# A parallel FreeFEM framework for topology optimization of structures into three spatial dimensions

J.M.M. Luz Filho , A.T.A. Gomes , and A.A. Novotny*

*Laboratório Nacional de Computação Científica LNCC/MCTI,*
*Coordenação de Métodos Matemáticos e Computacionais,*
*Av. Getúlio Vargas 333, 25651-075 Petrópolis - RJ, Brasil*
**Corresponding author: novotny@lncc.br*
*Contributing authors: jmorvan@lncc.br, atagomes@lncc.br*

**Abstract**

This article presents a simple and compact parallel FreeFEM implementation for topology optimization of structures into three spatial dimensions. The topology optimization algorithm relies on the topological derivative concept together with a level-set domain representation method, in which the topological derivative is used as a steepest descent direction evolving the level-set function within the optimization procedure. In addition, adaptive mesh refinement is performed in the optimization scheme for enhancing the boundary representation of the resulting structure and reducing the computational cost of structured mesh refinements. In addition, a rich and thorough discussion of the main aspects of the parallel FreeFEM implementation is given in full detail. In particular, a version of the resulting code is provided for the reader convenience, including two benchmark examples for structural stiffness maximization and for the design of compliant mechanisms, showing the simplicity and effectiveness of the proposed implementation. Besides, we also show how the resulting code can be easily converted into a SIMP-based topology design algorithm. Since all the background knowledge on the topological derivative method is presented and the complete FreeFEM implementation is provided as supplementary material, this paper may also be used as educational/pedagogical material for graduate students and/or newcomers to the field of topology optimization.

**Keywords:** Structural compliance minimization, design of compliant mechanisms, topological derivative method, parallel FreeFEM.

## 1 Introduction

Topology optimization of structures has been a topic of great interest and intense research for many decades. Since the seminal paper by Bendsøe and Kikuchi (1988), many different approaches have been proposed in the pursuit of new optimal and unintuitive designs for a variety of applications governed by different physics, such as solid and fluid mechanics, electromagnetism and others. Here, we present an overview of the main approaches in topology optimization, and the computational environments in which they have been implemented over the years. Let us start with the most widely used method within the density-based approaches in topology optimization, the penalized stiffness model written in terms of a density function known as Solid Isotropic Material with Penalization (SIMP) method. In this numerical scheme, for a fixed domain discretized by a mesh of finite elements, the variables to be optimized are the material densities associated with the finite elements (Bendsøe and Sigmund, 2003). In addition, the intermediate density values are penalized to enforce $0-1$ designs and in order to ensure the existence of solutions, the power-law approach must be combined with a perimeter constraint, a gradient constraint or with filtering techniques (Sigmund and Petersson, 1998). It is worth

mentioning that the pioneering educational paper by Sigmund (2001) with the famous 99 lines of topology optimization code written in Matlab relies on the SIMP method. Ten years later, Andreassen et al. (2011) proposed an 88-line topology optimization code to be the successor of the original code by Sigmund, extending it with a density filter and improving its efficiency by preallocating arrays and vectorizing loops. Another relevant paper with simple Matlab implementation for density-based approaches is the work by Liu and Tovar (2014), who proposes a compact code to solve three-dimensional topology optimization problems. The reader may refer to Wang et al. (2021) for a complete review on educational articles on structural and multidisciplinary optimization and Sigmund and Maute (2013) for an overview on topology optimization approaches.

Another major branch in topology optimization relies on the level-set method, which was first introduced by Osher and Sethian (1988) for moving interface problems. Its main feature is to enable an accurate description of the boundaries on a fixed mesh, which leads to fast numerical algorithms. In fact, the level-set method handles in a natural way topology changes. Level-set methods have been implemented in a variety of programming environments, such as FEMLAB (Liu et al., 2005), Matlab (Challis, 2010; Otomori et al., 2015), FEniCS (Laurain, 2018) and FreeFEM (Li et al., 2021). For a complete review of the level-set method, see the works by Allaire et al. (2021) and Van Dijk et al. (2013).

