Structural and Multidisciplinary Optimization (2018) 58:2727–2759
https://doi.org/10.1007/s00158-018-2094-0

EDUCATIONAL ARTICLE

`PolyMat`: an efficient Matlab code for multi-material topology optimization

Emily D. Sanders[1] · Anderson Pereira[2] · Miguel A. Aguiló[3] · Glaucio H. Paulino[1]

Received: 14 May 2018 / Revised: 16 August 2018 / Accepted: 5 September 2018 / Published online: 26 October 2018
© Springer-Verlag GmbH Germany, part of Springer Nature 2018

Abstract

We present a Matlab implementation of topology optimization for compliance minimization on unstructured polygonal finite element meshes that efficiently accommodates many materials and many volume constraints. Leveraging the modular structure of the educational code, `PolyTop`, we extend it to the multi-material version, `PolyMat`, with only a few modifications. First, a design variable for each candidate material is defined in each finite element. Next, we couple a Discrete Material Optimization interpolation with the existing penalization and introduce a new parameter such that we can employ continuation and smoothly transition from a convex problem without any penalization to a non-convex problem in which material mixing and intermediate densities are penalized. Mixing that remains due to the density filter operation is eliminated via continuation on the filter radius. To accommodate flexibility in the volume constraint definition, the constraint function is modified to compute multiple volume constraints and the design variable update is modified in accordance with the Zhang-Paulino-Ramos Jr. (ZPR) update scheme, which updates the design variables associated with each constraint independently. The formulation allows for volume constraints controlling any subset of the design variables, i.e., they can be defined globally or locally for any subset of the candidate materials. Borrowing ideas for mesh generation on complex domains from `PolyMesher`, we determine which design variables are associated with each local constraint of arbitrary geometry. A number of examples are presented to demonstrate the many material capability, the flexibility of the volume constraint definition, the ease with which we can accommodate passive regions, and how we may use local constraints to break symmetries or achieve graded geometries.

Keywords Topology optimization · Polygonal finite elements · Matlab · Multi-material · ZPR

1 Introduction

This paper adds to a series of educational papers and associated Matlab codes for topology optimization using unstructured polygonal finite element meshes. The first code in the series is `PolyMesher`, a general-purpose polygonal finite element mesh generator for possibly complex domains (Talischi et al. 2012a). `PolyTop` is a companion code that performs topology optimization for compliance minimization using a modular structure that separates the analysis routines from the optimization formulation and can accommodate unstructured meshes (Talischi et al. 2012b). The effectiveness of `PolyTop`'s modular framework is demonstrated by changing only a few lines in the analysis routines and arriving at an implementation for minimization of dissipated power in Stokes flow (Pereira et al. 2016). Here, we develop `PolyMat`, a code built on `PolyTop` that requires minimal modification to obtain minimum compliance designs with many materials and accommodating possibly many local or global volume constraints. In the same spirit of this paper, Tavakoli and Mohseni (2014) detail the implementation and provide a downloadable Matlab code for their Alternating Active Phase (AAP) algorithm, which handles multiple materials by performing a series of binary phase updates within each optimization step. The AAP code has

Responsible Editor: Xu Guo

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s00158-018-2094-0) contains supplementary material, which is available to authorized users.

✉ Glaucio H. Paulino
paulino@gatech.edu

[1] School of Civil and Environmental Engineering, Georgia Institute of Technology, 790 Atlantic Drive NW, Atlanta, GA 30332, USA

[2] Department of Mechanical Engineering, Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Rua Marquês de São Vicente, 225, Rio de Janeiro, RJ, 22451-900, Brazil

[3] Simulation and Modeling Sciences, Sandia National Laboratories, P.O. Box 5800, Albuquerque, NM 87185, USA

been adopted by many researchers in recent years for multi-material topology optimization (Park and Sutradhar 2015; Lieu and Lee 2017; Doan and Lee 2017), including one case in which it was coupled with multi-material topology optimization on polygonal finite element meshes (Chau et al. 2017).

In lieu of a full survey of the multi-material topology optimization literature, we highlight a few key references related to this work and refer the reader to Sanders et al. (2018) for a more comprehensive literature review. The multi-material formulation adopted here is based on that of Zhang et al. (2018), which was proposed in the context of the ground-structure method. Their problem statement includes a very general specification of the volume constraints that allows them to control any subset of the candidate materials in any sub-region of the domain. To efficiently accommodate such constraints, Zhang et al. (2018) introduce the Zhang-Paulino-Ramos Jr. (ZPR, pronounced "zipper") design variable update scheme, which is derived from a convex (linear) approximation to the problem. Due to separability of the dual objective associated with the linearized subproblems, Zhang et al. (2018) observe that each constraint is dependent only on the Lagrange multiplier of the constraint it is associated with, allowing the ZPR to update the design for each constraint independently. The ZPR update scheme was tailored for continuum problems by Sanders et al. (2018) and coupled with a Discrete Material Optimization (DMO) interpolation (Stegmann and Lund 2005) to obtain solutions containing little material mixing.

Here, we adopt both the ZPR update scheme and the DMO material interpolation scheme for volume-constrained compliance minimization with possibly many materials and many volume constraints. The main goal of this educational paper is to elaborate on the details required to implement the formulation within the PolyTop framework and also to provide an educational Matlab code that can be used and extended by others. Features of the educational PolyMat code include: (1) the ability to generate complex domains with complex sub-region geometries on which volume constraints and/or passive regions may be specified; (2) introduction of a new material interpolation parameter that, along with the penalty parameter, can be continued to achieve a smooth transition from the convex problem to a non-convex problem in which both intermediate densities and material mixing are penalized; (3) a method to obtain designs with crisp boundaries between the material phases via continuation on the filter radius; and (4) minimal increase in computational cost relative to the single-material implementation when the number of candidate materials is low (e.g., ≤ 5).

The remainder of this paper is organized as follows. In Section 2, we formulate the continuous volume-constrained compliance minimization problem with multiple materials and multiple volume constraints. The problem is discretized in Section 3. We elaborate on the PolyMat implementation in Section 4 and provide some extensions in Section 5. In Section 6, we make some comments on the code's efficiency and provide a number of illustrative examples in Section 7. Finally, Appendix A provides convergence plots for selected example problems, Appendix B summarizes three different filtering techniques that can easily be implemented in PolyMat, Appendix C lists the problems available with download of the code and summarizes the input information required to run each problem, Appendix D provides an example input file, PolyScript, Appendix E provides a code listing for the main kernel of PolyMat, and Appendices F, G, and H provide some additional Matlab functions that are new relative to PolyTop.

2 Problem setting

In this section, we formulate the volume-constrained compliance minimization problem in which we search for an optimal shape, ω, contained in the extended domain, $\Omega \subseteq \mathbb{R}^2$, where ω is the union of a finite number of non-intersecting partitions composed of m different materials, i.e, $\omega = \cup_i^m \omega_i$ and $\omega_i \cap \omega_j = \emptyset$ for $i \neq j$ (see Fig. 1). The optimization problem is stated as:

$$\inf_{\omega \in \mathcal{O}} f(\omega, \mathbf{u}_\omega) \quad \text{subject to} \quad g_j(\omega, \mathbf{u}_\omega) \leq 0, \quad j = 1, \ldots, K \tag{1}$$

where $\mathcal{O}$ is the space of admissible shapes and $\mathbf{u}_\omega \in \mathcal{V}_\omega$ satisfies the governing variational problem of linear elasticity:

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

with the space of admissible displacements defined as:

$$\mathcal{V}_\omega = \left\{ \mathbf{v} \in H^1\left(\omega, \mathbb{R}^2\right) : \mathbf{v}|_{\partial\omega \cap \Gamma_D} = \mathbf{0} \right\} \tag{3}$$

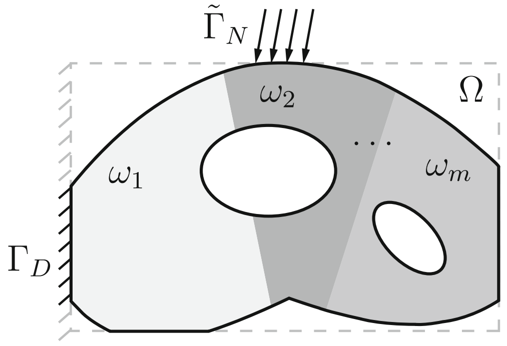


Fig. 1 Extended design domain, boundary conditions, and illustration of material partitions making up the optimal shape (adapted from Talischi et al. 2012b)

In (2), $\mathbf{C}(\mathbf{x})$ is the stiffness tensor that varies according to the material from which each partition, $\omega_i \subseteq \omega$, is made, Γ_D is the partition of $\partial\Omega$ on which displacements are prescribed, Γ_N is the complimentary partition of $\partial\Omega$ such that $\overline{\Gamma_D} \cup \overline{\Gamma_N} = \partial\Omega$ and $\Gamma_D \cap \Gamma_N = \emptyset$, and $\tilde{\Gamma}_N \subseteq \Gamma_N$ is the partition of $\partial\Omega$ on which non-zero tractions, $\mathbf{t}$, are prescribed.

The reader is referred to the discussions by Talischi et al. (2012b) that develop, in sufficient detail, the notions needed to arrive at the sizing problem from the problem described in (1) and (2). Here, we briefly highlight the key steps involved, emphasizing the additional requirements to incorporate multiple materials into the formulation.

First, we introduce a characteristic function, χ_ω, that recasts the boundary value problem in (2) onto Ω. Next, we define a choice function, ϕ_ω, that takes on a scalar value (material scale factor) corresponding to one of a finite number of candidate materials. Together, χ_ω and ϕ_ω scale a constant material tensor, $\mathbf{C}$, to arrive at its spatially varying counterpart, $\mathbf{C}(\mathbf{x})$. To enforce existence of solutions to the boundary value problem, we adopt an Ersatz approach by replacing $\chi_\omega\phi_\omega$ with $\varepsilon + (1-\varepsilon)\chi_\omega\phi_\omega$, where $\varepsilon\mathbf{C}$ defines a compliant material used to fill the void regions. Now, (2) becomes:

$$\int_\Omega [\varepsilon + (1-\varepsilon)\chi_\omega\phi_\omega]\mathbf{C}\nabla\mathbf{u} : \nabla\mathbf{v} d\mathbf{x} = \int_{\tilde{\Gamma}_N} \mathbf{t}\cdot\mathbf{v} ds, \quad \forall \mathbf{v} \in \mathcal{V}, \tag{4}$$

and the space of admissible displacements, $\mathcal{V}$, is independent of ω.

A continuous parameterization, $\rho_i \in [0,1]$, $i = 1,\ldots,m$, is introduced for each candidate material to avoid integer programming, which can be prohibitively expensive. A penalty function, m_W, (e.g., SIMP (Zhou and Rozvany 1991; Bendsøe 1989)) recovers the binary nature of the problem and an additional interpolation function, m_M, enforces selection of a single material at each point in ω.

Further, we adopt a restriction setting in which a regularization map, $\mathcal{P}$, enforces well-posedness of the optimization problem by introducing a measurable function, η_i, $i = 1,\ldots,m$, for each candidate material, such that each ρ_i in the admissible space of designs inherits the smoothness characteristics of the kernel used to define $\mathcal{P}$, i.e., $\rho_i = \mathcal{P}(\eta_i)$. Here, the regularization map is defined by convolution of the design functions, η_i, $i = 1,\ldots,m$, with a smooth kernel (filter), F, i.e., (Bourdin 2001; Borrvall and Petersson 2001):

$$\mathcal{P}_F(\eta_i) = \int_\Omega F(\mathbf{x},\overline{\mathbf{x}})\eta_i(\overline{\mathbf{x}})d\overline{\mathbf{x}} \tag{5}$$

where the filter of radius R is defined as:

$$F(\mathbf{x},\overline{\mathbf{x}}) = c(\mathbf{x})\max\left(1 - \frac{\mathbf{x}-\overline{\mathbf{x}}}{R}, 0\right) \tag{6}$$

and $c(\mathbf{x})$ is a normalizing coefficient.

Now, the volume-constrained compliance minimization problem is stated as:

$$\begin{aligned} \inf_{\{\rho\}_{i=1}^m \in \mathcal{A}} f(\rho_1,\ldots,\rho_m,\mathbf{u}) &= \int_{\tilde{\Gamma}_N} \mathbf{t}\cdot\mathbf{u} ds \quad \text{subject to} \\ g_j(\{\rho_i : i \in \mathcal{G}_j\}) &= \sum_{i\in\mathcal{G}_j} \frac{1}{|\Omega_j|}\int_{\Omega_j} m_V(\rho_i)d\mathbf{x} - \overline{v}_j \le 0, \\ j &= 1,\ldots,K \end{aligned} \tag{7}$$

where $\mathcal{G}_j$ is the set of material indices associated with constraint j, $\Omega_j \in \Omega$ is the partition of the domain for which constraint j is specified, m_V is the interpolation function for the volume constraint, $\overline{v}_j$ is the volume fraction limit for constraint j, the space of admissible designs is:

$$\mathcal{A} = \{\mathcal{P}_F(\eta_1)\circ\ldots\circ\mathcal{P}_F(\eta_m) : \eta_i \in L^\infty\ \forall i(\Omega; [\underline{\rho}, \overline{\rho}])\} \tag{8}$$

where $\circ$ indicates composition of functions, and $\mathbf{u} \in \mathcal{V}$ satisfies:

$$\int_\Omega m_E(\rho_1,\ldots,\rho_m)\mathbf{C}\nabla\mathbf{u} : \nabla\mathbf{v} d\mathbf{x} = \int_{\tilde{\Gamma}_N} \mathbf{t}\cdot\mathbf{v} ds, \quad \forall \mathbf{v} \in \mathcal{V}, \tag{9}$$

where $m_E = m_M \circ m_W$ and the space of admissible displacements is:

$$\mathcal{V} = \{\mathbf{v} \in H^1(\Omega, \mathbb{R}^2) : \mathbf{v}|_{\partial\Gamma_D} = \mathbf{0}\} \tag{10}$$

3 Discretization

A final step required to solve the problem numerically is to discretize the displacement field, $\mathcal{V}$, and design space, $\mathcal{A}$, on Ω. For convenience, we choose to discretize both spaces using the same fixed partition, $\mathcal{T}_h = \{\Omega_\ell\}_{\ell=1}^N$, for which h represents the characteristic mesh size, $\Omega_\ell \cap \Omega_k = \emptyset$ for $\ell \ne k$, and $\cup_\ell \overline{\Omega}_\ell = \overline{\Omega}$. With this partition, we define the piecewise constant discretization of $\mathcal{A}$:

$$\begin{aligned} \mathcal{A}_h = \{\mathcal{P}_F(\eta_1^h)\circ\cdots\circ\mathcal{P}_F(\eta_m^h) : \underline{\rho} \le \eta_i^h \le \overline{\rho}\forall i, \eta_i^h|_{\Omega_\ell} \\ = \text{const}\ \forall\ell\} \end{aligned} \tag{11}$$

where we define $\rho_i^h = \mathcal{P}_F(\eta_i^h), i = 1,\ldots,m$, for each candidate material. The matrix of design variables, $\mathbf{Z} = \{z_{\ell 1},\ldots,z_{\ell m}\}_{\ell=1}^N$, results from discretizing the design functions as follows:

$$\eta_i^h(\mathbf{x}) = \sum_{\ell=1}^N z_{\ell i}\chi_{\Omega_\ell}(\mathbf{x}) \tag{12}$$

where $z_{\ell i}$ is the constant value that η_i^h assumes over Ω_ℓ. The functions, ρ_i^h, are replaced by $\tilde{\rho}_i^h$, $i = 1, \ldots, m$, which are constant over each element:

$$\tilde{\rho}_i^h(\mathbf{x}) = \sum_{\ell=1}^{N} y_{\ell i} \chi_{\Omega_\ell}(\mathbf{x}) \tag{13}$$

according to an elemental value, $y_{\ell i} = \rho_i^h(\mathbf{x}_\ell^*)$, that we have defined using the value of ρ_i^h at the centroid, $\mathbf{x}_\ell^*$, of element ℓ. The set of element values can also be organized in matrix form as $\mathbf{Y} = \{y_{\ell 1}, \ldots, y_{\ell m}\}_{\ell=1}^N$.

