Structural and Multidisciplinary Optimization
https://doi.org/10.1007/s00158-020-02629-w

EDUCATIONAL PAPER

A new generation 99 line Matlab code for compliance topology optimization and its extension to 3D

Federico Ferrari[1] · **Ole Sigmund**[1]

Received: 18 February 2020 / Revised: 2 May 2020 / Accepted: 10 May 2020

Abstract
Compact and efficient Matlab implementations of compliance topology optimization (TO) for 2D and 3D continua are given, consisting of 99 and 125 lines respectively. On discretizations ranging from $3 \cdot 10^4$ to $4.8 \cdot 10^5$ elements, the 2D version, named `top99neo`, shows speedups from 2.55 to 5.5 times compared to the well-known `top88` code of Andreassen et al. (Struct Multidiscip Optim 43(1):1–16, 2011). The 3D version, named `top3D125`, is the most compact and efficient Matlab implementation for 3D TO to date, showing a speedup of 1.9 times compared to the code of Amir et al. (Struct Multidiscip Optim 49(5):815–829, 2014), on a discretization with $2.2 \cdot 10^5$ elements. For both codes, improvements are due to much more efficient procedures for the assembly and implementation of filters and shortcuts in the design update step. The use of an acceleration strategy, yielding major cuts in the overall computational time, is also discussed, stressing its easy integration within the basic codes.

Keywords Topology optimization · Matlab · Computational efficiency · Acceleration methods

1 Introduction

The celebrated `top99` Matlab code developed by Sigmund (2001) has certainly promoted the spreading of topology optimization among engineers and researchers, and the speedups carried by its heir, `top88` (Andreassen et al. 2011), substantially increased the scale of examples that can be solved on a laptop.

On these footprints, several other codes have followed, involving extension to 3D problems (Liu and Tovar 2014; Amir et al. 2014), material design (Andreassen and Andreasen 2014; Xia and Breitkopf 2015), level–set parametrizations (Wang 2007; Challis 2010), use of advanced discretization techniques (Talischi et al. 2012; Suresh 2010; Sanders et al. 2018), or integration of TO within some finite element frameworks.

With the evolution of TO and its application to more and more challenging problems, implementations in `top88` may have become outdated. Also, Matlab has improved in the last decade. Hence, we believe it is time to present a new "exemplary" code collecting shortcuts and speedups, allowing to tackle medium-/large-scale TO problems efficiently on a laptop. Preconditioned iterative solvers, applied for example in Amir and Sigmund (2011), Amir et al. (2014), Ferrari et al. (2018) and Ferrari and Sigmund (2020), allow the solution of the state equation with nearly optimal efficiency (Saad 1992). Thus, the computational bottleneck has been shifted on other operations, such as the matrix assembly or the repeated application of filters. Efficiency improvements for these operations were touched upon by Andreassen et al. (2011), however, without giving a quantitative analysis about time and memory savings.

Here, we provide compact Matlab codes for minimum compliance topology optimization of 2D and 3D continua which show a substantial speedup compared to the `top88` code. We include several extensions by default, such as specification of passive domains, a volume-preserving density projection (Guest et al. 2004; Wang et al. 2011) and continuation strategies for the penalization and projection parameters in a very compact, yet sharp, implementation.

Responsible Editor: Palaniappan Ramu

✉ Federico Ferrari
fferrar3@jh.edu

Ole Sigmund
sigmund@mek.dtu.dk

[1] Department of Mechanical Engineering, Technical University of Denmark, Nils Koppels Allé 404, 2800 Kongens Lyngby, Denmark

Published online: 24 August 2020

Coincidentally, the new 2D TO implementation consists of 99 lines of code and is thus named `top99neo`. We also show how to include an acceleration technique recently investigated for TO by Li et al. (2020), with a few extra lines of code and potentially carrying major speedups. Changes needed for the extension to 3D problems are remarkably small, making the corresponding code (`top3D125`) the most compact and efficient Matlab implementation for 3D compliance TO to date.

Our primary goal is not to present innovative new research. Rather, we aim at sharing some shortcuts and speedups that we have noticed through time, to the benefit of the research community. Improvements introduced by the present codes will be much useful also on more advanced problems, such as buckling optimization, which will be dealt with in an upcoming work.

The paper is organized as follows. In Section 2, we recall the setting of TO for minimum compliance. Section 3 is devoted to describe the overall structure of the 2D code, focusing on differences with respect to `top88`. Sections 3.1–3.5 give insights about the main speedups and show performance improvements with respect to `top88`. The very few changes needed for the 3D code are listed in Section 4, where an example is presented and the efficiency is compared to the previous code from Amir et al. (2014). Some final remarks are given in Section 5. Appendix A gives some details about the redesigns step that are useful for better understanding a method proposed in Section 3.2 and the Matlab codes are listed in Appendices B and C.

2 Problem formulation and solution scheme

We consider a 2D/3D discretization Ω_h consisting of m equi-sized quadrilateral elements Ω_e. Hereafter we denote by n the global number of Degrees of Freedom (DOFs) in the discretization and by d the number of (local) DOFs of each element.

Let $\mathbf{x} = \{x_e\}_{e=1:m} \in [0,1]^m$ be partitioned between $\mathbf{x}_\mathcal{A}$ and $\mathbf{x}_\mathcal{P}$, the sets of active (design) variables and passive elements, respectively. The latter may be further split in the sets of passive solid $\mathcal{P}_1$ ($x_e = 1$) and void $\mathcal{P}_0$ ($x_e = 0$) elements, of cardinalities $m_{\mathcal{P}_1}$ and $m_{\mathcal{P}_0}$, respectively (see Fig. 1a).

The set of physical variables $\hat{\mathbf{x}}_\mathcal{A} = \mathcal{H}(\tilde{\mathbf{x}})$ are defined by the relaxed Heaviside projection (Wang et al. 2011)

$$\mathcal{H}(\tilde{x}_e, \eta, \beta) = \frac{\tanh(\beta\eta) + \tanh(\beta(\tilde{x}_e - \eta))}{\tanh(\beta\eta) + \tanh(\beta(1-\eta))} \tag{1}$$

with threshhold η and sharpness factor β, where $\tilde{\mathbf{x}} = H\mathbf{x}$ is the filtered field, obtained by the linear operator

$$H\,(x_e, r_{\min}) := \frac{\sum_{i\in\mathcal{N}_e} h_{e,i} x_i}{\sum_{i\in\mathcal{N}_e} h_{e,i}} \tag{2}$$

where $\mathcal{N}_e = \{i \mid \mathrm{dist}(\Omega_i, \Omega_e) \le r_{\min}\}$ and $h_{e,i} = \max(0, r_{\min} - \mathrm{dist}(\Omega_i, \Omega_e))$.

Given a load vector $\mathbf{f} \in \mathbb{R}^n$ and the volume fraction $f \in (0,1)$, we consider the optimization problem

$$\begin{cases} \min\limits_{\mathbf{x}_\mathcal{A}\in[0,1]^{m_\mathcal{A}}} c\left(\hat{\mathbf{x}}\right) \\ \text{s.t.} \quad V\left(\hat{\mathbf{x}}\right) \le f|\Omega_h| \end{cases} \tag{3}$$

for the minimization of compliance $c\left(\hat{\mathbf{x}}\right) = \mathbf{u}^T\mathbf{f}$ with an upper bound on the overall volume

$$V\left(\hat{\mathbf{x}}\right) = \sum_{e=1}^{m} |\Omega_e| \hat{x}_e = \frac{1}{m}\left(m_{\mathcal{P}_1} + \sum_{e\in\mathcal{A}} \hat{x}_e\right) \le f \tag{4}$$

Problem (3) is solved with a nested iterative loop. At each iteration, the displacement $\mathbf{u}$ is computed by solving the equilibrium problem

$$K\mathbf{u} = \mathbf{f} \tag{5}$$

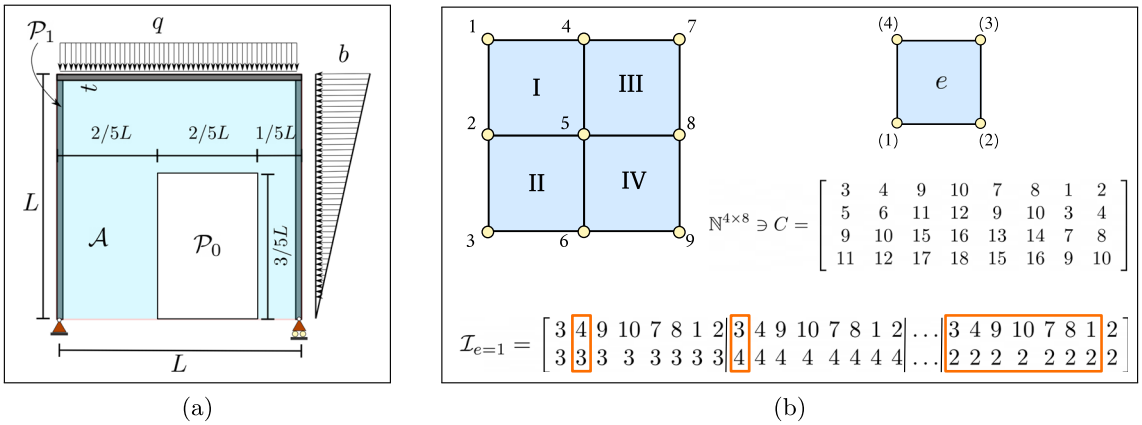


Fig. 1 Definition of the active $\mathcal{A}$, passive solid $\mathcal{P}_1$, and void $\mathcal{P}_0$ domains (**a**) and illustration of the connectivity matrix C for a simple discretization (**b**). The set of indices $\mathcal{I}$, here shown for the element $e = 1$, is used by the assembly operation. The symmetric repetitions in $\mathcal{I}$ are highlighted, and their elimination gives the reduced set $\mathcal{I}_r$ (see Section 3.1)

where the stiffness matrix $K = K(\hat{\mathbf{x}})$ depends on the physical variables through a SIMP interpolation (Bendsøe and Sigmund 1999) of the Young modulus

$$E(\hat{x}_e) = E_{\min} + \hat{x}_e^p (E_0 - E_{\min}) \tag{6}$$

with E_0 and $E_{\min}$ the moduli of solid and void ($E_{\min} \ll E_0$). The gradients of compliance and structural volume with respect to $\hat{\mathbf{x}}$ read ($\chi_e = 1$ if $e \in \mathcal{A}$ and 0 otherwise and $\mathbf{1}_m$ is the identity vector of dimension m)

$$\nabla_{\hat{\mathbf{x}}} c\left(\hat{\mathbf{x}}\right) = -\mathbf{u}^T \nabla_{\hat{\mathbf{x}}} K \mathbf{u} \chi_{\mathcal{A}}\,, \quad \nabla_{\hat{\mathbf{x}}} V\left(\hat{\mathbf{x}}\right) = \frac{1}{m} \mathbf{1}_m \chi_{\mathcal{A}} \tag{7}$$

and the sensitivities with respect to the design variables are recovered as

$$\begin{aligned} \nabla_{\mathbf{x}} c\left(\mathbf{x}\right) &= \nabla_{\tilde{\mathbf{x}}} \mathcal{H} \odot \left(H^T \nabla_{\hat{\mathbf{x}}} c\left(\hat{\mathbf{x}}\right)\right) \\ \nabla_{\mathbf{x}} V\left(\mathbf{x}\right) &= \nabla_{\tilde{\mathbf{x}}} \mathcal{H} \odot \left(H^T \nabla_{\hat{\mathbf{x}}} V\left(\hat{\mathbf{x}}\right)\right) \end{aligned} \tag{8}$$

where $\odot$ represents the element-wise multiplication and

$$\nabla_{\tilde{\mathbf{x}}} \mathcal{H} = \beta \frac{1 - \tanh(\beta(\tilde{\mathbf{x}} - \eta))^2}{\tanh(\beta\eta) + \tanh(\beta(1 - \eta))} \tag{9}$$

The active design variables $e \in \mathcal{A}$ are then updated by the optimality criterion rule (Sigmund 2001)

$$x_{k+1,e} = \mathcal{U}(x_{k,e}) = \begin{cases} \delta_- & \text{if } \mathcal{F}_{k,e} < \delta_- \\ \delta_+ & \text{if } \mathcal{F}_{k,e} > \delta_+ \\ \mathcal{F}_{k,e} & \text{otherwise} \end{cases} \tag{10}$$

where $\delta_- = \max(0, x_{k,e} - \mu)$, $\delta_+ = \min(1, x_{k,e} + \mu)$, for the fixed move limit $\mu \in (0, 1)$ and

$$\mathcal{F}_{k,e} = x_{k,e} \left(-\frac{\partial_e c_k}{\tilde{\lambda}_k \partial_e V_k} \right)^{1/2} \tag{11}$$

depends on the element sensitivities.

