Struct Multidisc Optim (2012) 45:329–357
DOI 10.1007/s00158-011-0696-x

EDUCATIONAL ARTICLE

PolyTop: a Matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes

Cameron Talischi · Glaucio H. Paulino · Anderson Pereira · Ivan F. M. Menezes

Received: 24 May 2011 / Revised: 12 July 2011 / Accepted: 19 July 2011 / Published online: 8 January 2012

Abstract We present an efficient Matlab code for structural topology optimization that includes a general finite element routine based on isoparametric polygonal elements which can be viewed as the extension of linear triangles and bilinear quads. The code also features a modular structure in which the analysis routine and the optimization algorithm are separated from the specific choice of topology optimization formulation. Within this framework, the finite element and sensitivity analysis routines contain no information related to the formulation and thus can be extended, developed and modified independently. We address issues pertaining to the use of unstructured meshes and arbitrary design domains in topology optimization that have received little attention in the literature. Also, as part of our examination of the topology optimization problem, we review the various steps taken in casting the optimal shape problem as a sizing optimization problem. This endeavor allows us to isolate the finite element and geometric analysis parameters and how they are related to the design variables of the discrete optimization problem. The Matlab code is explained in detail and numerical examples are presented to illustrate the capabilities of the code.

Keywords Topology optimization · Unstructured meshes · Polygonal finite elements · Matlab software

1 Introduction

In this work, we create a general framework for solving topology optimization problems and present a simple and efficient Matlab implementation that features a general finite element (FE) analysis routine and encompasses a large class of topology optimization formulations. Many engineering applications of topology optimization require the use of unstructured meshes for accurate description of the design domain geometry, specification of the loading and support conditions, and reliable analysis of the design response. We address the use of arbitrary meshes in topology optimization and discuss certain practical issues that have received little attention in the literature. For example, higher computational cost associated with unstructured meshes, cited as a motivation for adopting uniform grids, is sometimes attributed to the need for repeated calculations of the element stiffness matrices. However, as we demonstrate, the invariant quantities such as the element stiffness matrices and the global stiffness matrix connectivity can be computed once and stored for use in subsequent optimization iterations. This approach is feasible since the amount of memory required is relatively small considering the hardware capacities of current personal computers. Moreover, the overhead associated with computing multiple element stiffness matrices is small compared to the overall cost of the optimization algorithm which may require solving hundreds of linear systems. The Matlab code presented in this work retains the readability and efficiency of previous educational codes while offering a general FE framework. In

Electronic supplementary material The online version of this article (doi:10.1007/s00158-011-0696-x) contains supplementary material, which is available to authorized users.

C. Talischi · G. H. Paulino (✉)
Department of Civil and Environmental Engineering, University of Illinois at Urbana-Champaign, 205 North Mathews Avenue, Newmark Laboratory, MC-250, Urbana, IL 61801, USA
e-mail: paulino@uiuc.edu

A. Pereira · I. F. M. Menezes
Tecgraf, Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Rua Marquês de São Vicente, 225, Gávea, 22453-900, Rio de Janeiro, RJ, Brazil

terms of efficiency, our Matlab implementation has performance that is on par with the recently published 88 line code (Andreassen et al. 2010) and is faster for large meshes. As an example of arbitrary meshes, we have implemented the FE routine based on isoparametric polygonal finite elements.[1] This choice adds versatility to the code because the isoparametric formulation can support all linear polygons including the more commonly used linear triangles and bilinear quads. Moreover, a polygonal mesh generator based on the concept of Voronoi diagrams written in Matlab is presented in a companion paper (Talischi et al. 2011). The two codes make for a self-contained discretization and analysis package in Matlab. We note, however, that since the FE code is general, it is straightforward for the users to implement new elements (e.g. higher order, nonconforming, etc.) or use meshes generated by other software (an example of a Matlab mesh generator is `distMesh` by Persson and Strang 2004).

Another related goal of this work is to present a modular code structure for topology optimization in which the analysis routine for solving the state equation and computing the sensitivities is made independent of the specific formulation chosen. This means that the analysis routine need not know about the specific formulation used which in turn allows the code to accommodate various formulations without compromising its clarity or pragmatism. By contrast, previous educational software, e.g., the 99- and 88-line and similar codes, often mix the analysis and formulation—perhaps in the interest of keeping the code short and compact—and so require multiple versions (Andreassen et al. 2010). Within the present framework, the analysis routine can be developed, modified, or extended without any effect on the optimization code. Conversely, if a new problem demands a different type of analysis, a suitable analysis package can replace the one presented. We note that other Matlab codes for topology optimization can make use of our general analysis code. The formalism of this decoupling approach is crucial when one seeks to develop the framework for solving more complicated and involved topology optimization problems including multiphysics response, multiple geometry or state constraints, though for the simple compliance benchmarks the difference may seem minor.

As part of our examination of the topology optimization problem, we will review the various steps taken in casting the optimal shape problem as a sizing optimization problem. At certain points we depart from the usual narrative in order to address certain common misconceptions. For example, we note that the Ersatz approach, which consists of filling the void region with a compliant material, has to do with the approximation of the governing state equation and not the particular "sizing" formulation adopted. Also the purpose of filtering is shown to implicitly ensure smoothness of the design field. It is evident from formulations in which the positive lower bound on the design variables (used as a means to implement the Ersatz approach) and filtering parameters are related that these separate concepts are sometimes mixed up. Also in some papers, the use of filtering is inconsistent as it appears only in evaluating the stiffness terms and not the volume constraint (see Sigmund 2007 for a discussion on this issue). We will also discuss the rationale behind filtering at the continuum level, and show the approximation steps (often not explicitly presented) that lead to the discrete operation commonly used. Within this narrative, we will partially address the important but often neglected question of "what optimization problem is solved after the various parametrization and the restriction steps are taken."

The remainder of the paper is organized as follows: in Section 2, we review basic concepts in the formulation and discretization of the topology optimization problem. Next, in Section 3, we discuss the decoupling approach that forms the basis of the Matlab software. The details of the implementation are presented in Section 4, followed by numerical examples in Section 5 and discussion on issue of efficiency in Section 6. We conclude the paper with some remarks in Section 7.

2 Some theoretical considerations

The purpose of this section is to describe the classical continuum structural topology optimization problem and to identify the structure of the resulting discrete optimization problem in sufficient generality. The framework of the educational code reflects the findings of this section. Throughout, we shall also review some fundamental concepts in topology optimization problem formulation and make some comments regarding the various steps taken from the classical problem one sets out to solve, which suffers from several pathologies, to the final discrete "sizing" problem that is passed to a nonlinear programming algorithm.

2.1 Statement of the classical problem

The goal of topology optimization is to find the most efficient shape $\omega \subseteq \mathbb{R}^d, d = 2, 3$ of a physical system whose behavior is represented by the solution $\mathbf{u}_\omega$ to a

[1] Polygonal discretizations have been used in computational solid mechanics, see for example Ghosh (2010). In topology optimization, they have been shown to outperform linear triangles and quads since they eliminate numerical instabilities such as checkerboarding (Langelaar 2007; Saxena 2008; Talischi et al. 2009, 2010).

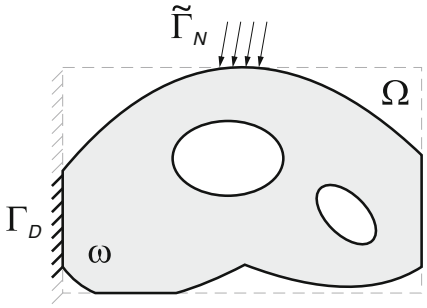


Fig. 1 Extended design domain and boundary conditions for the state equation

boundary value problem. More specifically, one deals with problems of the form:

$$\inf_{\omega\in\mathcal{O}} f(\omega, \mathbf{u}_\omega) \qquad \text{subject to}$$
$$g_i(\omega, \mathbf{u}_\omega) \le 0,\ i = 1, \ldots, K \tag{1}$$

Here $\mathcal{O}$ denotes the set of admissible shapes, f and g_i are, respectively, the objective and constraints functions that measure the performance of each candidate shape or design, ω, and K denotes the number of constraints. The geometric restrictions on the admissible shapes, such as bound on their volume or perimeter, are typically imposed through these constraint functions while other design requirements, such as symmetries or bound on feature size, are prescribed in $\mathcal{O}$. As shown in Fig. 1, one typically defines an extended design domain or a "hold-all" set Ω in which all the shapes lie, that is, $\omega \subseteq \Omega$ for all $\omega \in \mathcal{O}$. This working domain Ω facilitates the description of the governing boundary value problem.

It is well-known that existence of optimal solutions to this problem is, in general, not guaranteed if $\mathcal{O}$ is defined to be the set of all measurable subsets of Ω (see, for example, Kohn and Strang 1986a; Allaire 2001). Throughout this paper, we will assume that there are design or manufacturing constraints imposed on $\mathcal{O}$, directly or through the constraints in (1), such that the admissible shapes satisfy some uniform regularity property that makes the problem well-posed. This is typically known as the *restriction*[2] setting in the literature (Sigmund and Petersson 1998; Borrvall 2001). We will return to this issue later in this paper.

We consider linear elasticity as the governing state equation, which is typical in continuum structural optimization. The solution $\mathbf{u}_\omega \in \mathcal{V}_\omega$ satisfies the variational problem:

$$\int_\omega \mathbf{C}\nabla\mathbf{u}_\omega : \nabla\mathbf{v} d\mathbf{x} = \int_{\tilde{\Gamma}_N} \mathbf{t}\cdot\mathbf{v} ds, \quad \forall \mathbf{v} \in \mathcal{V}_\omega \tag{2}$$

[2]This is in contrast to the relaxation setting which begins with $\mathcal{O}$ defined as the set of all measurable subsets of Ω and addresses ill-posedness of the problem by further enlarging the space.

where

$$\mathcal{V}_\omega = \left\{\mathbf{v} \in H^1(\omega; \mathbb{R}^d) : \mathbf{v}|_{\partial\omega\cap\Gamma_D} = \mathbf{0}\right\} \tag{3}$$

is the space of admissible displacements, $\mathbf{C}$ is the stiffness tensor of the material from which the shape ω is made, Γ_D and Γ_N form a partition of $\partial\Omega$ and $\tilde{\Gamma}_N \subseteq \Gamma_N$ is where the non-zero tractions $\mathbf{t}$ is specified. For optimal problem (1) to be non-trivial, we shall assume that for each admissible shape ω, $\partial\omega \cap \Gamma_D$ has non-zero surface measure and $\tilde{\Gamma}_N \subseteq \partial\omega$. This reflects the desire that the admissible shapes are supported on Γ_D and subjected to the design loads defined on $\tilde{\Gamma}_N$. Note that in (2), the free boundary of ω, i.e., $\partial\omega\backslash\partial\Omega$, is traction-free. As we can see, the loading is assumed to be independent of the design. We refer to Bourdin and Chambolle (2003) and Bruyneel and Duysinx (2005) for formulation and analysis of problems with design-dependent loads such as pressure loading and self-weight.

The optimal design problem (1) with the boundary value problem constraint, as stated in (2), is not amenable to typical discretization and optimization strategies without an appropriate parametrization of the space of admissible shapes. For example, the implicit constraints on the space of admissible shapes $\mathcal{O}$ often cannot be directly enforced in the discrete setting. Note also that the internal virtual work term and the space of admissible displacements $\mathcal{V}_\omega$ change with the shape ω further complicating the prospects of discretization. Therefore it is useful to recast the boundary value problem on Ω by using the characteristic function χ_ω associated with ω. That is, we replace (2) with

$$\int_\Omega \chi_\omega \mathbf{C}\nabla\mathbf{u}_\omega : \nabla\mathbf{v} d\mathbf{x} = \int_{\tilde{\Gamma}_N} \mathbf{t}\cdot\mathbf{v} ds, \quad \forall \mathbf{v} \in \mathcal{V} \tag{4}$$

where now the space of admissible displacement,

$$\mathcal{V} = \left\{\mathbf{v} \in H^1(\Omega; \mathbb{R}^d) : \mathbf{v}|_{\Gamma_D} = \mathbf{0}\right\} \tag{5}$$

is independent of ω, unlike (3). Accordingly, for the optimal design problem (1), we define the admissible space $\mathcal{A}_\mathcal{O} = \{\chi_\omega : \omega \in \mathcal{O}\}$. Now the geometric attributes of shape ω can be described in terms of the associated characteristic functions in $\mathcal{A}_\mathcal{O}$. In fact, such distributed parametrization simplifies definition and imposition of such constraints. For example, the volume and perimeter of a shape ω can written as:

$$V(\omega) = \int_\Omega \chi_\omega d\mathbf{x}, \qquad P(\omega) = \int_\Omega |\nabla\chi_\omega| d\mathbf{x}$$

where the integral in the second expression is understood as the total variation of function χ_ω. We refer the reader to the monograph by Delfour and Zolésio (2001) for a more

detailed discussion of representing shapes by characteristic functions and the associated concepts of geometric measure theory.

The parametrization of the set of admissible shapes via characteristic functions, although a very useful starting point, does not address all the theoretical and practical issues. First, we note that existence and uniqueness of solutions to the extended boundary value problem (4) are not guaranteed as the energy bilinear form is no longer coercive (where χ_ω is zero, there is no contribution to the internal virtual work from the displacement; this loss of coercivity corresponds to the singularity of the stiffness matrix in the finite element discretization of this variational equation). The common approach in topology optimization, sometimes referred to the Ersatz material model in the level set literature, consists of filling these void regions with compliant material of stiffness $\varepsilon\mathbf{C}$. This amounts to replacing χ with $\varepsilon + (1-\varepsilon)\,\chi$ in the bilinear form of (4). The transmission conditions on the boundary $\partial\omega$ approximate the traction free state in (2) (see Allaire 2001 and Dambrine and Kateb 2009 for analysis of a scalar elliptic state equation in the degeneracy limit). The validity of this approximation for the optimal design problem (1) hinges on convergence of not only the solutions to the state equation but also the optimal shapes with this modified state equation as $\varepsilon \to 0$. It is clear that the ill-posedness of the optimal shape problem further complicates the analysis of the Ersatz approach. We refer the reader to Allaire and Francfort (1998) for a partial result on degeneracy of compliance minimization in the homogenization framework and to Bourdin and Chambolle (2003) in the restriction framework.

2.2 Continuous parametrization

A major drawback of parametrization of domains with characteristic functions from a practical perspective is that such parametrization does not lend itself to the use of typical nonlinear programming techniques. In particular, note that $\mathcal{A}_\mathcal{O}$ is not a vector space and natural discretization of characteristic functions leads to integer programming problems that are too large in practice to be tractable. For this reason, a "continuous" parametrization of the shape is often adopted by the topology optimization community. This essentially means that the control in the optimization problem is a function that can take values in some continuous interval. Henceforth we shall call this a *sizing function*. This reformulation must be accompanied by a modification of the state equation or addition of a new constraint to the problem so as to recover binary nature of the design. For example, in density methods, the sizing function is a volume fraction or density function, ρ, that takes values in $[0, 1]$ and replaces the characteristic function in the description of the state equation and the objective and constraint functions. Moreover, its appearance in the problem is modified via a *material interpolation function* such that optimal or near optimal density functions only take values of 0 and 1 throughout most of Ω and therefore approximate a characteristic function. For example, in the so-called SIMP formulation (in the restriction setting) the space of admissible designs $\mathcal{A}$ is a sufficiently regular subset of $L^\infty(\Omega; [0, 1])$,[3] and the state equation is given by:

$$\int_\Omega \left[\varepsilon + (1-\varepsilon)\,\rho^p\right] \mathbf{C}\nabla\mathbf{u} : \nabla\mathbf{v} d\mathbf{x} = \int_{\tilde{\Gamma}_N} \mathbf{t}\cdot\mathbf{v} ds, \quad \forall \mathbf{v} \in \mathcal{V}$$

where $p > 1$ is a penalization exponent (Bendsoe 1989; Rozvany et al. 1992; Rozvany 2009). Meanwhile, ρ enters linearly in the geometric constraints such as the volume or perimeter of the design:

$$V(\rho) = \int_\Omega \rho d\mathbf{x}, \qquad P(\rho) = \int_\Omega |\nabla\rho|\, d\mathbf{x}$$

Intuitively, the intermediate volume fractions are penalized as they are assigned a smaller stiffness compared to their contribution to volume. If one can establish that the optimal solution to the SIMP problem is a characteristic function, it follows from the fact that $\rho^p = \rho$ for all $\rho \in \mathcal{A} \cap L^\infty(\Omega; \{0, 1\})$, that enlarging the design space from characteristic functions to volume fraction functions is justified. Unfortunately, this is typically not the case when one is operating in a restriction setting since the admissible densities are smooth and the variation between the extreme values occurs over a transition region. Such results are, however, available for certain discrete formulations of SIMP (Stolpe and Svanberg 2001a; Rietz 2001; Martinez 2005).