In the same way that Talischi et al. (2012b) discretized the mapping $\mathcal{P}_F$, we relate the elemental values of η_i^h and ρ_i^h according to the following:

$$\mathbf{y}_i = \mathbf{P}\mathbf{z}_i, \quad i = 1, \ldots, m \tag{14}$$

where $\mathbf{y}_i$ and $\mathbf{z}_i$ are the i^{th} columns of $\mathbf{Y}$ and $\mathbf{Z}$, respectively, and:

$$P_{\ell k} = \int_{\Omega_k} F(\mathbf{x}_\ell^*, \overline{\mathbf{x}}) d\overline{\mathbf{x}} \tag{15}$$

The final discrete problem for volume-constrained compliance minimization that accommodates many candidate materials and many local or global volume constraints is expressed as:

$$\begin{aligned} \min_{\mathbf{Z} \in [0,1]^{N \times m}} \; & f = \mathbf{F}^T\mathbf{U} \quad \text{subject to} \\ & g_j = \frac{\sum_{i \in \mathcal{G}_j} \sum_{\ell \in \mathcal{E}_j} A_\ell m_V(y_{\ell i})}{\sum_{\ell \in \mathcal{E}_j} A_\ell} - \overline{v}_j \leq 0, \\ & j = 1, \ldots, K \end{aligned} \tag{16}$$

where $\mathbf{F}_i = \int_{\tilde{\Gamma}_N} \mathbf{t} \cdot \mathbf{N}_i ds$ is the vector of design-independent applied loads, $\mathcal{E}_j$ is the set of element indices associated with constraint j, A_ℓ is the volume of element ℓ, and $\mathbf{U}$ solves the discretized state equations, $\mathbf{KU} = \mathbf{F}$, in which the stiffness matrix is:

$$\mathbf{K} = \sum_{\ell=1}^{N} m_E(y_{\ell i}, \ldots, y_{\ell m}) \mathbf{k}_\ell \tag{17}$$

and $(\mathbf{k}_\ell)_{jk} = \int_{\Omega_\ell} \mathbf{C}\nabla\mathbf{N}_j : \nabla\mathbf{N}_k d\mathbf{x}$ is the constant element stiffness matrix of element ℓ. In (17), recall that the stiffness interpolation function, m_E, is the composition of the multi-material interpolation function and the penalty function, i.e., $m_E = m_M \circ m_W$.

4 Implementation in `PolyMat`

By leveraging the modular framework established in `PolyTop`, we arrive at `PolyMat`, an implementation of the problem stated in (16), with only a few modifications to `PolyTop`. In addition to some new required input data for multi-material problems, we add a new multi-material interpolation function, `MultiMatIntFnc`, that works with the existing penalty function, `MatIntFnc`, to interpolate the elemental stiffnesses. Additionally, we modify the constraint function, `ConstraintFnc`, so that it computes multiple volume fraction constraints. Lastly, we implement the ZPR design variable update scheme (Zhang et al. 2018), which requires no modification to the structure of the `UpdateScheme` function itself, but only modification to the main function in `PolyMat` such that we pass the subset of element variables associated with each constraint to `UpdateScheme`, one constraint at a time. We elaborate on these changes in the following subsections.

4.1 Input data and `PolyScript`

A Matlab script, `PolyScript`, is used to define all input data, which is passed to the `PolyMat` kernel to perform the analysis and optimization. Three inputs, defined in `PolyScript`, are required in the call to `PolyMat`: two Matlab `struct` arrays, `fem` and `opt`, which hold parameters related to the finite element analysis and the topology optimization, respectively, and `Color`, which is an `NMat`×3 matrix containing an RGB triplet for each candidate material (for visualization of results). The fields stored in `fem` and `opt` are provided in Table 1 and are almost identical to those needed for the single-material implementation in `PolyTop`. The only new fields are: (1) the array, `fem.Mat`, containing the Young's modulus of each candidate material; (2) the number of candidate materials, `fem.NMat`; (3) a handle to the multi-material interpolation function, `opt.MultiMatIntFnc`; (4) cell arrays, `opt.ElemInd` and `opt.MatInd`, holding the element indices and material indices, respectively, associated with each constraint; and (5) the number of volume constraints, `opt.NConstr`. Additionally, `opt.VolFrac` is modified such that it is an array containing a volume fraction for each of the volume constraints.

As before, a call to `PolyMesher` with a pre-specified domain file can be used to obtain the finite element mesh and boundary condition data for a desired problem. In a similar spirit, for the multi-material implementation with possibly many volume constraints, a call to a pre-specified constraint file can be used to obtain the necessary constraint information. The constraint specification is detailed in the next subsection.

4.2 Constraint specification

The formulation in (16) allows for a very general definition of the volume constraints in which a given constraint may control a subset of the candidate materials and/or a subset of the elements (i.e., sub-regions of the domain). The constraints are fully specified to `PolyMat` by the vector of volume fraction constraints, `opt.VolFrac`, and

Table 1 List of fields in the input structures

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 reference solid material
fem.Mat‡	Array of Young's moduli for candidate materials
fem.NMat‡	Number of candidate materials
fem.SElem‡	Elements in passive regions
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 penalty function
opt.MultiMatIntFnc‡	Handle to material interpolation function
opt.P	Matrix that maps design to element variables
opt.Volfrac‡	Array of specified volume fraction constraints
opt.NConstr‡	Number of volume constraints
opt.ElemInd‡	Cell array of elements associated with each constraint
opt.MatInd‡	Cell array of materials associated with each constraint
opt.Tol	Convergence tolerance on design variables
opt.MaxIter	Max. number of optimization iterations
opt.ZPRMove	Allowable move step in the ZPR update scheme
opt.ZPREta	Exponent used in the ZPR update scheme

The fields marked with the superscript †, if empty, are populated inside PolyMat. The fields marked with the superscript ‡, are new or modified relative to the single-material implementation in PolyTop

two cell arrays, opt.ElemInd and opt.MatInd. In the latter two arrays, each cell entry corresponds to a constraint and contains a vector of element (for opt.ElemInd) or material (for opt.MatInd) indices associated with that constraint. These vectors of element and material indices correspond to $\mathcal{E}_j$ and $\mathcal{G}_j$ in (16), respectively.

To avoid repeated modification of PolyScript for different problems, it is convenient to define the constraint information in a separate constraint file that is called from PolyScript. In addition to defining the required constraint data, VolFrac, ElemInd, and MatInd, in the constraint file, it is also convenient to specify the array of material moduli, Mat, and the array of RGB colors, Color, for each of the candidate materials.

In general, it is straightforward to specify the required constraint information; however, if the constraints are defined on many sub-regions or if the sub-regions have complex geometries, it may be difficult to determine the element indices associated with each constraint (i.e., the entries of ElemInd). One simple approach is to borrow some of the ideas used in `PolyMesher` for generating complex domains. In the same way that `PolyMesher` uses signed distance functions to implicitly represent the domain geometry, we use signed distance functions to implicitly represent the sub-region geometries. For the sub-region associated with each constraint, we construct a vector of distance values using the distance functions provided with `PolyMesher` (i.e., `dLine`, `dCircle`, `dRectangle`, `dDiff`, `dIntersect`, `dUnion`). Then, we can easily determine which elements are within the sub-region boundary by evaluating the sign of the distance values. For "nice" meshes, the seeds used to construct the mesh are very close to the element centroids and since the seeds are easily accessible from `PolyMesher`, we compute the distance values based on the locations of the element seeds. The indices of elements with seeds inside the boundary of constraint j's sub-region are returned as a vector entry in the j^{th} cell of `ElemInd`.

As an illustrative example, we construct the `ElemInd` cell array for the domain in Fig. 2a, which is constructed via a number of boolean operations on one rectangle and two circles (c_1 and c_3) using `PolyMesher`. To define the constraints shown in Fig. 2b, we define two additional circles, c_2 and c_4, and store the signed distance values of the element seeds with respect to these circles in variables `c2` and `c4`, respectively. The signed distance values for the portion of the domain between c_2 and c_4 is stored in variable `int`. With these signed distance values, we define `ElemInd` for constraints g_1 to g_5 using the Matlab code provided in Fig. 2b.

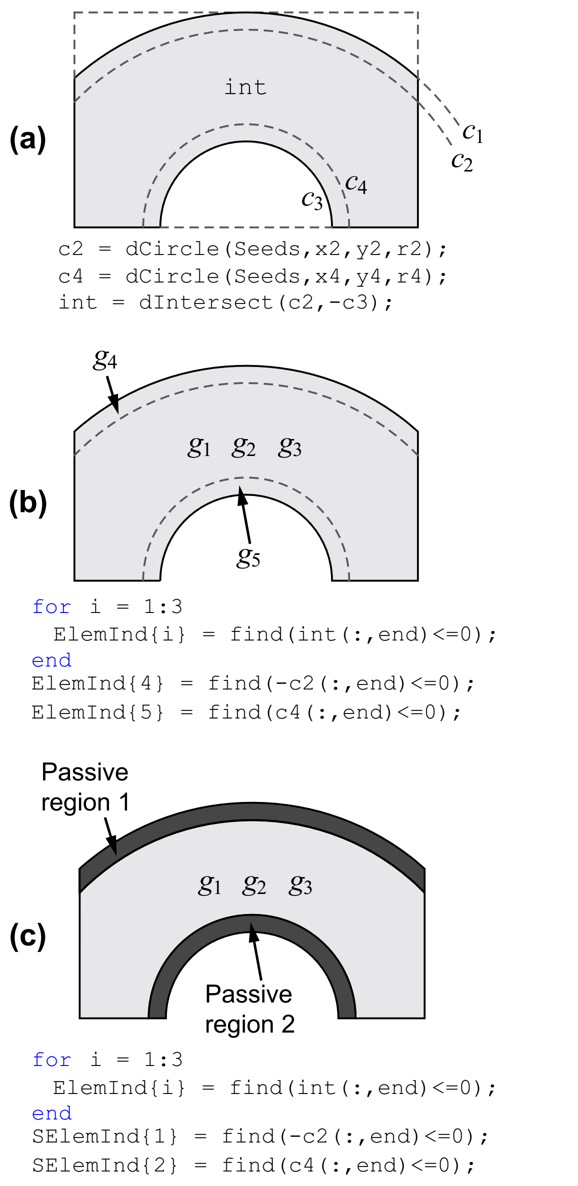
In addition to the domain files provided with download of `PolyTop`, we provide a number of associated constraint files with download of `PolyMat`. Refer to Appendix C to understand how to use these files to solve the various multi-material topology optimization problems provided with download of `PolyMat`.

4.3 Initial guess

The `NElem` × `NMat` matrix containing the initial guess, `zIni`, is specified in `PolyScript` by passing `VolFrac`, `ElemInd`, and `MatInd` to a function called `InitialGuess`, which evenly distributes the volume fraction specified for each constraint between the materials associated with that constraint (within the appropriate sub-region). The code listing for the `InitialGuess` function is included in Appendix F.

4.4 Interpolation functions

Function handles to two interpolation functions, `MatIntFnc` and `MultiMatIntFnc`, are stored in `opt`, allowing the user to specify parameters related to the



```
c2 = dCircle(Seeds,x2,y2,r2);
c4 = dCircle(Seeds,x4,y4,r4);
int = dIntersect(c2,-c3);
```

```
for i = 1:3
  ElemInd{i} = find(int(:,end)<=0);
end
ElemInd{4} = find(-c2(:,end)<=0);
ElemInd{5} = find(c4(:,end)<=0);
```

```
for i = 1:3
  ElemInd{i} = find(int(:,end)<=0);
end
SElemInd{1} = find(-c2(:,end)<=0);
SElemInd{2} = find(c4(:,end)<=0);
```

Fig. 2 Constraint specification: **a** curved beam domain and code needed to generate the distance values required to define the constraints and/or passive regions; **b** constraint specification and code needed to assemble `ElemInd`; and **c** constraint and passive region specification and code needed to assemble `ElemInd` and `SElemInd`

interpolations in `PolyScript`. Together, `MatIntFnc` and `MultiMatIntFnc`, perform the volume and stiffness interpolations, outputting the volume interpolation, m_V, and the stiffness interpolation, m_E, back to the kernel function, `PolyMat`. `MatIntFnc` computes quantities explicitly dependent on $\mathbf{Y}$, while `MultiMatIntFnc` computes quantities explicitly dependent on an intermediate variable, $\mathbf{W} = \{w_{\ell 1}, \ldots, w_{\ell m}\}_{\ell=1}^{N}$, introduced in the following.

The volume interpolation considered here is defined as:

$$m_V(y_{\ell i}) = y_{\ell i} \tag{18}$$

and is computed in MatIntFnc. The stiffness interpolation, m_E, is accomplished in two steps. First, we apply a penalty function, m_W, in MatIntFnc to push the element densities toward zero and one. For example, in the case of SIMP (Bendsøe 1989; Zhou and Rozvany 1991) and RAMP (Stolpe and Svanberg 2001), penalized element densities, $\mathbf{W}$, are computed as:

$$w_{\ell i} = m_W(y_{\ell i}) = \begin{cases} y_{\ell i}^p & \text{(SIMP)} \\ \frac{y_{\ell i}}{1+q(1-y_{\ell i})} & \text{(RAMP)} \end{cases} \quad \ell = 1, \ldots, N \quad i = 1, \ldots, m, \tag{19}$$

where $p > 1$ and $q > 0$ are penalty parameters for SIMP and RAMP, respectively, and $\mathbf{W}$ is introduced for convenience of notation. As in PolyTop, the penalty functions defined in MatIntFnc include SIMP and RAMP. Next, the multi-material interpolation, m_M, is performed in MultiMatIntFnc using the following function (Stegmann and Lund 2005):

$$m_M(\mathbf{w}'_\ell) = \varepsilon + (1-\varepsilon)\sum_{i=1}^{m} w_{\ell i} \prod_{\substack{j=1 \\ j\neq i}}^{m} (1-w_{\ell j}) E_i^0, \quad \ell = 1, \ldots, N \tag{20}$$

where $\mathbf{w}'_\ell$ is the ℓ^{th} row of $\mathbf{W}$, ε is a small number used to define the compliant (Ersatz) material in the void regions, E_i^0 is the Young's modulus of the i^{th} candidate material (specified in fem.Mat).

In Fig. 3, we plot the stiffness interpolation functions for a single element with two candidate materials ($E_1 = 0.5$, $E_2 = 1$) considering SIMP with $p = 4$ and $\varepsilon = 0$. As usual, the penalty function, m_W, in Fig. 3a makes intermediate values of $y_{\ell i}$ uneconomical (Bendsøe 1989; Zhou and Rozvany 1991). Note that the multi-material interpolation function, m_M, in Fig. 3b goes to zero when both materials are fully dense, i.e., the multi-material interpolation function makes material mixing uneconomical when the materials are fully dense. However, material mixing of intermediate densities is not significantly penalized. When composed to m_E, as shown in Fig. 3c, the two functions penalize both intermediate densities and material mixing.

The implementation of (20) in MultiMatIntFnc takes advantage of the matrix math capabilities in Matlab and avoids a nested for-loop over the elements and candidate materials. In lines 13 to 18 of MultiMatIntFnc, we assemble an NElem × NMat matrix, Prod, containing the product, $\prod_{j=1, j\neq i}^{m}(1-w_{\ell j})$, for each (ℓ, i) entry. Assembly of Prod requires a single for-loop over the number of materials minus one. Then, the entire NElem ×1 vector, $\mathbf{E} := m_E(\mathbf{Y}) = (m_M \circ m_W)(\mathbf{Y})$, is computed in one line of code:

```
E = eps + (1-eps).*((w.*Prod)*E0);
```

Aside from removing the parameter ε, MatIntFnc is identical to that used in PolyTop. The code listing for MultiMatIntFnc, which now accounts for ε, is provided in Appendix G.

4.5 Analysis functions

Since we seek to minimize compliance, no change is needed in the objective function, ObjectiveFnc; however, the constraint function, ConstraintFnc, is modified to compute multiple volume fraction constraints. The same information required by the analysis functions in the single material implementation is needed in the multi-material implementation, namely, the element stiffnesses, $\mathbf{E} = \{E_\ell\}_{\ell=1}^N := m_E(\mathbf{Y}) = (m_M \circ m_W)(\mathbf{Y})$, and element volume fractions $\mathbf{V} = \{V_{\ell 1}, \ldots, V_{\ell m}\}_{\ell=1}^N := m_V(\mathbf{Y})$. As before, the element areas, $\mathbf{A}$, needed to compute the volume constraints are computed only once. Both ObjectiveFnc and ConstraintFnc are provided with the PolyMat code listing in Appendix E.