In (11), $\tilde{\lambda}_k$ is the approximation to the current Lagrange multiplier λ_k^* associated with the volume constraint. This is obtained by imposing $V(\hat{\mathbf{x}}_{k+1}(\tilde{\lambda})) - f|\Omega_h| \approx 0$, e.g., by bisection on an interval $\Lambda_k^{(0)} \supset \lambda_k^*$.

3 Matlab implementation and speedups

The Matlab routine for 2D problems (see Appendix B) is called with the following arguments

```
top99neo(nelx,nely,volfrac,penal,rmin,ft,ftBC,eta,beta,
    move,maxit);
```

where `nelx` and `nely` define the physical dimensions and the mesh resolution, `volfrac` is the allowed volume fraction on the overall domain (i.e., $\mathcal{A} \cup \mathcal{P}$), `penal` the penalization used in (6), and `rmin` the filter radius for (2). The parameter `ft` is used to select the filtering scheme: density filtering alone if `ft=1`, whereas `ft=2` or `ft=3` also allows the projection (1), with `eta` and `beta` as parameters. `ftBC` specifies the filter boundary conditions (`'N'` for zero- Neumann or `'D'` for zero-Dirichlect), `move` is the move limit used in the OC update and `maxit` sets the maximum number of redesign steps.

The routine is organized in a set of operations which are performed only once and the loop for the TO iterative redesign. The initializing operations are grouped as follows

```
PRE.1) MATERIAL AND CONTINUATION PARAMETERS
PRE.2) DISCRETIZATION FEATURES
PRE.3) LOADS, SUPPORTS AND PASSIVE DOMAINS
PRE.4) DEFINE IMPLICIT FUNCTIONS
PRE.5) PREPARE FILTER
PRE.6) ALLOCATE AND INITIALIZE OTHER PARAMETERS
```

and below we give details only about parameters and instructions not found in the `top88` code.

To apply continuation on the generic parameter "`par`," a data structure is defined

```
parCont = {istart, maxPar, isteps, deltaPar};
```

such that the continuation starts when `loop=istart` and the parameter is increased by `deltaPar` each `isteps`, up to the value `maxPar`. This is implemented in Lines 6 and 7 for the penalization parameter p and the projection factor β, respectively. The update is then performed, by the instruction (see Line 92)

```
par=par+(loop>=parCnt{1}).*(par<parCnt{2}).*mod(
    loop,parCnt{3})==0).*parCnt{4}
```

making use of compact logical operations. Continuation can be switched off, e.g., by setting `maxPar<=par`, or `istart>=maxit`.

The blocks defining the discretization (`PRE.2`) contain some changes compared to `top88`. The number of elements (`nEl`), DOFs (`nDof`), and the set of node numbers (`nodeNrs`) are defined explicitly, to ease and shorten some following instructions. The setup of indices `iK` and `jK`, used for the sparse assembly, is performed in Lines 15–21 and follows the concept detailed in Section 3.1. The coefficients of the lower diagonal part of the elemental stiffness matrix are defined in vectorized form, such that `Ke` $= \mathcal{V}(K_e^{(s)})$ (see Lines 22–26). `Ke` is used for the assembly strategy described in Section 3.1. However, in Lines 27–29, we also recover the complete elemental matrix (`Ke0`), used to perform the double product $\mathbf{u}_e^T K_e \mathbf{u}_e$ when computing the compliance sensitivity (7). Although this could also be written in terms of the matrix $K_e^{(s)}$ only, this option would increase the number of matrix/vector multiplications.

In `PRE.3`), the user can specify the set of restrained (`fixed`) and loaded (`lcDof`) DOFs and passive regions ($\mathcal{P}_1 \leftrightarrow$ `pasS` and $\mathcal{P}_0 \leftrightarrow$ `pasV`) for the given configuration. Supports and loads are defined as in the `top88` code,

whereas passive domains may be specified targeting a set of column and rows from the array `elNrs`. Independently of the particular example, Lines 34–36 define the vector of applied loads, the set of free DOFs, and the sets of active $\mathcal{A} \leftrightarrow$ `act` design variables.

In order to make the code more compact and readable, operations which are repeatedly performed within the TO optimization loop are defined through inline functions in `PRE.4)` (Lines 38–43). The filter operator is built in `PRE.5)` making use of the built-in Matlab function `imfilter`, which represents a much more efficient alternative to the explicit construction of the neighboring array. A similar approach was already outlined by Andreassen et al. (2011), pointing to the Matlab function `conv2`, which is however not completely equivalent to the original operator, as it only allows zero-Dirichlet boundary conditions for the convolution operator. Here, we choose `imfilter`, which is essentially as efficient as `conv2`, but gives the flexibility to specify zero-Dirichlet (default option), or zero-Neumann boundary conditions.

Some final initializations and allocations are performed in `PRE.6)`. The design variables are initialized with the modified volume fraction, accounting for the passive domains (Line 52–53) and the constant volume sensitivity (7) is computed in Line 51.

Within the redesign loop, the following five blocks of operations are repeatedly performed

```
RL.1) COMPUTE PHYSICAL DENSITY FIELD
RL.2) SETUP AND SOLVE EQUILIBRIUM EQUATIONS
RL.3) COMPUTE SENSITIVITIES
RL.4) UPDATE DESIGN VARIABLES AND APPLY CONTINUATION
RL.5) PRINT CURRENT RESULTS AND PLOT DESIGN
```

In block `RL.1)`, the physical field is obtained, applying the density filter and, if selected, also the projection. If `ft=3`, the special value of the threshold `eta` giving a volume-preserving projection is computed, as discussed in Section 3.2.

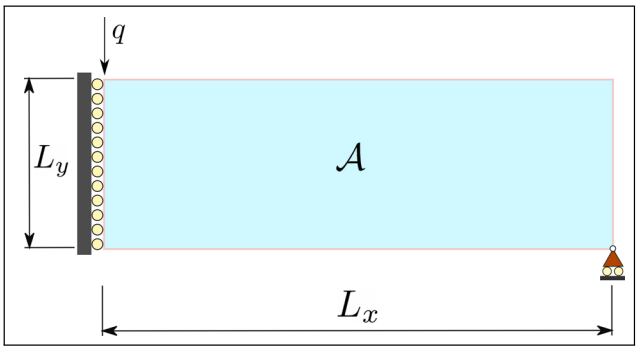


Fig. 2 Geometrical setting for the MBB example

The stiffness interpolation and its derivative (`sK`, `dsK`) are defined, and the stiffness matrix is assembled (see Lines 73–76). Ideally, one could also get rid of Lines 73–74 and directly define `sK` in Line 75 and `dsK` within Line 79. However, we decide to keep these operations apart, enhancing the readability of the code and to ease the specification of different interpolation schemes. Equation (5) is solved on Line 77 using the Matlab function `decomposition`, which can work with only half of the stiffness matrix (see Section 3.1). The sensitivity of compliance is computed, and the backfiltering operations (8) are performed in `RL.3)`.

The update (10), with the nested application of the bisection process for finding $\tilde{\lambda}_k$, is implemented in `RL.4)` (Lines 86–91), and we remark that `lm` represents $\sqrt{\lambda}$.

Some information about the process is printed and the current design is plotted in `RL.5)` (Lines 94–97). On small discretizations, repeated plotting operations absorb a significant fraction of the CPU time (e.g., 15% for $m = 4800$). Therefore, one might just plot the final design, moving Lines 96–97 outside the redesign loop.

The tests in the following have been run on a laptop equipped with an Intel(R) Core(TM) i7-5500U@2.40-GHz CPU, 15 GB of RAM, and Matlab 2018b running in serial mode under Ubuntu 18.04 (but a similar performance is expected in Windows setups). We will often refer to the half MBB beam example (see Fig. 2) for numerical testing. Unless stated otherwise, we choose $\Omega_h = 300 \times 100$, $f = 0.5$, and $r_{\min} = 8.75$ (Sigmund 2007). The load, having total magnitude $|q| = 1$ is applied to the first node. No passive domains are introduced for this example; therefore, `pasS=[];`, `pasV=[];` and we set $E_1 = 1$, $E_0 = 10^{-9}$, and $\nu = 0.3$ in all the tests.

3.1 Speedup of the assembly operation

In `top88`, the assembly of the global stiffness matrix is performed by the built-in Matlab function `sparse`

```
K = sparse( iK, jK, sK );
K = ( K +K' ) / 2;
```

where `sK` $\in \mathbb{R}^{m*d^2 \times 1}$ collects the coefficients of all the elemental matrices in a column-wise vectorized form (i.e., $\mathcal{V}(K_e)$) and `iK` and `jK` are the sets of indices mapping each `sK(i)` to the global location `K(iK(i),jK(i))`.

These two sets are set up through the operations

$$\texttt{iK} = \mathcal{V}\left[(C \otimes \mathbf{1}_d)^T\right], \qquad \texttt{jK} = \mathcal{V}\left[(C \otimes \mathbf{1}_d^T)^T\right] \tag{12}$$

where $C_{[m \times d]}$ is the connectivity matrix and "$\otimes$" is the Kronecker product (Horn and Johnson 2012). The size of the array $\mathcal{I} = [\texttt{iK}, \texttt{jK}] \in \mathbb{N}^{m*d^2 \times 2}$ grows very quickly with the number of elements m, especially for 3D discretizations

Table 1 Number of entries in the array $\mathcal{I}$ and corresponding memory requirement for the 2D and 3D test discretizations. White background refers to the F strategy with coefficients specified as `double`, cyan background to the H strategy, and light green to the H strategy and element specified as `int32`. The H strategy cuts $|\mathcal{I}|$ and memory of $\approx 44\%$ in 2D and $\approx 48\%$ in 3D. Then, specifying the indexes as `int32` further cuts memory of another 50%

2D	m	120^2	240^2	480^2	960^2	1920^2
	$\|\mathcal{I}\|$	1,843,200	7,372,800	29,491,200	117,964,800	–
		1,036,800	4,147,200	16,588,800	66,355,200	265,420,800
	memory (MB)	14.7	59	235	943	–
		8.3	33.2	132.7	530.8	2123
		4.1	16.6	66.3	265.4	1061
3D	m	8^3	16^3	32^3	64^3	128^3
	$\|\mathcal{I}\|$	589,824	4,718,592	37,748,736	301,898,888	2,415,919,104
		307,200	2,457,200	19,660,800	157,286,400	1,258,291,200
	memory (MB)	4.7	37.7	302	2416	9664
		2.5	19.7	157.3	1258	5033
		1.2	9.8	78.6	629.1	2516

(see Table 1), and even though its elements are integers, the `sparse` function requires them to be specified as `double` precision numbers. The corresponding memory burden slows down the assembly process and restricts the size of problems workable on a laptop.

The efficiency of the assembly can be substantially improved by

1. Acknowledging the symmetry of both K_e and K
2. Using an assembly routine working with `iK` and `jK` specified as integers

To understand how to take advantage of the symmetry of matrices, we refer to Fig. 1b and to the connectivity matrix C. Each coefficient $C_{ej} \in \mathbb{N}$ addresses the global DOF targeted by the jth local DOF of element e. Therefore, (12) explicitly reads

$$\begin{aligned} \texttt{iK}^e &= \{\underbrace{\mathbf{c}_e, \mathbf{c}_e, \ldots, \mathbf{c}_e}_{d \text{ times}}\} \\ \texttt{jK}^e &= \{\underbrace{c_{e1}, \ldots, c_{e1}}_{d \text{ times}}, \underbrace{c_{e2}, \ldots, c_{e2}}_{d \text{ times}}, \ldots, \underbrace{c_{ed}, \ldots, c_{ed}}_{d \text{ times}}\} \end{aligned} \tag{13}$$

where $\mathbf{c}_e = \{c_{e1}, c_{e2}, \ldots, c_{ed}\}$ is the row corresponding to element e.

If we only consider the coefficients of the (lower) symmetric part of the elemental matrix $K_e^{(s)}$ and their locations into the global one $K^{(s)}$, the set of indices can be reduced to

$$\begin{aligned} \texttt{iK}^e &= \{c_{e1}, \ldots, c_{ed}, c_{e2}, \ldots, c_{ed}, \ldots, c_{e3}, \ldots, c_{ed}, \ldots, c_{ed}\} \\ \texttt{jK}^e &= \{\underbrace{c_{e1}, \ldots, c_{e1}}_{d \text{ times}}, \underbrace{c_{e2}, \ldots, c_{e2}}_{(d-1) \text{ times}}, \underbrace{c_{e3}, \ldots, c_{e3}}_{(d-2) \text{ times}}, \ldots, c_{ed}\} \end{aligned} \tag{14}$$

and the overall indexing array becomes $\mathcal{I}_r = [\texttt{iK}, \texttt{jK}] \in \mathbb{N}^{\tilde{d}*m \times 2}$ where $\tilde{d} = \sum_{j=1}^{d} \sum_{i \le j} i$. The entries of the indexing array and the memory usage are reduced by approx. 45% (see Table 1).