We note that there is an alternative rationale for density methods, such as SIMP, that has its roots in the *relaxation* of the optimal design problem (1) (Kohn and Strang 1986a, b, c; Tartar 2000; Cherkaev 2000; Allaire 2001). This so-called material distribution perspective inherently involves the concept of homogenization. The intuition is that severe oscillations of near optimal solutions (the root-cause of the ill-posedness of the classical problem) is taken into account by enlarging the space of admissible designs to include the corresponding generalized shapes. Since these oscillations take place at a fine scale, their effect in such generalized

[3] Here $L^\infty(\Omega; K)$ denotes the space of measurable functions defined on Ω that take values in $K \subseteq \mathbb{R}$. For example, $L^\infty(\Omega; \{0, 1\})$ and $L^\infty(\Omega; [0, 1])$ denote the space of measurable functions that take values in $\{0, 1\}$ and interval $[0, 1]$, respectively.

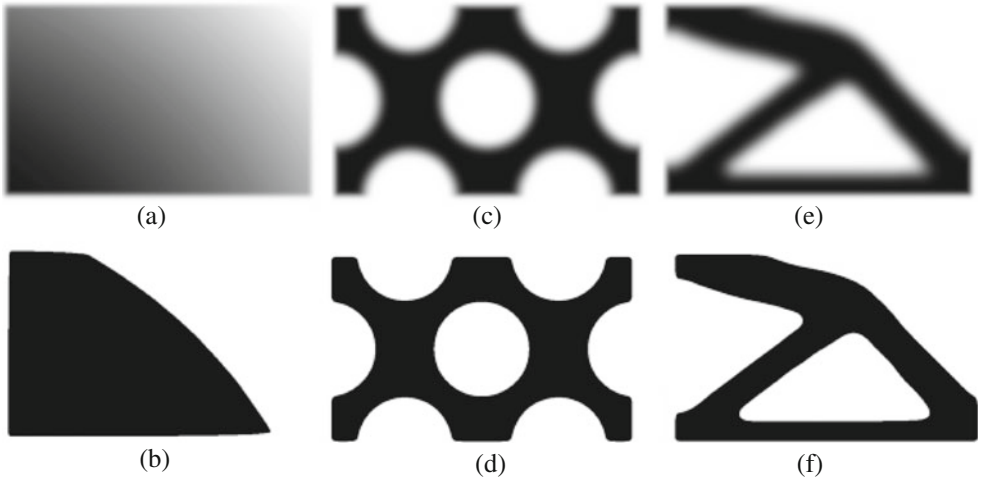


Fig. 2 **a**, **c**, **e** Greyscale plot of three "smooth" density functions ρ taking values in [0, 1]. **b**, **d**, **f** The corresponding interpreted shapes defined by $\chi_{\{\rho \geq 0.5\}}$

designs is captured through the homogenized properties of the corresponding (composite) material in the enlarged space. A density function that measures the volume fraction of these oscillations at the macroscopic scale characterizes each generalized design.

Here, we are adopting a different perspective—that of an approximation of a characteristic function—since we are operating in a restriction setting which is natural for most practical applications of topology optimization where layout or manufacturing constraints are present. It is difficult to extend the relaxation philosophy to the restriction framework since it amounts to distributing material freely at the microscale but subject to a restricted variation at the macroscale. In a sense, this undermines the notion of relaxation from both mathematical and physical perspectives. Nevertheless, one can explicitly construct composite materials that follow the SIMP model (Bendsoe and Sigmund 1999), i.e., ones whose stiffness and volume fraction coincide with SIMP, and so, in principle, the designs can be realized. For yet another physical justification of SIMP based on fictitious fabrication cost, we refer the reader to Rozvany (2009) and Zhou and Rozvany (1991). We note, however, that the SIMP problem without additional regularity conditions imposed on the admissible density functions is still ill-posed.

At this point a natural question arises: what is the space of admissible shapes $\mathcal{O}$ corresponding to the "continuous" and regularized design space $\mathcal{A}$? For example, considering the fact that the optimal density function ρ^* is typically not a characteristic function, one usually interprets the result to obtain a classical shape $\omega^* \subseteq \Omega$ via a post-processing routine. The question is: *in what sense is* ω^* *optimal?*

If the performance of the density functions, in the optimal regime, is well-approximated by that of the corresponding classical shapes, we can then argue that ω^* is "nearly" optimal *when the space of admissible shapes* $\mathcal{O}$ *consists of shapes corresponding to the approximate characteristic functions in* $\mathcal{A}$. Note that in such a scenario, the set of admissible shapes $\mathcal{O}$ is reasonably well-defined and perhaps captures the intention of the regularization scheme while the continuous parametrization setting is justified.

To further illustrate the notion of "post-processing" and what is meant by the "optimal regime," consider the three smooth density functions ρ taking values in [0, 1] shown in Fig. 2. The last two functions approximate a characteristic function: the second row shows the associated characteristic functions defined by $\chi_{\{\rho \geq 0.5\}}$.[4] It is clear that only in the last two cases, ρ and $\chi_{\{\rho \geq 0.5\}}$ are "close." Furthermore, the continuous density formulations of SIMP is set up so that in the optimal regime, only designs of this kind appear, i.e., ones for which $\rho \approx \chi_{\{\rho \geq 0.5\}}$. The continuous parametrization is acceptable if, in addition, the values of objective and constraints functions are well approximated, that is, $f(\rho) \approx f\left(\chi_{\{\rho \geq 0.5\}}\right)$. These conditions are evidently true for compliance minimization with SIMP though a mathematical proof is not available in the continuum setting.

So far in this discussion, the notion of sizing function may seem synonymous with densities.[5] However, we note that the continuous parametrization is not limited to density methods and level set formulations (Allaire et al. 2004; Wang et al. 2003; Belytschko et al. 2003; de Ruiter and Keulen 2004; Van Dijk et al. 2009) can be also placed in the same framework: the level set or implicit function φ can assume both positive and negative values and can be required to belong to the bounded interval $[-\alpha, \alpha]$ for some $\alpha > 0$ (Belytschko et al. 2003). What enters in the state equation and constraint functions, in place of χ_ω, is $H(\varphi)$,

[4] This function takes value of 1 at point $\mathbf{x} \in \Omega$ if $\rho(\mathbf{x}) \geq 0.5$ and is zero otherwise – this is a simple choice of post-processing.

[5] Variants of density methods involve different material interpolation functions, but are similar in spirit (Stolpe and Svanberg 2001b; Bruns 2005).

where H is an approximate Heaviside function. Note that we are not commenting on the choice of the optimization algorithm that is ultimately used in the discrete setting (the evolution of the optimal shape in level set methods in some of the references mentioned above is based on shape sensitivity analysis and motion of the shape boundary). Similar to the density formulations, some constraints must be imposed on the variation of the level set functions so that the resulting optimization problem is well-posed. Also some mechanism must ensure that the level set function is sufficiently steep near the boundary so that the stiffness is near binary throughout the domain.

2.3 Regularity of sizing functions

We now discuss the issue of the regularity of the space of admissible sizing functions. As mentioned before, this is relevant from a theoretical perspective since it is related to the well-posedness of the problem. We emphasize that continuous parametrization (by replacing characteristic functions with sizing functions) by itself does not address this issue. Moreover, restriction provides an appropriate setting for justifying the commonly used Ersatz approach. From a practical point of view, restricting the variations of the sizing function is related to manufacturing constraints on the admissible geometries considered.

Some formulations impose local or global regularity constraints explicitly with the aid of the "continuous" parametrization. For example, in the perimeter constraint setting, the addition of the constraint function (Ambrosio and Buttazzo 1993; Haber et al. 1996; Petersson 1999):

$$g_i(\rho) = \int_\Omega |\nabla \rho| \, d\mathbf{x} - \overline{P}$$

ensures that the admissible densities do not oscillate too much by requiring that the total variation of the design be bounded by the given perimeter $\overline{P}$. In the slope constraint formulation, $\mathcal{A} \subseteq W^{1,\infty}(\Omega)$ (admissible functions are weakly differentiable with essentially bounded derivatives) and

W^{1,inf} is sobolev space

$$g_i(\rho) = \operatorname*{ess\,sup}_{\mathbf{x}\in\Omega} \|\nabla \rho(\mathbf{x})\|_\infty - \overline{G}$$

which means that the gradient of $\rho \in \mathcal{A}$ cannot be too large (i.e., exceed the specified value $\overline{G}$) throughout the extended domain Ω. The discretization of this local constraint leads to a large number of linear constraints for the optimization problem and can be prohibitively expensive. We refer the interested reader to the review papers by Sigmund and Petersson (1998) and Borrvall (2001) on various restriction approaches.

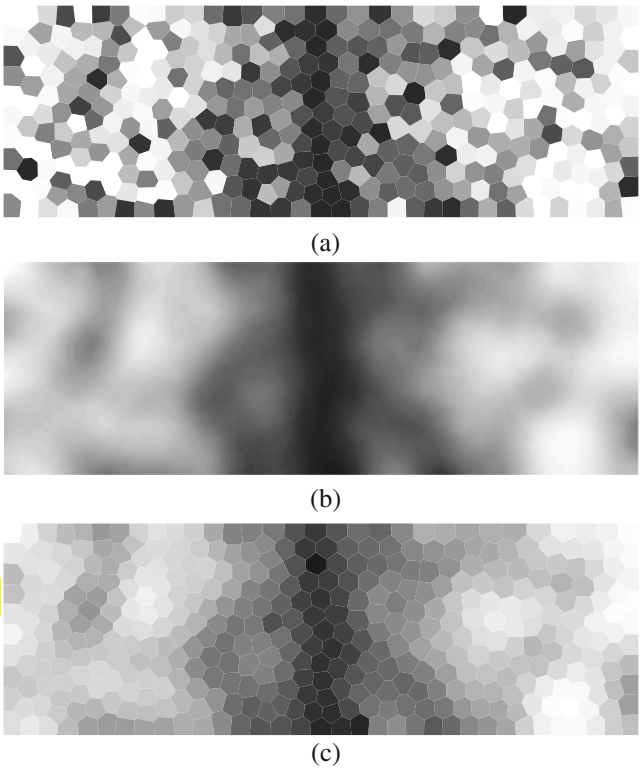
The alternative, emphasized here, is to impose regularity on $\mathcal{A}$ implicitly with the aid of a "regularization" map $\mathcal{P}$. For example, in the popular filtering formulation (Bourdin 2001; Borrvall and Petersson 2001), $\mathcal{A}$ consists of density functions that are produced via convolution with a smooth filter function F, that is,

$$\mathcal{A} = \left\{ \mathcal{P}_F(\eta) : \eta \in L^\infty(\Omega; [0,1]) \right\} \tag{6}$$

where $\mathcal{P}_F$ is the integral operator defined as:

$$\mathcal{P}_F(\eta)(\mathbf{x}) := \int_\Omega F(\mathbf{x}, \overline{\mathbf{x}}) \eta(\overline{\mathbf{x}}) d\overline{\mathbf{x}} \tag{7}$$

The expression (6) states that for each $\rho \in \mathcal{A}$, there exists a measurable function η such that $\rho = \mathcal{P}_F(\eta)$. By the virtue of the properties of the map $\mathcal{P}_F$, the sizing function ρ inherits the smoothness characteristics of the kernel F. Therefore, even if η is rough, ρ is guaranteed to be smooth (see Fig. 3a and b). As we shall discuss in the next section,



(a) (b) (c)

Fig. 3 Illustration of the effects of regularization mapping and its discretization: **a** $\eta_h = \sum_{\ell=1}^N z_\ell \chi_{\Omega_\ell}$ for a random vector of design variables $\mathbf{z} = (z_\ell)_{\ell=1}^N$; **b** $\mathcal{P}_F(\eta_h)$ for the design function η_h shown in **a** and linear filtering kernel F. Note that despite the severe oscillations of η_h, $\mathcal{P}_F(\eta_h)$ has smooth variation dictated by F; **c** $\mathcal{P}_F^h(\eta_h)$ which is the element-wise constant approximation to $\mathcal{P}_F(\eta_h)$ on the same finite element partition based on which η_h is defined

it is the discretization of η that produces the set of design variables for the optimization problem. Thus we can see that the use of $\mathcal{P}_F$ in defining the sizing functions eliminates the need to explicitly impose regularity on ρ.

The linear "hat" kernel of radius R, which is the common choice for filtering, is given by:

$$F(\mathbf{x}, \overline{\mathbf{x}}) = c(\mathbf{x}) \max\left(1 - \frac{|\mathbf{x} - \overline{\mathbf{x}}|}{R}, 0\right) \tag{8}$$

where $c(\mathbf{x})$ is defined as a normalizing coefficient such that

$$\int_\Omega F(\mathbf{x}, \overline{\mathbf{x}}) d\overline{\mathbf{x}} = 1 \tag{9}$$

for all $\mathbf{x} \in \Omega$. It is easy to see that the expression for this coefficient is

$$c(\mathbf{x}) = \left[\int_{B_R(\mathbf{x}) \cap \Omega} \left(1 - \frac{|\mathbf{x} - \mathbf{w}|}{R}\right) d\mathbf{w}\right]^{-1}$$

where $B_R(\mathbf{x})$ is the ball of radius R around $\mathbf{x}$. The condition (9) guarantees that the bounds for the filtered field $\mathcal{P}_F(\eta)$ are the same as those set for the design function η (e.g., 0 and 1 for design space in (6)). We emphasize, however, that in light of the previous discussion on definition of $\mathcal{A}$, the role of the map $\mathcal{P}_F$ is simply to enforce regularity of the density functions in $\mathcal{A}$. This distinction has been made in Sigmund (2007) where the author refers to $\rho \in \mathcal{A}$ as the "physical" density function to distinguish it from $\eta \in L^\infty(\Omega; [0, 1])$.

We can prescribe other layout or manufacturing constraints on admissible shapes via implicit maps in the same manner that the filtering approach enforces smoothness. We illustrate this idea via an example enforcing symmetry and remark that manufacturing constraints such as extrusion, pattern repetition and gradation can be imposed in a similar way (Kosaka and Swan 1999; Almeida et al. 2010; Stromberg et al. 2011). Suppose $\Omega \subseteq \mathbb{R}^2$ is a symmetric domain with respect to the x_1-axis and let $\Omega^+ = \{(x_1, x_2) \in \Omega : x_2 \geq 0\}$. Rather than adding constraints of the form

$$\rho(x_1, x_2) = \rho(x_1, -x_2) \tag{10}$$

for all $\mathbf{x} = (x_1, x_2) \in \Omega$ (which is conceivable in the discrete setting), we can build symmetry into the space of admissible sizing functions by defining the operator $\mathcal{P}_s$ that maps function η defined on Ω^+ to the function $\mathcal{P}_s(\eta)$ defined on Ω with the property that

$$\mathcal{P}_s(\eta)(\mathbf{x}) = \eta(x_1, |x_2|) \tag{11}$$

for every $\mathbf{x} = (x_1, x_2) \in \Omega$ and setting [6]

$$\mathcal{A} = \left\{\mathcal{P}_s(\eta) : \eta \in L^\infty(\Omega^+; [0, 1])\right\}$$

Again this means that $\rho \in \mathcal{A}$ if $\rho = \mathcal{P}_s(\eta)$ for some $\eta \in L^\infty(\Omega^+; [0, 1])$ and so ρ automatically satisfies (10). Given this discussion, it is easy to combine the symmetry and filtering using composition of the two maps $\mathcal{P} = \mathcal{P}_F \circ \mathcal{P}_s$.

