xi - lambda; psi(passive_node) = -alpha0; psi(active_node) = alpha0;
[vol,chi] = compute_volume (psi,connect);
Tol_constr = -(vol-t_ref);
if Tol_constr > 0, l1 = lambda; c1 = Tol_constr; else; l2 = lambda; c2 = Tol_constr; end
%%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%%
function [volume,chi] = compute_volume (psi,connect)
P = [-1 -1;1 -1;1 1;-1 1]; dvol = 1/4;
s = [-0.9324695142031521 -0.6612093864662645 -0.2386191860831969 0.2386191860831969 0.6612093864662645 0.9324695142031521]; [s,t] = meshgrid(s,s);
w = [ 0.1713244923791704  0.3607615730481386  0.4679139345726910 0.4679139345726910 0.3607615730481386 0.1713244923791704]; W=w'*w; W=W(:)';
psi_n = psi(connect); chi = sum((sign(psi_n)+1),2)'/8; id = chi~=1&chi~=0;
phi_x = psi_n(id,:)*((1+P(:,1)*s(:)').*(1+P(:,2)*t(:)')/4);
chi(1,id) = (W*(phi_x>0)'+ 0.5*W*(phi_x==0)')*dvol;
volume = 1 - sum(chi) / size(connect,1);
%%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%%
function [fig_handle,obj_handle] = plot_isosurface(fig_handle,obj_handle,iter,psi,coord,connect,J,opt)
if opt.Plot_top_iso
 if iter==0; fig_handle = figure(2); set(fig_handle,'Name','Topology'); caxis([-1 1]); colormap(flip(gray(2)));
  axis equal tight; xlabel('x'); ylabel('y'); title(['J = ',num2str(J)]);
  obj_handle = patch('Vertices',coord,'Faces',connect,'FaceVertexCData',psi,'EdgeColor',opt.EdgeColor,'FaceColor','interp');
 else
  set(0, 'CurrentFigure', fig_handle); set(get(gca,'Title'),'String',['J = ',num2str(J)]);
  set(obj_handle,'FaceVertexCData',psi);
 end
end
%%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%%
function [fig_handle,J_vec,vol_vec] = plot_volume_iter(fig_handle,iter,J_vec,J,vol_vec,vol,opt_plot,fig_num,fig_name,xlabel_name,linestyle)
J_vec = [J_vec,J]; vol_vec = [vol_vec,vol];
if opt_plot
 if iter==1; fig_handle = figure(fig_num); set(fig_handle,'Name',fig_name);
  subplot(2,1,1); plot(J_vec,linestyle);   ylabel('$\mathcal{J}_\chi$','Interp','Latex'); xlabel(xlabel_name); grid; grid minor;
  subplot(2,1,2); plot(vol_vec,linestyle); ylabel('|\Omega^-|'); xlabel(xlabel_name); grid; grid minor;
 else; set(0, 'CurrentFigure', fig_handle);
  subplot(2,1,1); set(findobj(gca,'Type','line'),'Xdata',1:numel(J_vec),'YData',J_vec);
  subplot(2,1,2); set(findobj(gca,'Type','line'),'Xdata',1:numel(vol_vec),'YData',vol_vec);
 end
end
%%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%% %%%%
function [vol] = set_reference_volume(iter,Vol0,Volf,nsteps,k)
if k==0; vol=Vol0+(Volf-Vol0)/nsteps*iter;
else; C1=(Vol0-Volf)/(1-exp(k)); C2=Vol0-C1; vol=C1*exp(k*iter/nsteps)+C2; end
```

Listing 1 UNVARTOP code written in Matlab

Replication of results

The Matlab codes provided in the paper, in Appendix A and GitHub repository, are the same ones used for obtaining the results here presented (Section 4). Therefore, they can be fully used as a replication tool, to reproduce those results, as well as to be used in additional numerical simulations.

References

1. G. Allaire, E. Bonnetier, G. Francfort, and F. Jouve. Shape optimization by the homogenization method. *Numerische Mathematik*, 76(1):27–68, mar 1997. doi: 10.1007/s002110050253.
2. G. Allaire, F. Jouve, and A.-M. Toader. A level-set method for shape optimization. *Comptes Rendus Mathematique*, 334(12):1125–1130, jan 2002. doi: 10.1016/s1631-073x(02)02412-3.
3. G. Allaire, F. Jouve, and A.-M. Toader. Structural optimization using sensitivity analysis and a level-set method. *Journal of Computational Physics*, 194(1):363–393, feb 2004. doi: 10.1016/j.jcp.2003.09.032.
4. N. Anderson and Å. Björck. A new high order method of regula falsi type for computing a root of an equation. *BIT*, 13(3):253–264, sep 1973. doi: 10.1007/bf01951936.