The set of indices (14) can be constructed by the following instructions (see Lines 15–21)

```
[ setI, setII ] = deal( [ ] );
for j=1:8
   setI=cat(2,setI,j:8);
   setII = cat(2,setII,repmat(j,1,8-j+1));
end
[iK , jK] = deal(cMat(:, sI)', cMat(:, sII)');
Iar = sort([iK(:),jK(:)], 2,'descend'); clear iK jK
```

which can be adapted to any isoparametric 2D/3D element just by changing accordingly the number `d` of elemental DOFs. In the attached scripts, based on 4-noded bilinear $Q4$ and 8-noded trilinear $H8$ elements, we set `d=8` and `d=24`, respectively. The last instruction sorts the indices as `iKr(i) > jKr(i)`, such that $K^{(s)}$ contains only sub-diagonal terms.

The syntax `K=sparse(iK,jK,sK)` now returns the lower triangular matrix $K^{(s)}$ and we remark that the full operator can be recovered by

$$K = K^{(s)} + (K^{(s)})^T - \text{diag}[K^{(s)}] \tag{15}$$

which costs as much as the averaging operation $\frac{1}{2}(K + K^T)$, performed in `top88` to get rid of roundoff errors. However, the Matlab built-in Cholesky solver and the corresponding `decomposition` routine can use just $K^{(s)}$, if called with the option `'lower'`.

Point 2 gives the most dramatic improvement, and can be accomplished by using routines developed by independent researchers. The `sparse2` function, from Suite Sparse (Davis 2019), was already pointed out by Andreassen et al. (2011) as a better alternative to the built-in Matlab `sparse`; however, no quantitative comparisons were performed. According to the CHOLMOD reference manual (Davis 2009), `sparse2` works exactly as `sparse`, but allowing the indices `iK` and `jK` to be specified as integers (accomplished by defining this type for the connectivity matrix, see Lines 11 and 13).

Here we suggest the "`fsparse`" routine, developed by Engblom and Lukarski (2016). Besides working with integers `iK` and `jK`, the function enhances the efficiency of the sparse assembly by a better sorting of the operations. From our experience on a single core process, `fsparse` gives a speedup of 170–250% compared to `sparse2`, and is also highly parallelizable (Engblom and Lukarski 2016). Defining the sets `ik` and `jk` as `int32` type, we can drastically cut the memory requirements, still representing $n \approx 2.1 \cdot 10^9$ numbers, far beyond the size of problems one can tackle in Matlab.

In order to use `fsparse`, one needs to download the "`stenglib`" library[1] and follow the installation instructions in the `README.md` file. The packages of the library can be installed by running the "`makeall.m`" file. As `fsparse` is contained within the folder "Fast," one may only select this folder when running `makeall.m`.

We test the efficiency of the assembly approaches on 2D and 3D uniform discretizations with m^2 and m^3 elements, respectively. Figure 3 shows time scalings for the different strategies: "F" corresponds to the assembly in `top88`, "H" takes advantage of the matrix symmetry only and "H,`fsparse`" correponds to the use of the `fsparse` routine (Engblom and Lukarski 2016) also. All the approaches exhibit a linear scaling of CPU time w.r.t the DOFs number. However, half the CPU time can be cut just by assembling $K^{(s)}$ (strategy H,`sparse`). Therefore, we definitely recommend this to users who aim to solve medium-size (10^5 to 10^6 DOFs) structural TO problems on a laptop. However, the most substantial savings follow from using `fsparse` (Engblom and Lukarski 2016) and by coupling these two strategies (H,`fsparse`) speedups of 10 for the 2D and 15 for 3D setting can be achieved. It is worth to highlight that a 3D stiffness matrix of the size of $\approx 9 \cdot 10^5$ can be assembled in less than a second and even one of size $6.2 \cdot 10^6$ can be assembled on a laptop in less than 10s. For this last case, the sole storage of the arrays `iK`, `jK`, and `sK` would cause a memory overflow, ruling out the "F" approach.

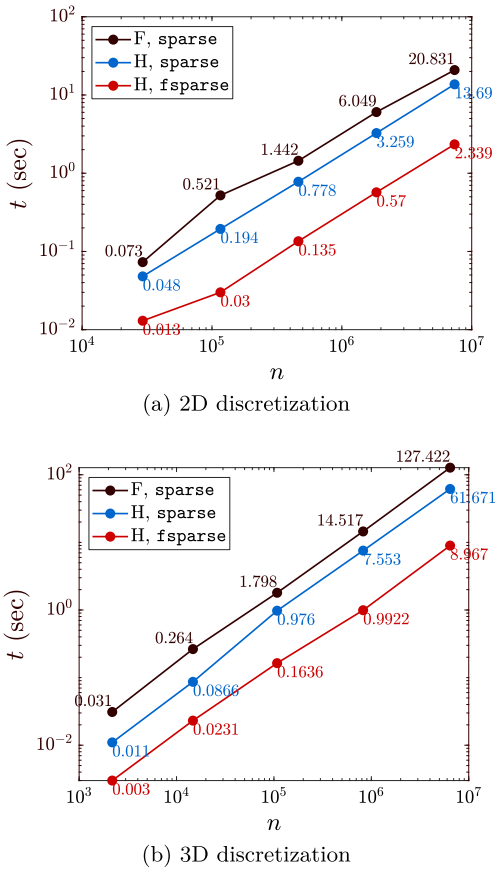


Fig. 3 Scaling of assembly time performed with the 3 strategies discussed in Section 3.1. Compared to the standard (F) assembly, the H strategy alone cuts near 50% of time and memory, and with the use of `fsparse` gives an overall efficiency improvement of 10–15 times

3.2 Speedup of the OC update

The cost of the redesign step $\mathbf{x}_{k+1} = \mathcal{U}(\mathbf{x}_k)$ is proportional to the number of bisections (n_{bs}) required for computing the approximation $\tilde{\lambda}_k \approx \lambda_k^*$. The following estimate (Quarteroni et al. 2000)

$$n_{\text{bs}} \geq \frac{\log(|\Lambda^{(0)}|) - \log(\tau)}{\log(2)} - 1 \tag{16}$$

is a lower bound to this number for a given accuracy $\tau > |\lambda_k^* - \tilde{\lambda}_k|$ and it is clear that n_{bs} would decrease if $\Lambda^{(0)}$, the initial guess for the interval bracketing λ_k^*, could be shrunk. Moreover, the volume constraint should be imposed on the physical field ($\tilde{\mathbf{x}}$ or $\hat{\mathbf{x}}$) and, in the original `top88` implementation, this requires a filter application at each bisection step, which may become expensive.

[1] https://github.com/stefanengblom/stenglib

The efficiency of the redesign step can be improved by a two-step strategy

1. Using volume-preserving filtering schemes
2. Estimating the interval $\Lambda_k^{(0)}$ bracketing the current Lagrange multiplier λ_k^*

Concerning point 1, the density filter is naturally volume-preserving (i.e., $V(\mathbf{x}_k) = V(\tilde{\mathbf{x}}_k)$) (Bourdin 2001; Bruns and Tortorelli 2001). Therefore, the volume constraint can be enforced on $V(\mathbf{x}_k)$ as long as the density filter alone is considered (`ft=1`). The relaxed Heaviside projection (1), on the other hand, is not volume-preserving for any η; thus, it would require one filter-and-projection application at each bisection step. However, (1) can also be made volume-preserving by computing, for each $\tilde{\mathbf{x}}_k$, the threshhold η_k^* such that (Xu et al. 2010; Li and Khandelwal 2015)

$$\eta_k^* \longrightarrow \min_{\eta\in[0,1]} |V(\hat{\mathbf{x}}_k(\eta)) - V(\tilde{\mathbf{x}}_k)| \tag{17}$$

This can be done, e.g., by the Newton method, starting from the last computed η_{k-1}^* and provided the derivative of (1) with respect to η

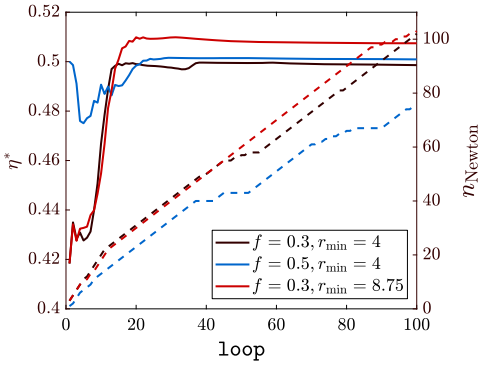
$$\frac{\partial V(\tilde{\mathbf{x}}(\eta))}{\partial \eta} = -2\beta \sum_{i\in\mathcal{A}} \frac{(e^{\beta(1-\tilde{x}_i)} - e^{\beta(\tilde{x}_i-1)})(e^{\beta x} - e^{\beta\tilde{x}_i})}{(e^{\beta} - e^{-\beta})[e^{\beta(\tilde{x}_i-\eta)} + e^{\beta(\eta-\tilde{x}_i)}]^2} \tag{18}$$

Existence of $\eta^* \in [0, 1]$ for all $\tilde{\mathbf{x}} \in [0, 1]^m$ follows from the fact that $g(\eta) = V(\hat{\mathbf{x}}(\eta)) - V(\tilde{\mathbf{x}})$ is continuous on $[0, 1]$ and $g(0)g(1) \leq 0$; uniqueness follows from the fact that $\frac{\partial g}{\partial \eta} < 0$ for all $\eta \in (0, 1)$.

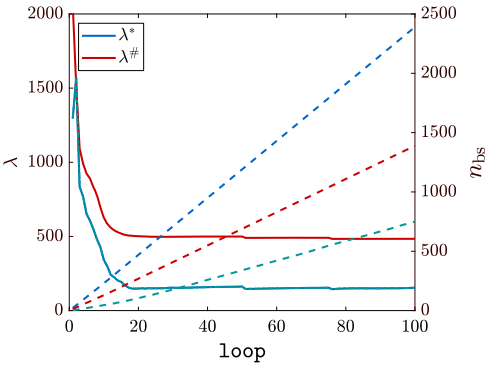
Numerical tests on the MBB beam show that generally $\eta_k^* \in [0.4, 0.52]$, the larger variability occurring for low volume fractions (see Fig. 4a). We also observe that η_k^* takes values slightly above 0.5 when $r_{\min}$ is increased or β is raised. Convergence to η_k^* is generally attained in 1–2 Newton iterations (see Fig. 4a).

The procedure for computing η_k^* from (17), with tolerance $\epsilon = 10^{-6}$ and initial guess $\eta_0 =$ `eta`, provided by the user, is implemented in Lines 63–67, that are executed if the routine `top99neo` is called with the parameter `ft=3`. Otherwise, if `ft=2`, the input threshhold `eta` is kept fixed. In case of the latter, the volume constraint should be consistently applied on $V(\hat{\mathbf{x}})$; otherwise, some violation or over-shooting of the constraint will happen. In particular, if the volume constraint is imposed on $\mathbf{x}$ and η is kept fixed, one has $V(\hat{\mathbf{x}}) > f|\Omega_h|$, if $\eta < 0.5$, and $V(\hat{\mathbf{x}}) < f|\Omega_h|$, if $\eta > 0.5$.

Even tough we usually observed small differences, these may result in local optima or bad designs, especially for low volume fractions or high β values. Therefore, accounting for this more general situation Lines 87–91 should be replaced by the following



(a) dashed lines show the cumulative number of Newton iterations



(b) dashed lines show the cumulative number of bi-section steps. Green lines refer to the use of the explicit primal-dual iteration discussed in Appendix A for computing λ^*

Fig. 4 Evolution of the parameter η^* realizing the equivalence $V(\tilde{\mathbf{x}}) = V(\hat{\mathbf{x}})$, for different volume fractions f and filter radii $r_{\min}$ (**a**) and evolution of the Lagrange multiplier estimate $\lambda^\#$ given by (19) compared to λ^* (**b**). For both plots, the cumulative number of Newton iterations n_{Newton} (viz. number of bisection steps n_{bs}) is shown against the right axis

```
while (l(2)-l(1))/(l(2)+l(1))>1e-4
  lmid=0.5*(l(1)+l(2));
  x=max(max(min(min(ocP/lmid,xU),1),xL),0);
  if ft > 0
    xf=imfilter(reshape(x,nely,nelx)./Hs,h);
    [xf(pasS),xf(pasV)]=deal(1,0);
    if ft>1, xf=prj(xf(:),eta,beta); end
  end
  if mean(xf(:))>volfrac,l(1)=lmid;else,l(2)=lmid;end
end
```

However, there could be other situations when one cannot rely on volume-preserving filters (e.g., when imposing length scale through robust design). Therefore, a more general strategy to reduce the cost of the OC update is to cut the number of bisection steps.