2.4 Setting for this paper

Two main features of common well-posed topology optimization formulations are interpolation models and enforcing of regularity of admissible designs. Many topology optimization formulations can be cast in the form of a sizing problem:

$$\inf_{\rho \in \mathcal{A}} f(\rho, \mathbf{u}) \qquad \text{subject to} \qquad g_i(\rho, \mathbf{u}) \leq 0,\ i = 1, \ldots, K \tag{12}$$

where the space of admissible sizing functions is

$$\mathcal{A} = \left\{\mathcal{P}(\eta) : \eta \in L^\infty\left(\Omega; \left[\underline{\rho}, \overline{\rho}\right]\right)\right\} \tag{13}$$

and $\mathbf{u} \in \mathcal{V}$ solves

$$\int_\Omega m_E(\rho) \mathbf{C} \nabla \mathbf{u} : \nabla \mathbf{v} d\mathbf{x} = \int_{\tilde{\Gamma}_N} \mathbf{t} \cdot \mathbf{v} ds, \quad \forall \mathbf{v} \in \mathcal{V} \tag{14}$$

Here m_E is the material interpolation function that relates the value of ρ at a point to the stiffness at that point.[7] Similarly, interpolation functions for other geometric measures such as bound on the perimeter may be needed for the formulation. For example, the compliance minimization problem

$$f(\rho, \mathbf{u}) = \int_{\tilde{\Gamma}_N} \mathbf{t} \cdot \mathbf{u} ds, \qquad g(\rho) = \frac{1}{|\Omega|} \int_\Omega m_V(\rho) d\mathbf{x} - \overline{v}$$

requires one additional interpolation function, m_V, for the volume constraint. Aside from m_V and m_E, in this framework we need to provide bounds, $\underline{\rho}$ and $\overline{\rho}$, as well as the mapping, $\mathcal{P}$.

[6] The distinction between the η and the admissible function $\mathcal{P}_s(\eta)$ should be more apparent here: η is defined on half of the domain while $\mathcal{P}_s(\eta)$ is defined over all of Ω.

[7] m_E essentially determines the dependence of the state equation on the design.

2.5 Discretization

The reformulation of the optimal design problem in the form of distributed sizing optimization (12) lends itself to a tractable discretization and optimization scheme. In particular, the finite element discretization of the design field $\mathcal{A}$ requires partitioning of the extended domain Ω without the need for remeshing as the design is evolving in the course of the optimization. Often the displacement field is discretized based on the FE same partition although this choice may lead to certain numerical artifacts such as checkerboard patterns. We remark that in the restriction setting, such coupled discretization strategies can be shown to be convergent under mesh refinement so the aforementioned numerical instabilities are expected to disappear when a sufficiently fine mesh is used (e.g. when the mesh size is smaller than the filtering radius).

We now discuss the various steps in the discretization process based on one finite element mesh. The goal is to identify the design variables for the discrete optimization problem and how they are related to the parameters defining the state equations and subsequently the cost functional. As we shall see, this involves an approximation of the mapping $\mathcal{P}$ which is usually not discussed in the topology optimization literature.

Let $\mathcal{T}_h = \{\Omega_\ell\}_{\ell=1}^N$ be a partition of Ω, i.e., $\Omega_\ell \cap \Omega_k = \emptyset$ for $\ell \neq k$ and $\cup_\ell \overline{\Omega}_\ell = \overline{\Omega}$ with h denoting the characteristic mesh size. In the analysis of convergence of FE solutions to the solution of (12), h is sent to zero. We intend to identify the discrete counterpart to (12) based on this partition and so in the remainder of this section $\mathcal{T}_h$ is fixed. The *piecewise constant* discretization of $\mathcal{A}$ in (13) is defined as:

$$\mathcal{A}_h = \left\{\mathcal{P}(\eta_h) : \underline{\rho} \leq \eta_h \leq \overline{\rho},\, \eta|_{\Omega_\ell} = \text{const}\ \forall \ell\right\} \tag{15}$$

discretization

In other words, each $\rho \in \mathcal{A}_h$ is the image of map $\mathcal{P}$ acting on a design function η_h that takes a constant value over each element Ω_ℓ. Note that η_h belongs to a finite dimensional space of functions of the form:

$$\eta_h(\mathbf{x}) = \sum_{\ell=1}^N z_\ell \chi_{\Omega_\ell}(\mathbf{x}) \tag{16}$$

where $\chi_{\Omega_\ell}(\mathbf{x})$ is the characteristic function associated with element Ω_ℓ and z_ℓ is the constant value that η_h assumes over Ω_ℓ. Furthermore, (15) is defined such that each $\rho_h \in \mathcal{A}_h$ can be written as $\rho_h = \mathcal{P}(\eta_h)$ for some η_h is the form of (16). Thus each candidate design in $\mathcal{A}_h$ can be defined by a set of design of variables $\mathbf{z} := (z_\ell)_{\ell=1}^N$, which is provided to the optimization algorithm for this partitioning. The only difference between $\mathcal{A}$ and $\mathcal{A}_h$ with the above definition is the restriction placed on η. It is precisely the discretization of the functions η that produces the design variables $\mathbf{z}$ that on the one hand, completely characterize the discrete space of admissible designs $\mathcal{A}_h$ and on the other hand become the parameters passed to the optimization algorithm for sizing.

As mentioned before, it is convenient that the state equation, more specifically the displacement field $\mathcal{V}$, is discretized on the same partition $\mathcal{T}_h$. Let $\mathcal{V}_h$ be this finite dimensional subspace and suppose each $\mathbf{u}_h \in \mathcal{V}_h$ has the expansion

$$\mathbf{u}_h(\mathbf{x}) = \sum_{i=1}^M U_i \mathbf{N}_i(\mathbf{x})$$

that is, $\{\mathbf{N}_i\}_{i=1}^M$ is the basis for $\mathcal{V}_h$ and M is the number of displacement degrees of freedom. The Galerkin approximation to the state equation (14) with $\rho = \rho_h \in \mathcal{A}_h$ can be written as:

$$\mathbf{K}\mathbf{U} = \mathbf{F} \tag{17}$$

where $\mathbf{U} = (U_i)_{i=1}^M$ is the vector of nodal displacements,

$$\mathbf{F}_i = \int_{\tilde{\Gamma}_N} \mathbf{t} \cdot \mathbf{N}_i ds \tag{18}$$

are the nodal loads, and

$$\begin{aligned}\mathbf{K}_{ij} &= \int_\Omega m_E(\rho_h)\mathbf{C}\nabla\mathbf{N}_i : \nabla\mathbf{N}_j d\mathbf{x} \\ &= \sum_{\ell=1}^N \int_{\Omega_\ell} m_E(\rho_h)\mathbf{C}\nabla\mathbf{N}_i : \nabla\mathbf{N}_j d\mathbf{x}\end{aligned} \tag{19}$$

is the stiffness matrix. The integral inside the summation is the (i, j)-th entry of the stiffness matrix for element Ω_ℓ in the global node numbering.

We note that $\rho_h = \mathcal{P}(\eta_h)$ may not be constant over Ω_ℓ and therefore cannot be pulled outside of this integral. Similarly, quantities like the volume of the design cannot be readily computed given the finite element partition. This highlights the fact that in common algorithms for topology optimization an additional approximation step is used beyond what is explicitly stated in the definition of $\mathcal{A}_h$ in (15). This approximation in solving the state equation in fact amounts to the discretization of the operator $\mathcal{P}$ and therefore could be explicitly expressed in the definition of the space of admissible designs $\mathcal{A}_h$.

Typically, in practice, the sizing function ρ_h is replaced by a function $\tilde{\rho}_h$ that is constant over each finite element, that is,

$$\tilde{\rho}_h(\mathbf{x}) = \sum_{\ell=1}^{N} y_\ell \chi_{\Omega_\ell}(\mathbf{x}) \tag{20}$$

One choice for the elemental values y_ℓ is to sample the value of ρ_h at the centroid of element ℓ,

$$y_\ell = \rho_h(\mathbf{x}_\ell^*)$$

where we have denoted by $\mathbf{x}_\ell^*$ the location of the centroid of the element Ω_ℓ. For example, in the case of filtering, we have

$$\begin{aligned} y_\ell &= \rho_h(\mathbf{x}_\ell^*) \\ &= \mathcal{P}_F(\eta_h)(\mathbf{x}_\ell^*) \\ &= \int_\Omega F(\mathbf{x}_\ell^*, \overline{\mathbf{x}})\eta_h(\overline{\mathbf{x}})d\overline{\mathbf{x}} \\ &= \sum_{k=1}^{N} z_k \underbrace{\int_{\Omega_k} F(\mathbf{x}_\ell^*, \overline{\mathbf{x}})d\overline{\mathbf{x}}}_{:=w_{\ell k}} \end{aligned} \tag{21}$$

Collecting the weights computed in a matrix $\mathbf{P} = (w_{\ell k})$, we can relate the elemental values of $\tilde{\rho}_h$ to those of η_h by

$$\mathbf{y} = \mathbf{P}\mathbf{z} \tag{22}$$

Note that the weights $w_{\ell m}$ are independent of the design variables $\mathbf{z}$ and can be computed once at the beginning of the algorithm. Also many of the weights are zero since $\mathbf{x}_\ell^*$ is not in the support of $\mathcal{P}_F(\chi_k)$ when Ω_ℓ and Ω_k are far apart in the mesh. Therefore, $\mathbf{P}$ can be efficiently stored in a sparse matrix. In the Appendix A, it is shown that these weights correspond to the usual discrete filtering formulas used for the linear hat function (8).

The matrix $\mathbf{P}$ must be viewed as the discrete counterpart to the mapping $\mathcal{P}_F$. In fact, such discretization of any *linear* map[8] $\mathcal{P}$ in the definition of $\mathcal{A}$ produces a constant matrix. Let us define the "discretized" map[9]

$$\mathcal{P}_h : \eta \to \sum_{\ell=1}^{N} \mathcal{P}(\eta)(\mathbf{x}_\ell^*)\chi_{\Omega_\ell}$$

[8] Filtering, symmetry, pattern repetition, and extrusion constraints can all be implemented by means of such linear maps

[9] Alternatively we can view $\mathcal{P}_h = \mathcal{I}_h \circ \mathcal{P}$ where $\mathcal{I}_h$ maps any ρ_h to $\tilde{\rho}_h$, that is, $\mathcal{I}_h(\rho) = \sum_{\ell=1}^{N} \rho(\mathbf{x}_\ell^*)\chi_{\Omega_\ell}$

Note that for each piecewise constant design function η_h and linear map $\mathcal{P}$,

$$\begin{aligned} \mathcal{P}(\eta_h)(\mathbf{x}_\ell^*) &= \mathcal{P}\left(\sum_{k=1}^{N} z_k \chi_{\Omega_k}\right)(\mathbf{x}_\ell^*) \\ &= \sum_{k=1}^{N} z_k \mathcal{P}(\chi_{\Omega_k})(\mathbf{x}_\ell^*) = \mathbf{P}\mathbf{z} \end{aligned}$$

where

$$(\mathbf{P})_{\ell k} = \mathcal{P}(\chi_{\Omega_k})(\mathbf{x}_\ell^*) \tag{23}$$

This illustrates the relationship between $\mathcal{P}_h$ and matrix $\mathbf{P}$. Just as the vector $\mathbf{z}$ represents the elemental values of design function η_h, vector $\mathbf{P}\mathbf{z}$ gives the elemental values of the piecewise constant function $\mathcal{P}_h(\eta_h)$.

A more precise description of the space of admissible designs $\mathcal{A}_h$ associated with $\mathcal{T}_h$ that accounts for this approximation is thus given by:

$$\mathcal{A}_h = \left\{\mathcal{P}_h(\eta_h) : \underline{\rho} \le \eta_h \le \overline{\rho},\ \eta_h|_{\Omega_\ell} = \text{const}\ \forall \ell\right\} \tag{24}$$

Figure 3 illustrates the effects of the regularization mapping $\mathcal{P}_F$ with a linear hat kernel F and its discretization $\mathcal{P}_F^h$.

With $\tilde{\rho}_h$ replacing ρ_h in expression (19), which amounts to replacing $\mathcal{P}(\eta_h)$ with $\mathcal{P}_h(\eta_h)$, the stiffness matrix is simplified to:

$$\mathbf{K} = \sum_{\ell=1}^{N} m_E(y_\ell)\mathbf{k}_\ell \tag{25}$$

where $(\mathbf{k}_\ell)_{ij} = \int_{\Omega_\ell} \mathbf{C}\nabla\mathbf{N}_i : \nabla\mathbf{N}_j d\mathbf{x}$ is the ℓth element stiffness matrix. As mentioned before, it is not just the calculation of stiffness matrix that benefits from the approximation $\mathcal{P}$. Evaluation of objective and constraint functions that involve volume and surface integrals are also now greatly simplified. For example,

$$g(\tilde{\rho}_h) = \frac{1}{|\Omega|}\int_\Omega m_V(\tilde{\rho}_h)d\mathbf{x} - \overline{v} = \frac{\sum_{\ell=1}^{N} m_V(y_\ell)\,|\Omega_\ell|}{\sum_{\ell=1}^{N}|\Omega_\ell|} - \overline{v}$$

We note that this additional approximation is sometimes not explicitly considered in the convergence proofs (e.g. in Borrvall and Petersson 2001) but is justifiable since its effect disappears as the mesh size h goes to zero. However, accuracy considerations become relevant. Since there

is only one mesh used, the variation of η_h and accuracy of $\tilde{\rho}_h$ and $\mathbf{u}_h$ are tied to the same FE mesh.

2.6 Discrete form of the minimum compliance problem

We conclude this section by stating the discrete compliance minimization problem defined on $\mathcal{T}_h$:

$$\inf_{\rho_h \in \mathcal{A}_h} \int_{\tilde{\Gamma}_N} \mathbf{t} \cdot \mathbf{u}_h ds \qquad \text{subject to}$$

$$\frac{1}{|\Omega|} \int_{\Omega} m_V(\rho_h) d\mathbf{x} - \overline{v} \leq 0 \tag{26}$$

where the space of admissible functions $\mathcal{A}_h$ is given in (24), and $\mathbf{u}_h \in \mathcal{V}_h = \text{span}\{\mathbf{N}_i\}_{i=1}^{M}$ solves the discretized state equation:

$$\int_{\Omega} m_E(\rho_h) \mathbf{C} \nabla \mathbf{u}_h : \nabla \mathbf{v} d\mathbf{x} = \int_{\tilde{\Gamma}_N} \mathbf{t} \cdot \mathbf{v} ds, \quad \forall \mathbf{v} \in \mathcal{V}_h$$

The only difference between (12) and the continuum problem (26) is that spaces $\mathcal{A}$ and $\mathcal{V}$ are replaced by their finite dimensional counterparts $\mathcal{A}_h$ and $\mathcal{V}_h$.

As shown in the previous subsection, this problem is completely equivalent to the familiar discrete form:[10]

$$\min_{\mathbf{z} \in \left[\underline{\rho}, \overline{\rho}\right]^N} \mathbf{F}^T \mathbf{U} \quad \text{subject to} \quad \frac{\mathbf{A}^T m_V(\mathbf{P}\mathbf{z})}{\mathbf{A}^T \mathbf{1}} - \overline{v} \leq 0 \tag{27}$$

where $\mathbf{F}$ given by (18) is independent of the design variables $\mathbf{z}$, $\mathbf{U}$ solves (17) in which $\mathbf{K}$ depends "linearly" on $m_E(\mathbf{P}\mathbf{z})$ as stated in (25), $\mathbf{A} = (|\Omega_\ell|)$ is the vector of element volumes, and $\mathbf{P}$ is defined in (23).

3 Modular framework, formulation, and optimizer

The structure of the discrete optimization problem (27) allows for separation of the analysis routine from the particular topology optimization formulation chosen. The analysis routine is defined to be the collection of functions in the code that compute the objective and constraint functions and thus require access to mesh information (e.g. the FE analysis and functions computing the volume or perimeter of the design). We can see that for the minimum compliance problem vectors $\mathbf{E} := m_E(\mathbf{y})$ and $\mathbf{V} := m_V(\mathbf{y})$, i.e., the element stiffnesses and volume fractions, are the only "design" related information that need to be provided to the analysis functions. The analysis functions need not know about the choice of interpolations functions which corresponds to the choice of sizing parametrization or the mapping $\mathcal{P}$ that places constraints on the design space. Therefore a general implementation of topology optimization in the spirit of this discussion must be structured in such a way that the finite element routines contain no information related to the specific topology optimization formulation. A clear advantage of this approach is that the analysis functions can be extended, developed and modified independently.