5. E. Andreassen, A. Clausen, M. Schevenels, B. S. Lazarov, and O. Sigmund. Efficient topology optimization in MATLAB using 88 lines of code. *Structural and Multidisciplinary Optimization*, 43(1):1–16, nov 2010. doi: 10.1007/s00158-010-0594-7.
6. M. P. Bendsøe. Optimal shape design as a material distribution problem. *Structural Optimization*, 1(4):193–202, dec 1989. doi: 10.1007/bf01650949.
7. M. P. Bendsøe and O. Sigmund. *Topology Optimization.* Springer Berlin Heidelberg, 2004. doi: 10.1007/978-3-662-05086-6.
8. E. Biyikli and A. C. To. Proportional topology optimization: A new non-sensitivity method for solving stress constrained and minimum compliance problems and its implementation in MATLAB. *PLOS ONE*, 10(12): e0145041, dec 2015. doi: 10.1371/journal.pone.0145041.
9. R. Bulirsch and J. Stoer. *Introduction to Numerical Analysis.* Springer New York, 2010. ISBN 144193006X.
10. V. J. Challis. A discrete level-set topology optimization code written in matlab. *Structural and Multidisciplinary Optimization*, 41(3):453–464, sep 2009. doi: 10.1007/s00158-009-0430-0.
11. D. Da, L. Xia, G. Li, and X. Huang. Evolutionary topology optimization of continuum structures with smooth boundary representation. *Structural and Multidisciplinary Optimization*, 57(6):2143–2159, nov 2017. doi: 10.1007/s00158-017-1846-6.
12. M. Dowell and P. Jarratt. A modified regula falsi method for computing the root of an equation. *BIT*, 11(2):168–174, jun 1971. doi: 10.1007/bf01934364.
13. R. A. Feijoo, A. A. Novotny, E. Taroco, and C. Padra. The topological-shape sensitivity method in two-dimensional linear elasticity topology design. *Applications of Computational Mechanics in Structures and Fluids*, 2005.
14. S. Giusti, A. Novotny, and C. Padra. Topological sensitivity analysis of inclusion in two-dimensional linear elasticity. *Engineering Analysis with Boundary Elements*, 32(11):926–935, nov 2008. doi: 10.1016/j.enganabound.2007.12.007.
15. J. K. Guest, J. H. Prévost, and T. Belytschko. Achieving minimum length scale in topology optimization using nodal design variables and projection functions. *International Journal for Numerical Methods in Engineering*, 61(2):238–254, aug 2004. doi: 10.1002/nme.1064.
16. B. S. Lazarov and O. Sigmund. Filters in topology optimization based on helmholtz-type differential equations. *International Journal for Numerical Methods in Engineering*, 86(6):765–781, dec 2010. doi: 10.1002/nme.3072.
17. Q. Q. Liang and G. P. Steven. A performance-based optimization method for topology design of continuum structures with mean compliance constraints. *Computer Methods in Applied Mechanics and Engineering*, 191(13-14):1471–1489, jan 2002. doi: 10.1016/s0045-7825(01)00333-4.
18. K. Liu and A. Tovar. An efficient 3d topology optimization code written in matlab. *Structural and Multidisciplinary Optimization*, 50(6):1175–1196, jun 2014. doi: 10.1007/s00158-014-1107-x.
19. C. G. Lopes, R. B. dos Santos, and A. A. Novotny. Topological derivative-based topology optimization of structures subject to multiple load-cases. *Latin American Journal of Solids and Structures*, 12(5):834–860, may 2015. doi: 10.1590/1679-78251252.
20. D. G. Luenberger and Y. Ye. *Linear and Nonlinear Programming.* Springer International Publishing, 2016. doi: 10.1007/978-3-319-18842-3.
21. A. Novotny, R. Feijóo, E. Taroco, and C. Padra. Topological sensitivity analysis. *Computer Methods in Applied Mechanics and Engineering*, 192(7-8):803–829, feb 2003. doi: 10.1016/s0045-7825(02)00599-6.
22. J. Oliver, D. Yago, J. Cante, and O. Lloberas-Valls. Variational approach to relaxed topological optimization: Closed form solutions for structural problems in a sequential pseudo-time framework. *Computer Methods in Applied Mechanics and Engineering*, 355:779–819, oct 2019. doi: 10.1016/j.cma.2019.06.038.
23. S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of Computational Physics*, 79(1):12–49, nov 1988. doi: 10.1016/0021-9991(88)90002-2.
24. M. Otomori, T. Yamada, K. Izui, and S. Nishiwaki. Matlab code for a level set-based topology optimization method using a reaction diffusion equation. *Structural and Multidisciplinary Optimization*, 51(5):1159–1172, nov 2014. doi: 10.1007/s00158-014-1190-z.
25. G. Patanè and B. Falcidieno. Computing smooth approximations of scalar functions with constraints. *Computers & Graphics*, 33(3):399–413, jun 2009. doi: 10.1016/j.cag.2009.03.014.