To this end, the selection of the initial bracketing interval $\Lambda_k^{(0)}$ may build upon the upper bound estimate for λ_k^* (Hestenes 1969; Arora et al. 1991)

$$\lambda_k^\# = \left[\frac{1}{mf}\sum_{e=1}^{m} x_{k,e}\left(-\frac{\partial_e c_k}{\partial_e V_k}\right)^{1/2}\right]^2 \tag{19}$$

More details on the derivation of (19) are given in Appendix A. The behavior of the estimate (19) is shown in Fig. 4b for the MBB example. The overall number of bisections (n_{bs}) in order to compute λ_k^* meeting the tolerance $\tau = 10^{-8}$ when considering $\Lambda_k^{(0)} = [0, \lambda_k^*]$ is cut by about 50%, compared with the one required by starting from $\Lambda^{(0)} = [0, 10^9]$ as in `top88`. Moreover, if no projection is applied, (19) could be used together with (10) to perform an explicit Primal-Dual iteration to compute $(\mathbf{x}_{k+1}, \lambda_k^*)$ and this would reduce the number of steps even more (see green curve in Fig. 4b).

However, in the basic versions of the codes, given in Appendices B and C, we consider the bisection process and (19) is used to bracket the search interval, as this procedure is more general.

3.3 Acceleration of the OC iteration

The update rule (10) resembles a fixed-point (FP) iteration $\mathbf{x}_{k+1} = \mathcal{U}(\mathbf{x}_k)$, generating a sequence $\{\mathbf{x}_k\}$ converging to a point such that $\mathbf{r} = \mathcal{U}(\mathbf{x}^*) - \mathbf{x}^* = \mathbf{0}$.

Several methods are available to speedup the convergence of such a sequence (Brezinski and Chehab 1998; Ramiere and Helfer 2015), somehow belonging to the family of quasi-Newton methods (Eyert 1996). The acceleration proposed by Anderson (1965), for instance, is nowadays experiencing a renewed interest (Fang and Saad 2009; Pratapa et al. 2016; Peng et al. 2018) and has recently been applied to TO by Li et al. (2020).

Anderson acceleration takes into account the residuals $\mathbf{r}_i$, their differences $\Delta\mathbf{r}_i = \mathbf{r}_{i+1} - \mathbf{r}_i$ and the differences of the updates $\Delta\mathbf{x}_i = \mathbf{x}_{i+1} - \mathbf{x}_i$ for the last m_r iterations (i.e. $i = k - m_r, \ldots, k-1$), and obtains the new element of the vector sequence as

$$\mathbf{x}_{k+1} = \mathbf{x}_k^{\#} + \zeta \mathbf{r}_k^{\#} \tag{20}$$

where $\zeta \in [0, 1]$ is a damping coefficient and

$$\begin{aligned} \mathbf{x}_k^{\#} &= \mathbf{x}_k - \sum_{i=k-m_r}^{k-1} \gamma_i^{(k)} \Delta\mathbf{x}_i = \mathbf{x}_k - X_k \boldsymbol{\gamma}_k \\ \mathbf{r}_k^{\#} &= \mathbf{r}_k - \sum_{i=k-m_r}^{k-1} \gamma_i^{(k)} \Delta\mathbf{r}_i = \mathbf{r}_k - R_k \boldsymbol{\gamma}_k \end{aligned} \tag{21}$$

The coefficients $\gamma_i^{(k)}$ minimize the following

$$\{\gamma_i^{(k)}\}_{i=1}^{m_r} \rightarrow \min_{\boldsymbol{\gamma}} \|\mathbf{r}_k^{\#}(\boldsymbol{\gamma})\|_2^2 \tag{22}$$

The rationale behind the method is to compute a rank-m_r update of the inverse Jacobian matrix J_k^{-1} of the nonlinear system $\mathbf{r}_k = \mathbf{0}$. This has been shown to be equivalent to a multi-secant Broyden method (Eyert 1996; Fang and Saad 2009) starting from $J_0^{-1} = -\zeta I$.

The update rule (20) is usually applied only once each q steps. Thus, we can write more generally $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{z}_k$, where (Pratapa et al. 2016)

$$\mathbf{z}_k = \begin{cases} \alpha \mathbf{r}_k & \text{if } \frac{k+1}{q} \notin \mathbb{N} \\ \zeta I - (X_k + \zeta F_k)\boldsymbol{\gamma}_k & \text{if } \frac{k+1}{q} \in \mathbb{N} \end{cases} \tag{23}$$

($\alpha \in (0, 1)$) obtaining the so-called periodic Anderson extrapolation (PAE) (Pratapa et al. 2016; Li et al. 2020).

The implementation can be obtained, e.g., by adding the following few lines after the OC step (Line 91)

```
fres = x( act ) - xT( act );
 if loop >= q0
    sel = mod(loop - q0, q)==0;
    mix = sel*xi + alpha*(1-sel);
    x(act) = pae(xT(act),fres,mr,loop-q0,mix,q);
    x(x > 1) = 1; x(x < 0) = 0;
 end
% --------------------------
function [ xnew ] = pae( x, r, m, it, mix, q )
persistent X R Xold Rold;  dp = 0;
if it > 1
   k = mod( it - 1, m ) + 1;
   [X(:, k), R(:, k)] = deal(x - Xold, r - Rold);
   if rem( it-1, q ) == 0
       dp = (X+mix*R)* ((R'*R)\(R'*r));
   end
end
xnew = x + mix * r - dp; Xold = x; Rold = r;
end
```

where the part solving (22) and the update has been put in a separate routine for better efficiency.

In the above, we use the "\" for solving the least squares problem (22); however, strategies based on a QR (or SVD) decomposition may be preferred in terms of numerical stability. We refer to Fang and Saad (2009) for a deeper discussion on this point.

In order to assess the effect of different filtering schemes and the introduction of parameter continuation, Anderson acceleration is tested on the MBB example considering the following options

- T1 Density filter alone, $p = 3$;
- T2 Density-and-projection filter, with η^* computed from (17) and $\beta = 2$
- T3 As T2, but with continuation on both β and p, defined by the parameters `betaCnt={250,16,25,2}` and `penalCnt={50,3,25,0.25}`
- T4 As T2, but for the discretization $\Omega_h = 600 \times 200$

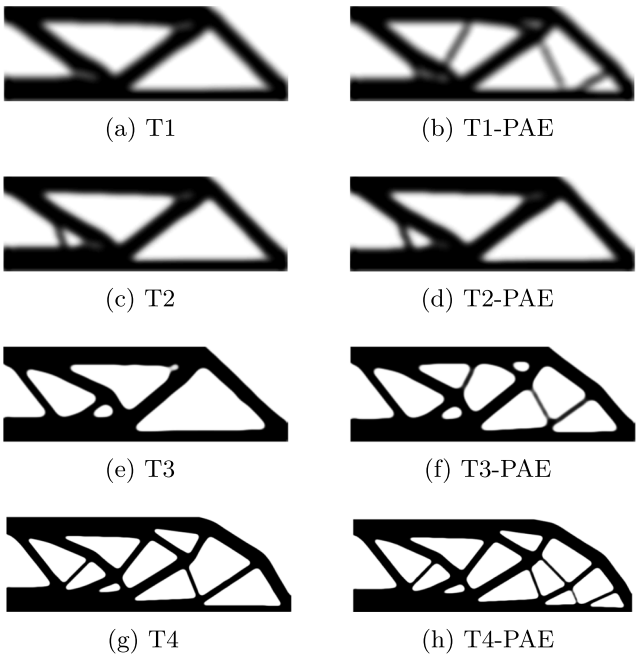
For all the cases, the TO loop stops when $\|\mathbf{r}_k\|_2/\sqrt{m} < 10^{-6}$, where the residual is defined with respect to the physical variables (i.e., $\mathbf{r}_k = \tilde{\mathbf{x}}_k - \tilde{\mathbf{x}}_{k-1}$ for T1 and $\mathbf{r}_k = \hat{\mathbf{x}}_k - \hat{\mathbf{x}}_{k-1}$ for T2–T4). The acceleration is applied each $q = 4$ steps, considering the last $m_r = 4$ residuals, starting from iteration $q_0 = 20$ for T1–T2 and from $q_0 = 500$ for T3–T4, when both continuations have finished. We set $\alpha = 0.9$ for the non-accelerated steps. The choice $m_r = 4$ is based on the observation that convergence improvements increase very slowly for $m_r > 3$ (Anderson 1965; Eyert

Table 2 Comparison of convergence-related parameters for the standard (T) and accelerated (T-PAE) TO tests, for the MBB example

	it.	c	Δc	$\|\mathbf{r}\|_2/\sqrt{m}$	m_{ND}
T1	2500	252.7	$4.2 \cdot 10^{-8}$	$1.03 \cdot 10^{-5}$	0.025
T1-PAE	828	258.9	$4.2 \cdot 10^{-10}$	$9.95 \cdot 10^{-7}$	0.021
T2	2500	246.1	$5.1 \cdot 10^{-8}$	$3.21 \cdot 10^{-5}$	0.023
T2-PAE	352	253.9	$6.2 \cdot 10^{-9}$	$9.97 \cdot 10^{-7}$	0.014
T3	2500	199.6	$1.1 \cdot 10^{-4}$	$1.91 \cdot 10^{-3}$	0.014
T3-PAE	752	197.5	$3.7 \cdot 10^{-8}$	$8.72 \cdot 10^{-7}$	0.007
T4	2500	191.8	$2.0 \cdot 10^{-7}$	$3.21 \cdot 10^{-5}$	0.006
T4-PAE	818	192.1	$2.5 \cdot 10^{-7}$	$9.97 \cdot 10^{-7}$	0.001

1996). However, a deeper discussion about the influence of all parameters on the convergence is outside the scope of the present work and we refer to Li et al. (2020) or, in a more general context, to Walker and Ni (2011) for this .

Results are collected in Table 2 and Figs. 5 and 6, showing the evolution of the norm of the residual, the flatness of the normalized compliance $\Delta c_k/c_0 = (c_k - c_{k-1})/c_0$ and the non-discreteness measure $m_{\text{ND}} = 100 \cdot 4\mathbf{x}^T(1-\mathbf{x})/m$. We observe how Anderson acceleration substantially reduces the number of iterations needed to fulfill the stopping criterion, at the price of just a moderate increase in compliance (0.2–3%). Moreover, starting the acceleration just a few iterations later (e.g., it = 50 or it = 100 for T1) gives much lower compliance values ($c = 254.3$ and $c = 252.9$, respectively) and for T3 and T4 when the acceleration is started as the design has stabilized, compliance differences are negligible.



(a) T1 (b) T1-PAE (c) T2 (d) T2-PAE (e) T3 (f) T3-PAE (g) T4 (h) T4-PAE

Fig. 5 Optimized designs obtained without (left column) and with Anderson acceleration (right column) of the TO loop

From Fig. 5 it is easy to notice the trend of PAE of producing a design with some more bars. This may even give slightly stiffer structures, such as for case T3, where the non accelerated approach removes some bars after it = 2000, whereas stopping at the design of T3–PAE gives a stiffer structure.

A comment is about the convergence criterion used, which is different from the one in `top88` (maximum absolute change of the design variables ($\|\mathbf{x}_{k+1} - \mathbf{x}_k\|_\infty$). Here, we consider it more appropriate to check the residual with respect to the *physical* design field, and the 2-norm seems to give a more global measure, less affected by local oscillations.

3.4 Performance comparison to `top88`

We compare the performance of `top99neo` to the previous `top88` code. In the following, we will refer to "`top88`" as the original code provided by Andreassen et al. (2011) and to "`top88U`" as its updated version making use of the `sparse2` function (Davis 2009) for the assembly, with `iK` and `jK` specified as integers, and the filter implemented by using `conv2`.

The codes are tested by running 100 iterations for the MBB beam example (see Fig. 2), for the discretizations 300×100, 600×200, and 1200×400, a volume fraction $f = 0.5$ and considering mesh independent filters of radii $r_{\min} = 4$, 8, and 16, respectively. For `top88` and `top88U`, we only consider density filtering, whereas for the new `top99neo`, we also consider the Heaviside projection, with the η^* computed as described in Section 3.2. It will be apparent that the cost of this last operation is negligible.

Timings are collected in Table 3 where t_{it} is the average cost per iteration, t_A and t_S are the overall time spent by the assembly and solver, respectively, and t_U is the overall time spent for updating the design variables. For `top88` and `top88U`, the latter consists of the OC updating and the filtering operations performed when applying the bisection on the volume constraint. For `top99neo`, this term accounts for the cost of the OC updating, that for estimating the Lagrange multiplier λ^* as discussed in Section 3.2 and the filter and projection (Lines 59–70). t_P collects all the preliminary operations, such as the set up of the discretization, and filter, repeated only once, before the TO loop starts.