4.6 Sensitivity analysis

Sensitivities of the objective and constraints with respect to the design variables, $\mathbf{z}_i$, $i = 1, \ldots, m$, can be expressed via the chain rule in an identical manner as the single-material case, i.e.,:

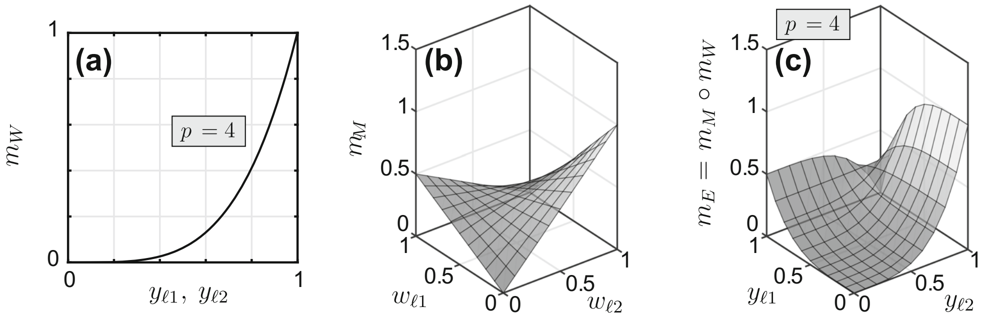


Fig. 3 Material interpolation functions for two candidate materials ($E_1 = 0.5$, $E_2 = 1$) in element ℓ considering SIMP with $p = 4$ and $\varepsilon = 0$: **a** m_W (see (19)); **b** m_M (see (20)); and **c** $m_E = m_M \circ m_W$

$$\frac{\partial f}{\partial \mathbf{z}_i} = \frac{\partial \mathbf{y}_i}{\partial \mathbf{z}_i}\frac{\partial \mathbf{E}}{\partial \mathbf{y}_i}\frac{\partial f}{\partial \mathbf{E}} + \frac{\partial \mathbf{y}_i}{\partial \mathbf{z}_i}\frac{\partial \mathbf{V}_i}{\partial \mathbf{y}_i}\frac{\partial f}{\partial \mathbf{V}_i}, \quad i = 1,\dots,m \tag{21}$$

$$\frac{\partial g_j}{\partial \mathbf{z}_i} = \frac{\partial \mathbf{y}_i}{\partial \mathbf{z}_i}\frac{\partial \mathbf{E}}{\partial \mathbf{y}_i}\frac{\partial g_j}{\partial \mathbf{E}} + \frac{\partial \mathbf{y}_i}{\partial \mathbf{z}_i}\frac{\partial \mathbf{V}_i}{\partial \mathbf{y}_i}\frac{\partial g_j}{\partial \mathbf{V}_i}, \quad i = 1,\dots,m \tag{22}$$

Such separation of the derivatives allows ObjectiveFnc and ConstraintFnc to compute only the derivatives with respect to their internal parameters, **E** and **V**. For volume-constrained compliance minimization, these derivatives are:

$$\frac{\partial f}{\partial E_\ell} = -\mathbf{U}^T\frac{\partial \mathbf{K}}{\partial E_\ell}\mathbf{U}, \quad \frac{\partial f}{\partial V_{\ell i}} = 0 \tag{23}$$

$$\frac{\partial g_j}{\partial E_\ell} = 0, \quad \frac{\partial g_j}{\partial V_{\ell i}} = \frac{A_\ell}{\sum_{\ell\in\mathcal{E}_j} A_\ell} \tag{24}$$

Then, the derivatives of **E** and **V** with respect to **Y** can be computed in MatIntFnc and MultiMatIntFnc, such that each function is only required to compute the derivatives with respect to its internal parameters, **Y** and **W**, respectively. Since **V** is not a function of **W**, we immediately compute:

$$\frac{\partial V_{kj}}{\partial y_{\ell i}} = \begin{cases} 1 \text{ if } \ \ell = k \ \text{ and } \ i = j \\ 0 \text{ otherwise} \end{cases} \tag{25}$$

in MatIntFnc. Using the chain rule again, we separate the sensitivities of the design parameter, **E**:

$$\frac{\partial \mathbf{E}}{\partial \mathbf{y}_i} = \frac{\partial \mathbf{w}_i}{\partial \mathbf{y}_i}\frac{\partial \mathbf{E}}{\partial \mathbf{w}_i}, \quad i = 1,\dots,m \tag{26}$$

such that MatIntFnc computes $\partial \mathbf{w}_i/\partial \mathbf{y}_i, i = 1,\dots,m$ and MultiMatIntFnc computes $\partial \mathbf{E}/\partial \mathbf{w}_i, i = 1,\dots,m$. For example, in the case of SIMP, MatIntFnc computes:

$$\frac{\partial w_{kj}}{\partial y_{\ell i}} = \begin{cases} p y_{\ell i}^{p-1} & \text{if } \ \ell = k \ \text{ and } \ i = j \\ 0 & \text{otherwise} \end{cases} \quad \text{(SIMP)} \tag{27}$$

and MultiMatIntFnc computes:

$$\frac{\partial E_k}{\partial w_{\ell i}} = (1-\varepsilon)\begin{cases} \prod_{\substack{j=1\\ j\neq i}}^{m}(1-w_{\ell j})E_i^0 \\ \quad -\sum_{\substack{p=1\\ p\neq i}}^{m} w_{\ell p}\prod_{\substack{r=1\\ r\neq p\\ r\neq i}}^{m}(1-w_{\ell r})E_p^0 & \text{if } \ \ell = k \\ 0 & \text{otherwise} \end{cases} \tag{28}$$

on lines 21 to 32, using assembly of a dProd matrix, in a similar manner as was done to compute **E** with a Prod matrix. Note that $\partial \mathbf{y}_i/\partial \mathbf{z}_i = \mathbf{P}^T$ is already known (see (14)), and thus, we have computed all terms necessary to obtain the sensitivities of the objective and constraints with respect to the design variables, **Z**, in (21) and (22). Noting the diagonal nature of the derivatives in (25), (27), and (28), we store these derivatives as NElem × NMat matrices. Then, the derivatives of the objective and constraints, stored as an NElem × NMat matrix and an NElem × NConstr × NMat matrix, respectively, are computed directly in the main function of PolyMat as follows:

```
%Compute design sensitivities
dfdz = P'*(dEdy.*repmat(dfdE,1,fem.NMat) + dVdy.*dfdV);
for c=1:opt.NConstr
  dgdz(:,c,:) = P'*(dEdy.*repmat(dgdE(:,c),1,fem.NMat) +...
                    dVdy.*reshape(dgdV(:,c,:),[fem.NElem,fem.NMat]));
end
```

4.7 Update scheme

We adopt the ZPR design variable update scheme, which allows us to efficiently handle multiple volume constraints (Zhang et al. 2018). The ZPR update is derived using Lagrangian duality of a series of linearized subproblems and takes advantage of the fact that the Lagrange multipliers associated with each constraint in these subproblems are independent of one another. As a result, the set of design variables associated with each constraint can be updated independently. From an implementation point of view, we loop over all volume constraints, each time passing only the design variables associated with the current constraint to UpdateScheme. The lines of code in PolyMat used to call the update scheme for each constraint are as follows:

```
%Update design variable and analysis parameters
for c=1:opt.NConstr
  ElemIndices = cell2mat(opt.ElemInd(c));
  MatIndices = cell2mat(opt.MatInd(c));
  [z(ElemIndices,MatIndices),Change(c)] = UpdateScheme(...
      dfdz(ElemIndices,MatIndices),g(c),...
      dgdz(ElemIndices,c,MatIndices),z(ElemIndices,MatIndices),opt);
end
```

Note that the structure of the UpdateScheme function itself remains unchanged from the original PolyTop code, with one exception. The multi-material interpolation function, m_M, defined in (20) is a non-monotonous function of $\mathbf{W}$, allowing $\partial m_M/\partial \mathbf{w}_i$ and $\partial m_E/\partial \mathbf{y}_i, i = 1, \ldots, m$ to become negative (see Fig. 3). Thus, the derivatives, dfdz, may become positive, leading to difficulties in the recursive update term when the updated design variable becomes undefined (or negative). Note that in the composition, $m_E = m_M \circ m_W$, with $p > 1$ (SIMP) or $q > 0$ (RAMP) as shown in Fig. 3c for SIMP, the derivatives, $\partial m_E/\partial \mathbf{y}_i, i = 1, \ldots, m$, become negative in regions in which the densities of both materials are relatively high. Also note, however, that this situation is penalized and, in practice, material mixing only results due to overlapping "gray" regions arising due to the filter operation. Thus, although not ideal, on line 77 of PolyMat (inside the UpdateScheme function), we simply ignore the sensitivities of element variables that have positive dfdz (Bendsøe and Sigmund 2003).

5 Extensions

We discuss a few extensions which, by design, can easily be handled in the PolyMat framework. The extensions include alternative filtering schemes, a means to avoid mixing at material interfaces, continuation on both the SIMP penalty parameter (p) and the mixing penalty parameter (γ), which is specific to multi-material problems, and, finally, incorporation of passive regions. The filtering schemes in Section 5.1 include the standard density filtering, the density + ZPR filter by Sanders et al. (2018), and a new combination of the density + sensitivity filter.

5.1 Filtering

In addition to the density filter implemented in PolyTop, we discuss two additional filters that can be used to achieve (possibly) improved designs. One is the "density + sensitivity filter", in which we combine the original density filter with the sensitivity filter proposed by Sigmund (1994), (1997). This combination of the density + sensitivity filter is a new approach that has not been explored elsewhere. The second is the "density + ZPR filter" proposed by Sanders et al. (2018). In addition to generating the element density field, $\mathbf{Y}$, with application of the density filter, these two additional filters modify inputs to the design variable update scheme, leading to different paths toward an optimal design. First, we briefly review the design variable update scheme and note how the two filters affect the update. Refer to Appendix B of PolyTop (Talischi et al. 2012b) and Zhang et al. (2018) for additional detail on derivation of the update scheme.

The design variable update is defined as:

$$z_{\ell i}^{\text{new}} = \begin{cases} z_{\ell i}^{+}, & z_{\ell i}^{*} \geq z_{\ell i}^{+} \\ z_{\ell i}^{-}, & z_{\ell i}^{*} \leq z_{\ell i}^{-} \\ z_{\ell i}^{*}, & \text{otherwise} \end{cases} \tag{29}$$

where $z_{\ell i}^{\text{new}}$ is the design point for the next iteration and $z_{\ell i}^{*}$ is a candidate design point for the next iteration that is accepted only if it does not violate lower and upper bounds, $z_{\ell i}^{-}$ and $z_{\ell i}^{+}$, of the search region, which are defined according to the box constraints, $\overline{\rho}$ and $\underline{\rho}$, and a move limit, M (opt.ZPRMove), as follows:

$$\begin{aligned} z_{\ell i}^{-} &= \max\left(\underline{\rho}, z_{\ell i}^{0} - M\right) \\ z_{\ell i}^{+} &= \min\left(\overline{\rho}, z_{\ell i}^{0} + M\right) \end{aligned} \tag{30}$$

The candidate design point, $z_{\ell i}^{*}$, is obtained from a fixed-point iteration:

$$z_{\ell i}^{*} = \underline{\rho} + (B_{\ell i})^{\eta}(z_{\ell i}^{0} - \underline{\rho}) \tag{31}$$

where η (opt.ZPREta) is a damping parameter and, according to the ZPR design variable update scheme,

$$B_{\ell i} = -\frac{\frac{\partial f}{\partial z_{\ell i}}|_{\mathbf{Z}=\mathbf{Z}_0}}{\lambda_j \frac{\partial g_j}{\partial z_{\ell i}}|_{\mathbf{Z}=\mathbf{Z}_0}} \tag{32}$$

Introducing a sensitivity filter, $\tilde{\mathbf{P}}$, the fixed-point iteration becomes:

$$z_{\ell i}^{*} = \underline{\rho} + (\tilde{B}_{\ell i})^{\eta}(z_{\ell i}^{0} - \underline{\rho}) \tag{33}$$

where

$$\tilde{B}_{\ell i} = -\frac{\sum_{k=1}^{N} (\tilde{P}_i)_{\ell k} \frac{\partial f}{\partial z_{ki}}|_{\mathbf{Z}=\mathbf{Z}_0}}{\lambda_j \frac{\partial g_j}{\partial z_{\ell i}}|_{\mathbf{Z}=\mathbf{Z}_0}} \tag{34}$$

and the components of the sensitivity filter are:

$$(\tilde{P}_i)_{\ell k} = \frac{1}{z_{\ell i}} P_{\ell k} z_{ki} \tag{35}$$

Implementation of the density + sensitivity filter amounts to addition of the following line of code:

```
dfdz = (P*(z.*dfdz))./z;
```

immediately following computation of dfdz on line 18 of PolyMat.

The density + ZPR filter introduced by Sanders et al. (2018) applies an additional density filter during the fixed-point iteration, as follows:

$$z_{\ell i}^{*} = \underline{\rho} + (B_{\ell i})^{\eta}\left(\sum_{k=1}^{N} P_{\ell k} z_{ki}^{0} - \underline{\rho}\right) \tag{36}$$

Implementation of the density + ZPR filter in PolyMat requires passing both z and V to UpdateScheme, which are re-named as z0 and V0, respectively, inside of

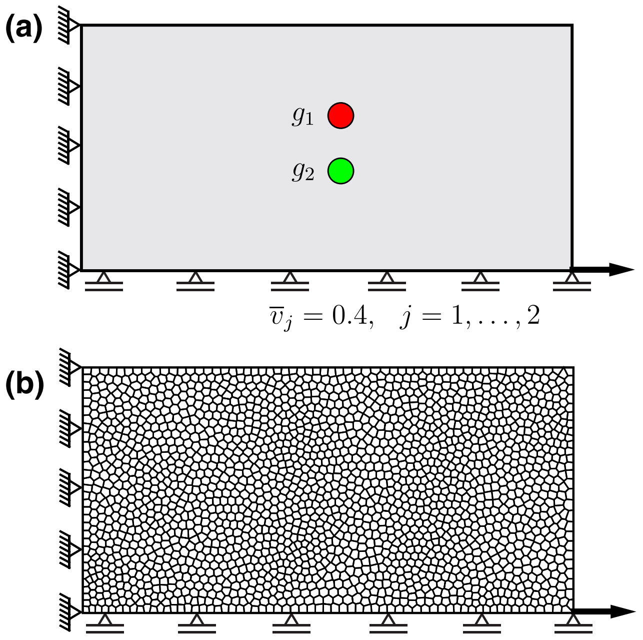


Fig. 4 Short column: **a** Domain, boundary conditions, and constraint specification (the red material controlled by g_1 is 6.67 times stiffer than the green material controlled by g_2); **b** finite element mesh (color online)

`UpdateScheme`. Then after reshaping the array on line 70 of `PolyMat` (inside of the `UpdateScheme` function), `z0` should be replaced by `V0` on line 77. Note that the code provided in Appendix E and in the downloadable version, adopt this density + ZPR filter implementation.

The three filtering techniques discussed here are summarized in Table 8 in Appendix B. A comparison of the density filter acting alone with the density + sensitivity filter and the density + ZPR filter is provided in Section 7.1.

5.2 Avoiding mixing at material interfaces

Although composition of the penalty function, m_W, and material interpolation function, m_M, penalizes material mixing (see Fig. 3c), some mixing may occur due to overlapping “gray” regions that arise due to the filter operation. We optimize the symmetric short column shown in Fig. 4 with two candidate materials (red and green represent materials with $E_1 = 1$ and $E_2 = 0.15$, respectively), and vary the filter radius to show the effect it has on material mixing. For this demonstration, we adopt the density filter + ZPR filter, but all three filtering schemes discussed in Section 5.1 yield similar results for this simple example. The optimization parameters used in this study are provided in Table 2. As the filter radius is reduced in Fig. 5a, b, and c, the length scale of the mixing region decreases until there is no mixing in Fig. 5c (when no filter is applied). Note, however, that when no filter is considered, some voids appear at the interface of the two materials (Fig. 5c). To achieve the crisp boundaries shown in Fig. 5d and e, without any material mixing and without the appearance of voids, we use the converged filtered solutions from Fig. 5a and b, respectively, as the initial guess and re-run without any filter. In general, this approach is effective at removing the material mixing caused by the filter. Note that in some problems small oscillations are observed when the filter is turned off and the maximum number of iterations must be used as the stopping criterion. Alternatively, gradually reducing the filter radius tends to mitigate the oscillatory behavior.