We can also see that certain quantities used in the analysis functions, such as element areas $\mathbf{A}$ and stiffness matrices $\mathbf{k}_\ell$ as well as the "connectivity" of the global stiffness matrix $\mathbf{K}$ need to be computed only once in the course of the optimization algorithm. Therefore, for the sake of efficiency of the implementation, one should store the invariant quantities. We emphasize that this is independent of the uniformity of the FE partition $\mathcal{T}_h$, i.e., whether or not it is a structured mesh. Similarly the matrix $\mathbf{P}$ can be computed once in a pre-processing step and stored.

To use a gradient-based optimization algorithm for solving the discrete problem (27), which is the case in our Matlab code, we must also compute the gradient of the cost functions with respect to the design variables $\mathbf{z}$. The sensitivity analysis can be "separated" along the same lines. The analysis functions would compute the sensitivities of the cost functions *with respect to their own internal parameters*. Note that by the chain rule:

$$\frac{\partial g_i}{\partial z_k} = \sum_{\ell=1}^{N} \left(\frac{\partial E_\ell}{\partial z_k} \frac{\partial g_i}{\partial E_\ell} + \frac{\partial V_\ell}{\partial z_k} \frac{\partial g_i}{\partial V_\ell} \right)$$

or in equivalent vector form:

$$\frac{\partial g_i}{\partial \mathbf{z}} = \frac{\partial \mathbf{E}}{\partial \mathbf{z}} \frac{\partial g_i}{\partial \mathbf{E}} + \frac{\partial \mathbf{V}}{\partial \mathbf{z}} \frac{\partial g_i}{\partial \mathbf{V}} \tag{28}$$

Therefore the analysis functions would only compute the sensitivities of g_i with respect to the internal parameters $\mathbf{E}$ and $\mathbf{V}$. In the compliance problem, $f = \mathbf{F}^T \mathbf{U}$ and we have:

$$\frac{\partial f}{\partial E_\ell} = -\mathbf{U}^T \frac{\partial \mathbf{K}}{\partial E_\ell} \mathbf{U} = -\mathbf{U}^T \mathbf{k}_\ell \mathbf{U}, \qquad \frac{\partial f}{\partial V_\ell} = 0 \tag{29}$$

This means that the FE function that computes the objective function also returns the negative of the element strain

[10] In the remainder of the paper, we understand $m_E(\mathbf{y})$ and $m_V(\mathbf{y})$ as vectors with entries $m_E(y_\ell)$ and $m_V(y_\ell)$.

energies as the vector of sensitivities $\partial f/\partial \mathbf{E}$. The remaining terms in (28) depend on the formulation, i.e., how the design variables $\mathbf{z}$ are related to the analysis parameters. For example, $\mathbf{E} = m_E(\mathbf{Pz})$ and $\mathbf{V} = m_V(\mathbf{Pz})$ implies:

$$\frac{\partial \mathbf{E}}{\partial \mathbf{z}} = \mathbf{P}^T J_{m_E}(\mathbf{Pz}), \qquad \frac{\partial \mathbf{V}}{\partial \mathbf{z}} = \mathbf{P}^T J_{m_V}(\mathbf{Pz}) \tag{30}$$

where $J_{m_E}(\mathbf{y}) := \mathrm{diag}(m'_E(y_1), \ldots, m'_E(y_N))$ is the Jacobian matrix of map m_E. The evaluation of expression (28) is carried out outside the analysis routine and the result, $\partial g_i/\partial \mathbf{z}$, is passed to the optimizer to update the values of the design variables.

In the same way that the analysis routine ought to be separated from the rest of topology optimization algorithm, the optimizer responsible for updating the values of the design variables should also be kept separate. This is perhaps better known in the topology optimization community as optimizers like MMA (Svanberg 1987) are generally used as black-box routines. Of course, it is important to know and understand the inner workings of the optimizer. As noted in Groenwold and Etman (2008), certain choices of approximations of cost functionals in a sequential optimization algorithm lead to an Optimality-Criteria type update expression when there is only one constraint. An attempt to directly change the OC expression, for example, to accommodate a more general formulation such as the one presented here may not be readily obvious. In order to account for the general box constraints (i.e., upper and lower bounds other than 0–1), one needs an additional intermediate variable. Also, the volume constraint in the compliance problem should be linearized if one wishes to adopt the sequential approximation interpretation. This is not an issue in SIMP since the volume functional is already linear in the design variables. A brief discussion on the update scheme is provided in the Appendix B.

4 Matlab implementation

In this section we describe the Matlab implementation of the discrete topology optimization problem (27). The function `PolyTop` is the kernel of the code that contains the optimizer and analysis routines, including the FE routine and functions responsible for computing cost functional and their sensitivities. The prototype is written to solve the compliance minimization problem, but specific functions are designated to compute the objective and constraint functions. All the parameters related to topology optimization that link design variables with the analysis parameters (e.g. filter matrix, material interpolation functions), as well as the finite element model (e.g. the mesh, load and support boundary conditions for the state equation) are defined outside in `PolyScript`, a Matlab script that calls the kernel. This functional decoupling allows the user to run various formulations or discretizations without the need to change the kernel. Also interpolation and regularization functions can be changed easily for the purposes of continuation, for example, on the penalization parameter or filtering radius.

4.1 Input data and `PolyScript`

All the input and internal parameters of the code are collected in two Matlab `struct` arrays. One `struct`, called `fem`, contains all the FE-related parameters while the other, `opt`, has the variables pertaining to the topology optimization formulation and optimizer. Table 1 shows the list of fields in these structure arrays. Note that some of the `fem` fields are populated inside the `PolyTop` kernel unless they are already specified. Also the user has access to all the model parameters since these structures reside in the Matlab workspace.

In the representative implementation of `PolyScript`, the auxiliary functions `PolyMesher` and `PolyFilter` are called to initialize the finite element mesh and construct the linear filtering matrix $\mathbf{P}$ with input radius R. The polygonal mesh generator, `PolyMesher`, is described in the companion paper (Talischi et al. 2011).[11] The implementation of `PolyFilter` is short but efficient and is provided as supplementary material. To construct the filtering matrix, this function simply computes the distances between the centroids of the elements in the mesh and defines the filtering weight based on the input filtering radius (cf. (23) and Appendix A). If the filtering radius is too small, the function returns the identity matrix. One can also intentionally input a negative value to get the identity matrix. In such a case, each design variable corresponds to an element property—this is sometimes known as the "element-based" approach in the literature and, as discussed in Section 2, corresponds to an ill-posed continuum formulation and not surprisingly suffers from mesh-dependency. However, it may be used, for example, to recover Michell-type solutions, provided that numerical instabilities such as checkerboard patterns are suppressed.

[11] We caution that the `PolyMesher` files should be added to the Matlab path in order for this call to be successful. This mesh generator can be of course replaced by any other mesh generator (e.g. `distMesh` (Persson and Strang 2004)) as long as the node list, element connectivity cell and load and support vectors follow the same format.

Table 1 List of fields in the input structures. The fields marked with the superscript †, if empty, are populated inside `PolyTop`

`fem` field	
`fem.NNode`	Number of nodes
`fem.NElem`	Number of elements
`fem.Node`	[`NNode` × 2] array of nodes
`fem.Element`	[`NElem` × Var] cell array of elements
`fem.Supp`	[`NSupp` × 3] support array
`fem.Load`	[`NLoad` × 3] load array
`fem.Nu0`	Poisson's ratio of solid material
`fem.E0`	Young's modulus of solid material
`fem.Reg`	Tag for regular meshes
`fem.ElemNDof`†	Array showing number of DOFs of elements
`fem.ShapeFnc`†	Cell array with tabulated shape functions and weights
`fem.k`†	Array of local stiffness matrix entries
`fem.i`†	Index array for sparse assembly of `fem.k`
`fem.j`†	Index array for sparse assembly of `fem.k`
`fem.e`†	Array of element IDs corresponding to `fem.k`
`fem.ElemArea`†	Array of element areas
`fem.F`†	Global load vector
`fem.FreeDofs`†	Array of free degrees of freedom
`opt` field	
`opt.zMin`	Lower bound for design variables
`opt.zMax`	Upper bound for design variables
`opt.zIni`	Initial array of design variables
`opt.MatIntFnc`	Handle to material interpolation function
`opt.P`	Matrix that maps design to element variables
`opt.VolFrac`	Specified volume fraction constraint
`opt.Tol`	Convergence tolerance on design variables
`opt.MaxIter`	Max. number of optimization iterations
`opt.OCMove`	Allowable move step in the OC update scheme
`opt.OCEta`	Exponent used in the OC update scheme

Another auxiliary function given is the material interpolation function, whose function handle is passed to the kernel via the `opt.MatIntFnc` field. Given an input vector $\mathbf{y}$, this function must return the vector of the corresponding stiffnesses and volume fractions arrays $\mathbf{E} = m_E(\mathbf{y})$ and $\mathbf{V} = m_V(\mathbf{y})$ of the same length as $\mathbf{y}$ and the sensitivity vectors $\partial\mathbf{E}/\partial\mathbf{y} := m'_E(\mathbf{y})$ and $\partial\mathbf{V}/\partial\mathbf{y} := m'_V(\mathbf{y})$.[12] For example in the case of SIMP:

```
function [E,dEdy,V,dVdy]
 = MatIntFnc(y,penal)
  eps = 1e-4;
  E = eps+(1-eps)*y.^penal;
  V = y;
  dEdy = (1-eps)*penal*y.^(penal-1);
  dVdy = ones(size(y,1),1);
```

[12] Again we understand $m'_E(\mathbf{y})$ and $m'_V(\mathbf{y})$ as vectors with elements $m'_E(y_\ell)$ and $m'_V(y_\ell)$. Essentially $m'_E(\mathbf{y})$ and $m'_V(\mathbf{y})$ are the diagonal entries of Jacobian matrices $J_{m_E}(\mathbf{y})$ and $J_{m_V}(\mathbf{y})$.

Note also that the stiffness of the void region can be set here. *This again highlights the fact that the material interpolation model and the Ersatz approximation are independent of the choice of regularization scheme or bounds on the design variables*. The material interpolation function can accept input parameters (e.g. the penalization exponent `penal` in SIMP) so that, if desired, they can be changed by the user in the workspace (e.g., for the purpose of continuation).

4.2 Comments on the functions in `PolyTop`

The kernel `PolyTop` possesses fewer than 190 lines, of which 116 lines pertain to the finite element analysis including 81 lines for the element stiffness calculations for polygonal elements. In this section we explain the implementation of various functions in the kernel.

Main function The function begins with initialization of the iteration parameters `Iter`, `Tol` and `Change` as

well as the initial analysis parameters for the initial guess `z=opt.zIni` on line 8. The function `InitialPlot`, executed on line 10, plots a triangulation of the mesh using the `patch` function and outputs a handle to the figure and also vector `FigData` that will be used to update the patch colors. The iterative optimization algorithm is nested inside the while-loop that terminates when either the maximum number of iterations, `opt.MaxIter`, is exceeded or the change in design variables, `Change`, is smaller than the prescribed tolerance. In each iteration, the current values of `E` and `V` are passed to analysis functions `ObjectiveFnc` and `ConstraintFnc` to, as their names suggest, compute the values and sensitivities of the objective and constraint functions (lines 14 and 15). The sensitivities with respect to the design variables are computed on lines 17 and 18 according to expressions (28) and (30). Note that the entry-by-entry multiplication `dEdy.*dfdE` produces the same vector as the matrix-vector multiplication $J_{m_E}(\mathbf{y})\,\frac{\partial f}{\partial \mathbf{E}}$ because the Jacobian matrix $J_{m_E}(\mathbf{y})$ is diagonal and has `dEdy` as its diagonal elements. The design sensitivities are then passed to `UpdateScheme` to obtain the next vector design variables. The analysis parameters for the new design are computed on line 21. Finally, the new value of the objective function and the maximum change in the current iteration are printed to the screen and element patch colors in the plot are updated to reflect the updated design.

`FEAnalysis` Before describing the functions responsible for computing the objective and constraint functions, we discuss the implementation of the FE analysis in function `FEAnalysis` and its specific features pertaining to the topology optimization. As noted before, the structure of the global stiffness matrix is such that not only element stiffness matrices are invariant but also the connectivity of the mesh is fixed.

Efficient assembly of the global stiffness matrix in Matlab makes use of the built-in function `sparse` that generates a sparse matrix from an input array of values using two index arrays of the same length. To assemble the global stiffness matrix, we place all the entries of the local stiffness matrices (with the stiffness of reference solid material) in a single vector `fem.k`.[13] The index vectors `fem.i` and `fem.j` contain the global degrees of freedom of each entry in `fem.k`. In particular, this indicates to the `sparse` function that `fem.k(q)` should be placed in row `fem.i(q)` and column `fem.j(q)` of the global stiffness matrix. These vectors are computed and stored on lines 69–79. Our convention is that $2n-1$ and $2n$ are the horizontal and vertical degrees of freedom corresponding to the nth node. By design, the `sparse` function sums the entries in `fem.k` that have the same corresponding indices. To account for the different values of `E` that must be assigned to each element to represent the current design, we compute and store an additional index vector `fem.e`. This array keeps track of the elements to which the local stiffness matrix entries in `fem.k` belong. So the expression `E(fem.e)` returns the elongated list of element stiffnesses that has the same size as the index vectors. The entry-by-entry multiplication `E(fem.e).*fem.k` then appropriately scales the local stiffness matrix values in `fem.k`. The *assembly* of the global stiffness matrix is therefore accomplished with the following line of code:

```
K = sparse(fem.i,fem.j,E(fem.e).*fem.k);
```

Lines 80–89, executed only during the first iteration, compute the list of free degrees of freedoms `fem.FreeDofs` and global load vector `fem.F` from the given `Supp` and `Load` matrices.[14] Note that only four lines of code are executed in the `FEAnalysis` function to obtain the nodal displacement after the initialization in the first iteration (lines 91–94). We have included line 92, following the recommendation of (Andreassen et al. 2010), to ensure that the "backslash" solver recognizes the stiffness matrix `K` as symmetric, which in turn reduces the time for solving the linear system.

`ObjectiveFnc` This function computes the objective function of the optimization problem using the current values of `E` and `V`. Note again that this function is not given any information related to the optimization formulation. In the prototype implementation, we evaluate the compliance simply using the inner product of the global force and displacement vector, which are computed in call to the `FEAnalysis` function. The function also returns

[13] The local stiffness matrices are obtained by calling the function `LocalK` on either line 68 or line 70. If the mesh is known to be uniform, by setting the `fem.Reg` tag equal to 1 in the initialization of the `fem` structure, only one such call is made (on line 68) and thus there is no overhead for repeated calculation of the same element stiffness matrices.

[14] These matrices are expected to have the following format: `Supp` must have three columns, the first holding the node number, the second and third columns giving support conditions for that node in the x- and y-direction, respectively. Value of 0 indicates that the node is free, and value of 1 specifies a fixed node. The nodal load vector `Load` is structured in a similar way, except for the values in the second and third columns, which represent the magnitude of the x- and y-components of the force.

the sensitivity of the objective function with respect to `E` and `V`. In the case of compliance, `dfdV` is the zero vector while `dfdE` is the array consisting of negative of the element strain energies (cf. (29)). Since the index vectors `fem.i`, `fem.j`, and `fem.k` contain all the relevant FE information, they can be used to efficiently compute the strain energies. For element Ω_ℓ, we need to compute $-\sum \mathbf{U}_i\,(\mathbf{k}_\ell)_{ij}\,\mathbf{U}_j$ where the sum is taken over all DOFs i and j of element Ω_ℓ. This requires summing the block of `-U(fem.i).*fem.k.*U(fem.j)` that corresponds to element ℓ. Lines 30-33 carry out this computation using `cumsum`, the cumulative sum function in Matlab.

`ConstraintFnc` This function computes the constraint function of the optimization problem, which for the compliance problem, is the volume fraction constraint (cf. (27)). Similar to the initialization step in `FEAnalysis`, the vector of areas is computed only once. Lines 43–45 compute the value and sensitivity of this function with respect to `E` and `V`.

`UpdateScheme` The input to this function consists of the gradients of the objective and constraint functions `dfdz` and `dgdz`, the current set of design variables `z0`, and the current value of the constraint function `g`. With this information, we can compute, for example, the approximate constraint function: the expression `g+dgdz'*(zNew-z0)` on line 56 gives $g_{\text{app}}(\mathbf{z}^{\text{new}})$, the value of the linearized constraint function at the candidate design variables. The implementation of the update scheme is consistent with the material in Appendix B. The bisection method is used (similar to the 88- and 99-line codes) to solve the dual problem. The move limit M is defined to be `opt.OCMove*(zMax-zMin)` on line 49.