From t_{it}, we clearly see that `top99neo` enhances the performance of the original `top88` by 2.66, 3.85, and 5.5 times on the three discretizations, respectively. Furthermore, timings of `top88` on the largest discretization (1200×400), relate to a smaller filter size ($r_{\min} = 12$), because of memory issues; thus, the speedup is even underestimated in this case. Comparing to `top88U` version, the improvements are less pronounced (i.e., 1.55, 1.57, and 1.78 times) but still

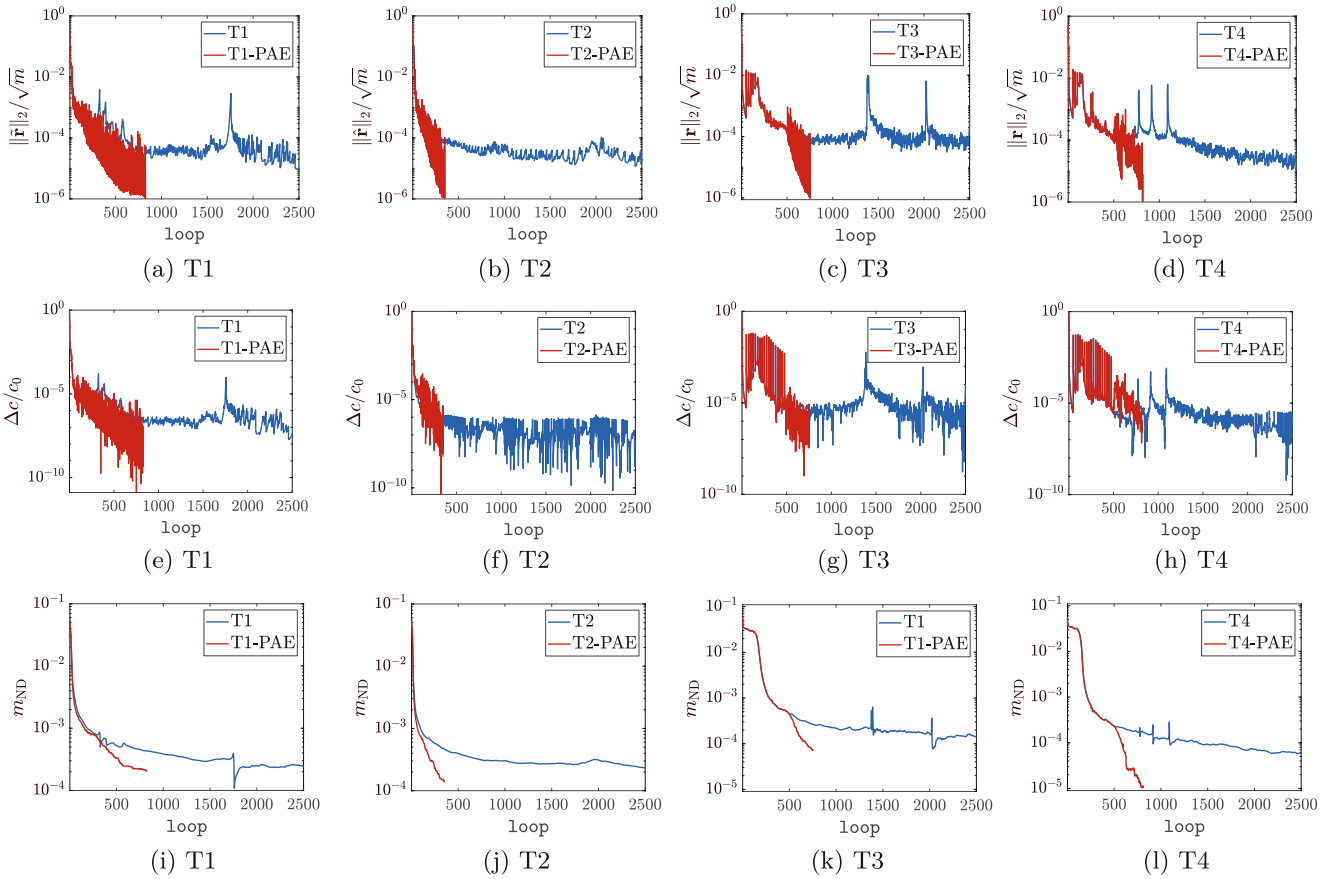


Fig. 6 Evolution of some parameters related to convergence for the standard and Anderson accelerated TO process. The first row shows the normalized norm of the residual defined on physical variables, the second row shows a measure of the flatness of the objective function and the last row shows the non-discreteness measure

substantial. The computational cost of the new assembly strategy is very low, even comparing to the `top88U` version, and its weight on the overall computational cost is basically constant. Also, from Table 3, it is clear that the design variables update weighs a lot on the overall CPU time, for both `top88` and `top88U`. On the contrary, this becomes very cheap in the new `top99neo` thanks to the strategies discussed in Section 3.2; t_U takes about 4–5% of the overall CPU time.

Computational savings would become even higher when adopting the larger filter size $r_{\min} = 8.75$ for the mesh 300×100, and scaling to $r_{\min} = 17.5$ and $r_{\min} = 35$ on the two finer discretizations. For these cases, speedups with respect to `top88` amount to 4.45 and 10.35 on the first two meshes, whereas for the larger one, the setup of the filter in `top88` causes a memory overflow. Speedups with respect to `top88U` amount to 1.55, 2.55 and 3.6 times respectively.

Table 3 Comparison of numerical performance between the old `top88`/`top88U` and new `top99neo` Matlab code. t_{it} is the cost per iteration, t_A, t_S, t_U are the overall times for assembly, equilibrium equation solve, and design update, respectively. t_P is the time spent for all the preliminary operations. Values within brackets represent the % weight of the corresponding operation on the overall CPU. On the larger mesh, `top88` is run with $r_{\min} = 12$, because of memory issues

Ω_h	300×100, $r_{\min} = 4$			600×200, $r_{\min} = 8$			1200×400, $r_{\min} = 16$		
	`top88`	`top88U`	`top99neo`	`top88`	`top88U`	`top99neo`	`top88`	`top88U`	`top99neo`
t_{it}	0.615	0.358	0.231	4.57	1.87	1.19	31.3	10.1	5.69
t_A	19.4(31.5)	5.4(15.0)	1.4 (6.1)	83.1(18.2)	31.3(16.7)	5.6 (4.7)	361.1(11.6)	151.5(15.2)	30.7 (5.4)
t_S	23.1(37.4)	22.9(59.3)	19.7(85.3)	122.4(26.8)	109.3(58.4)	106.9(89.7)	592.5(19.0)	513.2(50.9)	510.5(89.6)
t_U	13.3(21.6)	4.8(13.5)	1.2 (4.8)	223.8(48.8)	38.0(20.3)	5.2 (4.4)	1164.2(37.4)	310.4(31.4)	29.2 (5.1)
t_P	0.8(1.3)	0.06 (0.2)	0.1 (0.3)	12.9 (2.8)	0.1(< 0.1)	0.2(< 0.1)	92.3 (3.1)	0.5(< 0.1)	0.6(< 0.1)

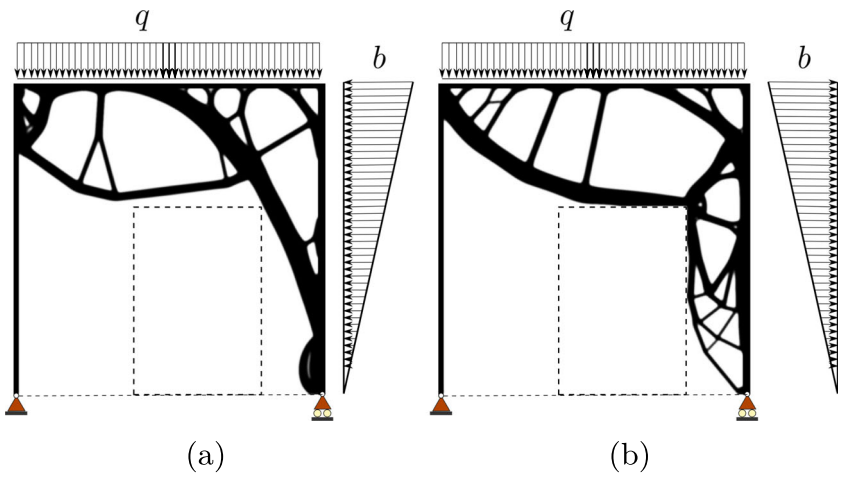


Fig. 7 Designs obtained for the frame reinforcement problem sketched in Fig. 1a. In **a**, the horizontal, triangular load distribution is pointing leftwards, whereas in **b,** it is pointing rightwards

3.5 Frame reinforcement problem

Let us go back to the example of Fig. 1a, adding the specification of passive domains and a different loading condition.

We may think of a practical application like a reinforcement problem for the solid frame, with thickness $t = L/50$ ($\mathcal{P}_1$), subjected to two simultaneous loads. A vertical, uniformly distributed load with density $q = -2$ and a horizontal height-proportional load, with density $b = \pm y/L$. Some structural material has to be optimally placed within the active design domain $\mathcal{A}$ in order to minimize the compliance, while keeping the void space ($\mathcal{P}_0$), which may represent a service opening.

To describe this configuration, we only need to replace Lines 31–33 with the following

```
elNrs = reshape(1:nEl,nely,nelx);
[lDofv,lDofh]=deal(2*nodeNrs(1,:),2*nodeNrs(:,end)-1);
fixed = [1,2,nDof];
a1=elNrs(1:nely/50,:);
a2=elNrs( :,[1:nelx/50,end-nelx/50+1:end]);
a3=elNrs(2*nely/5:end,2*nelx/5:end-nelx/5);
[pasS,pasV]=deal(unique([a1(:);a2(:)]),a3(:));
```

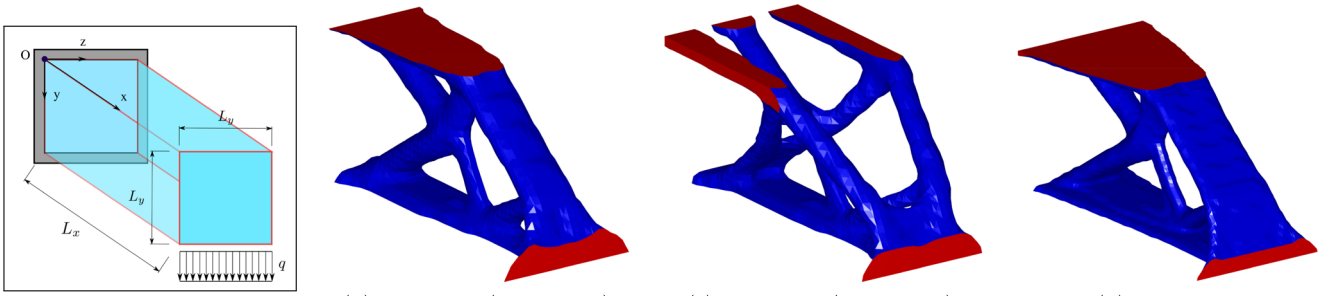
where `lDofv` and `lDofh` target the DOFs subjected to vertical and horizontal forces, respectively. Then, the load (Line 34) is replaced with

```
F=fsparse(lDofv',1,-2/(nelx+1),[nDof,1]) + fsparse(...
lDofh,1,-[0:1/(nely).^2:1/nely]',[nDof,1]);
```

Figure 7 shows the two optimized design corresponding to the two orientations of the horizontal load b, after 100 redesign steps. The routine `top99neo` has been called with the following arguments `nely=nelx=900`, `volfrac=0.2`, `penal=3`, `rmin=8`, `ft=3`, `eta=0.5`, `beta=2` and no continuation is applied. The cost per iteration is about 10.8 s and, considering the fairly large discretization of $1.62 \cdot 10^6$ DOFs, is very reasonable.

4 Extension to 3D

The implementation described in Section 3 is remarkably easy to be extended to 3D problems (see Section Appendix).



(a) (b) Neumann ($c = 3993.7$) (c) Dirichlect ($c = 4074.3$) (d) Neumann

Fig. 8 Geometrical sketch of the 3D cantilever example (**a**) and optimized topology for $\Omega_h = 48 \times 24 \times 24$ and considering the two filter boundary conditions (**b**, **c**). The design in **d** corresponds to the finer mesh $\Omega_h = 96 \times 48 \times 48$ and has been obtained by replacing the direct solver with the multigrid–preconditioned CG (see Amir et al. 2014 for details)

Notable modifications are the definition of $K_e^{(s)}$ for the 8-node hexahedron (Lines 24–47) and the solution of the equilibrium (5), now performed by

```
L = chol( K( free, free ), 'lower' );
U( free ) = L' \ ( L \ F( free ) );
```

which in this context has been observed to be faster than the `decomposition` routine. Then, apart from the plotting instructions, all the operations are the same as in the 2D code and only 12 lines need minor modifications, basically to account for the extra space dimension (see tags "#3D#" in Section Appendix).

We test the 3D implementation on the cantilever example shown in Fig. 8a, for the same data considered in Amir et al. (2014). The discretization is set to $\Omega_h = 48 \times 24 \times 24$, the volume fraction is $f = 0.12$, and the filter radius $r_{\min} = \sqrt{3}$. We also consider the volume-preserving Heaviside projection, (`ft=3`). Figure 8b and c show the designs obtained after 100 redesign steps, for the two different filter boundary conditions. The design in (b), identical to the one in Amir et al. (2014), corresponds to zero-Neumann boundary conditions (i.e., the option "`symmetric`" was used in `imfilter`). The design in (c) on the other hand, corresponds to zero-Dirichlect boundary conditions for the filter operator and is clearly a worse local minimum.