Table 2 Input parameters used for the short column

Number of elements	2,000
SIMP penalty parameter, p	3
Material interpolation factor, γ	1
ZPR move limit, δ	0.2
Convergence tolerance	0.01
max. num. iterations (per R)	100

5.3 Continuation on the material interpolation parameters

Continuation on the penalty parameter is a technique often used for single-material topology optimization in which the penalty parameter is set to recover the convex problem at the start (e.g., $p = 1$ for SIMP) and is gradually increased to achieve the required penalization on intermediate densities as the optimization progresses. For multi-material topology optimization, we introduce a second mixing penalty parameter, $0 \leq \gamma \leq 1$, that leads to a convex formulation when $\gamma = 0$ and penalizes material mixing when $\gamma > 0$. The material interpolation function, re-written to accommodate continuation:

$$m_M\left(\mathbf{w}'_\ell\right) = \varepsilon + (1-\varepsilon)\sum_{i=1}^{m} w_{\ell i}\prod_{\substack{j=1\\ j\neq i}}^{m}(1-\gamma w_{\ell j})E_i^0, \quad \ell = 1, \ldots, N \tag{37}$$

is identical to (20) when $\gamma = 1$. Figure 6 shows how the material interpolation function, m_E, changes as p and γ are increased from 1 to 4 and 0 to 1, respectively. Note that the penalty parameter, p, controls the curvature of the surface of m_E and the material interpolation parameter, γ, controls the allowable magnitude of m_E when both materials are fully dense (e.g., for the 2-material case in

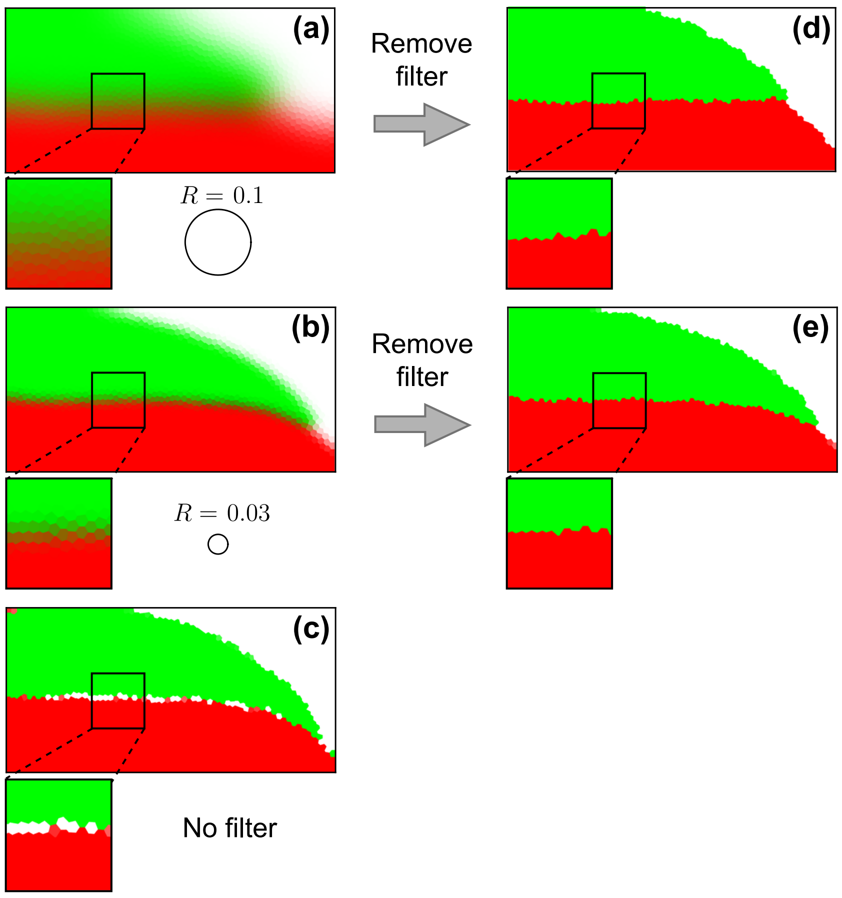


Fig. 5 Demonstration of how material mixing is influenced by the filter radius, R, using the short column in Fig. 4. The results in the left column are based on a constant filter with radius **a** $R = 0.1$; **b** $R = 0.03$; and **c** no filter. The results in the right column, **d** and **e**, are obtained from the converged solutions in **(a)** and **(b)**, respectively, without any filter. (color online)

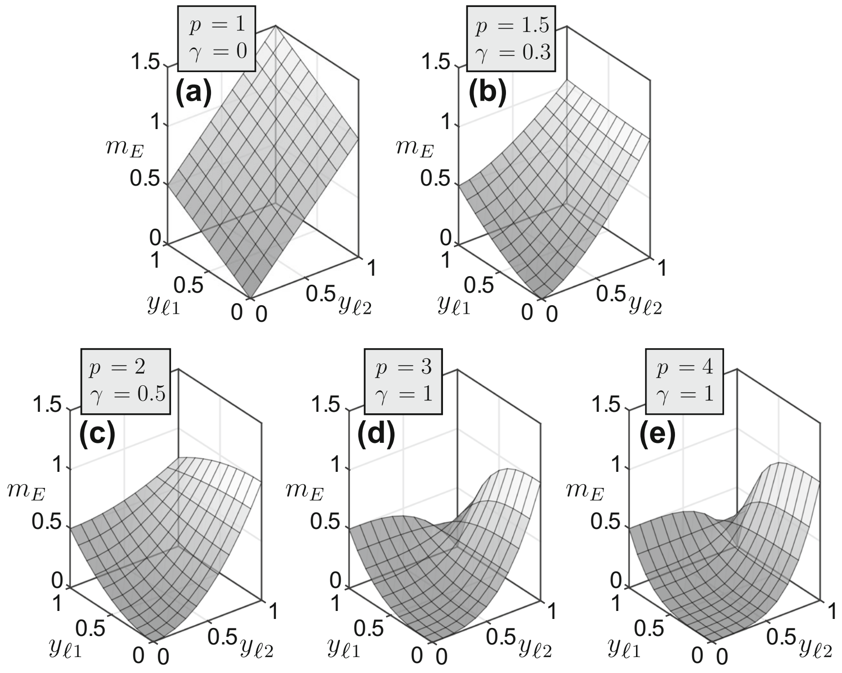


Fig. 6 Continuation on the material interpolation parameters: **a** $p = 1, \gamma = 0$; **b** $p = 1.5, \gamma = 0.3$; **c** $p = 2, \gamma = 0.5$; **d** $p = 3, \gamma = 1$; **e** $p = 4, \gamma = 1$

Fig. 6, γ controls the height of the point at $y_{\ell 1} = 1$ and $y_{\ell 2} = 1$). In Fig. 6a, when $p = 1$ and $\gamma = 0$, the material interpolation is convex and material mixing is favorable. As γ is increased toward 1, mixing is increasingly penalized. Note that the specific continuation scheme (i.e., increments of p and γ) is heuristic and different schemes will lead to different results. Similarly to continuation for single-material topology optimization, continuation on the multi-material interpolation parameters often leads to an improved objective value in the final design, but not always.

5.4 Passive regions

The multi-material formulation provides a simple framework for incorporating passive regions, i.e., regions that are assigned solid or void at the start and do not participate in the optimization. Since `PolyMesher` can be used to define meshes on complex domains (e.g., with holes), we focus here on passive regions in which the elements in the passive region are solid. When multiple materials are available, a passive solid region implies that all of the design variables in that region associated with one candidate material are equal to one and the design variables in that region associated with all other candidate materials are equal to zero.

To consider passive regions in the implementation, the design variables associated with the passive elements must be set to zero or one during initialization. Thus, we define two additional cell arrays, `SElemInd` and `SMatInd`, which contain a vector of element indices for each passive region and the corresponding solid material index for each passive region, respectively. These cell arrays can be defined within the constraint files in a similar way to how we defined `ElemInd` and `MatInd` (see an example in Fig. 2c). Then, all four cell arrays, `ElemInd`, `MatInd`, `SElemInd`, and `SMatInd`, are passed to the `InitialGuess` function to set the initial densities of both the optimizable and non-optimizable regions. It is also important to ensure that element indices corresponding to the passive regions are left out of the `ElemInd` array so that they are never passed to `UpdateScheme` and will remain as initialized throughout the optimization.

One additional detail is that the matrix associated with the regularization mapping, `P`, should be modified so that it does not alter the design variables associated with elements in the passive regions. To do so, we pass `fem.SElem`, a vector of all passive element indices, to `PolyFilter`, so that all entries associated with indices of the passive elements are assigned zero during assembly of `P` (refer to Talischi et al. (2012b) for additional detail on the `PolyFilter` function).

6 Efficiency

We study the efficiency of `PolyMat` using the MBB beam problem in which the design domain is a rectangle of length 6 and height 1. Due to symmetry, only half of the domain is discretized into a 300 × 100 orthogonal mesh. In each case, the optimization is run for 200 iterations and no continuation of the material interpolation parameters or filter radius are considered. We use SIMP with $p = 3$ and a constant filter radius of $R = 0.025$. All runs are performed using Matlab R2018a on a desktop computer with an Intel(R) Xeon(R) CPU E5-1660 v3, 3.00 GHz processor and 64.0 GB of RAM.

Since `PolyMat` is structured similarly to `PolyTop`, we expect the two codes to perform similarly when the number of materials and constraints are relatively low. In Table 3,

Table 3 Code runtime breakdown comparison of `PolyTop` (1 material, 1 constraint), `PolyMat` (1 material, 1 constraint), and `PolyMat` (5 materials, 2 constraints) for the MBB beam design run for 200 iterations (times are in seconds with percentage of total runtime of `PolyScript` in parentheses)

	`PolyTop` (1 mat., 1 constr.)	`PolyMat` (1 mat., 1 constr.)	`PolyMat` (5 mat., 2 constr.)
Computing **P**	2.94 (2.6%)	3.01 (2.7%)	2.94 (2.4%)
Assembling **K**	46.99 (42.4%)	47.11 (42.3%)	46.62 (37.8%)
Solving **KU** = **F**	36.67 (33.1%)	37.96 (34.0%)	38.75 (31.4%)
Mapping **Z**, **E**, and **V**	0.125 (0.1%)	0.16 (0.1%)	1.33 (1.1%)
Computing constraints	0.02 (0.0%)	0.13 (0.1%)	0.29 (0.2%)
Computing constraint sensitivities	0.05 (0.0%)	0.15 (0.1%)	1.95 (1.6%)
Computing compliance sensitivities	8.42 (7.6%)	8.63 (7.7%)	14.08 (11.4%)
Design update	6.53 (5.9%)	1.79 (1.6%)	3.96 (3.2%)
Plotting the solutions	7.09 (6.4%)	7.83 (7.0%)	8.09 (6.6%)
Total time of `PolyScript`	**110.91**	**111.50**	**123.33**

The bold in table is to emphasize the total runtimes we compare the breakdown of code runtime for a single-material design with a single global volume constraint and note that the performance of the two codes is indeed similar. Note that the two codes differ by less than one second. The last column of Table 3 also breaks down the code runtime for a PolyMat run of the same MBB beam problem with five candidate materials and two volume constraints, and it is shown that the increased CPU time for a typical multi-material problem (about 12 s) is small relative to PolyTop.

Additionally, we study the efficiency degradation of PolyMat as the number of constraints and the number of materials increases. In Fig. 7a, the total CPU time is reported for the MBB beam problem, considering a single candidate material and a local volume constraint controlling each of 1, 2, 4, 8, and 16 sub-regions, that is, 1×1, 2×1, 2×2, 4×2, and 4×4 sub-regions, respectively. Increased cost due to an increased number of constraints manifests itself in additional time for the ZPR update, since the UpdateScheme function must be accessed a number of times in each iteration. However, the ZPR update accounts for a relatively small percentage of the total runtime, and thus, as indicated in Fig. 7a, the computational cost does not significantly increase as the number of constraints increases. In fact, it takes less than 10 additional seconds to run the problem with 16 constraints as compared to the problem with a single constraint.

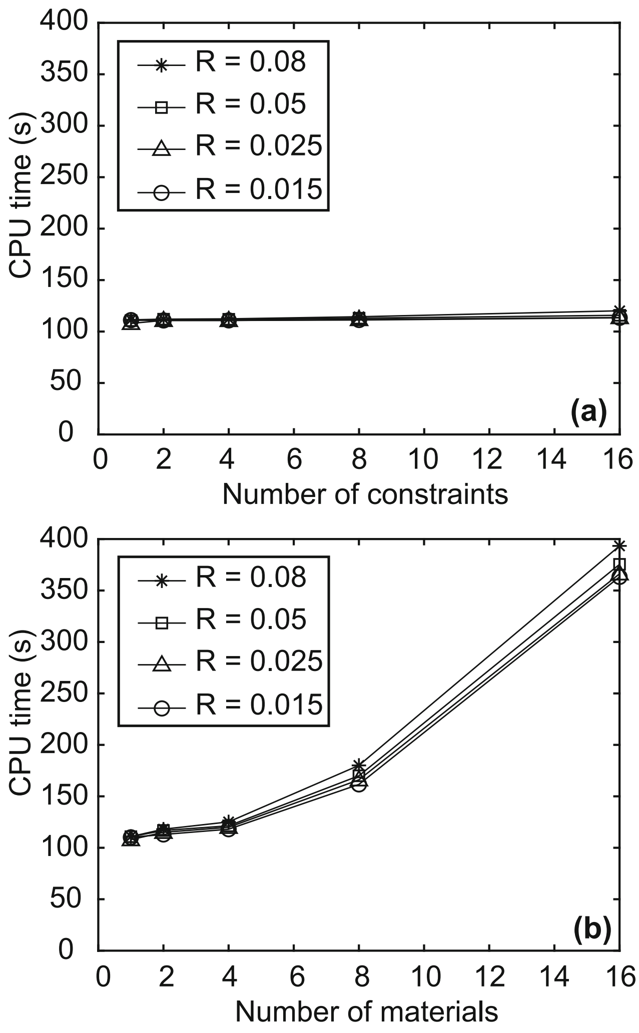


Fig. 7 CPU time (seconds) for the MBB beam problem with a 300×100 orthogonal mesh run for 200 iterations: **a** a single candidate material controlled by a varying number of local volume constraints on sub-regions of the domain; and **b** a single global volume constraint controlling a varying number of candidate materials

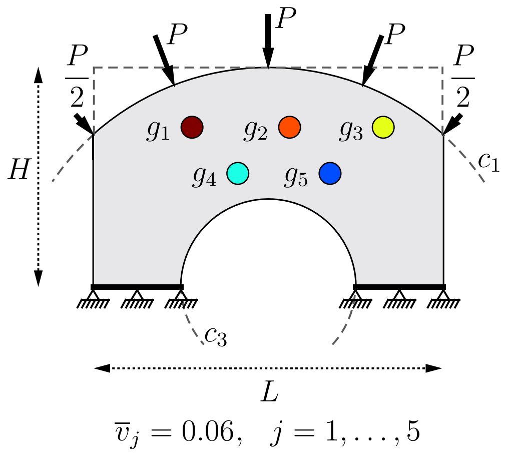


Fig. 8 Domain, boundary conditions, and constraint specification for the curved beam problem in which five global constraints ($\overline{v}_j = 0.06, \quad j = 1, \ldots, 5$) each control a single candidate material (color online)

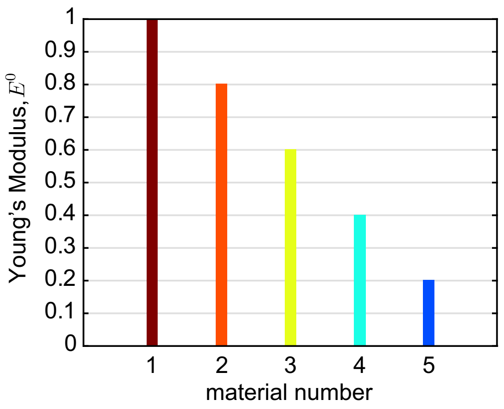


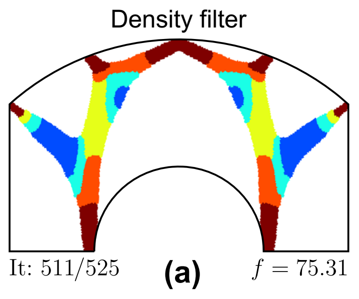
Fig. 9 Young's modulus (and color used in the result visualization) of the five candidate materials available to the curved beam problem (color online)

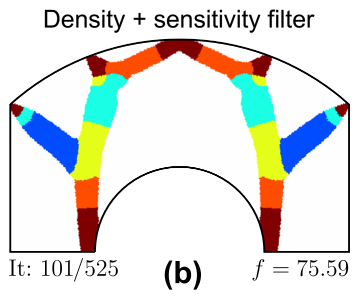
In Fig. 7b, the total CPU time is reported for the MBB beam problem, considering a single global volume constraint controlling 1, 2, 4, 8, and 16 candidate materials, that is, 30,000, 60,000, 120,000, 240,000, and 480,000 design variables, respectively. As the number of materials increases, more memory is required to store the design variables. Thus, as illustrated in Fig. 7b, the computational cost increases from about 2 min for a single candidate material to about 6.5 min for sixteen candidate materials.

Figure 7 also shows that an increase in the filter radius leads to increased computational time for a given number of constraints and materials. The increase is more pronounced as the number of materials increases; however, based on numerical experimentation, the increased CPU time due to an increased filter radius is negligible. As a side note, we also observe that although only the stiffest material arises in all of the designs obtained in this efficiency study (aside from the 5-material, 1-constraint example in Table 3), the designs are similar, but not identical, since the initial guess is different in each case.