`LocalK` This function computes the local stiffness matrix of the isoparametric polygonal elements. Because the polygonal shape functions are affine, an isoparametric mapping from regular n-gons (the so-called "reference" elements) to any convex polygon can be constructed using these shape functions. Furthermore, for $n = 3$ and $n = 4$, the resulting elements coincide with the well-known linear triangle and bilinear quadrilateral. The implementation of this isoparametric formulation and notation used follows the standard conventions[15] in most finite element textbooks (see, for example, Hughes 2000).

[15]Regarding the connection between (25) and the well-known expression $\int_{\Omega_\ell} \mathbf{B}_I^T \mathbf{D} \mathbf{B}_J d\mathbf{x}$ for the element stiffness matrix, we refer the reader to Section 2.8 of Hughes (2000).

Though closed-form expressions for the polygonal shape functions can be obtained (see appendix of Tabarraei and Sukumar 2006), we use a more costly geometric construction in this code for the sake of brevity, as discussed below. However, we eliminate the overhead associated with redundant shape function calculations by computing the needed quantities only once. For an isoparametric element, only the values of shape functions and their gradients at the integration points of the reference element are needed. This can be seen from the quadrature loop on lines 100–110 for computing the local stiffness matrix. The shape function values and the quadrature weights are computed in function `TabShapeFnc` and stored in `fem.ShapeFnc` (the description is given below). Note that this approach only affects the overhead of computing the element stiffness matrices in the initialization process.

`TabShapeFnc` This function populates the field `fem.ShapeFnc` which contains the tabulated values of shape functions and their gradients at the integration points of the reference element along with the associated quadrature weights. `fem.ShapeFnc` is a cell of length $N_{\max}$, the maximum number of nodes for an element in the input mesh. The nth cell, `fem.ShapeFnc{n}`, is itself a structure array with three fields `N`, `dNdxi`, and `W` whose values are obtained from the reference n-gon (see lines 115–125). The only uses of these shape function values in `PolyTop` are in `LocalK` function on lines 99 and 101.

`PolyShapeFnc` This function computes the set of linear shape functions for a reference n-gon at an interior point $\boldsymbol{\xi}$. The Wachspress shape function corresponding to node i, $1 \le i \le n$, is defined as (Sukumar and Tabarraei 2004):

$$N_i(\boldsymbol{\xi}) = \frac{\alpha_i(\boldsymbol{\xi})}{\sum_{j=1}^{n} \alpha_j(\boldsymbol{\xi})} \tag{31}$$

where α_i are the interpolants of the form:[16]

$$\alpha_i(\boldsymbol{\xi}) = \frac{A(\mathbf{p}_{i-1}, \mathbf{p}_i, \mathbf{p}_{i+1})}{A(\mathbf{p}_{i-1}, \mathbf{p}_i, \boldsymbol{\xi}) A(\mathbf{p}_i, \mathbf{p}_{i+1}, \boldsymbol{\xi})}$$

Here A denotes the area of the triangle formed by its arguments (see Fig. 4a). Since the reference element is a regular polygon, $A(\mathbf{p}_{i-1}, \mathbf{p}_i, \mathbf{p}_{i+1})$ is the same for all i and thus can be factored out of expression (31). Adopting the notation

[16]By convention, we set $\mathbf{p}_{n+1} = \mathbf{p}_1$ in this expression.

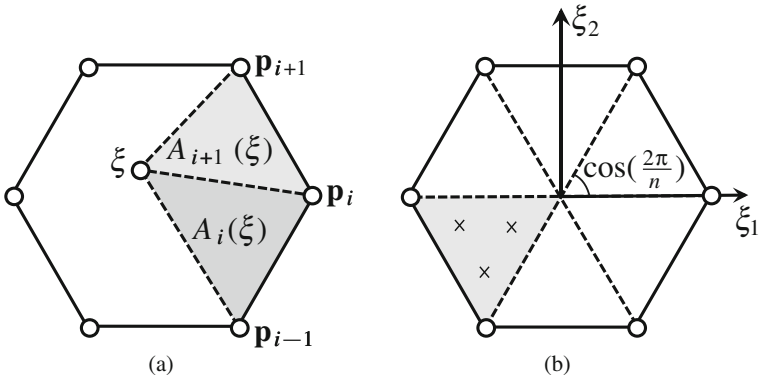


Fig. 4 **a** Illustration of the triangular areas used to define α_i in expression (32). **b** Triangulation of the reference regular polygon and integration points defined on each triangle

$A_i(\boldsymbol{\xi}) := A(\mathbf{p}_{i-1}, \mathbf{p}_i, \boldsymbol{\xi})$, we obtain the simplified formula for the interpolants:

$$\alpha_i(\boldsymbol{\xi}) = \frac{1}{A_i(\boldsymbol{\xi}) A_{i+1}(\boldsymbol{\xi})} \tag{32}$$

In the code, on lines 131–136, the area of the triangles formed by $\boldsymbol{\xi}$ and the vertices as well as their gradient with respect to it are computed using expressions:

$$A_i(\boldsymbol{\xi}) = \frac{1}{2} \begin{vmatrix} \xi_1 & \xi_2 & 1 \\ p_{1,i-1} & p_{2,i-1} & 1 \\ p_{1,i} & p_{2,i} & 1 \end{vmatrix},$$

$$\frac{\partial A_i}{\partial \xi_1} = \frac{1}{2}\left(p_{2,i-1} - p_{2,i}\right), \quad \frac{\partial A_i}{\partial \xi_2} = \frac{1}{2}\left(p_{1,i} - p_{1,i-1}\right)$$

The derivatives of the interpolant are simply given by (computed on lines 140 and 141)

$$\frac{\partial \alpha_i}{\partial \xi_k} = -\alpha_i \left(\frac{1}{A_i} \frac{\partial A_i}{\partial \xi_k} + \frac{1}{A_{i+1}} \frac{\partial A_{i+1}}{\partial \xi_k} \right), \quad k = 1, 2$$

and from (31) we have the following expression for the shape function gradients (lines 147):

$$\frac{\partial N_i}{\partial \xi_k} = \frac{1}{\sum_{j=1}^{n} \alpha_j} \left(\frac{\partial \alpha_i}{\partial \xi_k} - N_i \sum_{j=1}^{n} \frac{\partial \alpha_j}{\partial \xi_k} \right), \quad k = 1, 2$$

`PolyTrnglt` This function generates a directed triangulation of the reference n-gon by connecting its vertices to the input point $\boldsymbol{\xi}$ that lies in its interior. As shown in the Fig. 4b, the nodes of the reference n-gon are located at $\mathbf{p}_i = (\cos 2\pi i/n, \sin 2\pi i/n)$. This function is used both in the definition of the polygonal shape functions and the quadrature rule.

`PolyQuad` One scheme to carry out the integration on the reference n-gon is to divide it into n triangles (by connecting the origin to the vertices) and use well-known quadrature rules on each triangle. For the verification problem in the next section, we have used three integration points per triangle (see Fig. 4b). We note that numerical integration can alternatively be carried out using special quadrature rules recently developed for polygonal domains (see, for example, Mousavi et al. 2009; Natarajan et al. 2009) that are more accurate.

`TriQuad, TriShape` These functions are called by `PolyQuad` and provide the usual quadrature rule for the reference triangle and its linear shape functions.

5 Numerical results

In this section, we present numerical results for benchmark compliance minimization problems to demonstrate the versatility of the code. For all results, the Ersatz parameter ε was set to 10^{-4} and the Young's modulus and Poisson's ratio of the solid phase were taken to be $E_0 = 1$ and $\nu = 0.3$, respectively. Also the maximum tolerance for the change in design variables was taken to be 1%. Unless otherwise stated, `opt.P` was set to the linear filtering matrix $\mathbf{P}$ given by (33) and computed by the auxiliary function `PolyFilter`.

The first example is the MBB beam problem (Olhoff et al. 1991) whose domain, loading and support conditions are shown in Fig. 5a. A mesh of 5,000 polygonal elements was generated using `PolyMesher` (Talischi et al. 2011). The final result shown in Fig. 5b was obtained using a linear filter of radius 0.04 (the rectangular domain has unit height) and the SIMP model with continuation performed on the penalty parameter p as follows: the value of p was increased

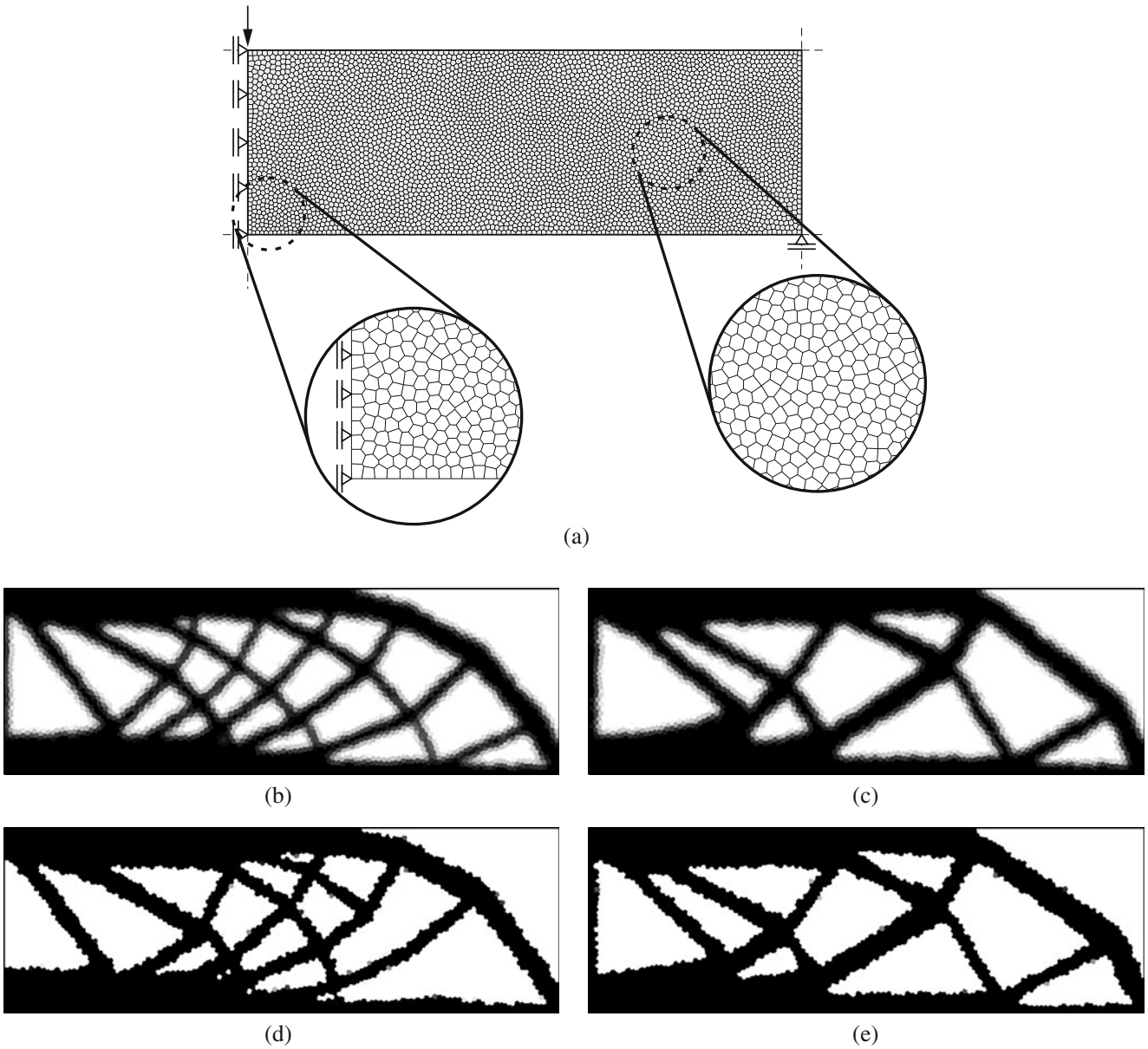


Fig. 5 MBB beam problem **a** domain geometry, loading and boundary conditions; The mesh is composed of 5,000 elements (27 4-gons, 1,028 5-gons, 3,325 6-gons, 618 7-gons, and 2 8-gons) and 9,922 nodes. Final topologies using **b** SIMP, **c** RAMP, **d** SIMP with Heaviside filtering and **e** RAMP with Heaviside filtering

from 1 to 4 using increments of size 0.5 and for each value of p, a maximum of 150 iterations was allowed (by setting `opt.MaxIter=150`). The continuation was implemented in `PolyScript`, outside the `PolyTop` kernel, as follows:

```
for penal = 1:0.5:4
  opt.MatIntFnc
    = @(y)MatIntFnc(y,'SIMP',penal);
  [opt.zIni,V,fem] = PolyTop(fem,opt);
end
```

This is the default example in `PolyScript` provided in the supplementary material. Upon running the script (for example, by simply entering `PolyScript` in the command prompt), a call is made to `PolyMesher` to generate the mesh,[17] the struct arrays `fem` and `opt` are defined and the `PolyTop` kernel is executed inside this continuation loop.

[17] The Voronoi mesh may be different from the one used in our results due to the random placement of seeds in `PolyMesher`.

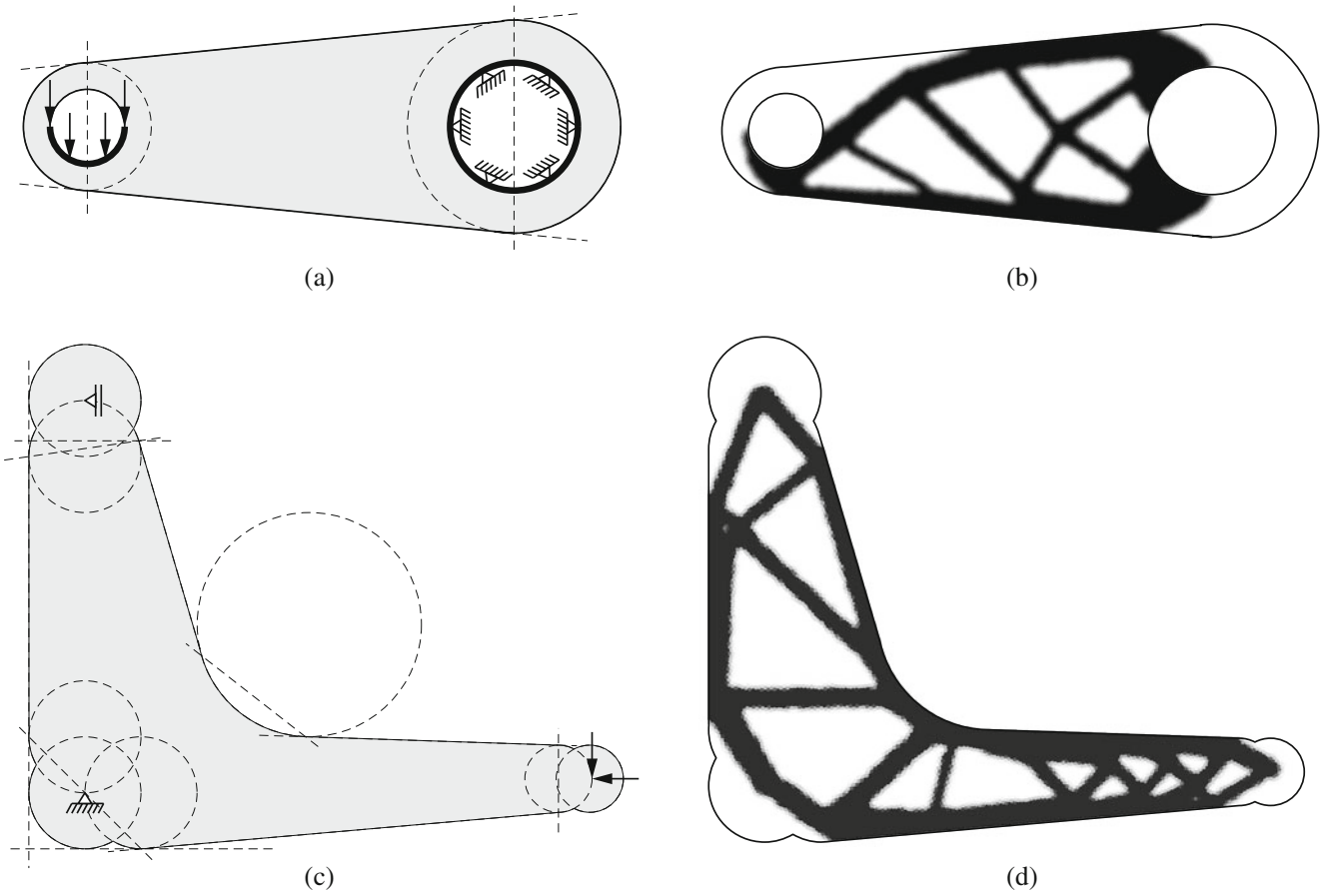


Fig. 6 Compliance problems with non-trivial domain geometries: **a** domain of wrench problem, $R = 0.03$, $\overline{v} = 0.4$; **b** final topology for wrench problem with RAMP functions; **c** domain of suspension triangle problem, $R = 0.25$, $\overline{v} = 0.45$, magnitude of horizontal load is eight times larger than the vertical load; **d** final topology for suspension problem with RAMP functions

To use another material interpolation function, we only need to change the `MatIntFnc` outside the `PolyTop` kernel. For example, the RAMP functions (Stolpe and Svanberg 2001b; Bendsøe and Sigmund 2003) defined by

$$m_E(\rho) = \varepsilon + (1-\varepsilon)\,\frac{\rho}{1+q(1-\rho)}, \quad m_V(\rho) = \rho$$

were used to generate the result shown in Fig. 5c. The parameter q was initially set to zero and continuation was subsequently carried out by doubling its value from 1 to 128. As the same filtering radius was used, the difference between the two results highlights the difference between the performance of these interpolation functions.