The overall CPU time spent over 100 iterations is 1741 s and about 96% of this is due to the solution of the state equation. Only 1.2% of the CPU time is taken by matrix assemblies and 0.4% by filtering and the design update processes.

Upon replacing the direct solver in `top3D125` with the same multigrid preconditioned CG solver of Amir et al. (2014), we can compare the efficiency of the two codes. We refer to Table 4 for the CPU timings, considering the discretizations $\Omega_h = 48 \times 24 \times 24$ ($l = 3$ multigrid levels) and $\Omega_h = 96 \times 48 \times 48$ ($l = 4$ multigrid levels). `top3D125` shows speedups of about 1.8 and 1.9, respectively, and most of the time is cut on the matrix assembly. In the code of Amir et al. (2014), this operation takes about 50% of the overall time (and notably has the same weight as the state equation solve) whereas in `top3D125` this weight is cut to $7 - 10\%$. Also, the time spent for the OC update is reduced, even though the code of Amir et al. (2014) already implemented a strategy for avoiding filtering at each bisection step.

5 Concluding remarks

We have presented new Matlab implementations of compliance topology optimization for 2D and 3D domains. Compared to the previous `top88` code (Andreassen et al. 2011) and available 3D codes (e.g., by Liu and Tovar 2014 or Amir et al. 2014), the new codes show remarkable speedups.

Improvements are mainly due to the following:

1. The matrix assembly is made much more efficient by defining mesh-related quantities as integers (Matlab `int32`) and assembling just one half of the matrix.
2. The number of OC iterations is drastically cut by looking at the explicit expression of the Lagrange multiplier for the problem at hand.
3. Filter implementation and volume-preserving density projection allow to speed up the redesign step.

The new codes are computationally well balanced and as the problem size increases the majority of the time (85 to 90% for 2D and even 96% for 3D discretizations) is spent on the solution of the equilibrium system. This is precisely what we aimed at, as this step can be dealt with efficiently by preconditioned iterative solvers (Amir et al. 2014; Ferrari et al. 2018; Ferrari and Sigmund 2020). We also discussed Anderson acceleration, that has recently been applied to TO also by Li et al. (2020), to accelerate the convergence of the overall optimization loop.

We point out that even if we specifically addressed volume constrained compliance minimization and density-based TO the methods above can be applied also to level-set and other TO approaches. Point 1 can be extended to all problems governed by symmetric matrices. Points 2 and 3

Table 4 Performance comparison between the new `top3D125` code and the one from Amir et al. (2014). t_{it}, t_A, t_S, t_U, and t_P have the same meaning as in Table 3 and numbers between brackets denote the % weight of the operations on the overall CPU time

Ω_h	$48 \times 24 \times 24$, $r_{\min} = \sqrt{3}$		$96 \times 48 \times 48$, $r_{\min} = 2\sqrt{3}$	
	`top3dmgcg`	`top3D125`	`top3dmgcg`	`top3D125`
t_{it}	3.19	1.79	27.33	14.20
t_A	160.6(50.3)	13.1 (7.4)	1369(50.1)	137.2(9.7)
t_S	148.1(46.4)	151.7(84.7)	1250(45.7)	1272(89.5)
t_U	1.97 (0.6)	0.7 (0.4)	21.2 (0.8)	15.12(1.1)
t_P	0.74 (0.4)	0.24 (0.1)	39.2 (1.4)	0.29(<0.1)

can also be extended to other problems, to some extent, and Anderson acceleration is also usable in a more general setting (e.g., within MMA).

Therefore, we believe that this contribution should be helpful to all researchers and practitioners who aim at tackling TO problems on laptops, and set a solid framework for the efficient implementation of more advanced procedures.

Acknowledgments The project is supported by the Villum Fonden through the Villum Investigator Project "InnoTop." The authors are grateful to members of the TopOpt group for their useful testing of the code.

Compliance with ethical standards

Conflict of interests The authors declare that they have no conflict of interest.

Replication of results Matlab codes are listed in the Appendix and available at `www.topopt.dtu.dk`. The `stenglib` package, containing the `fsparse` function, is avaialble for download at `https://github.com/stefanengblom/stenglib`.

Appendix A: Elaboration on the OC update

Let us consider (3) at a given design point $\mathbf{x}_k$ assuming the reciprocal and linear approximation for the compliance and volume functions, respectively (Christensen and Klarbring 2008)

$$\begin{cases} \min\limits_{\mathbf{x}\in[\delta_-,\delta_+]^m} c(\mathbf{x}) \simeq c_k + \sum_{e=1}^m (-x_{k,e}^2 \partial_e c(\mathbf{x}_k)) x_e^{-1} \\ \text{s.t.} \quad \sum_{e=1}^m \partial_e V(\mathbf{x}_k) x_e - f|\Omega_h| \le 0 \end{cases} \tag{24}$$

We set up the Lagrangian associated with (24)

$$L(\mathbf{x},\lambda) = c(\mathbf{x}) + \lambda \left(\sum_{e=1}^m \partial_e V(\mathbf{x}_k) x_e - f|\Omega_h| \right)$$

and seek the pair $(\mathbf{x}_{k+1}, \lambda_k^*) \in \mathbb{R}^m \times \mathbb{R}_+$ solving the subproblem

$$\max_{\lambda>0} \left\{ \psi(\lambda) := \min_{\mathbf{x}\in\mathcal{C}} L(\mathbf{x},\lambda) \right\} \tag{25}$$

where $\mathcal{C} = \{\mathbf{x} \in \mathbb{R}^m \mid \delta_- \le x_e \le \delta_+,\ e = 1\ldots,m\}$ and $\psi(\lambda)$ is the dual function. Equation (25) is solved by primal-dual (PD) iterations, as $\mathbf{x}$ and λ are interlaced. Replacing $\boldsymbol{\xi} = \mathbf{x}_k$ and using subscripts (j) to denote inner PD iterations, we have

1. Fixed $\lambda = \lambda_{(j)}$, the inner minimization in (25) gives

$$\xi_e^2 \partial_e c(\boldsymbol{\xi}) x_e^{-2} + \lambda \partial_e V(\boldsymbol{\xi}) = 0 \Longrightarrow x_e = \xi_e \left(-\frac{\partial_e c(\boldsymbol{\xi})}{\lambda \partial_e V(\boldsymbol{\xi})} \right)^{\frac{1}{2}}$$

due to separability of the approximation. Let us denote the rightmost expression $x_e = \mathcal{F}_{(j)e}(\lambda)$, and taking into account the box constraints in $\mathcal{C}$, we have

$$\mathcal{U}(x_e) = \begin{cases} x_{(j+1),e} = \delta_- & \text{if } e \in \mathcal{L} = \{e \mid x_{(j+1),e} \le \delta_-\} \\ x_{(j+1),e} = \delta_+ & \text{if } e \in \mathcal{U} = \{e \mid x_{(j+1),e} \ge \delta_+\} \\ x_{(j+1),e} = \mathcal{F}_{(j),e} & \text{if } e \in \mathcal{M} = \{e \mid \delta_- < x_{(j+1),e} < \delta_+\} \end{cases} \tag{26}$$

where $\mathcal{C} = \mathcal{L}+\mathcal{U}+\mathcal{M}$. The above is equivalent to (10).

2. We then evaluate the dual function for $x_{(j+1)}$ given by (26), and the stationarity $(\partial_\lambda \psi = 0)$ gives

$$\sum_{e=1}^m \partial_e V(\boldsymbol{\xi})(\chi_{\mathcal{U}}\delta_+ + \chi_{\mathcal{L}}\delta_- + \mathcal{F}_{(j),e}(\lambda)\chi_{\mathcal{M}}) - f|\Omega_h| = 0$$

where $\chi_{[\cdot]}$ is the characteristic function of a set. In this simple case, the above can be solved for $\lambda_{(j+1)}$, the Lagrange multiplier enforcing the volume constraint for the updated density $x_{(j+1)}$, and after some simplifications, we obtain

$$\lambda_{(j+1)} = \left(\frac{\sum_{e\in\mathcal{M}} x_{(j+1)e}(\partial_e c(\boldsymbol{\xi})/\partial_e V(\boldsymbol{\xi}))^{1/2}}{f|\Omega_h|/\partial_e V(\boldsymbol{\xi}) - |\mathcal{L}|\delta_- - |\mathcal{U}|\delta_+} \right)^2 \tag{27}$$

where $|\cdot|$ denotes the number of elements in a set.

Equations (26) and (27) can be iteratively used to compute the new solution $(\mathbf{x}_{k+1}, \lambda_k^*)$, as implemented in the code here below (again, note that `lm` here represents $\sqrt{\lambda}$)

```
u=min(xT+move,1); l=max(xT-move,0);
ocP=@(s) xT(s).*sqrt(-dc(s)./dV0(s));
lm=mean(ocP(act))/volfrac;
lmOld = 0;
while abs(lm-lmOld)>1e-10
  tmp=ocP(act)/lm;
  [setu,setl]=deal(find(tmp>u),find(tmp<l));
  setM=not(abs(sign(sign(l-tmp)+sign(u-tmp))));
  den=volfrac-(sum(u(setu))+sum(l(setl)))/nEl;
  lmOld=lm;
  lm=(sum(ocP(setM))/den)/nEl;
end
x=ocP(act)/lm;
[setu, setl]=deal(find(x>u),find(x<l));
x(setl)=l(setl); x(setu)=u(setu);
```

and, for the MBB beam example, this performs as shown by the green curves in Fig. 4b.

However, a closed form expression such as (27) cannot be obtained for more involved constraint expressions and therefore a root finding strategy must be employed to approximate the Lagrange multiplier. The application of (27) to the current, feasible design point $(\mathbf{x}_{(j+1)} = \mathbf{x}_k)$ reduces to

$$\lambda^\# = \left[\frac{1}{mf} \sum_{e=1}^m x_{k,e} \left(-\frac{\partial_e c(\boldsymbol{\xi})}{\partial_e V(\boldsymbol{\xi})} \right)^{1/2} \right]^2 \tag{28}$$

since $|\mathcal{M}| = |\Omega_h| = m$, $|\mathcal{L}| = |\mathcal{U}| = 0$ and we made use of (7). We immediately verify that (28) is identical to (19).