7 Numerical examples

Five numerical examples illustrate use of the `PolyMat` code. Domain and constraint files for each example are provided with download of `PolyMat` (see Appendix C for a list of examples provided with `PolyMat`). In all cases, the box constraints, $\underline{\rho}$ and $\overline{\rho}$, are 0 and 1, respectively, the move limit used in the ZPR update is 0.2, and the stopping criterion is either the maximum number of iterations or





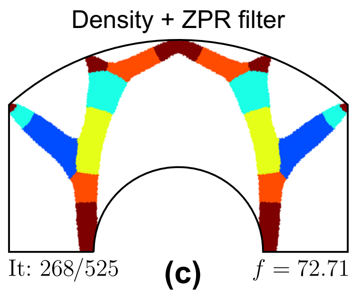
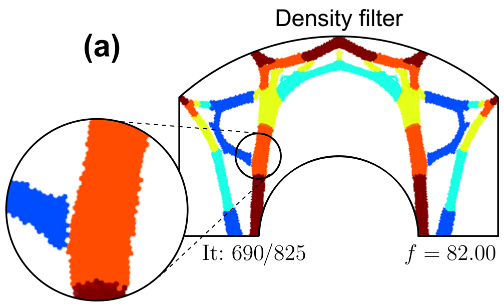
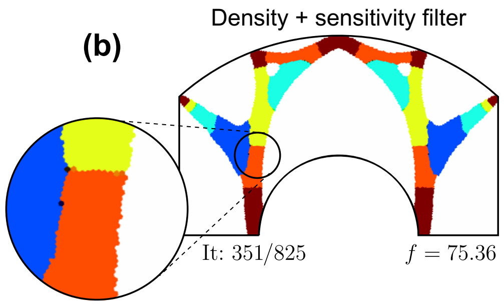


Fig. 10 Curved beam optimized for five global constraints ($\overline{v}_j = 0.06, \quad j = 1, \ldots, 5$), each controlling one of the 5 candidate materials in the entire domain, without continuation on the material interpolation parameters ($p = [3, 3]$ and $\gamma = [1, 1]$), and with filter reduction ($R = [0.03, -1]$). The designs are shown for the **a** density filter; **b** density + sensitivity filter; and **c** density + ZPR filter. (color online)





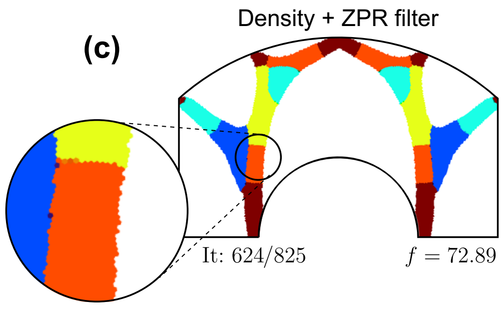


Fig. 11 Curved beam optimized for five global constraints ($\overline{v}_j = 0.06, \quad j = 1, \ldots, 5$), each controlling one of the 5 candidate materials in the entire domain, with continuation on the material interpolation parameters ($p = [1, 1.5, 2, 3, 4]$, $\gamma = [0, 0.3, 0.5, 1, 1]$), and with filter reduction ($R = [0.03, 0.03, 0.03, 0.03, -1]$). The designs are shown for the **a** density filter, **b** density + sensitivity filter, and **c** density + ZPR filter. (color online)

the infinity norm of the change in design variables with convergence tolerance of 0.01 (whichever is met first). Convergence plots for selected examples are provided in Appendix A.

7.1 Comparison of the filtering techniques

The first example is used to compare the density filter acting alone to the density filter in combination with a sensitivity filter (i.e., density + sensitivity filter) and the density filter in combination with a ZPR filter (i.e., density + ZPR filter). Additionally, we compare the results using the continuation scheme presented in Section 5.3 to those with constant material interpolation parameters (p and γ). For these comparisons we consider the curved beam problem shown in Fig. 8 with dimensional parameters $H = 1.25$, $L = 2$, $c_1 = (x_{c1}, y_{c1}, r_{c1}) = (0, 0, 1.5)$, $c_3 = (0, 0.25, 0.5)$, and the load parameter $P = 1$. Using PolyMesher, the domain is discretized into 30,000 polygonal finite elements with a vertical line of symmetry (symmetry is on the mesh only, i.e., symmetry is not enforced on the design variables). As shown in Fig. 9, the candidate materials have Young's moduli regularly spaced in the range [0.2, 1] and each material is assigned a color according to Matlab's "jet" colormap, such that a material with Young's modulus equal to one is shown in dark red and a material with Young's modulus equal to zero is shown in dark blue.

In Fig. 10 the density filter is compared with the density + sensitivity filter and the density + ZPR filter considering constant SIMP penalty parameter ($p = [3, 3]$), mixing penalty parameter ($\gamma = [1, 1]$), and filter reduction ($R = [0.03, -1]$). The problem is run for a maximum of 500 iterations before turning the filter off, after which is it run for a maximum of 25 additional iterations. This example captures the key observations made from comparisons of many different problems using various parameters. First, the density filter acting alone tends to result in designs with smaller strips or islands of different materials than the density + sensitivity filter or the density + ZPR filter. Additionally, the objective function tends to be lowest with the density + ZPR filter and highest with the density + sensitivity filter. Lastly, the density filter acting alone tends to require more iterations to converge according to the infinity norm of the change in design variables, and as a result, the maximum number of iterations often controls. The latter two observations also hold, in general, for single-material problems.

In Fig. 11 the density filter is compared with the density + sensitivity filter and the density + ZPR filter considering continuation on the SIMP and mixing penalty parameters ($p = [1, 1.5, 2, 3, 4]$, $\gamma = [0, 0.3, 0.5, 1, 1]$) and filter reduction ($R = [0.03, 0.03, 0.03, 0.03, -1]$). The first four continuation steps are each run for a maximum of 200 iterations and the final continuation step (when the filter is turned off) is run for a maximum of 25 additional iterations.

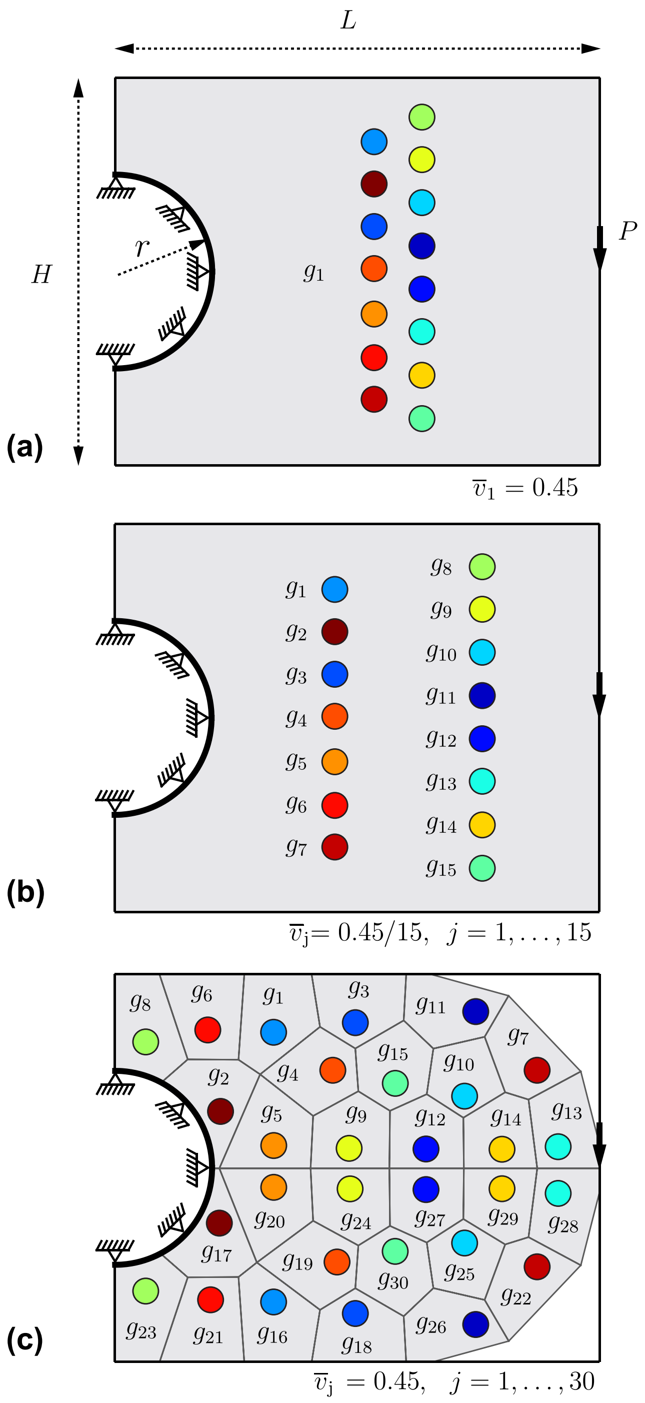

L
H
P
r
g_1
(a)
$\overline{v}_1 = 0.45$
g_1 g_2 g_3 g_4 g_5 g_6 g_7 g_8 g_9 g_{10} g_{11} g_{12} g_{13} g_{14} g_{15}
(b)
$\overline{v}_j = 0.45/15, \ j = 1, \ldots, 15$
g_1 … g_{30}
(c)
$\overline{v}_j = 0.45, \ j = 1, \ldots, 30$

Fig. 12 Domain and boundary conditions for the Michell cantilever with circular support: **a** in case 1, a single global constraint ($\overline{v}_1 = 0.45$) controls all fifteen candidate materials; **b** in case 2, fifteen global constraints ($\overline{v}_j = 0.45/15, \ j = 1, \ldots, 15$) each control a single candidate material; and **c** in case 3, thirty local constraints ($\overline{v}_j = 0.45, \ j = 1, \ldots, 30$) that are symmetric about the horizontal center line each control a single candidate material (color online)

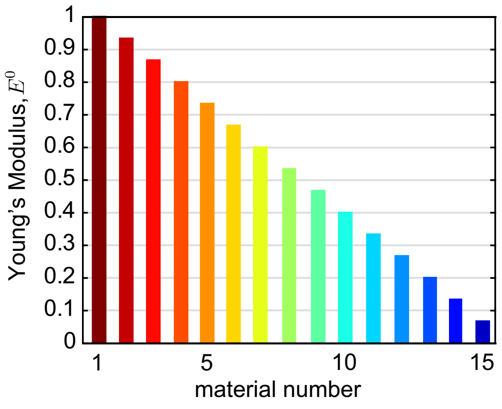


Fig. 13 Young's modulus (and color used in the result visualization) of the fifteen candidate materials available in all three of the Michell cantilever cases (color online)

Here, similar observations are made to those without continuation on the material interpolation parameters. One additional note is that filter reduction in combination with continuation on the material interpolation parameters tends to cause disconnected members when the density filter acts alone. Note also that in this case, the chosen continuation scheme on the material interpolation parameters causes increased objective values in all three cases.

Another observation that is not illustrated with this example is that even for a symmetric problem with a symmetric mesh, the density + ZPR filter is more inclined to result in slightly asymmetric results (an example is provided in Section 7.2). The remaining examples adopt the density + ZPR filter with filter reduction, and all but the serpentine example in Section 7.4 adopt continuation on the material interpolation parameters.

7.2 Michell cantilever

In this example, we use a "Michell" cantilever with circular support (Michell 1904) to illustrate the three main types of volume constraints that can be specified with `PolyMat`: (1) global constraints that control many materials; (2) global

Table 4 Input parameters used for all three cases of the Michell cantilever with circular support (brackets indicate continuation and $R = -1$ indicates no filter)

Number of elements	90,000
SIMP penalty parameter, p	[1, 1.5, 2, 3, 4]
Material interpolation factor, γ	[0, 0.3, 0.5, 1, 1]
Filter radius, R	[0.05, 0.05, 0.05, 0.05, −1]
Max. number of iterations (per continuation step)	200

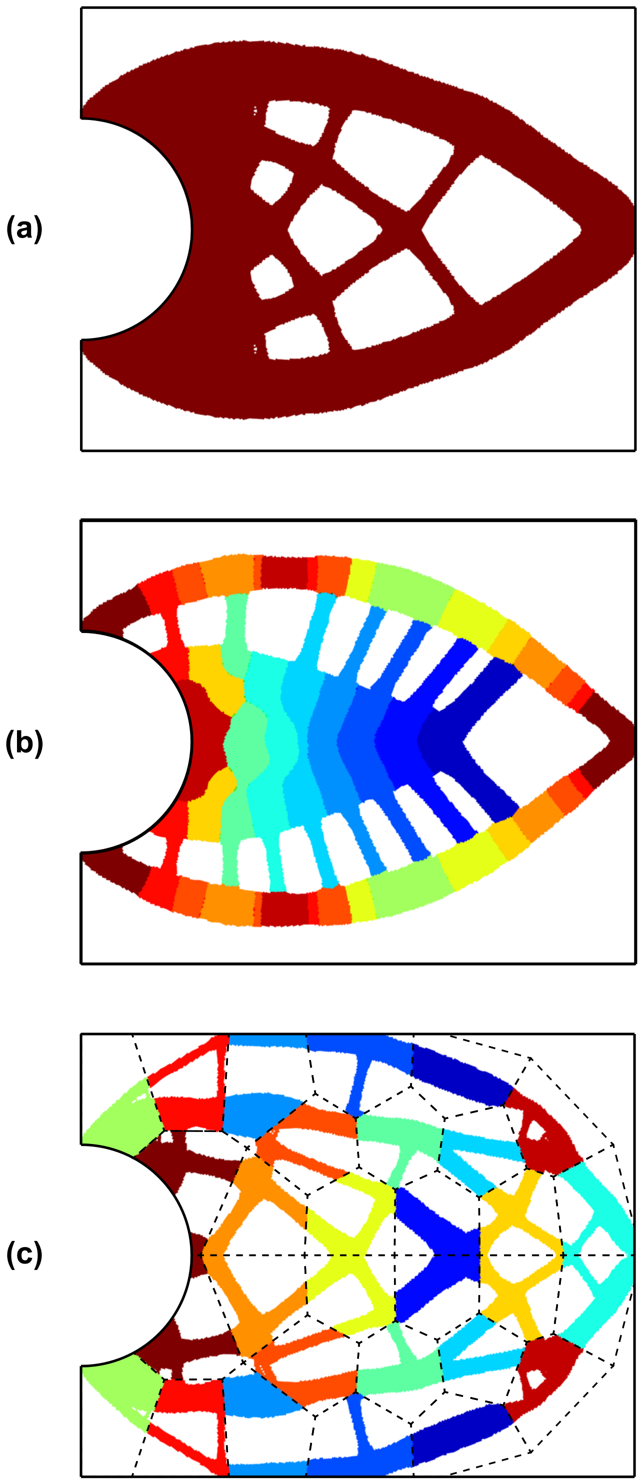


Fig. 14 Michell cantilever with circular support, optimized for: **a** a single global constraint ($\overline{v}_1 = 0.45$) controlling all 15 materials in the entire domain; **b** 15 global constraints ($\overline{v}_j = 0.45/15, \quad j = 1, \ldots, 15$), each controlling one of the 15 candidate materials in the entire domain; and **c** 30 local constraints ($\overline{v}_j = 0.45, \quad j = 1, \ldots, 30$), each controlling a single candidate material in a sub-region of the domain (color online)

constraints that control a single material; and (3) local constraints that control one or more materials. The domain and boundary conditions for the Michell cantilever are provided in Fig. 12a, where the dimensional parameters are defined as $H = 4$, $L = 5$, and $r = 1$, and the load $P = 1$. The sub-regions and materials associated with each constraint for cases 1, 2, and 3 are provided in Fig. 12a–c, respectively. In all three cases, fifteen candidate materials are available. As shown in Fig. 13, the candidate materials have Young's moduli regularly spaced in the range $[0.0667, 1]$ and each material is assigned a color according to Matlab's "jet" colormap, such that a material with Young's modulus equal to one is shown in dark red and a material with Young's modulus equal to zero is shown in dark blue. Note that the materials are assigned in random order to the constraints to highlight the fact that the ZPR algorithm is order-independent. Using PolyMesher, we construct a 90,000-element polygonal finite element mesh with a horizontal line of symmetry (symmetry is on the mesh only, i.e., symmetry is not enforced on the design variables). Continuation on the material interpolation parameters (p and γ) is considered. In the final continuation step, the filter is turned off. The optimization parameters used for all three cases are provided in Table 4.