An interesting and rich branch within the topology optimization methods is the so-called topological derivative. The main feature of this approach is to nucleate or remove holes/inclusions in the interior of a geometrical domain according to the sign of the topological derivative field. A FreeFEM code for the topology optimization of two-dimensional structures based on the topological derivative together with a level-set domain representation method is presented in details by Filho et al. (2023). For the reader convenience, a Matlab version of this code can be found on the webpage `https://novotny.lncc.br/#matlab`, whereas a very similar topological derivative-based implementation for topology optimization is also available in the NGSolve programming environment on its Interactive NGSolve Tutorial webpage, `https://docu.ngsolve.org/latest/i-tutorials/index.html`. The reader may also find in this tutorial a whole section dedicated to the implementation of topology optimization algorithms, including a framework for fully and semi-automated shape differentiation (Gangl et al., 2021). For a comprehensive introduction to the topological derivative method, the reader may refer to the book by Novotny and Sokołowski (2020).

FreeFEM is an open-source program written in C++ for solving partial differential equations (PDEs) by the finite element method (Hecht, 2012). A defining feature of FreeFEM is its *embedded domain-specific language* (EDSL), a C++ idiom whose syntax bears great resemblance with the mathematical formulation of the problem of interest. This EDSL allows its users to write *scripts* containing only a few commands that easily handle the complex processes required for finite element analysis (Kim et al., 2020). In particular, this EDSL allows the implementation of three dimensional problems with almost the same simplicity of two dimensional implementations. To cater for more complex or computationally expensive problems, FreeFEM integrates with many external libraries, such as, ARPACK, METIS, MUMPS, Mmg, PETSc, Gmsh and Paraview, to cite a few. In the context of topology optimization, a considerable number of educational articles with implementations based on FreeFEM can be found in the literature, such as the works by Allaire and Pantz (2006); Kim et al. (2020); Filho et al. (2023). The work by Allaire and Pantz (2006) is the first educational paper for structural topology optimization using FreeFEM. It presents two classical structural optimization methods, namely, the Hadamard method for geometric optimization (Sokołowski and Zolésio, 1992) and the homogenization method for topology optimization (Allaire, 2002). On the other hand, in the work by Kim et al. (2020) the design algorithm relies on a reaction-diffusion equation-based topology optimization and adaptive mesh refinement for an enhanced boundary representation.

The work by Filho et al. (2023) presents a FreeFEM topological derivative-based sequential implementation for structural compliance minimization in two spatial dimensions with volume constraint imposed via linear penalization. As a natural sequence of our work, we propose here an extension of the computational framework from Filho et al. (2023) to parallel computing.

In addition to the structural stiffness maximization, we also consider the design of compliant mechanisms which leads to a non-self adjoint formulation. Finally, in contrast to Filho et al. (2023), both problems are here posed into three spatial dimensions and the volume constraint is efficiently imposed via the augmented Lagrangian method. For the state of the art on volume control methods, see for instance the paper by Cui et al. (2023).

In the parallel version of FreeFEM, the parallelism relies on the distributed memory paradigm. In this paradigm, a set of processes—each one running a copy of FreeFEM—are dispatched in different processors, exchanging data and synchronizing with each other by means of some inter-process communication technology. In the case of FreeFEM, the Message Passing Interface (MPI) standard is employed. These features are deeply explored for solving the elasticity system and the topological derivative filtering problem, which represent the computational bottlenecks in our particular case. Briefly, the central idea of the proposed implementation is to perform parallel computing for solving the PDEs posed into three spatial dimensions, whereas simple procedures, such as evaluating the topological derivative field, updating the level-set function and computing the cost functional, are sequentially performed on a single process. Therefore, the FreeFEM script reported here would be of particular relevance for those involved in the analysis and design of structures in general, independently of the specific topology optimization approach to be adopted.