AppendixB: The 2D code for compliance minimization

```
function top99neo(nelx,nely,volfrac,penal,rmin,ft,ftBC,eta,beta,move,maxit)
% ------------------------------ PRE. 1) MATERIAL AND CONTINUATION PARAMETERS
E0 = 1;                                                    % Young modulus of solid
Emin = 1e-9;                                               % Young modulus of "void"
nu = 0.3;                                                  % Poisson ratio
penalCnt = { 1,  3, 25, 0.25 };                            % continuation scheme on penal
betaCnt  = { 1,  2, 25,    2 };                            % continuation scheme on beta
if ftBC == 'N', bcF = 'symmetric'; else, bcF = 0; end      % filter BC selector
% ------------------------------ PRE. 2) DISCRETIZATION FEATURES
nEl = nelx * nely;                                         % number of elements
nodeNrs = int32( reshape( 1 : (1 + nelx) * (1 + nely), 1+nely, 1+nelx ) ); % nodes numbers (defined as int32)
cVec = reshape( 2 * nodeNrs( 1 : end - 1, 1 : end - 1 ) + 1, nEl, 1 );
cMat = cVec + int32( [ 0, 1, 2 * nely + [ 2, 3, 0, 1 ], -2, -1 ] );  % connectivity matrix
nDof = ( 1 + nely ) * ( 1 + nelx ) * 2;                    % total number of DOFs
[ sI, sII ] = deal( [ ] );
for j = 1 : 8
    sI = cat( 2, sI, j : 8 );
    sII = cat( 2, sII, repmat( j, 1, 8 - j + 1 ) );
end
[ iK , jK ] = deal( cMat( :,  sI )', cMat( :, sII )' );
Iar = sort( [ iK( : ), jK( : ) ], 2, 'descend' ); clear iK jK   % reduced assembly indexing
c1 = [12;3;-6;-3;-6;-3;0;3;12;3;0;-3;-6;-3;-6;12;-3;0;-3;-6;3;12;3;...
    -6;3;-6;12;3;-6;-3;12;3;0;12;-3;12];
c2 = [-4;3;-2;9;2;-3;4;-9;-4;-9;4;-3;2;9;-2;-4;-3;4;9;2;3;-4;-9;-2;...
    3;2;-4;3;-2;9;-4;-9;4;-4;-3;-4];
Ke = 1/(1-nu^2)/24*( c1 + nu .* c2 );                      % lower sym. part of el. matrix
Ke0( tril( ones( 8 ) ) == 1 ) = Ke';
Ke0 = reshape( Ke0, 8, 8 );
Ke0 = Ke0 + Ke0' - diag( diag( Ke0 ) );                    % recover full elemental matrix
% ------------------------------ PRE. 3) LOADS, SUPPORTS AND PASSIVE DOMAINS
lcDof = 2 * nodeNrs( 1, 1 );                               % DOFs with applied load
fixed = union( 1 : 2 : 2*( nely + 1 ), 2 * nodeNrs( end, end ) );  % restrained DOFs
[ pasS, pasV ] = deal( [], [] );                           % UD, passive solid and void el.
F = fsparse( lcDof', 1, -1, [ nDof, 1 ] );                 % define load vector
free = setdiff( 1 : nDof, fixed );                         % set of free DOFs
act = setdiff( (1 : nEl )', union( pasS, pasV ) );         % set of active d.v.
% ------------------------------ PRE. 4) DEFINE IMPLICIT FUNCTIONS
prj = @(v,eta,beta) (tanh(beta*eta)+tanh(beta*(v(:)-eta)))./...
    (tanh(beta*eta)+tanh(beta*(1-eta)));                   % projection
deta = @(v,eta,beta) - beta * csch( beta ) .* sech( beta * ( v( : ) - eta ) ).^2 .* ...
    sinh( v( : ) * beta ) .* sinh( ( 1 - v( : ) ) * beta );  % projection eta-derivative
dprj = @(v,eta,beta) beta*(1-tanh(beta*(v-eta)).^2)./(tanh(beta*eta)+tanh(beta*(1-eta)));% proj. x-derivative
cnt = @(v,vCnt,l) v+(l>=vCnt{1})*(v<vCnt{2})*(mod(l,vCnt{3})==0)*vCnt{4};  % apply continuation
% ------------------------------ PRE. 5) PREPARE FILTER
[dy,dx] = meshgrid(-ceil(rmin)+1:ceil(rmin)-1,-ceil(rmin)+1:ceil(rmin)-1);
h = max( 0, rmin - sqrt( dx.^2 + dy.^2 ) );                % conv. kernel
Hs = imfilter( ones( nely, nelx ), h, bcF );               % matrix of weights (filter)
dHs = Hs;
% ------------------------------ PRE. 6) ALLOCATE AND INITIALIZE OTHER PARAMETERS
[ x, dsK, dV ] = deal( zeros( nEl, 1 ) );                  % initialize vectors
dV( act, 1 ) = 1/nEl/volfrac;                              % derivative of volume (constant)
x( act ) = ( volfrac*( nEl - length(pasV) ) - length(pasS) )/length( act );% volume fraction on active set
x( pasS ) = 1;                                             % set x = 1 on pasS set
[ xPhys, xOld, ch, loop, U ] = deal( x, 1, 1, 0, zeros( nDof, 1 ) );  % old x, x change, it. counter, U
% ================================================ START OPTIMIZATION LOOP
while ch > 1e-6 && loop < maxit
  loop = loop + 1;                                         % update iter. counter
  % ---------- RL. 1) COMPUTE PHYSICAL DENSITY FIELD (AND ETA IF PROJECT.)
  xTilde = imfilter( reshape( x, nely, nelx ), h, bcF ) ./ Hs;
  xPhys( act ) = xTilde( act );                            % reshape to column vector
  if ft > 1                                % compute optimal eta* with Newton
      f = ( mean( prj( xPhys, eta, beta ) ) - volfrac ) * ( ft == 3 );  % function (volume)
      while abs( f ) > 1e-6           % Newton process for finding opt. eta
          eta = eta - f / mean( deta( xPhys( : ), eta, beta ) );
          f = mean( prj( xPhys, eta, beta ) ) - volfrac;
      end
      dHs = Hs ./ reshape( dprj( xTilde, eta, beta ), nely, nelx );  % modification of the sensitivity
      xPhys = prj( xPhys, eta, beta );                     % projected (physical) field
```

```
    end
    ch = norm( xPhys - xOld ) ./ sqrt( nEl );
    xOld = xPhys;
    % ------------------------------ RL. 2) SETUP AND SOLVE EQUILIBRIUM EQUATIONS
    sK = ( Emin + xPhys.^penal * ( E0 - Emin ) );                          % stiffness interpolation
    dsK( act ) = -penal * ( E0 - Emin ) * xPhys( act ) .^ ( penal - 1 );   % derivative of stiffness interp.
    sK = reshape( Ke( : ) * sK', length( Ke ) * nEl, 1 );
    K = fsparse( Iar( :, 1 ), Iar( :, 2 ), sK, [ nDof, nDof ] );           % assemble stiffness matrix
    U( free ) = decomposition( K( free, free ), 'chol','lower' ) \ F( free );% solve equilibrium system
    % ------------------------------------------------ RL. 3) COMPUTE SENSITIVITIES
    dc = dsK .* sum( ( U( cMat ) * Ke0 ) .* U( cMat ), 2 );                % derivative of compliance
    dc = imfilter( reshape( dc, nely, nelx ) ./ dHs, h, bcF );             % filter objective sensitivity
    dV0 = imfilter( reshape( dV, nely, nelx ) ./ dHs, h, bcF );            % filter compliance sensitivity
    % ------------------- RL. 4) UPDATE DESIGN VARIABLES AND APPLY CONTINUATION
    xT = x( act );
    [ xU, xL ] = deal( xT + move, xT - move );                             % current upper and lower bound
    ocP = xT .* real( sqrt( -dc( act ) ./ dV0( act ) ) );                  % constant part in resizing rule
    l = [ 0, mean( ocP ) / volfrac ];                                      % initial estimate for LM
    while ( l( 2 ) - l( 1 ) ) / ( l( 2 ) + l( 1 ) ) > 1e-4                 % OC resizing rule
        lmid = 0.5 * ( l( 1 ) + l( 2 ) );
        x( act ) = max( max( min( min( ocP / lmid, xU ), 1 ), xL ), 0 );
        if mean( x ) > volfrac, l( 1 ) = lmid; else, l( 2 ) = lmid; end
    end
    [penal,beta] = deal(cnt(penal,penalCnt,loop), cnt(beta,betaCnt,loop));  % apply conitnuation on parameters
    % ------------------------------ RL. 5) PRINT CURRENT RESULTS AND PLOT DESIGN
    fprintf( 'It.:%5i C:%7.4f V:%7.3f ch.:%0.2e penal:%7.2f beta:%7.1f eta:%7.2f \n', ...
        loop, F'*U, mean( xPhys ), ch, penal, beta, eta );
    colormap( gray ); imagesc( 1 - reshape( xPhys, nely, nelx ) );
    caxis([0 1]); axis equal off; drawnow;
end
end
```