In case 1, we consider a single global volume constraint ($\overline{v}_1 = 0.45$) that controls all elements in the domain and all candidate materials. The Matlab code used to generate the cell arrays, ElemInd and MatInd, for case 1 is as follows:

```
%Case 1: Single global constraint
NMat = 15;
VolFrac = 0.45;
mat = linspace(NMat,1,NMat)'./NMat;
matorder = randperm(NMat);
```

In case 2, we consider fifteen global volume constraints ($\overline{v}_j = 0.45/15$, $j = 1, \ldots, 15$): one for each of the fifteen candidate materials. The Matlab code used to generate the cell arrays, ElemInd and MatInd, for case 2 is as follows:

Table 5 Input parameters used for the curved beam problem (brackets indicate continuation and $R = -1$ indicates no filter)

Number of elements	100,000
SIMP penalty parameter, p	[1, 1.5, 2, 3, 4]
Material interpolation factor, γ	[0, 0.3, 0.5, 1, 1]
Filter radius, R	[0.016, 0.016, 0.016, 0.016, −1]
Max. number of iterations (per continuation step)	200

```
% Case 2: Fifteen global constraints
%   (one for each material)
NMat = 15;
NConstr = NMat;
VolFrac = 0.45/NMat*ones(NMat,1);
mat = linspace(NMat,1,NMat)'./NMat;
matorder = randperm(NMat);
for c = 1:opt.NConstr
  ElemInd{c} = linspace(1,NElem,NElem);
  MatInd{c} = matorder(c);
end
```

Finally, in case 3, we consider fifteen local volume constraints on each side of the domain's horizontal line of

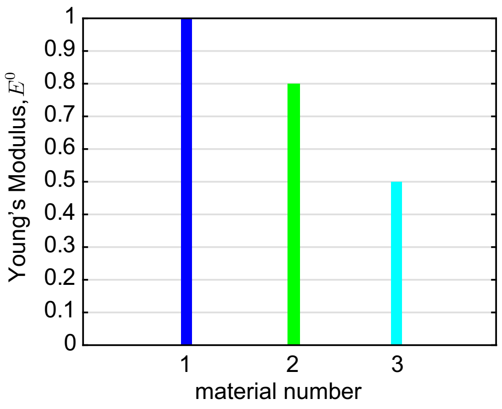


Fig. 15 Young's modulus (and color used in the result visualization) of the three candidate materials available in the curved beam problem (color online)

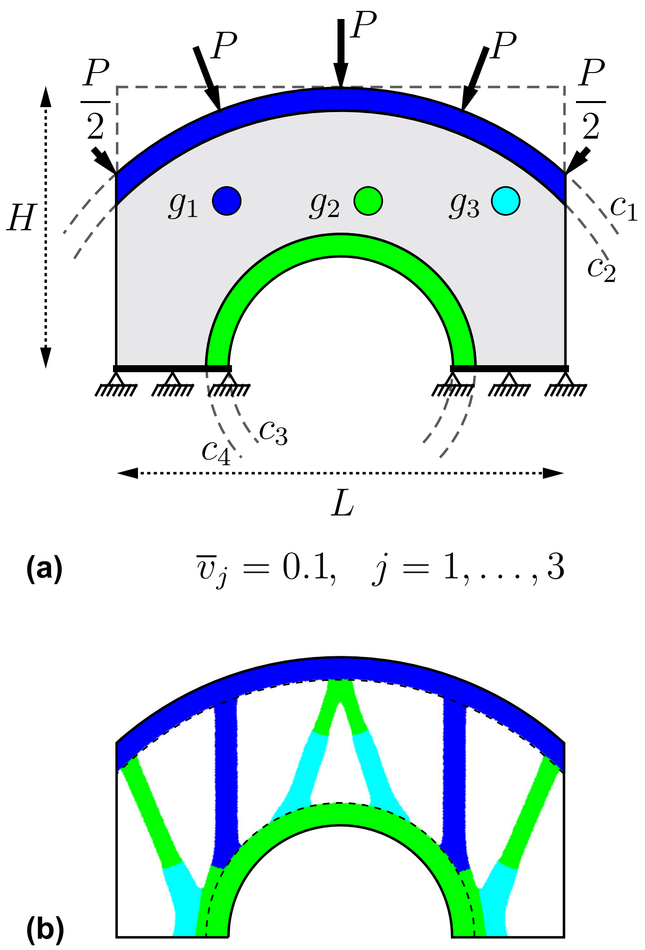


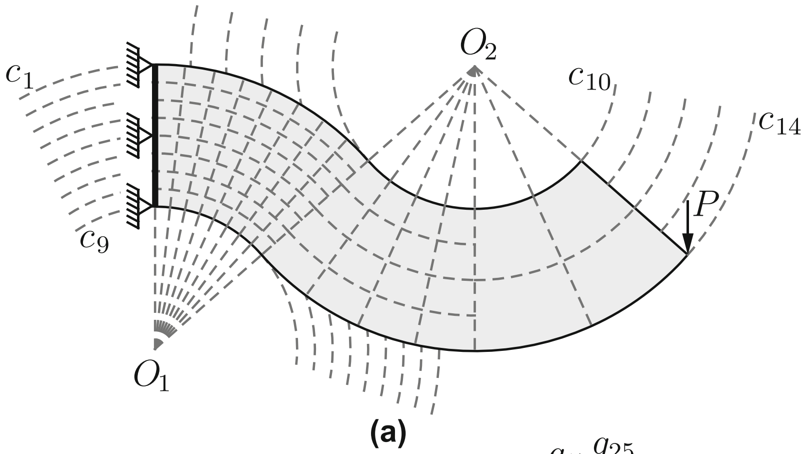
Fig. 16 Curved beam problem: **a** domain, boundary conditions, constraint specification, and passive region specification; and **b** optimized design (color online)

symmetry (i.e., thirty local constraints, $\overline{v}_j = 0.45, \quad j = 1, \ldots, 30$), such that a single material is available in each sub-region. The Matlab code used to generate the cell arrays, ElemInd and MatInd, for case 3 is as follows:

```
% Case 3: Thirty local constraints
NMat = 15;
NConstr = 2*NMat;
VolFrac = 0.45*ones(NConstr,1);
mat = linspace(NMat,1,NMat)'./NMat;
matorder = [randperm(NMat) randperm(NMat)];
for c = 1:NConstr
  ElemInd{c} = find(Dist{c} <= 0);
  MatInd{c} = matorder(c);
end
```

Note that in the code listing for case 3, Dist is a cell array in which each cell represents a constraint and contains the distance values of each element for the sub-region associated with that constraint. In particular, PolyMesher is used to generate the sub-regions for case 3, which are defined by the "nodes" and "elements" of a polygonal mesh with 15 "elements" on each side of the horizontal line of symmetry of the Michell domain intersected with a circle of radius H that is tangent to the point of load application.

The designs obtained for cases 1, 2, and 3 are provided in Fig. 14. As expected, when one global constraint controls all fifteen candidate materials (Fig. 14a), the final design is composed only of the stiffest candidate material. When



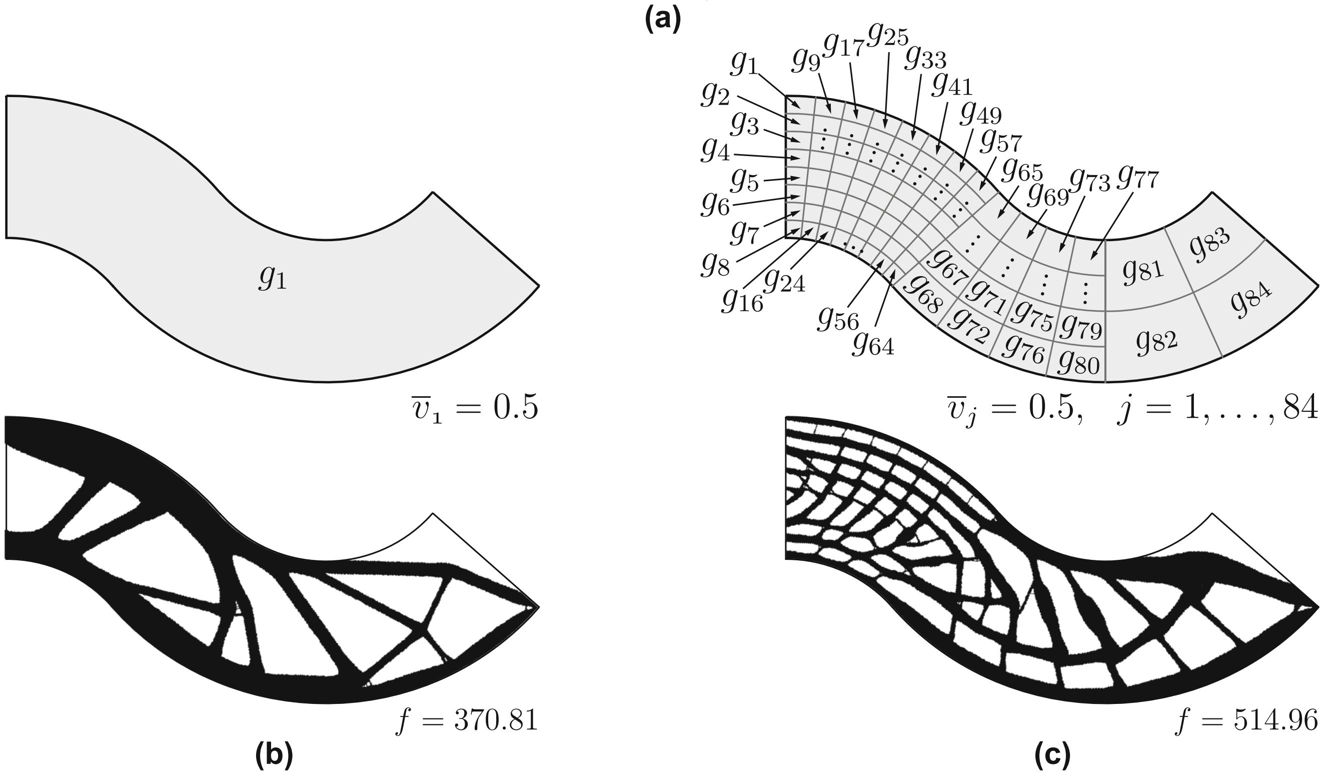

g_1
$\overline{v}_1 = 0.5$
$f = 370.81$
(b)
g_1 g_9 g_{17} g_{25} g_{33} g_{41} g_{49} g_{57} g_{65} g_{69} g_{73} g_{77}
g_2 g_3 g_4 g_5 g_6 g_7 g_8 g_{16} g_{24} g_{56} g_{64}
g_{67} g_{68} g_{71} g_{72} g_{75} g_{76} g_{79} g_{80}
g_{81} g_{82} g_{83} g_{84}
$\overline{v}_j = 0.5, \quad j = 1, \ldots, 84$
$f = 514.96$
(c)

Fig. 17 Serpentine problem: **a** domain and boundary conditions; **b** the single region used to define a single global volume constraint with volume fraction, $\overline{v}_1 = 0.5$, (top) and resulting optimized design (bottom); and **c** the 84 sub-regions used to define 84 local volume constraints each with volume fraction, $\overline{v}_j = 0.5$, $j = 1, \ldots, 84$, (top) and resulting optimized design (bottom)

each material is controlled by a separate global constraint (Fig. 14b), all fifteen candidate materials appear in the final design and they are distributed such that the stiffer materials are located where stress is expected to be high and more compliant materials are located where stress is expected to be low. In the case of local volume constraints (Fig. 14c), all of the constraint boundaries are respected. Also note that the design in Fig. 14b is not perfectly symmetric about the horizontal centerline.

7.3 Curved beam with passive regions

Here, we use the curved beam problem described in Fig. 16a to demonstrate the ease with which we may specify passive regions (i.e., regions that do not participate in the optimization) using PolyMat. The dimensional parameters defining the domain are $H = 1.25$, $L = 2$, $c_1 = (x_{c1}, y_{c1}, r_{c1}) = (0, 0, 1.5)$, $c_2 = (0, 0, 1.4)$, $c_3 = (0, 0.25, 0.5)$, and $c_4 = (0, 0.25, 0.6)$, and the load parameter $P = 1$. The curved beam is discretized into 100,000 polygonal finite elements via PolyMesher and is designed considering three candidate materials with properties given in Fig. 15. The top and bottom curved surfaces are considered passive regions and are assigned materials 1 and 2, respectively, in the constraint file. Note that the elements with seeds inside of the passive regions are assigned to SElemInd and not to ElemInd; thus, they are not updated after initialization. Three constraints each control one of the three candidate materials in the optimizable region such that each of the three candidate materials is limited to occupy no more than 10% of the optimizable region volume, i.e., $\overline{v}_j = 0.1, j = 1, \ldots, 3$. Continuation on the material interpolation parameters (p and γ) is considered. In the final continuation step, the filter is turned off. The optimization parameters used for the curved beam problem are provided in Table 5. The resulting design is shown in Fig. 16c where it is clear that the passive regions have been respected.

7.4 Exploring local volume constraints

In the final example, we explore how specifying the constraint sub-regions and material properties in a clever way can provide control over geometric features. First, we consider a single candidate material and work with the serpentine domain shown with boundary conditions in Fig. 17a, where the centers of the circles are $O_1 = (0, -2.6458)$ and $O_2 = (9, 5.2915)$, circles c_1 and c_{14} each have radius 8, circles c_9 and c_{10} each have radius 4, and the load $P = 1$. Figure 17b shows the result of an optimization considering a single global constraint that limits the material

Table 6 Input parameters used for the serpentine problem (brackets indicate continuation and $R = -1$ indicates no filter)

Number of elements	75,000
SIMP penalty parameter, p	[3, 3]
Material interpolation factor, γ	[1, 1]
Filter radius, R	[0.1, −1]
Max. number of iterations (per continuation step)	500

to occupy no more than 50% of the total domain volume, i.e., $\overline{v}_1 = 0.5$. Seeking a structure with finer details at the left end, the domain is divided into the 84 sub-regions shown in the top portion of Fig. 17c. The single candidate material is limited to occupy no more than 50% of each sub-region volume, i.e., $\overline{v}_j = 0.5, j = 1, \ldots, 84$, and the resulting design is provided in the bottom portion of Fig. 17c. As expected, the design with more constraints has a higher objective value. Table 6 lists additional parameters input to PolyMat for both of the serpentine designs.