Yet another example of a material interpolation model is the Heaviside projection (Guest et al. 2004). Even though it is typically considered a nonlinear filtering approach, it is easy to see that it can be cast in the framework of Section 2.5 where the regularization map is linear. More specifically, if it is used in conjunction with SIMP, the material interpolation functions are given as:

$$m_E(\rho) = \varepsilon + (1-\varepsilon)\,[h(\rho)]^p$$
$$m_V(\rho) = h(\rho)$$

where

$$h(x) = 1 - \exp(-\beta x) + x\exp(-\beta)$$

is the approximate Heaviside and ρ belong to the space of admissible designs defined in (6), that is, ρ is obtained from the usual linear filtering. This shows that *the Heaviside filtering essentially amounts to a modification of the material interpolation functions.* The additional parameter β controls the amount of grey scales that appears in the optimal solutions. Note, however, that SIMP penalization plays a crucial role since with $p = 1$, we have $m_E(\rho) \approx m_V(\rho)$

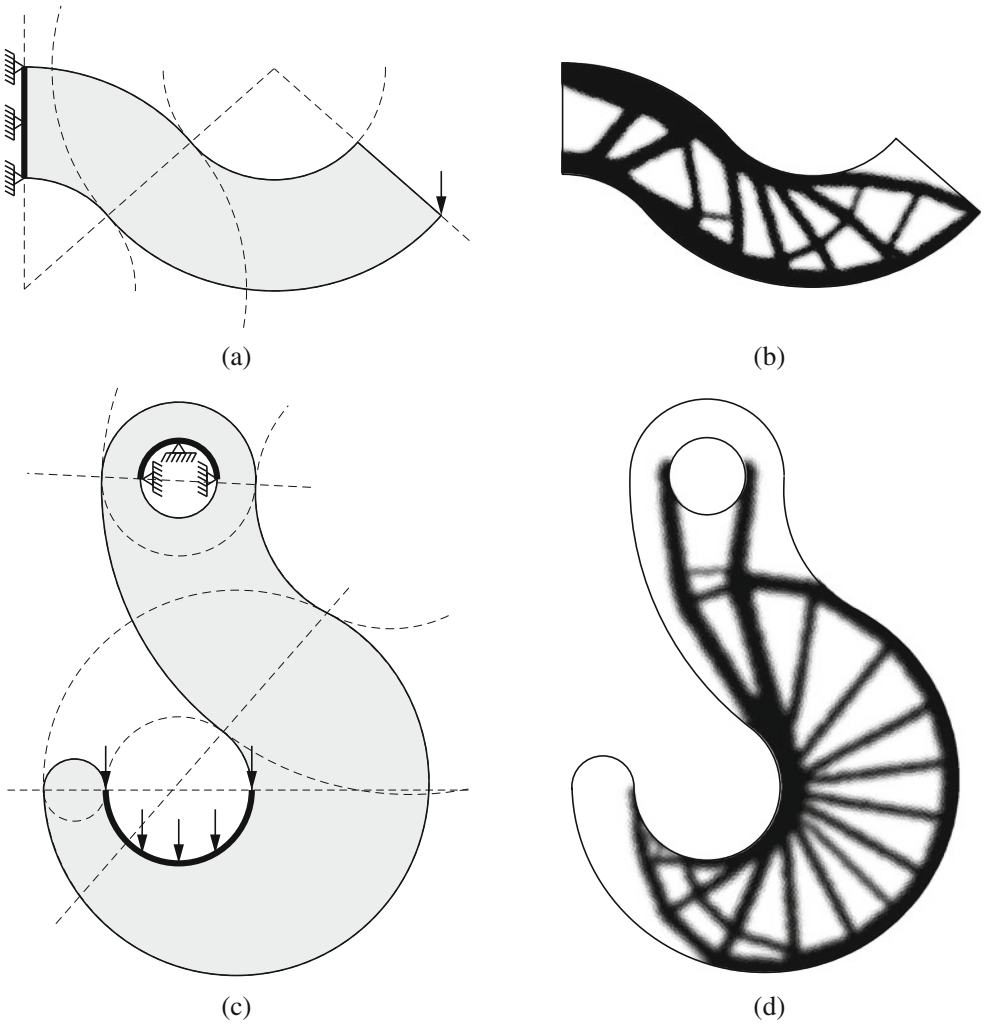


Fig. 7 Compliance problems with non-trivial domain geometries: **a** domain of serpentine beam problem, $R = 0.25$, $\overline{v} = 0.55$; **b** final topology for serpentine beam problem with SIMP functions; **c** domain of hook problem, $R = 2.0$, $\overline{v} = 0.40$; **d** final topology for hook beam problem with SIMP functions

for any ρ and so the optimal solution will consist mostly of intermediate densities no matter how large β is. One can similarly define material interpolation functions for the Heaviside scheme based on the RAMP functions. Figure 5d and 5e show the Heaviside filtering results with SIMP and RAMP with the same penalty parameters and radius of filtering as the previous results. Moreover the continuation of p and q were interleaved with the continuation on value of β as follows: the value of β is initially set to one and for each increment of p or q, it is doubled. As expected, the results depend on the manner in which the continuation of the penalty parameter and β is carried out.

The next set of results are for problems with non-trivial domain geometries or loading and support conditions. The design domains with the boundary conditions are shown in the left column of Figs. 6 and 7. The wrench and suspension triangle were solved with RAMP functions while the hook and serpentine beam problems were solved with the SIMP interpolation function, both using the same continuation schemes as before. Even though in the 99- and 88-line codes, location of holes can be prescribed by means of passive elements, it is evident that describing arbitrary geometries such as those shown here on a regular grid becomes cumbersome if not intractable. Moreover, the use of an unstructured mesh is necessary if one is to resolve

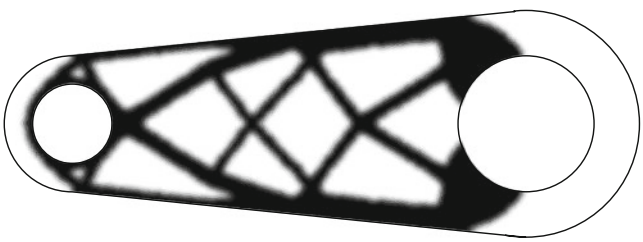


Fig. 8 Symmetric solution to the wrench problem

Table 2 Breakdown of the code runtime for 200 optimization iterations: times are in seconds with percentage of total runtime of PolyScript provided in the parentheses

Mesh size	90 × 30	150 × 50	300 × 100	600 × 200
Computing **P**	0.25 (1.6%)	0.876 (2.1%)	9.12 (4.9%)	93.79 (9.2%)
Populating fem	0.20 (1.3%)	0.50 (1.2%)	2.05 (1.1%)	8.08 (0.8%)
Assembling **K**	5.86 (37.8%)	16.73 (41.1%)	69.50 (37.2%)	289.09 (28.5%)
Solving $\mathbf{KU} = \mathbf{F}$	3.56 (23.0%)	11.64 (28.6%)	60.06 (32.1%)	302.95 (29.8%)
Mapping **z** and **E**, **V**	0.17 (1.1%)	0.86 (2.1%)	12.86 (6.9%)	203.82 (20.1%)
Compliance sensitivities	0.94 (6.1%)	2.76 (6.8%)	12.94 (6.9%)	50.82 (5.0%)
Plotting the solutions	2.53 (16.3%)	3.15 (7.7%)	6.20 (3.3%)	18.76 (1.8%)
OC update	0.96 (6.2%)	1.99 (4.9%)	5.17 (2.8%)	14.39 (1.4%)
Total time of PolyScript	15.50	40.74	187.02	1,015.89

the domain geometry, accurately specify the design loads and compute the structure's response. Problems of this sort typically arise in the practical applications, for example, the suspension triangle (cf. Fig. 6c and 6d) is an industrial application of topology optimization presented by Allaire and Jouve (2005).

Next, we show how symmetry and similar layout constraints can be enforced in the code. We consider the wrench problem and wish to enforce symmetry along the horizontal axis. An unstructured but symmetric polygonal mesh was generated using PolyMesher (see Section 6.2 of Talischi et al. 2011). For $\ell = 1, \ldots, N/2$, element $\Omega_{\ell+N/2}$ is the reflection of element Ω_ℓ about the horizontal axis. As discussed in Section 2.3, symmetry can be enforced via the mapping $\mathcal{P}_s$ given in (11). For this mesh, the matrix associated with the discretization of $\mathcal{P}_s$ (cf. (23)) is defined by:

$$(\mathbf{P}_s)_{\ell k} = \begin{cases} 1, & \text{if } \ell = k \text{ or } \ell = k + N/2 \\ 0, & \text{otherwise} \end{cases}$$

where $\ell = 1, \ldots N$ and $k = 1, \ldots, N/2$. To apply filtering as well, opt.P was set as:

$$\mathbf{P} = \mathbf{P}_F \mathbf{P}_s$$

where $\mathbf{P}_F$ is the linear filtering matrix. Figure 8 shows the optimal wrench topology with symmetry enforced along the horizontal axis. This problem has asymmetric loading and imposing symmetry amounts to requiring the optimal design to also withstand identical upward loads. One can show that this symmetric topology is identical to the solution for the multiple load problem consisting of mirrored loading cases with equal weights.

6 Efficiency

Finally, we discuss the breakdown of the computational cost of the code and compare its efficiency to the 88 line code (Andreassen et al. 2010). For the purposes of comparison, the MBB problem was solved using regular square meshes (see Section 6.5 of Talischi et al. 2011 on how to generate such meshes using PolyMesher). A filtering radius of $R = 0.12$ was used.[18] In the case of the 88 line, the input tag ft was set to 2 to use the "consistent" density filter. In both cases, the SIMP penalty parameter was fixed at $p = 3$ and 200 optimizations iterations were performed on a machnine with an Intel(R) Core i7, 3.33 GHz processor and 24.0 GB of RAM running Matlab R2009b. Of course, the two codes produced identical topologies at each iteration.

The breakdown of the runtime of PolyScript is shown in Table 2. We can see that the initialization time, which includes populating the fem structure and computing the filtering matrix, were modest for all mesh sizes. During subsequent iterations of PolyTop, assembling the stiffness matrix and solving the linear system of FE equations constituted the largest portion of the code runtime. The relative cost of mapping the design variables **z** to analysis quantities **E** and **V** and associated analysis sensitivities (lines 14, 15 and 21 of the code) increased with the mesh size. Note that both operations involve multiplication with the filtering matrix which demands more memory for larger meshes. This increase was most significant for the 600 × 200 mesh perhaps due to the large size of the filtering matrix relative to available memory.

[18] The larger the radius of filtering, the longer it takes to compute the filtering matrix and also the larger the amount of memory needed to store it.

Table 3 Runtime comparison of `PolyScript` with the 88 line code (Andreassen et al. 2010) (times are reported in seconds for 200 optimization iterations)

Mesh size	90 × 30	150 × 50	300 × 100	600 × 200
Total time of `PolyScript`	15.5	**40.7**	**187**	**1,016**
Total time of 88 line	**14.8**	44.4	360	4,463

Table 3 lists the runtime of `PolyScript` and the 88 line code. We note that the present Matlab implementation was faster than the 88 line for every mesh except the smallest mesh. Also, the difference between the runtime of the two codes became larger as the mesh size grew. This suggests that as the size of the problem increases, `PolyTop` becomes more efficient than the 88 line—notice that there is more than a four-fold speedup for the 600 × 200 mesh. This observation prompted an investigation to identify the source of discrepancy.

We noticed that in every bisection iteration inside the OC update function of the 88 line code, the design volume is computed by summing the "physical" densities obtained from multiplying the candidate design variables by the filtering matrix. But this volume function can be rewritten as

$$V(\mathbf{z}) = \sum_{\ell=1}^{N} (\mathbf{P}\mathbf{z})_\ell = \mathbf{1}^T (\mathbf{P}\mathbf{z}) = \left(\mathbf{1}^T\mathbf{P}\right)\mathbf{z} = \left(\mathbf{P}^T\mathbf{1}\right)^T \mathbf{z}$$

where $\mathbf{1}$ is the vector of length N with unit entries. Observe that the vector $\mathbf{P}^T\mathbf{1}$ can be computed once so that the calculation of V is subsequently reduced to taking the inner product of this vector with $\mathbf{z}$, thereby minimizing the need for costly multiplications by $\mathbf{P}$ in every bisection step.

Though the above expression is not explicitly used in `PolyTop`, the decoupling of the OC scheme from the analysis routine naturally leads to this more efficient calculation. Note that $\mathbf{P}^T\mathbf{1}$ is in fact the sensitivity vector `dgdz` provided to the `UpdateScheme` function in `PolyTop`.[19] The update equations are based on the linearization of the constraint function $g(\mathbf{z})$ in the form (see Appendix B):

$$g(\mathbf{z}) = g(\mathbf{z}_0) + \left(\frac{\partial g}{\partial \mathbf{z}}\right)^T (\mathbf{z} - \mathbf{z}_0)$$

where $\mathbf{z}_0$ is the design variables from the current iteration. Since the volume constraint is linear, this is identical to the expression for the volume function (the reader can verify the equivalence of the two expressions). This observation is perhaps further illustration of the virtue of decoupling philosophy advocated here.

We conclude this section with a final remark regarding the overhead associated with computing multiple element stiffness matrices. As mentioned before, the initialization of `fem.k` was done computing the element stiffness matrix only once (by setting `fem.Reg=1`) for the purposes of comparison with the 88 line. However, even without the use of `fem.Reg` tag, the cost associated with the repeated element stiffness matrix calculations as a percentage of total cost is small. For example, for the mesh of 300 × 100 quads, this took 6.12 s which constituted 3.3% of the total cost of 200 optimization iterations. Likewise, for a polygonal mesh of 10,000 elements, the element matrix calculations took 8.39 s which was 5.8% of the total cost of 200 optimization iterations.

7 Conclusions and extensions

We have presented a general framework for topology optimization using unstructured polygonal FE meshes in arbitrary domains, including a modular Matlab code called `PolyTop`. In this code, the analysis routine and optimization algorithm are separated from the specific choice of topology optimization formulation. In fact, the FE and sensitivity analysis routines contain no information related to the formulation and thus can be extended, maintained, developed, and/or modified independently. In a companion paper, we have provided a general purposed mesh generator for polygonal elements, called `PolyMesher` (Talischi et al. 2011), which was also written in Matlab. Both `PolyMesher` and `PolyTop` allow users to solve topology optimization problems in arbitrary domains, rather than the Cartesian domains that have plagued the literature. With these codes, we hope that the community will move beyond Cartesian domains and explore general domains, which are typical of practical engineering problems (see Figs. 6–8). We hope that the modularity and flexibility offered by `PolyTop` will be a motivating factor for the community to explore its framework in other problems (implicit function formulations, topology optimization for fluids, etc.) beyond what is covered in this paper.