AppendixC: 3D code for compliance minimization

```
function top3D125(nelx,nely,nelz,volfrac,penal,rmin,ft,ftBC,eta,beta,move,maxit)
%------------------------------------ PRE. 1) MATERIAL AND CONTINUATION PARAMETERS
E0 = 1;                                                    % Young modulus of solid
Emin = 1e-9;                                               % Young modulus of "void"
nu = 0.3;                                                  % Poisson ratio
penalCnt = { 1, 3, 25, 0.25 };                             % continuation scheme on penal
betaCnt  = { 1, 2, 25,    2 };                             % continuation scheme on beta
if ftBC == 'N', bcF = 'symmetric'; else, bcF = 0; end      % filter BC selector
%------------------------------------------- PRE. 2) DISCRETIZATION FEATURES
nEl = nelx * nely * nelz;                                  % number of elements          #3D#
nodeNrs = int32( reshape( 1 : ( 1 + nelx ) * ( 1 + nely ) * ( 1 + nelz ), ...
    1 + nely, 1 + nelz, 1 + nelx ) );                      % nodes numbering             #3D#
cVec = reshape( 3 * nodeNrs( 1 : nely, 1 : nelz, 1 : nelx ) + 1, nEl, 1 ); %              #3D#
cMat = cVec+int32( [0,1,2,3*(nely+1)*(nelz+1)+[0,1,2,-3,-2,-1],-3,-2,-1,3*(nely+...
   1)+[0,1,2],3*(nely+1)*(nelz+2)+[0,1,2,-3,-2,-1],3*(nely+1)+[-3,-2,-1]]);% connectivity matrix  #3D#
nDof = ( 1 + nely ) * ( 1 + nelz ) * ( 1 + nelx ) * 3;     % total number of DOFs        #3D#
[ sI, sII ] = deal( [ ] );
for j = 1 : 24
    sI = cat( 2, sI, j : 24 );
    sII = cat( 2, sII, repmat( j, 1, 24 - j + 1 ) );
end
[ iK, jK ] = deal( cMat( :, sI )', cMat( :, sII )' );
Iar = sort( [ iK( : ), jK( : ) ], 2, 'descend' ); clear iK jK          % reduced assembly indexing
Ke = 1/(1+nu)/(2*nu-1)/144 *( [ -32;-6;-6;8;6;6;10;6;3;-4;-6;-3;-4;-3;-6;10;...
    3;6;8;3;3;4;-3;-3; -32;-6;-6;-4;-3;6;10;3;6;8;6;-3;-4;-6;-3;4;-3;3;8;3;...
    3;10;6;-32;-6;-3;-4;-3;-3;4;-3;-6;-4;6;6;8;6;3;10;3;3;8;3;6;10;-32;6;6;...
    -4;6;3;10;-6;-3;10;-3;-6;-4;3;6;4;3;3;8;-3;-3;-32;-6;-6;8;6;-6;10;3;3;4;...
    -3;3;-4;-6;-3;10;6;-3;8;3;-32;3;-6;-4;3;-3;4;-6;3;10;-6;6;8;-3;6;10;-3;...
    3;8;-32;-6;6;8;6;-6;8;3;-3;4;-3;3;-4;-3;6;10;3;-6;-32;6;-6;-4;3;3;8;-3;...
    3;10;-6;-3;-4;6;-3;4;3;-32;6;3;-4;-3;-3;8;-3;-6;10;-6;-6;8;-6;-3;10;-32;...
    6;-6;4;3;-3;8;-3;3;10;-3;6;-4;3;-6;-32;6;-3;10;-6;-3;8;-3;3;4;3;3;-4;6;...
    -32;3;-6;10;3;-3;8;6;-3;10;6;-6;8;-32;-6;6;8;6;-6;10;6;-3;-4;-6;3;-32;6;...
    -6;-4;3;6;10;-3;6;8;-6;-32;6;3;-4;3;3;4;3;6;-4;-32;6;-6;-4;6;-3;10;-6;3;...
    -32;6;-6;8;-6;-6;10;-3;-32;-3;6;-4;-3;3;4;-32;-6;-6;8;6;6;-32;-6;-6;-4;...
    -3;-32;-6;-3;-4;-32;6;6;-32;-6;-32]+nu*[ 48;0;0;0;-24;-24;-12;0;-12;0;...
    24;0;0;0;24;-12;-12;0;-12;0;0;-12;12;12;48;0;24;0;0;0;-12;-12;-24;0;-24;...
    0;0;24;12;-12;12;0;-12;0;-12;-12;0;48;24;0;0;12;12;-12;0;24;0;-24;-24;0;...
    0;-12;-12;0;0;-12;-12;0;-12;48;0;0;0;-24;0;-12;0;12;-12;12;0;0;0;-24;...
    -12;-12;-12;-12;0;0;48;0;24;0;-24;0;-12;-12;-12;-12;12;0;0;24;12;-12;0;...
    0;-12;0;48;0;24;0;-12;12;-12;0;-12;-12;24;-24;0;12;0;-12;0;0;-12;48;0;0;...
    0;-24;24;-12;0;0;-12;12;-12;0;0;-24;-12;-12;0;48;0;24;0;0;0;-12;0;-12;...
    -12;0;0;0;-24;12;-12;-12;48;-24;0;0;0;0;-12;12;0;-12;24;24;0;0;12;-12;...
    48;0;0;-12;-12;12;-12;0;0;-12;12;0;0;0;24;48;0;12;-12;0;0;-12;0;-12;-12;...
    -12;0;0;-24;48;-12;0;-12;0;0;-12;0;12;-12;-24;24;0;48;0;0;0;-24;24;-12;...
    0;12;0;24;0;48;0;24;0;0;0;-12;12;-24;0;24;48;-24;0;0;-12;-12;-12;0;-24;...
    0;48;0;0;0;-24;0;-12;0;-12;48;0;0;24;0;24;0;-12;12;48;0;-24;0;12;-12;-12;...
    48;0;0;0;-24;-24;48;0;24;0;0;48;24;0;0;48;0;0;48;0;48 ] );% elemental stiffness matrix  #3D#
Ke0( tril( ones( 24 ) ) == 1 ) = Ke';
Ke0 = reshape( Ke0, 24, 24 );
Ke0 = Ke0 + Ke0' - diag( diag( Ke0 ) );                    % recover full matrix
%------------------------------------ PRE. 3) LOADS, SUPPORTS AND PASSIVE DOMAINS
lcDof = 3 * nodeNrs( 1 : nely + 1, 1, nelx + 1 );
fixed = 1 : 3 * ( nely + 1 ) * ( nelz + 1 );
[ pasS, pasV ] = deal( [], [] );                           % passive solid and void elements
F = fsparse( lcDof, 1, -sin((0:nely)/nely*pi)', [ nDof, 1 ] );% define load vector
free = setdiff( 1 : nDof, fixed );                         % set of free DOFs
act = setdiff( ( 1 : nEl )', union( pasS, pasV ) );        % set of active d.v.
%------------------------------------ PRE. 4) DEFINE IMPLICIT FUNCTIONS
prj = @(v,eta,beta) (tanh(beta*eta)+tanh(beta*(v(:)-eta)))./...
    (tanh(beta*eta)+tanh(beta*(1-eta)));                   % projection
deta = @(v,eta,beta) - beta * csch( beta ) .* sech( beta * ( v( : ) - eta ) ).^2 .* ...
    sinh( v( : ) * beta ) .* sinh( ( 1 - v( : ) ) * beta );% projection eta-derivative
dprj = @(v,eta,beta) beta*(1-tanh(beta*(v-eta)).^2)./(tanh(beta*eta)+tanh(beta*(1-eta)));% proj. x-derivative
cnt = @(v,vCnt,l) v+(l>=vCnt{1}).*(v<vCnt{2}).*(mod(l,vCnt{3})==0).*vCnt{4};
%------------------------------------------- PRE. 5) PREPARE FILTER
[dy,dz,dx]=meshgrid(-ceil(rmin)+1:ceil(rmin)-1,...
    -ceil(rmin)+1:ceil(rmin)-1,-ceil(rmin)+1:ceil(rmin)-1 );
h = max( 0, rmin - sqrt( dx.^2 + dy.^2 + dz.^2 ) );        % conv. kernel                #3D#
```

```
Hs = imfilter( ones( nely, nelz, nelx ), h, bcF );                    % matrix of weights (filter)  #3D#
dHs = Hs;
% ------------------------- PRE. 6) ALLOCATE AND INITIALIZE OTHER PARAMETERS
[ x, dsK, dV ] = deal( zeros( nEl, 1 ) );                             % initialize vectors
dV( act, 1 ) = 1/nEl/volfrac;                                         % derivative of volume
x( act ) = ( volfrac*( nEl - length(pasV) ) - length(pasS) )/length( act );% volume fraction on active set
x( pasS ) = 1;                                                        % set x = 1 on pasS set
[ xPhys, xOld, ch, loop, U ] = deal( x, 1, 1, 0, zeros( nDof, 1 ) );  % old x, x change, it. counter, U
% ============================================== START OPTIMIZATION LOOP
while ch > 1e-6 && loop < maxit
  loop = loop + 1;                                                    % update iter. counter
  % ---------- RL. 1) COMPUTE PHYSICAL DENSITY FIELD (AND ETA IF PROJECT.)
  xTilde = imfilter( reshape( x, nely, nelz, nelx ) ./ Hs, h, bcF );  % filtered field              #3D#
  xPhys( act ) = xTilde( act );                                       % reshape to column vector
  if ft > 1                                % compute optimal eta* with Newton
      f = ( mean( prj( xPhys, eta, beta ) ) - volfrac ) * (ft == 3);  % function (volume)
      while abs( f ) > 1e-6          % Newton process for finding opt. eta
          eta = eta - f / mean( deta( xPhys, eta, beta ) );
          f = mean( prj( xPhys, eta, beta ) ) - volfrac;
      end
      dHs = Hs ./ reshape( dprj( xPhys, eta, beta ), nely, nelz, nelx );  % sensitivity modification   #3D#
      xPhys = prj( xPhys, eta, beta );                                % projected (physical) field
  end
  ch = norm( xPhys - xOld ) ./ nEl;
  xOld = xPhys;
  % ------------------------- RL. 2) SETUP AND SOLVE EQUILIBRIUM EQUATIONS
  sK = ( Emin + xPhys.^penal * ( E0 - Emin ) );
  dsK( act ) = -penal * ( E0 - Emin ) * xPhys( act ) .^ ( penal - 1 );
  sK = reshape( Ke( : ) * sK', length( Ke ) * nEl, 1 );
  K = fsparse( Iar( :, 1 ), Iar( :, 2 ), sK, [ nDof, nDof ] );
  L = chol( K( free, free ), 'lower' );
  U( free ) = L' \ ( L \ F( free ) );                                 % f/b substitution
  % ------------------------------------- RL. 3) COMPUTE SENSITIVITIES
  dc = dsK .* sum( ( U( cMat ) * Ke0 ) .* U( cMat ), 2 );             % derivative of compliance
  dc = imfilter( reshape( dc, nely, nelz, nelx ), h, bcF ) ./ dHs;     % filter objective sens.     #3D#
  dV0 = imfilter( reshape( dV, nely, nelz, nelx ), h, bcF ) ./ dHs;    % filter compliance sens.    #3D#
  % ------------------- RL. 4) UPDATE DESIGN VARIABLES AND APPLY CONTINUATION
  xT = x( act );
  [ xU, xL ] = deal( xT + move, xT - move );                          % current upper and lower bound
  ocP = xT .* real( sqrt( -dc( act ) ./ dV0( act ) ) );               % constant part in resizing rule
  l = [ 0, mean( ocP ) / volfrac ];                                   % initial estimate for LM
  while ( l( 2 ) - l( 1 ) ) / ( l( 2 ) + l( 1 ) ) > 1e-4              % OC resizing rule
      lmid = 0.5 * ( l( 1 ) + l( 2 ) );
      x( act ) = max( max( min( min( ocP/lmid, xU ), 1 ), xL ), 0 );
      if mean( x ) > volfrac, l( 1 ) = lmid; else, l( 2 ) = lmid; end
  end
  [penal,beta] = deal(cnt(penal,penalCnt,loop), cnt(beta,betaCnt,loop));  % apply conitnuation on parameters
  % ------------------------- RL. 5) PRINT CURRENT RESULTS AND PLOT DESIGN
  fprintf( 'It.:%5i C:%6.5e V:%7.3f ch.:%0.2e penal:%7.2f beta:%7.1f eta:%7.2f lm:%0.2e \n', ...
      loop, F'*U, mean(xPhys(:)), ch, penal, beta, eta, lmid );
  isovals = shiftdim( reshape( xPhys, nely, nelz, nelx ), 2 );
  isovals = smooth3( isovals, 'box', 1 );
  patch(isosurface(isovals, .5),'FaceColor','b','EdgeColor','none');
  patch(isocaps(isovals, .5),'FaceColor','r','EdgeColor','none');
  drawnow; view( [ 145, 25 ] ); axis equal tight off; cla();
end
end
```

References

Amir O, Sigmund O (2011) On reducing computational effort in topology optimization: how far can we go? Struct Multidiscip Optim 44(1):25–29. https://doi.org/10.1007/s00158-010-0586-7

Amir O, Aage N, Lazarov BS (2014) On multigrid–CG for efficient topology optimization. Struct Multidiscip Optim 49(5):815–829. https://doi.org/10.1007/s00158-013-1015-5

Anderson DG (1965) Iterative procedures for nonlinear integral equations. J Assoc Comput Mach 12(4):547–560

Andreassen E, Andreasen CS (2014) How to determine composite material properties using numerical homogenization. Comput Mater Sci 83:488–495

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

Arora JS, Chahande AI, Paeng JK (1991) Multiplier methods for engineering optimization. Int J Numer Methods Eng 32(7):1485–1525

Bendsøe MP, Sigmund O (1999) Material interpolation schemes in topology optimization. Arch Appl Mech 69(9):635–654. https://doi.org/10.1007/s004190050248

Bourdin B (2001) Filters in topology optimization. Int J Numer Methods Eng 50(9):2143–2158. https://doi.org/10.1002/nme.116

Brezinski C, Chehab JP (1998) Nonlinear hybrid procedures and fixed point iterations. Numer Funct Anal Optim 19(5–6):465–487. https://doi.org/10.1080/01630569808816839

Bruns TE, Tortorelli DA (2001) Topology optimization of non-linear elastic structures and compliant mechanisms. Comput Methods Appl Mech Eng 190(26):3443–3459. https://doi.org/10.1016/S0045-7825(00)00278-4. http://www.sciencedirect.com/science/article/pii/S0045782500002784

Challis VJ (2010) A discrete level-set topology optimization code written in matlab. Struct Multidiscip Optim 41(3):453–464. https://doi.org/10.1007/s00158-009-0430-0

Christensen P, Klarbring A (2008) An introduction to structural optimization. Solid mechanics and its applications. Springer, Netherlands

Davis TA (2009) User guide for CHOLMOD: a sparse Cholesky factorization and modification package

Davis T (2019) Suitesparse: a suite of sparse matrix software. http://faculty.cse.tamu.edu/davis/suitesparse.html

Engblom S, Lukarski D (2016) Fast matlab compatible sparse assembly on multicore computers. Parallel Comput 56:1–17

Eyert V (1996) A comparative study on methods for convergence acceleration of iterative vector sequences. J Comput Phys 124(2):271–285. https://doi.org/10.1006/jcph.1996.0059

Fang HR, Saad Y (2009) Two classes of multisecant methods for nonlinear acceleration. Numer Linear Algebra Appl 16(3):197–221. https://doi.org/10.1002/nla.617

Ferrari F, Sigmund O (2020) Towards solving large-scale topology optimization problems with buckling constraints at the cost of linear analyses. Comput Methods Appl Mech Eng 363:112,911. https://doi.org/10.1016/j.cma.2020.112911

Ferrari F, Lazarov BS, Sigmund O (2018) Eigenvalue topology optimization via efficient multilevel solution of the frequency response. Int J Numer Methods Eng 115(7):872–892

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

Hestenes MR (1969) Multiplier and gradient methods. J Optim Theory Appl 4(5):303–320. https://doi.org/10.1007/BF00927673

Horn RA, Johnson CR (2012) Matrix analysis, 2nd edn. Cambridge University Press, New York

Li L, Khandelwal K (2015) Volume preserving projection filters and continuation methods in topology optimization. Engineering Stru 85:144–161

Li W, Suryanarayana P, Paulino G (2020) Accelerated fixed–point formulation of topology optimization: application to compliance minimization problems. Mech Rese Commun 103:103,469

Liu K, Tovar A (2014) An efficient 3d topology optimization code written in matlab. Struct Multidiscip Optim 50(6):1175–1196. https://doi.org/10.1007/s00158-014-1107-x

Peng Y, Deng B, Zhang J, Geng F, Qui W, Liu L (2018) Anderson acceleration for geometry optimization and physics simulation. ACM Trans Graph 37(4):42:1–42:14

Pratapa PP, Suryanarayana P, Pask JE (2016) Anderson acceleration of the jacobi iterative method: An efficient alternative to krylov methods for large, sparse linear systems. J Comput Phys 306:43–54. https://doi.org/10.1016/j.jcp.2015.11.018

Quarteroni A, Sacco R, Saleri F (2000) Numerical mathematics. Texts in applied mathematics. Springer

Ramiere I, Helfer T (2015) Iterative residual–based vector methods to accelerate fixed point iterations. Comput Math Appl 70:2210–2226

Saad Y (1992) Numerical methods for large eigenvalue problems. Manchester University Press

Sanders ED, Pereira A, Aguiló MA, Paulino GH (2018) Polymat: an efficient Matlab code for multi–material topology optimization. Struct Multidiscip Optim 58:2727–2759

Sigmund O (2001) A 99 line topology optimization code written in Matlab. Struct Multidiscip Optim 21(2):120–127. https://doi.org/10.1007/s001580050176

Sigmund O (2007) Morphology–based black and white filters for topology optimization. Struct Multidiscip Optim 33(4):401–424

Suresh K (2010) A 199–line Matlab code for Pareto–optimal tracing in topology optimization. Struct Multidiscip Optim 42(5):665–679

Talischi C, Paulino GH, Pereira A, Menezes IF (2012) Polytop: a matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes. Struct Multidiscip Optim 45(3):329–357. https://doi.org/10.1007/s00158-011-0696-x

Walker HF, Ni P (2011) Anderson acceleration for fixed point iterations. SIAM J Numer Anal 49(4):1715–1735

Wang MY (2007) Structural topology optimization using level set method. In: Computational methods in engineering & science. Springer, Berlin, pp 310–310

Wang F, Lazarov B, Sigmund O (2011) On projection methods, convergence and robust formulations in topology optimization. Struct Multidiscip Optim 43(6):767–784

Xia L, Breitkopf P (2015) Design of materials using topology optimization and energy-based homogenization approach in matlab. Struct Multidiscip Optim 52(6):1229–1241. https://doi.org/10.1007/s00158-015-1294-0

Xu S, Cai Y, Cheng G (2010) Volume preserving nonlinear density filter based on Heaviside functions. Struct Multidiscip Optim 41:495–505