Next, we consider the donut-shaped domain with torsion boundary conditions shown in Fig. 19a (Zegard and Paulino 2014), where the diameters of the inner and outer circles are 0.25 and 2, respectively, and the load $P = 1$. Five constraints are defined, each in a different sub-region of the domain, and each controlling one of the five candidate materials described in Fig. 18. Additional input parameters are provided in Table 7. The resulting design in Fig. 19b has no lines of symmetry, in contrast to the single-material design provided in Fig. 19c, which has five lines of symmetry and resembles a flower (perfect symmetry is not achieved due to the polygonal mesh). Notice that the complexity of features in the five-material design is correlated with the stiffness of the candidate material. For example, the portion of the structure made

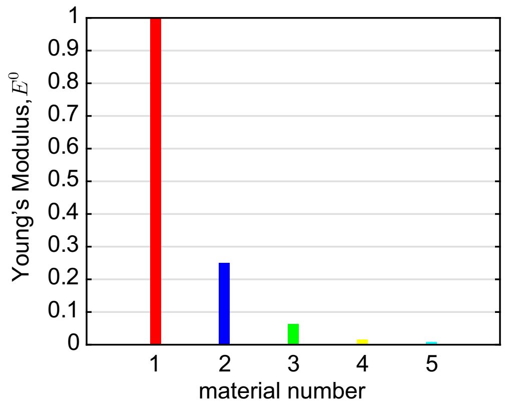
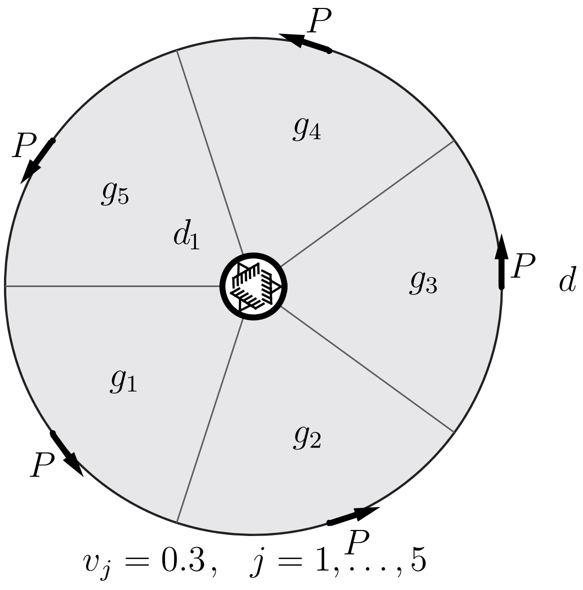
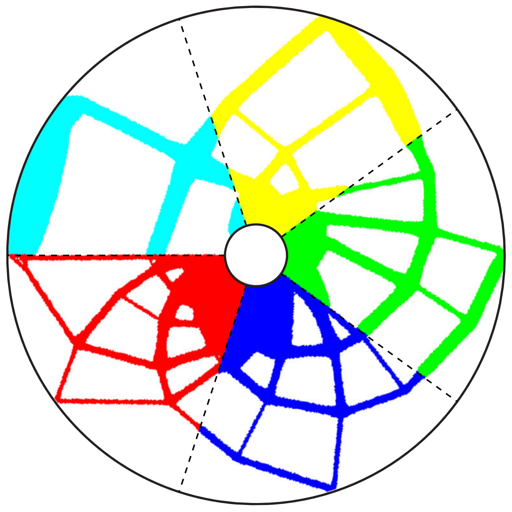


Fig. 18 Young's modulus (and color used in the result visualization) of the five candidate materials available in the flower problem (color online)

(a)



(b)



(c)

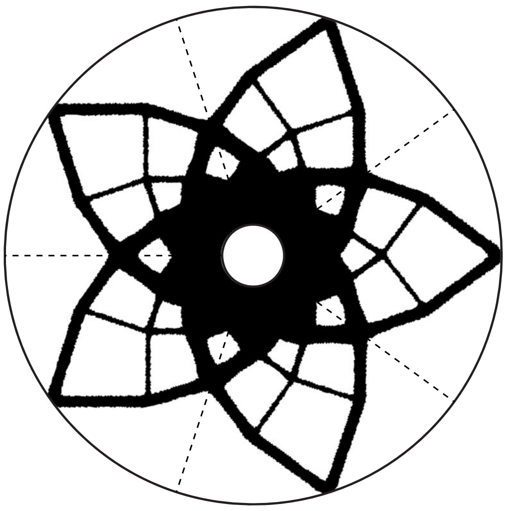


Fig. 19 Flower problem: **a** domain, boundary conditions, and constraint specification for the multi-material design; **b** multi-material design; **c** corresponding single-material design (color online)

Table 7 Input parameters used for the flower problem (brackets indicate continuation and $R = -1$ indicates no filter)

Number of elements	100,000
SIMP penalty parameter, p	[1, 1.5, 2, 3, 4]
Material interpolation factor, γ	[0, 0.3, 0.5, 1, 1]
Filter radius, R	[0.01, 0.01, 0.01, 0.01, −1]
Max. number of iterations (per continuation step)	200

of the most compliant material, shown in cyan, has the least complex geometry, and complexity gradually increases clockwise around the domain as the materials become stiffer. Additionally, the members tend to become thinner as the materials increase in stiffness. Finally, if we identify the "flower pedals" of the multi-material design as each "Michell-like" structure touching each load point, the pedals tend to become smaller moving clockwise around the domain from the most compliant material (cyan) to the stiffest material (red).

8 Conclusions

We have presented a multi-material framework for topology optimization on unstructured polygonal meshes along with an educational code written in Matlab called `PolyMat`. `PolyMat` is built on top of the single-material version, `PolyTop` (Talischi et al. 2012b), and adopts the modular structure defined there in which the optimization and analysis routines are separate. A key difference between `PolyMat` and `PolyTop` is that `PolyMat` can accommodate *many* materials and *many* volume constraints, which may be specified to control any subset of the design variables, i.e., the volume constraints may be global or local and may control a single candidate material or multiple candidate materials. Using similar ideas as those used in `PolyMesher` (Talischi et al. 2012a) for generating complex domains, we can systematically define sub-regions of arbitrary geometries within the design domain. These sub-regions serve as boundaries of the local volume constraints and/or passive regions, i.e., regions in which the design variables do not participate in the optimization. To accommodate such general volume constraints and passive regions, we adopt the ZPR design variable update scheme, which leverages separability of the dual subproblem to update the design for each constraint, independently. As such, the ZPR update only needs to know which design variables are associated with each constraint, that is, the element and material indices associated with each constraint. The design variables within a passive region are simply not assigned to any constraint, and thus, are not updated after initialization.

We provide results for examples with up to 15 candidate materials and up to 84 constraints, but the formulation itself does not limit the number of candidate materials or the number of volume constraints. Two techniques are introduced to prevent material mixing. A DMO interpolation scheme (Stegmann and Lund 2005) is effective at achieving designs with discrete material regions; however, it cannot remove the small regions of mixing that arise at the interfaces between materials due to the density filter. Thus, we also perform continuation on the filter radius such that filter radius is smaller than the mesh size in the final iterations, leading to final designs that contain no mixing. Additionally, a new parameter introduced to the DMO interpolation allows us to perform continuation on both the material interpolation and penalization such that we start with a convex problem and transition to a non-convex problem in which both material mixing and intermediate densities are penalized.

The ability to consider many materials, specify many global or local volume constraints, easily define passive regions, and ensure designs are (close to) free of material mixing, leads to a very general and flexible setting for multi-material topology optimization in which the designer has increased freedom to control both the material and geometric characteristics of the optimized design.

Acknowledgments GHP and EDS acknowledge the financial support from the US National Science Foundation (NSF) under project #1663244, and the endowment provided by the Raymond Allen Jones Chair at the Georgia Institute of Technology. AP appreciates the financial support from the Carlos Chagas Filho Research Foundation of Rio de Janeiro State (FAPERJ) under grant E-26/203.189/2016, and from Tecgraf/PUC-Rio (Group of Technology in Computer Graphics), Rio de Janeiro, Brazil. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. The information provided in this paper is the sole opinion of the authors and does not necessarily reflect the views of the sponsoring agencies.

Appendix A: Convergence plots for selected problems

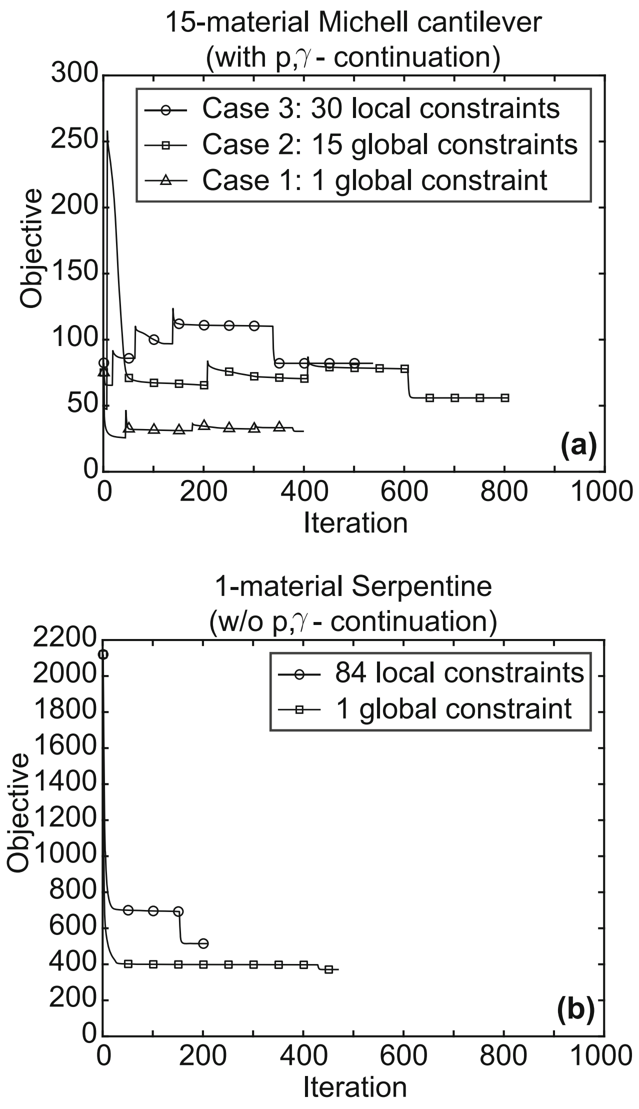


Fig. 20 Convergence plots: **a** 15-material Michell cantilever with continuation on the material interpolation parameters and with filter reduction in the final continuation step and **b** 1-material serpentine without continuation on the material interpolation parameters and with filter reduction after convergence

Appendix B: Summary of filtering techniques

The three filtering techniques discussed in Section 5.1 are summarized in Table 8.

Table 8 Summary of filtering techniques

	Density filter	Density + Sensitivity filter	Density + ZPR filter
Design variables	$\mathbf{Z} = \{z_{\ell 1}, \ldots, z_{\ell m}\}_{\ell=1}^{N}$	$\mathbf{Z} = \{z_{\ell 1}, \ldots, z_{\ell m}\}_{\ell=1}^{N}$	$\mathbf{Z} = \{z_{\ell 1}, \ldots, z_{\ell m}\}_{\ell=1}^{N}$
Element densities	$\mathbf{Y} = \{y_{\ell 1}, \ldots, y_{\ell m}\}_{\ell=1}^{N}$	$\mathbf{Y} = \{y_{\ell 1}, \ldots, y_{\ell m}\}_{\ell=1}^{N}$	$\mathbf{Y} = \{y_{\ell 1}, \ldots, y_{\ell m}\}_{\ell=1}^{N}$
	$\mathbf{y}_i = \mathbf{P}\mathbf{z}_i, \quad i = 1, \ldots, m$	$\mathbf{y}_i = \mathbf{P}\mathbf{z}_i, \quad i = 1, \ldots, m$	$\mathbf{y}_i = \mathbf{P}\mathbf{z}_i, \quad i = 1, \ldots, m$
Element stiffnesses	$\mathbf{E} = m_E\left(\mathbf{y}_1, \ldots, \mathbf{y}_m\right)$	$\mathbf{E} = m_E\left(\mathbf{y}_1, \ldots, \mathbf{y}_m\right)$	$\mathbf{E} = m_E\left(\mathbf{y}_1, \ldots, \mathbf{y}_m\right)$
Sensitivities of objective	$\frac{\partial f}{\partial \mathbf{z}_i}$	$\frac{\widetilde{\partial f}}{\partial \mathbf{z}_i} = \tilde{\mathbf{P}}_i \frac{\partial f}{\partial \mathbf{z}_i}$	$\frac{\partial f}{\partial \mathbf{z}_i}$
		$\tilde{\mathbf{P}}_i = \text{diag}\left(\frac{1}{\mathbf{z}_i}\right) \mathbf{P}\text{diag}\left(\mathbf{z}_i\right)$	
Fixed-point iteration	$z_{\ell i}^{*} = \underline{\rho} + \left(B_{\ell i}\right)^{\eta} \left(z_{\ell i}^{0} - \underline{\rho}\right)$	$z_{\ell i}^{*} = \underline{\rho} + \left(\tilde{B}_{\ell i}\right)^{\eta} \left(z_{\ell i}^{0} - \underline{\rho}\right)$	$z_{\ell i}^{*} = \underline{\rho} + \left(B_{\ell i}\right)^{\eta} \left(\sum_{k=1}^{N} P_{\ell k} z_{ki}^{0} - \underline{\rho}\right)$
	$B_{\ell i} = -\frac{\frac{\widetilde{\partial f}}{\partial z_{\ell i}} \vert_{\mathbf{Z}=\mathbf{Z}_0}}{\lambda_j \frac{\partial g_j}{\partial z_{\ell i}} \vert_{\mathbf{Z}=\mathbf{Z}_0}}$	$\tilde{B}_{\ell i} = -\frac{\frac{\partial f}{\partial z_{\ell i}} \vert_{\mathbf{Z}=\mathbf{Z}_0}}{\lambda_j \frac{\partial g_j}{\partial z_{\ell i}} \vert_{\mathbf{Z}=\mathbf{Z}_0}}$	$B_{\ell i} = -\frac{\frac{\partial f}{\partial z_{\ell i}} \vert_{\mathbf{Z}=\mathbf{Z}_0}}{\lambda_j \frac{\partial g_j}{\partial z_{\ell i}} \vert_{\mathbf{Z}=\mathbf{Z}_0}}$
Reference	Bendsøe and Sigmund (2003)	This paper	Sanders et al. (2018)

Appendix C: Library of example problems

The Matlab files needed to run each of the problems discussed herein are provided with download of `PolyMat`. Table 9 summarizes the problems and indicates which files need to be called from `PolyScript` to run each.

Table 9 Examples provided with `PolyMat`

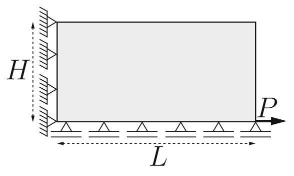
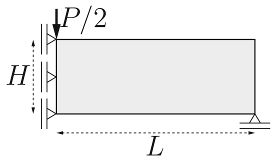
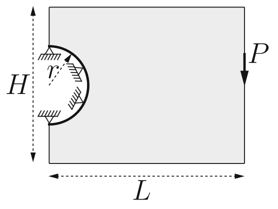
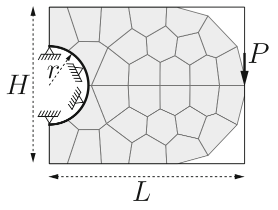
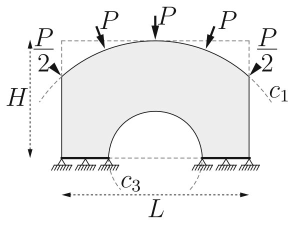
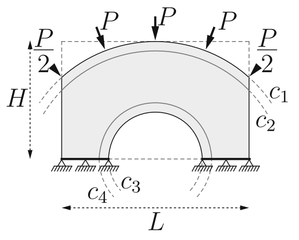
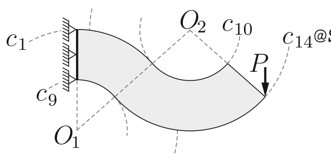
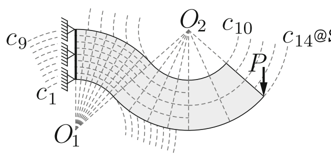
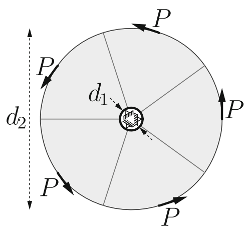
Domain	`PolyMesher` domain file	`PolyMat` constraint file	Description
	`@ShortColumnDomain`	`ShortColumnConstraints`	• Defaults: $H = 0.5$, $L = 1$, $P = 1$ • 2 global volume constraints, each controlling 1 of 2 candidate materials
	`@MbbDomain`	`MbbConstraints`	• Defaults: $H = 1$, $L = 3$, $P = 1$ • 1 global volume constraint controlling 1 candidate material
	`@MichellDomain`	Case 1: `Michell1Constraints` Case 2: `Michell2Constraints`	• Defaults: $H = 4$, $L = 5$, $r = 1$, $P = 1$ • Case 1: 1 global volume constraint controlling 15 candidate materials • Case 2: 15 global volume constraints, each controlling 1 of 15 candidate materials
	`@MichellDomain`	Case 3: `Michell3Constraints`	• Defaults: $H = 4$, $L = 5$, $r = 1$, $P = 1$ • Case 3: 30 local volume constraints, each controlling a sub-region of the domain and 1 of 15 candidate materials
	`@CurvedBeamDomain`	`CurvedBeamConstraints`	• Defaults: $H = 1.25$, $L = 2$, $c_1 = (x_{c1}, y_{c1}, r_{c1}) = (0, 0, 1.5)$, $c_3 = (0, 0.25, 0.5)$, $P = 1$ • 5 volume constraints, each controlling 1 of 5 candidate materials • 5 volume constraints, each controlling 1 of 5 candidate materials

Table 9 (continued)

Domain	`PolyMesher` domain file	`PolyMat` constraint file	Description
 P $\frac{P}{2}$ H c_1 c_2 c_3 c_4 L	`@CurvedBeamDomain`	`CurvedBeamPassiveConstraints`	• Defaults: $H = 1.25$, $L = 2$, $c_1 = (x_{c1}, y_{c1}, r_{c1}) = (0, 0, 1.5)$, $c_2 = (0, 0, 1.4)$, $c_3 = (0, 0.25, 0.5)$, $c_4 = (0, 0.25, 0.6)$, $P = 1$ • 2 passive regions • 3 volume constraints, each controlling 1 of 3 candidate materials in the optimizable region
 c_1 c_9 O_1 O_2 c_{10} c_{14} P	`@SerpentineDomain`	`Serpentine1Constraints`	• Defaults: $O_1 = (0, -2.6458)$, $O_2 = (9, 5.2915)$, $r_{c_1} = 8$, $r_{c_9} = 4$, $r_{c_{10}} = 4$, $r_{c_{14}} = 8$, $P = 1$ • 1 global volume constraint controlling 1 candidate material
 c_9 c_1 O_1 O_2 c_{10} c_{14} P	`@SerpentineDomain`	`Serpentine84Constraints`	• Defaults: $O_1 = (0, -2.6458)$, $O_2 = (9, 5.2915)$, $r_{c_1} = 8$, $r_{c_9} = 4$, $r_{c_{10}} = 4$, $r_{c_{14}} = 8$, $P = 1$ • 84 local volume constraints, each controlling 1 of 1 candidate material in each of 84 sub-regions
	`@FlowerDomain`	Case 1: `Flower1MatConstraints` Case 2: `Flower5MatConstraints`	• Defaults: $d_1 = 0.25$, $d_2 = 2$, $P = 1$ • Case 1: 5 local volume constraints, each controlling 1 of 5 candidate materials in each of 5 sub-regions • Case 2: 5 local volume constraints, each controlling the same candidate material in 5 sub-regions