Acknowledgments The first two authors acknowledge the support by the Department of Energy Computational Science Graduate Fellowship Program of the Office of Science and National Nuclear Security Administration in the Department of Energy under contract DE-FG02-97ER25308. The last two authors acknowledge the financial support by Tecgraf (Group of Technology in Computer Graphics), PUC-Rio, Rio de Janeiro, Brazil.

[19] Note, however, that in `PolyTop` the volume function is normalized by the volume of the entire domain and elements can have different areas (so $\mathbf{1}$ is replaced by $\mathbf{A}$ in `PolyTop`). Also, the constraint function is defined as the difference between the normalized volume and specified volume fraction $\overline{v}$.

Appendix A: Filtering matrix for the linear kernel

We can derive the usual discrete filtering formulas, corresponding to the linear hat filter (8), by sampling the integrand in (21) at the element centroids:

$$
\begin{aligned}
w_{\ell k} &= \int_{\Omega_k} F(\mathbf{x}_\ell^*, \overline{\mathbf{x}}) d\overline{\mathbf{x}} \\
&= c(\mathbf{x}_\ell^*) \int_{\Omega_k} \max\left(1 - \frac{\left|\mathbf{x}_\ell^* - \overline{\mathbf{x}}\right|}{R}, 0\right) d\overline{\mathbf{x}} \\
&\approx c(\mathbf{x}_\ell^*) \left|\Omega_k\right| \max\left(1 - \frac{\left|\mathbf{x}_\ell^* - \mathbf{x}_k^*\right|}{R}, 0\right)
\end{aligned}
$$

If we similarly approximate the integral defining $c(\mathbf{x}_\ell^*)$ and denote by $S(\ell)$ the set of indices of the elements Ω_k whose centroid falls within radius R of the centroid of element Ω_ℓ, i.e., $\left|\mathbf{x}_\ell^* - \mathbf{x}_k^*\right| \le R$, we can then write filtered field as:

$$
y_\ell = \frac{\sum_{k \in S(\ell)} z_k \left|\Omega_k\right| \left(1 - \left|\mathbf{x}_\ell^* - \mathbf{x}_k^*\right| / R\right)}{\sum_{k \in S(\ell)} \left|\Omega_k\right| \left(1 - \left|\mathbf{x}_\ell^* - \mathbf{x}_k^*\right| / R\right)}
$$

which is the commonly used expression. We note that often the $|\Omega_k|$ terms are omitted because the underlying mesh is uniform. We also reiterate that the vector representation of this expression is more appropriate for implementation in Matlab. Thus, the filter matrix $\mathbf{P}$ given by

$$
(\mathbf{P})_{\ell k} = \frac{\max\left(0, \left|\Omega_k\right| \left(1 - \left|\mathbf{x}_\ell^* - \mathbf{x}_k^*\right| / R\right)\right)}{\sum_{k \in S(\ell)} \left|\Omega_k\right| \left(1 - \left|\mathbf{x}_\ell^* - \mathbf{x}_k^*\right| / R\right)} \tag{33}
$$

is stored as a sparse matrix. The `PolyFilter` function used to compute this matrix for the examples in this paper is provided below:

```
%-----------------------------------------------------------------------%
function [P] = PolyFilter(fem,R)
if R<0, P = speye(fem.NElem); return; end %P is set to identity when R<0
ElemCtrd = zeros(fem.NElem,2);
for el = 1:fem.NElem        %Compute the centroids of all the elements
  vx=fem.Node(fem.Element{el},1); vy=fem.Node(fem.Element{el},2);
  temp = vx.*vy([2:end 1])-vy.*vx([2:end 1]);
  A = 0.5*sum(temp);
  ElemCtrd(el,1) = 1/(6*A)*sum((vx+vx([2:end 1])).*temp);
  ElemCtrd(el,2) = 1/(6*A)*sum((vy+vy([2:end 1])).*temp);
end
[d] = DistPntSets(ElemCtrd,ElemCtrd,R);  %Obtain distance values & indices
P = sparse(d(:,1),d(:,2),1-d(:,3)/R);    %Assemble the filtering matrix
P = spdiags(1./sum(P,2),0,fem.NElem,fem.NElem)*P;
%---------------------------------- COMPUTE DISTANCE BETWEEN TWO POINT SETS
function [d] = DistPntSets(PS1,PS2,R)
d = cell(size(PS1,1),1);
for el = 1:size(PS1,1)       %Compute the distance information
    dist = sqrt((PS1(el,1)-PS2(:,1)).^2 + (PS1(el,2)-PS2(:,2)).^2);
    [I,J] = find(dist<=R);   %Find the indices for distances less that R
    d{el} = [I,J+(el-1),dist(I)];
end
d = cell2mat(d);             %Matrix of indices and distance value
%-----------------------------------------------------------------------%
```

Appendix B: Update scheme

A basic iterative method in structural optimization consists of replacing the objective and constraint functions with their approximations at the current design point. That is, one solves the following approximate problem in each iteration:

$$
\min_{\mathbf{z}} f_{\text{app}}(\mathbf{z}) \quad \text{subject to} \quad g_{\text{app}}(\mathbf{z}) \le \mathbf{0}, \quad \mathbf{z} \in \left[\underline{\rho}, \overline{\rho}\right]^N \tag{34}
$$

where f_{app} and g_{app} are approximations to the objective and constraint functions in (27) obtained from the first order Taylor expansion in some suitable (and physically reasonable) intermediate variables. To obtain the OC-type update scheme, f is linearized in exponential intermediate variables[20]

$$\left(\frac{z_\ell - \underline{\rho}}{\overline{\rho} - \underline{\rho}}\right)^a$$

using the current value of the design variables $\mathbf{z} = \mathbf{z}^0$, which yields:

$$f_{\text{app}}(\mathbf{z}) = f\left(\mathbf{z}^0\right) + \sum_{\ell=1}^{N} \left.\frac{\partial f}{\partial z_\ell}\right|_{\mathbf{z}=\mathbf{z}^0} \frac{1}{a}\left(z_\ell^0 - \underline{\rho}\right) \times \left[\left(\frac{z_\ell - \underline{\rho}}{z_\ell^0 - \underline{\rho}}\right)^a - 1\right] \tag{35}$$

The constraint function is approximated linearly in the design variables:

$$g_{\text{app}}(\mathbf{z}) = g\left(\mathbf{z}^0\right) + \sum_{\ell=1}^{N} \left.\frac{\partial g}{\partial z_\ell}\right|_{\mathbf{z}=\mathbf{z}^0} \left(z_\ell - z_\ell^0\right) \tag{36}$$

The condition of optimality provides the relationship between the Lagrange multiplier λ and each design variable z_ℓ (separability of the approximation is used here):

$$\frac{\partial f_{\text{app}}}{\partial z_\ell} + \lambda \frac{\partial g_{\text{app}}}{\partial z_\ell} = 0, \quad \ell = 1, \ldots, N$$

The value of the Lagrange multiplier is obtained by solving the dual problem, for example, via the bi-section method.

[20]The significance of approximating response dependent cost functions in "reciprocal" variables, i.e., when $a = -1$, are discussed in Groenwold and Etman (2008) and references therein. For $a = 1$, one recovers the usual Taylor linearization.

Upon substitution of (35) and (36), the above equation reduces to:

$$\left(\frac{z_\ell - \underline{\rho}}{z_\ell^0 - \underline{\rho}}\right)^{1-a} = \frac{-\left.\frac{\partial f}{\partial z_\ell}\right|_{\mathbf{z}=\mathbf{z}^0}}{\lambda \left.\frac{\partial g}{\partial z_\ell}\right|_{\mathbf{z}=\mathbf{z}^0}} := B_\ell$$

Thus, we get an explicit expression for optimal z_ℓ (in terms of λ), denoted by z_ℓ^*, which will be the candidate for the next iteration:

$$z_\ell^* = \underline{\rho} + (B_\ell)^{\frac{1}{1-a}}\left(z_\ell^0 - \underline{\rho}\right) \tag{37}$$

The quantity $\eta = 1/(1-a)$ is sometimes referred to as the damping coefficient. For reciprocal approximation, $a = -1$ and so $\eta = 1/2$. This is the value usually used for compliance minimization.

Since the approximation of the objective and constraints functions are generally accurate only near the current design point $\mathbf{z}^0$, the candidate design variable is not accepted unless it lies in a search region, defined based on a move limit. Also, we need to make sure that z_ℓ^{new} satisfies the box constraints. With this in mind, we define the next design point as:

$$z_\ell^{\text{new}} = \begin{cases} z_\ell^+, & z_\ell^* \geq z_\ell^+ \\ z_\ell^-, & z_\ell^* \leq z_\ell^- \\ z_\ell^*, & \text{otherwise} \end{cases}$$

where the z_ℓ^+ and z_ℓ^- are the bounds for the search region given by:

$$z_\ell^- = \max\left(\underline{\rho}, z_\ell^0 - M\right)$$

$$z_\ell^+ = \min\left(\overline{\rho}, z_\ell^0 + M\right)$$

where M is the move limit, specified as a certain fixed fraction of $\left(\overline{\rho} - \underline{\rho}\right)$. For more information on this derivation, we refer the reader to Groenwold and Etman (2008).

Appendix C: PolyScript

```
%------------------------------ PolyScript --------------------------------%
% Ref: C Talischi, GH Paulino, A Pereira, IFM Menezes, "PolyTop: A Matlab %
% implementation of a general topology optimization framework using       %
% unstructured polygonal finite element meshes", Struct Multidisc Optim,  %
% DOI 10.1007/s00158-011-0696-x                                           %
%-------------------------------------------------------------------------%

%% ------------------------------------------------------ CREATE 'fem' STRUCT
[Node,Element,Supp,Load] = PolyMesher(@MbbDomain,5000,30);
fem = struct(...
  'NNode',size(Node,1),...     % Number of nodes
  'NElem',size(Element,1),...  % Number of elements
  'Node',Node,...              % [NNode x 2] array of nodes
  'Element',{Element},...      % [NElement x Var] cell array of elements
  'Supp',Supp,...              % Array of supports
  'Load',Load,...              % Array of loads
  'Nu0',0.3,...                % Poisson's ratio of solid material
  'E0',1.0,...                 % Young's modulus of solid material
  'Reg',0 ...                  % Tag for regular meshes
   );
%% ------------------------------------------------------ CREATE 'opt' STRUCT
R = 0.04;
VolFrac = 0.5;
m = @(y)MatIntFnc(y,'SIMP',3);
P = PolyFilter(fem,R);
zIni = VolFrac*ones(size(P,2),1);
opt = struct(...
  'zMin',0.0,...               % Lower bound for design variables
  'zMax',1.0,...               % Upper bound for design variables
  'zIni',zIni,...              % Initial design variables
  'MatIntFnc',m,...            % Handle to material interpolation fnc.
  'P',P,...                    % Matrix that maps design to element vars.
  'VolFrac',VolFrac,...        % Specified volume fraction cosntraint
  'Tol',0.01,...               % Convergence tolerance on design vars.
  'MaxIter',150,...            % Max. number of optimization iterations
  'OCMove',0.2,...             % Allowable move step in OC update scheme
  'OCEta',0.5 ...              % Exponent used in OC update scheme
   );
%% ------------------------------------------------------------ RUN 'PolyTop'
figure;
for penal = 1:0.5:4        %Continuation on the penalty parameter
   disp(['current p: ', num2str(penal)]);
   opt.MatIntFnc = @(y)MatIntFnc(y,'SIMP',penal);
   [opt.zIni,V,fem] = PolyTop(fem,opt);
end
%% -------------------------------------------------------------------------
```