26. S. S. Rao. *The Finite Element Method in Engineering.* Butterworth-Heinemann, 2004. ISBN 0-7506-7828-3.
27. O. Sigmund. A 99 line topology optimization code written in matlab. *Structural and Multidisciplinary Optimization*, 21(2):120–127, apr 2001. doi: 10.1007/s001580050176.
28. O. Sigmund and J. Petersson. Numerical instabilities in topology optimization: A survey on procedures dealing with checkerboards, mesh-dependencies and local minima. *Structural Optimization*, 16(1):68–75, aug 1998. doi: 10.1007/bf01214002.
29. J. Sokolowski and A. Zochowski. On the topological derivative in shape optimization. *SIAM Journal on Control and Optimization*, 37(4):1251–1272, jan 1999. doi: 10.1137/s0363012997323230.
30. K. Suresh. A 199-line matlab code for pareto-optimal tracing in topology optimization. *Structural and Multidisciplinary Optimization*, 42(5):665–679, jul 2010. doi: 10.1007/s00158-010-0534-6.
31. A. Takezawa, S. Nishiwaki, and M. Kitamura. Shape and topology optimization based on the phase field method and sensitivity analysis. *Journal of Computational Physics*, 229(7):2697–2718, apr 2010. doi: 10.1016/j.jcp.2009.12.017.

32. R. Tavakoli and S. M. Mohseni. Alternating active-phase algorithm for multimaterial topology optimization problems: a 115-line MATLAB implementation. *Structural and Multidisciplinary Optimization*, 49(4):621–642, oct 2013. doi: 10.1007/s00158-013-0999-1.
33. L. J. van Vliet, I. T. Young, and G. L. Beckers. A nonlinear laplace operator as edge detector in noisy images. *Computer Vision, Graphics, and Image Processing*, 45(2):167–195, feb 1989. doi: 10.1016/0734-189x(89) 90131-x.
34. M. Y. Wang and S. Zhou. Synthesis of shape and topology of multi-material structures with a phase-field method. *Journal of Computer-Aided Materials Design*, 11(2-3):117–138, jan 2004. doi: 10.1007/ s10820-005-3169-y.
35. M. Y. Wang, S. Chen, and Q. Xia. Toplsm, a 199-line matlab program, July 2004.
36. S. Wang and M. Y. Wang. Radial basis functions and level set method for structural topology optimization. *International Journal for Numerical Methods in Engineering*, 65(12):2060–2090, 2006. doi: 10.1002/nme.1536.
37. S. Wang, K. Lim, B. Khoo, and M. Wang. An extended level set method for shape and topology optimization. *Journal of Computational Physics*, 221(1):395–421, jan 2007. doi: 10.1016/j.jcp.2006.06.029.
38. P. Wei, Z. Li, X. Li, and M. Y. Wang. An 88-line MATLAB code for the parameterized level set method based topology optimization using radial basis functions. *Structural and Multidisciplinary Optimization*, 58(2):831–849, feb 2018. doi: 10.1007/s00158-018-1904-8.
39. Y. M. Xie and G. P. Steven. *Evolutionary Structural Optimization*. Springer London, 1997. doi: 10.1007/ 978-1-4471-0985-3.
40. D. Yago, J. Cante, O. Lloberas-Valls, and J. Oliver. Topology optimization of thermal problems in a nonsmooth variational setting: closed-form optimality criteria. *Computational Mechanics*, 66(2):259–286, jun 2020. doi: 10.1007/s00466-020-01850-0.
41. T. Yamada, K. Izui, S. Nishiwaki, and A. Takezawa. A topology optimization method based on the level set method incorporating a fictitious interface energy. *Computer Methods in Applied Mechanics and Engineering*, 199(45-48):2876–2891, nov 2010. doi: 10.1016/j.cma. 2010.05.013.
42. X. Y. Yang, Y. M. Xie, G. P. Steven, and O. M. Querin. Bidirectional evolutionary method for stiffness optimization. *AIAA Journal*, 37:1483–1488, jan 1999. doi: 10.2514/3.14346.
43. W. Zhang, J. Yuan, J. Zhang, and X. Guo. A new topology optimization approach based on moving morphable components (MMC) and the ersatz material model. *Structural and Multidisciplinary Optimization*, 53(6): 1243–1260, dec 2015. doi: 10.1007/s00158-015-1372-3.
44. O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu. *The Finite Element Method: Its Basis and Fundamentals*. Elsevier LTD, Oxford, 2013. ISBN 1856176339.
45. Z. H. Zuo and Y. M. Xie. A simple and compact python code for complex 3d topology optimization. *Advances in Engineering Software*, 85:1–11, jul 2015. doi: 10.1016/j. advengsoft.2015.02.006.