Appendix D: PolyScript

```
%------------------------------- PolyScript -------------------------------%
% Ref: ED Sanders, A Pereira, MA Aguilo, GH Paulino, "PolyMat: an          %
%      efficient Matlab code for multi-material topology optimization,"    %
%      Struct Multidisc Optim, DOI 10.1007/s00158-018-2094-0, 2018         %
%--------------------------------------------------------------------------%

%% ---------------------------------------------------- CREATE 'fem' STRUCT
[Node,Element,Supp,Load,Seeds] = PolyMesher(@MichellDomain,5000,30);
[VolFrac,ElemInd,MatInd,SElemInd,SMatInd,Mat,Color] = Michell2Constraints(Seeds);
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,...                    % Young's modulus of solid material
  'Mat',Mat,...                 % Material properties
  'NMat',size(Mat,1),...        % Number of candidate materials
  'Reg',0 ...                   % Tag for regular meshes
  );
%% ---------------------------------------------------- CREATE 'opt' STRUCT
R = 0.08;
P = PolyFilter(fem,R);
zIni = zeros(fem.NElem,fem.NMat);
zIni = InitialGuess(VolFrac,ElemInd,MatInd,SElemInd,SMatInd,zIni);
opt = struct(...
  'zMin',0.0,...                % Lower bound for design variables
  'zMax',1.0,...                % Upper bound for design variables
  'zIni',zIni,...               % Initial design variables
  'P',P,...                     % Matrix that maps design to element vars.
  'VolFrac',VolFrac,...         % Specified volume fraction constraint
  'NConstr',size(VolFrac,1),...% Number of volume constraints
  'ElemInd',{ElemInd},...       % Element indices assoc. with each constr.
  'MatInd',{MatInd},...         % Material indices assoc. with each constr.
  'Tol',0.01,...                % Convergence tolerance on design vars.
  'MaxIter',100,...             % Max. number of optimization iterations
  'ZPRMove',0.2,...             % Allowable move step in ZPR update scheme
  'ZPREta',1/2 ...              % Exponent used in ZPR update scheme
  );
%% ---------------------------------------------------------- RUN 'PolyMat'
figure;
Param = [1 1.5 2 3 4;
         0 0.3 0.5 1 1]; %continuation on material interpolation parameters
for i = 1:size(Param,2)
  penal = Param(1,i); gamma = Param(2,i);
  if i == size(Param,2); opt.P = PolyFilter(fem,-1); end
  disp(['current p: ', num2str(penal), ...
        ', current gamma: ', num2str(gamma)]);
  opt.MatIntFnc = @(y)MatIntFnc(y,'SIMP',penal);
  opt.MultiMatIntFnc = @(y)MultiMatIntFnc(y,opt.MatIntFnc,fem.Mat,gamma);
  [opt.zIni,V,fem] = PolyMat(fem,opt,Color);
end
%% ------------------------------------------------------------------------
```

Appendix E: PolyMat

```
%-------------------------------- PolyMat -------------------------------%
% Ref: ED Sanders, A Pereira, MA Aguilo, GH Paulino, "PolyMat: an         %
%      efficient Matlab code for multi-material topology optimization,"   %
%      Struct Multidisc Optim, DOI 10.1007/s00158-018-2094-0, 2018        %
%-------------------------------------------------------------------------%
function [z,V,fem] = PolyMat(fem,opt,Color)
Iter=0; Tol=opt.Tol*(opt.zMax-opt.zMin); Change=2*Tol*ones(1,opt.NConstr); z=opt.
    zIni; P=opt.P;
[E,dEdy,V,dVdy] = opt.MultiMatIntFnc(P*z);
[FigHandle,FigData,Color,RBM] = InitialPlot(fem,V,Color);
while (Iter<opt.MaxIter) && (max(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,opt,E,V);
  %Compute design sensitivities
  dfdz = P'*(dEdy.*repmat(dfdE,1,fem.NMat) + dVdy.*dfdV);
  for c=1:opt.NConstr
    dgdz(:,c,:) = P'*(dEdy.*repmat(dgdE(:,c),1,fem.NMat) + dVdy.*reshape(dgdV(:,c
        ,:),[fem.NElem,fem.NMat]));
    %Update design variable and analysis parameters
    ElemIndices = cell2mat(opt.ElemInd(c));
    MatIndices = cell2mat(opt.MatInd(c));
    [z(ElemIndices,MatIndices),Change(c)] = UpdateScheme(...
        dfdz(ElemIndices,MatIndices),g(c),...
        dgdz(ElemIndices,c,MatIndices),z(ElemIndices,MatIndices),V(ElemIndices,
            MatIndices),opt);
  end
  [E,dEdy,V,dVdy] = opt.MultiMatIntFnc(P*z);
  %Output results
  fprintf('It: %i \t Objective: %1.3f \t Max. Constraint: %1.3f \t Change: %1.3f\n
      ',Iter,f,max(g),max(Change));
  rgb=[zeros(fem.NElem,3),ones(size(V,1),1)];
  for i=1:fem.NMat, rgb(:,1:3) = rgb(:,1:3) + V(:,i)*Color(i,1:3); end
  rgbT = RBM*rgb'; rgb = rgbT';
  I = reshape(rgb(:,1:3),fem.NElem,1,3);
  set(FigHandle,'FaceColor','flat','CData',I(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));
%----------------------------------------------------- CONSTRAINTS FUNCTION
function [g,dgdE,dgdV,fem] = ConstraintFnc(fem,opt,E,V)
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 = zeros(opt.NConstr,1);
```

```
dgdV = zeros(fem.NElem,opt.NConstr,fem.NMat);
for c=1:opt.NConstr
  ElemIndices = cell2mat(opt.ElemInd(c));
  MatIndices = cell2mat(opt.MatInd(c));
  for m = 1:size(MatIndices,2)
    g(c) = g(c) + sum(fem.ElemArea(ElemIndices).*V(ElemIndices,MatIndices(m)))./
        sum(fem.ElemArea(ElemIndices));
    dgdV(ElemIndices,c,MatIndices(m)) = fem.ElemArea(ElemIndices)/sum(fem.ElemArea
        (ElemIndices));
  end
  g(c) = g(c) - opt.VolFrac(c,1);
end
dgdE = zeros(size(E,1),opt.NConstr);
%-------------------------------------------------------------- ZPR UPDATE
function [zNew,Change] = UpdateScheme(dfdz,g,dgdz,z0,V0,opt)
nelem = size(dfdz,1); nmat = size(dfdz,2);
dfdz = reshape(dfdz,nelem*nmat,1); dgdz = reshape(dgdz,nelem*nmat,1);
z0 = reshape(z0,nelem*nmat,1); V0 = reshape(V0,nelem*nmat,1);
zMin=opt.zMin; zMax=opt.zMax;
move=opt.ZPRMove*(zMax-zMin); eta=opt.ZPREta;
l1=0; l2=1e6;
while l2-l1 > 1e-4
  lmid = 0.5*(l1+l2);
  B = -(dfdz./dgdz)/lmid;
  zCnd = zMin+(V0-zMin).*max(0,B).^eta; %MultiMatIntFnc may cause non-negative
      sensitivities
  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);
zNew = reshape(zNew,nelem,nmat,1);
%------------------------------------------------------------- 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
  for el = 1:fem.NElem
    if ~fem.Reg || el==1,  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)*[1 fem.Nu0 0;fem.Nu0 1 0;0 0 (1-fem.Nu0)/2];
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,Color,RBM] = InitialPlot(fem,z0,Color)
[Color,RBM] = ConvertColors(Color);
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
I=[zeros(fem.NElem,3),ones(size(z0,1),1)];
for i=1:fem.NMat
    I(:,1:3) = I(:,1:3) + z0(:,i)*Color(i,1:3);
end
IT = RBM*I';
I = IT';
handle = patch('Faces',Tri,'Vertices',fem.Node,'FaceVertexCData',...
               I(map,1:3),'FaceColor','flat','EdgeColor','none');
axis equal; axis off; axis tight; caxis([0 1]);
%--------------------------------------------------------------------------%
```

Appendix F: `InitialGuess`

```
%-------------------------------- PolyMat --------------------------------%
% Ref: ED Sanders, A Pereira, MA Aguilo, GH Paulino, "PolyMat: an         %
%      efficient Matlab code for multi-material topology optimization,"   %
%      Struct Multidisc Optim, DOI 10.1007/s00158-018-2094-0, 2018        %
%-------------------------------------------------------------------------%
function [zIni] = InitialGuess(VolFrac,ElemInd,MatInd,SElemInd,SMatInd,zIni)
NConstr = size(VolFrac,1);
for i = 1:NConstr
    MatIndices = cell2mat(MatInd(i));
    ElemIndices = cell2mat(ElemInd(i));
    for m = 1:size(MatIndices,2)
        zIni(ElemIndices,MatIndices(m)) = VolFrac(i)./size(MatIndices,2);
    end
end
NFixed = size(SElemInd,1);
for i = 1:NFixed
    MatIndices = cell2mat(SMatInd(i));
    ElemIndices = cell2mat(SElemInd(i));
    for m = 1:size(MatIndices,2)
        zIni(ElemIndices,MatIndices(m)) = 1;
    end
end
%-------------------------------------------------------------------------%
```

Appendix G: MultiMatIntFnc

```
%-------------------------------- PolyMat --------------------------------%
% Ref: ED Sanders, A Pereira, MA Aguilo, GH Paulino, "PolyMat: an         %
%      efficient Matlab code for multi-material topology optimization,"   %
%      Struct Multidisc Optim, DOI 10.1007/s00158-018-2094-0, 2018        %
%-------------------------------------------------------------------------%
function [E,dEdy,V,dVdy,w] = MultiMatIntFnc(y,MatIntFnc,E0,param)
eps = 1e-4;  %Ersatz stiffness
gamma = param;
[w,dwdy,V,dVdy] = MatIntFnc(y);
NElem = size(y,1); NMat = size(y,2);
dEdw = zeros(NElem,NMat);
% Compute E
S = 1-(gamma.*w);
Prod = ones(NElem,NMat);
for m = 1:NMat-1
  S = [S(:,NMat),S(:,1:NMat-1)];
  Prod = Prod.*S;
end
E = eps + (1-eps).*((w.*Prod)*E0);
% Compute dEdy
for m = 1:NMat
  S2 = 1-(gamma.*w);
  S2(:,m) = ones(NElem,1);
  dProd = ones(NElem,NMat);
  for j = 1:NMat-1
    S2 = [S2(:,NMat),S2(:,1:NMat-1)];
    dProd = dProd.*S2;
  end
  w_tmp = -gamma.*w;
  w_tmp(:,m) = ones(NElem,1);
  dEdw(:,m) = (w_tmp.*dProd)*E0;
end
dEdw = dEdw - eps.*dEdw;
dEdy = dwdy.*dEdw;
%-------------------------------------------------------------------------%
```

Appendix H: `ConvertColors`

The `ConvertColors` function is used to enable a result visualization in which white represents void. To plot the multi-material density in a given element, we sum the RGB `Colors` associated with each material in the element, scaled by the corresponding material densities, to find a point on the unit RGB color cube representing the mixing in that element. In general, the unit RGB color cube is defined such that black represents void (0, 0, 0) and white represents full mixing (1, 1, 1). Thus, `ConvertColors` is called before the optimization loop in `PolyMat` to compute a rigid body motion matrix, `RBM`, that can be applied to the RGB color cube so that white is at the origin. Using `RBM`, the RGB `Colors` specified by the user are converted to the corresponding colors on the rotated unit RGB color cube (e.g., red (1, 0, 0) goes to cyan (0, 1, 1), green (0, 1, 0) goes to yellow (1, 1, 0), and blue (0, 0, 1) goes to magenta (1, 0, 1)). These converted colors are scaled by the density of each material and summed in line 30 of `PolyMat` to find a point on the rotated unit RGB color cube representing the mixing in a given element, where void elements have RGB value (0, 0, 0). When rotated back to the original unit RGB color cube, we re-gain the specified color scheme and the void elements are now at (1, 1, 1), which represents white. The code used to compute the `RBM` matrix and convert the user-specified colors is provided in this Appendix.

```matlab
%-------------------------------- PolyMat --------------------------------%
% Ref: ED Sanders, A Pereira, MA Aguilo, GH Paulino, "PolyMat: an         %
%      efficient Matlab code for multi-material topology optimization,"   %
%      Struct Multidisc Optim, DOI 10.1007/s00158-018-2094-0, 2018         %
%-------------------------------------------------------------------------%
function [Color,RBM] = ConvertColors(Color)
%This function applies a rigid body motion to the RGB cube such that white
%  is at [0 0 0] and black is at [1 1 1]. That way, we plot the
%  multi-material results on a white background.
%Translation Matrix
T1 = eye(4,4); T1(1:3,4) = [-0.5;-0.5;-0.5];
%Rotation Matrix
thz = pi; thx = pi/2;
Rz = [cos(thz) -sin(thz) 0 0;sin(thz) cos(thz) 0 0;0 0 1 0;0 0 0 1];
Rx = [1 0 0 0;0 cos(thx) -sin(thx) 0;0 sin(thx) cos(thx) 0;0 0 0 1];
%Translation Matrix
T2 = eye(4,4); T2(1:3,4) = [0.5;0.5;0.5];
%Full rigid body motion matrix
RBM = T2*Rx*Rz*T1;
%Converted colors
Color = [Color, ones(size(Color,1),1)];
ColorT = RBM*Color';
Color = ColorT';
%-------------------------------------------------------------------------%
```

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

References

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

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

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

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

Chau KN, Chau KN, Ngo T, Hackl K, Nguyen-Xuan H (2017) A polytree-based adaptive polygonal finite element method for multi-material topology optimization. Comput Methods Appl Mech Eng 332:712–739

Doan QH, Lee D (2017) Optimum topology design of multi-material structures with non-spurious buckling constraints. Adv Eng Softw 114:110–120

Lieu QX, Lee J (2017) A multi-resolution approach for multi-material topology optimization based on isogeometric analysis. Comput Methods Appl Mech Eng 323:272–302

Michell AG (1904) The limits of economy of material in frame structures. Philos Mag 8(6):589–597

Park J, Sutradhar A (2015) A multi-resolution method for 3D multi-material topology optimization. Comput Methods Appl Mech Eng 285:571–586

Pereira A, Talischi C, Paulino GH, Menezes IFM, Carvalho MS (2016) Fluid flow topology optimization in PolyTop: stability and computational implementation. Struct Multidiscip Optim 54(5):1345–1364

Sanders ED, Aguiló MA, Paulino GH (2018) Multi-material continuum topology optimization with arbitrary volume and mass constraints. Comput Methods Appl Mech Eng 340:798–823

Sigmund O (1994) Design of material structures using topology optimization. PhD thesis, Department of Solid Mechanics Technical University of Denmark

Sigmund O (1997) On the design of compliant mechanisms using topology optimization. J Struct Mech 25(4):493–524

Stegmann J, Lund E (2005) Discrete material optimization of general composite shell structures. Int J Numer Methods Eng 62(14):2009–2027

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

Talischi C, Paulino GH, Pereira A, Menezes IFM (2012a) PolyMesher: a general-purpose mesh generator for polygonal elements written in Matlab. Struct Multidiscip Optim 45(3):309–328

Talischi C, Paulino GH, Pereira A, Menezes IFM (2012b) PolyTop: a Matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes. Struct Multidiscip Optim 45(3):329–357

Tavakoli R, Mohseni SM (2014) Alternating active-phase algorithm for multimaterial topology optimization problems: a 115-line matlab implementation. Struct Multidiscip Optim 49(4):621–642

Zegard T, Paulino GH (2014) GRAND – ground structure based topology optimization for arbitrary 2D domains using MATLAB. Struct Multidiscip Optim 50(5):861–882

Zhang XS, Paulino GH, Ramos AS Jr (2018) Multi-material topology optimization with multiple volume constraints: a general approach applied to ground structures with material nonlinearity. Struct Multidiscip Optim 57:161–182

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