Appendix D: `PolyTop`

```
%---------------------------- PolyTop ----------------------------------%
% Ref: C Talischi, GH Paulino, A Pereira, IFM Menezes, "PolyTop: A Matlab %
% implementation of a general topology optimization framework using       %
% unstructured polygonal finite element meshes", Struct Multidisc Optim,  %
% DOI 10.1007/s00158-011-0696-x                                           %
%-----------------------------------------------------------------------%
function [z,V,fem] = PolyTop(fem,opt)
Iter=0; Tol=opt.Tol*(opt.zMax-opt.zMin); Change=2*Tol; z=opt.zIni; P=opt.P;
[E,dEdy,V,dVdy] = opt.MatIntFnc(P*z);
[FigHandle,FigData] = InitialPlot(fem,V);
while (Iter<opt.MaxIter) && (Change>Tol)
  Iter = Iter + 1;
  %Compute cost functionals and analysis sensitivities
  [f,dfdE,dfdV,fem] = ObjectiveFnc(fem,E,V);
  [g,dgdE,dgdV,fem] = ConstraintFnc(fem,E,V,opt.VolFrac);
  %Compute design sensitivities
  dfdz = P'*(dEdy.*dfdE + dVdy.*dfdV);
  dgdz = P'*(dEdy.*dgdE + dVdy.*dgdV);
  %Update design variable and analysis parameters
  [z,Change] = UpdateScheme(dfdz,g,dgdz,z,opt);
  [E,dEdy,V,dVdy] = opt.MatIntFnc(P*z);
  %Output results
  fprintf('It: %i \t Objective: %1.3f\tChange: %1.3f\n',Iter,f,Change);
  set(FigHandle,'FaceColor','flat','CData',1-V(FigData)); drawnow
end
%-------------------------------------------------------- OBJECTIVE FUNCTION
function [f,dfdE,dfdV,fem] = ObjectiveFnc(fem,E,V)
[U,fem] = FEAnalysis(fem,E);
f = dot(fem.F,U);
temp = cumsum(-U(fem.i).*fem.k.*U(fem.j));
temp = temp(cumsum(fem.ElemNDof.^2));
dfdE = [temp(1);temp(2:end)-temp(1:end-1)];
dfdV = zeros(size(V));
%------------------------------------------------------ CONSTRAINT FUNCTION
function [g,dgdE,dgdV,fem] = ConstraintFnc(fem,E,V,VolFrac)
if ~isfield(fem,'ElemArea')
  fem.ElemArea = zeros(fem.NElem,1);
  for el=1:fem.NElem
    vx=fem.Node(fem.Element{el},1); vy=fem.Node(fem.Element{el},2);
    fem.ElemArea(el) = 0.5*sum(vx.*vy([2:end 1])-vy.*vx([2:end 1]));
  end
end
g = sum(fem.ElemArea.*V)/sum(fem.ElemArea)-VolFrac;
dgdE = zeros(size(E));
dgdV = fem.ElemArea/sum(fem.ElemArea);
```

```
%--------------------------------------------- OPTIMALITY CRITERIA UPDATE
function [zNew,Change] = UpdateScheme(dfdz,g,dgdz,z0,opt)
zMin=opt.zMin; zMax=opt.zMax;
move=opt.OCMove*(zMax-zMin); eta=opt.OCEta;
l1=0; l2=1e6;
while l2-l1 > 1e-4
  lmid = 0.5*(l1+l2);
  B = -(dfdz./dgdz)/lmid;
  zCnd = zMin+(z0-zMin).*B.^eta;
  zNew = max(max(min(min(zCnd,z0+move),zMax),z0-move),zMin);
  if (g+dgdz'*(zNew-z0)>0),  l1=lmid;
  else                       l2=lmid;  end
end
Change = max(abs(zNew-z0))/(zMax-zMin);
%------------------------------------------------------------------ FE-ANALYSIS
function [U,fem] = FEAnalysis(fem,E)
if ~isfield(fem,'k')
  fem.ElemNDof = 2*cellfun(@length,fem.Element); % # of DOFs per element
  fem.i = zeros(sum(fem.ElemNDof.^2),1);
  fem.j=fem.i; fem.k=fem.i; fem.e=fem.i;
  index = 0;
  if ~isfield(fem,'ShapeFnc'), fem=TabShapeFnc(fem); end
  if fem.Reg, Ke=LocalK(fem,fem.Element{1}); end
  for el = 1:fem.NElem
    if ~fem.Reg,  Ke=LocalK(fem,fem.Element{el}); end
    NDof = fem.ElemNDof(el);
    eDof = reshape([2*fem.Element{el}-1;2*fem.Element{el}],NDof,1);
    I=repmat(eDof ,1,NDof); J=I';
    fem.i(index+1:index+NDof^2) = I(:);
    fem.j(index+1:index+NDof^2) = J(:);
    fem.k(index+1:index+NDof^2) = Ke(:);
    fem.e(index+1:index+NDof^2) = el;
    index = index + NDof^2;
  end
  NLoad = size(fem.Load,1);
  fem.F = zeros(2*fem.NNode,1);  %external load vector
  fem.F(2*fem.Load(1:NLoad,1)-1) = fem.Load(1:NLoad,2);  %x-crdnt
  fem.F(2*fem.Load(1:NLoad,1))   = fem.Load(1:NLoad,3);  %y-crdnt
  NSupp = size(fem.Supp,1);
  FixedDofs = [fem.Supp(1:NSupp,2).*(2*fem.Supp(1:NSupp,1)-1);
               fem.Supp(1:NSupp,3).*(2*fem.Supp(1:NSupp,1))];
  FixedDofs = FixedDofs(FixedDofs>0);
  AllDofs   = [1:2*fem.NNode];
  fem.FreeDofs = setdiff(AllDofs,FixedDofs);
end
K = sparse(fem.i,fem.j,E(fem.e).*fem.k);
K = (K+K')/2;
U = zeros(2*fem.NNode,1);
U(fem.FreeDofs,:) = K(fem.FreeDofs,fem.FreeDofs)\fem.F(fem.FreeDofs,:);
```

```
%---------------------------------------------- ELEMENT STIFFNESS MATRIX
function [Ke] = LocalK(fem,eNode)
D=fem.E0/(1-fem.Nu0^2)*[1fem.Nu00;fem.Nu0 1 0;00(1-fem.Nu0)/2];%plane stress
nn=length(eNode); Ke=zeros(2*nn,2*nn);
W = fem.ShapeFnc{nn}.W;
for q = 1:length(W)  %quadrature loop
  dNdxi = fem.ShapeFnc{nn}.dNdxi(:,:,q);
  J0 = fem.Node(eNode,:)'*dNdxi;
  dNdx = dNdxi/J0;
  B = zeros(3,2*nn);
  B(1,1:2:2*nn) = dNdx(:,1)';
  B(2,2:2:2*nn) = dNdx(:,2)';
  B(3,1:2:2*nn) = dNdx(:,2)';
  B(3,2:2:2*nn) = dNdx(:,1)';
  Ke = Ke+B'*D*B*W(q)*det(J0);
end
%---------------------------------------------- TABULATE SHAPE FUNCTIONS
function fem = TabShapeFnc(fem)
ElemNNode = cellfun(@length,fem.Element); % number of nodes per element
fem.ShapeFnc = cell(max(ElemNNode),1);
for nn = min(ElemNNode):max(ElemNNode)
  [W,Q] = PolyQuad(nn);
  fem.ShapeFnc{nn}.W = W;
  fem.ShapeFnc{nn}.N = zeros(nn,1,size(W,1));
  fem.ShapeFnc{nn}.dNdxi = zeros(nn,2,size(W,1));
  for q = 1:size(W,1)
    [N,dNdxi] = PolyShapeFnc(nn,Q(q,:));
    fem.ShapeFnc{nn}.N(:,:,q) = N;
    fem.ShapeFnc{nn}.dNdxi(:,:,q) = dNdxi;
  end
end
%---------------------------------------------- POLYGONAL SHAPE FUNCTIONS
function [N,dNdxi] = PolyShapeFnc(nn,xi)
N=zeros(nn,1); alpha=zeros(nn,1); dNdxi=zeros(nn,2); dalpha=zeros(nn,2);
sum_alpha=0.0; sum_dalpha=zeros(1,2); A=zeros(nn,1); dA=zeros(nn,2);
[p,Tri] = PolyTrnglt(nn,xi);
for i=1:nn
  sctr = Tri(i,:); pT = p(sctr,:);
  A(i) = 1/2*det([pT,ones(3,1)]);
  dA(i,1) = 1/2*(pT(3,2)-pT(2,2));
  dA(i,2) = 1/2*(pT(2,1)-pT(3,1));
end
A=[A(nn,:);A]; dA=[dA(nn,:);dA];
for i=1:nn
  alpha(i) = 1/(A(i)*A(i+1));
  dalpha(i,1) = -alpha(i)*(dA(i,1)/A(i)+dA(i+1,1)/A(i+1));
  dalpha(i,2) = -alpha(i)*(dA(i,2)/A(i)+dA(i+1,2)/A(i+1));
  sum_alpha = sum_alpha + alpha(i);
  sum_dalpha(1:2) = sum_dalpha(1:2)+dalpha(i,1:2);
end
for i=1:nn
  N(i) = alpha(i)/sum_alpha;
  dNdxi(i,1:2) = (dalpha(i,1:2)-N(i)*sum_dalpha(1:2))/sum_alpha;
end
```

```
%------------------------------------------------------ POLYGON TRIANGULATION
function [p,Tri] = PolyTrnglt(nn,xi)
p = [cos(2*pi*((1:nn))/nn); sin(2*pi*((1:nn))/nn)]';
p = [p; xi];
Tri = zeros(nn,3); Tri(1:nn,1)=nn+1;
Tri(1:nn,2)=1:nn; Tri(1:nn,3)=2:nn+1; Tri(nn,3)=1;
%------------------------------------------------------ POLYGONAL QUADRATURE
function [weight,point] = PolyQuad(nn)
[W,Q]= TriQuad;                    %integration pnts & wgts for ref. triangle
[p,Tri] = PolyTrnglt(nn,[0 0]);  %triangulate from origin
point=zeros(nn*length(W),2); weight=zeros(nn*length(W),1);
for k=1:nn
  sctr = Tri(k,:);
  for q=1:length(W)
    [N,dNds] = TriShapeFnc(Q(q,:));  %compute shape functions
    J0 = p(sctr,:)'*dNds;
    l = (k-1)*length(W) + q;
    point(l,:) = N'*p(sctr,:);
    weight(l) = det(J0)*W(q);
  end
end
%------------------------------------------------------ TRIANGULAR QUADRATURE
function [weight,point] = TriQuad
point=[1/6,1/6;2/3,1/6;1/6,2/3]; weight=[1/6,1/6,1/6];
%------------------------------------------------ TRIANGULAR SHAPE FUNCTIONS
function [N,dNds] = TriShapeFnc(s)
N=[1-s(1)-s(2);s(1);s(2)]; dNds=[-1,-1;1,0;0,1];
%------------------------------------------------------------- INITIAL PLOT
function [handle,map] = InitialPlot(fem,z0)
Tri = zeros(length([fem.Element{:}])-2*fem.NElem,3);
map = zeros(size(Tri,1),1); index=0;
for el = 1:fem.NElem
  for enode = 1:length(fem.Element{el})-2
    map(index+1) = el;
    Tri(index+1,:) = fem.Element{el}([1,enode+1,enode+2]);
    index = index + 1;
  end
end
handle = patch('Faces',Tri,'Vertices',fem.Node,'FaceVertexCData',...
                1-z0(map),'FaceColor','flat','EdgeColor','none');
axis equal; axis off; axis tight; colormap(gray);
%-------------------------------------------------------------------------%
```

References

Allaire G (2001) Shape optimization by the homogenization method. Springer, Berlin

Allaire G, Francfort GA (1998) Existence of minimizers for non-quasiconvex functionals arising in optimal design. Ann Inst Henri Poincare Anal 15(3):301–339

Allaire G, Jouve F (2005) A level-set method for vibration and multiple loads structural optimization. Comput Methods Appl Mech Eng 194(30–33):3269–3290. doi:10.1016/j.cma.2004.12.018

Allaire G, Jouve F, Toader AM (2004) Structural optimization using sensitivity analysis and a level-set method. J Comput Phys 194(1):363–393. doi:10.1016/j.jcp.2003.09.032

Almeida SRM, Paulino GH, Silva ECN (2010) Layout and material gradation in topology optimization of functionally graded structures: a global-local approach. Struct Multidisc Optim 42(6):885–868. doi:10.1007/s00158-010-0514-x

Ambrosio L, Buttazzo G (1993) An optimal design problem with perimeter penalization. Calc Var Partial Differ Equ 1(1):55–69

Andreassen E, Clausen A, Schevenels M, Lazarov B, Sigmund O (2011) Efficient topology optimization in MATLAB using 88 lines of code. Struct Multidisc Optim 43(1):1–16. doi:10.1007/s00158-010-0594-7

Belytschko T, Xiao SP, Parimi C (2003) Topology optimization with implicit functions and regularization. Int J Numer Methods Eng 57(8):1177–1196. doi:10.1002/nme.824

Bendsoe MP (1989) Optimal design as material distribution problem. Struct Optim 1:193–202. doi:10.1007/BF01650949

Bendsoe MP, Sigmund O (1999) Material interpolation schemes in topology optimization. Arch Appl Mech 69(9–10):635–654

Bendsøe MP, Sigmund O (2003) Topology optimization: theory, methods and applications. Springer, Berlin

Borrvall T (2001) Topology optimization of elastic continua using restriction. Arch Comput Methods Eng 8(4):251–285

Borrvall T, Petersson J (2001) Topology optimization using regularized intermediate density control. Comput Methods Appl Mech Eng 190(37–38):4911–4928

Bourdin B (2001) Filters in topology optimization. Int J Numer Methods Eng 50(9):2143–2158

Bourdin B, Chambolle A (2003) Design-dependent loads in topology optimization. ESAIM, Controle Optim Calc Var 9(2):19–48

Bruns TE (2005) A reevaluation of the simp method with filtering and an alternative formulation for solid-void topology optimization. Struct Multidisc Optim 30(6):428–436. doi:10.1007/s00158-005-0537-x

Bruyneel M, Duysinx P (2005) Note on topology optimization of continuum structures including self-weight. Struct Multidisc Optim 29(4):245–256. doi:10.1007/s00158-004-0484-y

Cherkaev A (2000) Variational methods for structural optimization. Springer, New York

Dambrine M, Kateb D (2009) On the Ersatz material approximation in level-set methods. ESAIM, Controle Optim Calc Var 16(3):618–634. doi:10.1051/cocv/2009023

de Ruiter MJ, Van Keulen F (2004) Topology optimization using a topology description function. Struct Multidisc Optim 26(6):406–416. doi:10.1007/s00158-003-0375-7

Delfour MC, Zolésio JP (2001) Shapes and geometries: analysis, differential calculus, and optimization. Society for Industrial and Applied Mathematics, Philadelphia

Ghosh S (2010) Micromechanical analysis and multi-scale modeling using the Voronoi cell finite element method. In: Computational mechanics and applied analysis. CRC Press, Boca Raton

Groenwold AA, Etman LFP (2008) On the equivalence of optimality criterion and sequential approximate optimization methods in the classical topology layout problem. Int J Numer Methods Eng 73(3):297–316. doi:10.1002/nme.2071

Guest JK, Prevost JH, Belytschko T (2004) Achieving minimum length scale in topology optimization using nodal design variables and projection functions. Int J Numer Methods Eng 61(2):238–254. doi:10.1002/nmc.1064

Haber RB, Jog CS, Bendsoe MP (1996) A new approach to variable-topology shape design using a constraint on perimeter. Struct Optim 11(1):1–12

Hughes TJR (2000) The finite element method: linear static and dynamic finite elemnt analysis. Dover, New York

Kohn RV, Strang G (1986a) Optimal design and relaxation of variational problems I. Commun Pure Appl Math 39(1):113–137

Kohn RV, Strang G (1986b) Optimal design and relaxation of variational problems. II. Commun Pure Appl Math 38(1):139–182

Kohn RV, Strang G (1986c) Optimal design and relaxation of variational problems. III. Commun Pure Appl Math 39:353–377

Kosaka I, Swan CC (1999) A symmetry reduction method for continuum structural topology optimization. Comput Struct 70(1):47–61

Langelaar M (2007) The use of convex uniform honeycomb tessellations in structural topology optimization. In: 7th world congress on structural and multidisciplinary optimization, Seoul, South Korea, May 21–25

Martinez JM (2005) A note on the theoretical convergence properties of the SIMP method. Struct Multidisc Optim 29(4):319–323. doi:10.1007/s00158-004-0479-8

Mousavi SE, Xiao H, Sukumar N (2009) Generalized gaussian quadrature rules on arbitrary polygons. Int J Numer Methods Eng. doi:10.1002/nme.2759

Natarajan S, Bordas SPA, Mahapatra DR (2009) Numerical integration over arbitrary polygonal domains based on Schwarz–Christoffel conformal mapping. Int J Numer Methods Eng. doi:10.1002/nme.2589

Olhoff N, Bendsoe MP, Rasmussen J (1991) On cad-integrated structural topology and design optimization. Comput Methods Appl Mech Eng 89(1–3):259–279

Persson P, Strang G (2004) A simple mesh generator in MATLAB. Siam Rev 46(2):329–345. doi:10.1137/S0036144503429121

Petersson J (1999) Some convergence results in perimeter-controlled topology optimization. Comput Methods Appl Mech Eng 171 (1–2):123–140

Rietz A (2001) Sufficiency of a finite exponent in SIMP (power law) methods. Struct Multidisc Optim 21:159–163

Rozvany GIN (2009) A critical review of established methods of structural topology optimization. Struct Multidisc Optim 37(3):217–237. doi:10.1007/s00158-007-0217-0

Rozvany GIN, Zhou M, Birker T (1992) Generalized shape optimization without homogenization. Struct Optim 4(3–4):250– 252

Saxena A (2008) A material-mask overlay strategy for continuum topology optimization of compliant mechanisms using honeycomb discretization. J Mech Des 130(8):2304-1-9. doi:10.1115/1.2936891

Sigmund O (2007) Morphology-based black and white filters for topology optimization. Struct Multidisc Optim 33(4–5):401–424. doi:10.1007/s00158-006-0087-x

Sigmund O, Petersson J (1998) Numerical instabilities in topology optimization: a survey on procedures dealing with checkerboards, mesh-dependencies and local minima. Struct Optim 16(1):68–75

Stolpe M, Svanberg K (2001a) On the trajectories of penalization methods for topology optimization. Struct Multidisc Optim 21(2):128–139

Stolpe M, Svanberg K (2001b) An alternative interpolation scheme for minimum compliance topology optimization. Struct Multidisc Optim 22(2):116–124

Stromberg LL, Beghini A, Baker WF, Paulino GH (2011) Application of layout and topology optimization using pattern gradation for the conceptual design of buildings. Struct Multidisc Optim 43(2):165–180. doi:10.1007/s00158-010-0563-1

Sukumar N, Tabarraei A (2004) Conforming polygonal finite elements. Int J Numer Methods Eng 61(12):2045–2066. doi:10.1002/nme.1141

Svanberg K (1987) The method of moving asymptotes—a new method for structural optimization. Int J Numer Methods Eng 24(2):359–373

Tabarraei A, Sukumar N (2006) Application of polygonal finite elements in linear elasticity. Int J Comput Methods 3(4):503–520. doi:10.1142/S021987620600117X

Talischi C, Paulino GH, Le CH (2009) Honeycomb wachspress finite elements for structural topology optimization. Struct Multidisc Optim 37(6):569–583. doi:10.1007/s00158-008-0261-4

Talischi C, Paulino GH, Pereira A, Menezes IFM (2010) Polygonal finite elements for topology optimization: a unifying paradigm. Int J Numer Methods Eng 82(6):671–698. doi:10.1002/nme.2763

Talischi C, Paulino GH, Pereira A, Menezes IFM (2011) PolyMesher: a general-purpose mesh generator for polygonal elements written in Matlab. Struct Multidisc Optim. doi:10.1007/s00158-011-0706-z

Tartar L (2000) An introduction to the homogenization method in optimal design. In: Optimal shape design: lecture notes in mathematics, no. 1740. Springer, Berlin, pp 47–156

Van Dijk NP, Langelaar M, Van Keulen F (2009) A discrete formulation of a discrete level-set method treating multiple constraints. In: 8th World Congress on Structural and Multidisciplinary Optimization, June 1-5, 2009, Lisbon, Portugal

Wang MY, Wang XM, Guo DM (2003) A level set method for structural topology optimization. Comput Methods Appl Mech Eng 192(1–2):227–246

Zhou M, Rozvany GIN (1991) The COC algorithm, part II: topological, geometrical and generalized shape optimization. Comput Methods Appl Mech Eng 89(1–3):309–336