The resulting parallel FreeFEM script for topology optimization of three dimensional structures developed in this work can be executed in any standard multi-core PCs as well as in supercomputer architectures. In particular, we have tested our implementation in the Santos Dumont (SDumont) supercomputing facility at LNCC in Brazil.[1] This facility comprises a hybrid configuration including different models of thin nodes, fat nodes, and nodes with GPUs. All of these nodes are interconnected through an Infiniband network. We ran our tests on thin nodes with the following configuration: 2× CPU Intel Xeon Cascade Lake Gold (24 cores each CPU) and 384GB RAM. To avoid the need for compiling the FreeFEM program on SDumont, we used a Docker container made available by the FreeFEM developers.[2]

The remainder of this paper is organized as follows. The problem formulation is introduced in Section 2. In Section 3 a brief introduction to the concept of the topological derivative is provided as well as the topological derivative formula associated with the shape functional of interest. The topology optimization algorithm based on the topological derivative and a level-set domain representation method is presented in Section 4. Section 5 provides a complete discussion on the main aspects of the FreeFEM implementation of the 3D topology optimization algorithm. The complete 3D topology optimization FreeFEM code is available as supplementary material (`elas3d.edp`) for the reader convenience. In order to show the simplicity and effectiveness of the proposed implementation, two benchmark examples are presented in Section 6. Finally, a straightforward conversion of the resulting code into a SIMP-based topology design algorithm is presented for the reader convenience, which in this context combines the density-based approach with a level-set domain representation method.

## 2 Formulation of a model problem in elasticity

Let us consider an open and bounded domain $\mathcal{D} \subset \mathbb{R}^3$ with Lipschitz boundary denoted as $\Gamma := \partial\mathcal{D}$. The boundary $\Gamma$ is the union of two given non-overlapping subsets $\Gamma_D$ and $\Gamma_N$, that is $\Gamma = \overline{\Gamma}_D \cup \overline{\Gamma}_N$ and $\Gamma_D \cap \Gamma_N = \varnothing$, where $\Gamma_D$ and $\Gamma_N$ are Dirichlet and Neumann boundaries, respectively. See sketch in Figure 1.

[1] `http://sdumont.lncc.br`

[2] `https://github.com/FreeFem/FreeFem-docker/releases`

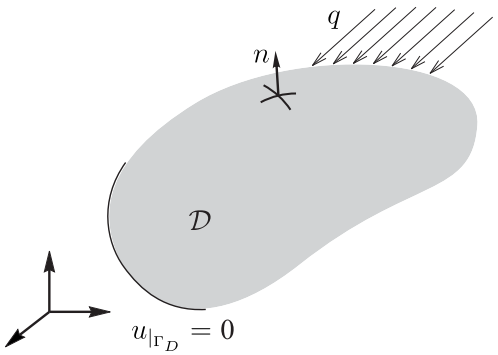



Figure 1: The elasticity problem.

Using the Einstein summation convention over repeated indices $i, j, k, l = \overline{1,3}$, we specifically have $u = u_i e_i$ for vectors and $U = U_{ij}(e_i \otimes e_j)$ for second order tensors, where $e_j$ is the unit vector in the (Cartesian) $j$th coordinate direction. The inner product between vectors $u, v$ and second order tensors $U, V$ are respectively defined as $u \cdot v = u_i v_i$ and $U \cdot V = U_{ij} V_{ij}$. In addition, $u_i = u \cdot e_i$ and $U_{ij} = e_i \cdot U e_j$ are the Cartesian components of $u$ and $U$, respectively. Finally, $\mathrm{I} = \delta_{ij} e_i \otimes e_j$ and $\mathbb{I} = (\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk})\ e_i \otimes e_j \otimes e_k \otimes e_l$, where $\delta_{ij}$ is the Kronecker delta.

We are interested in solving a topology optimization problem which consists in finding a subdomain $\Omega \subset \mathcal{D}$ that solves the following minimization problem:

$$\begin{cases} \underset{\Omega \subset \mathcal{D}}{\text{Minimize}}\ \mathcal{J}(u) \\ \text{subject to } g_\Omega \leq 0\ , \end{cases} \tag{2.1}$$

where $g_\Omega$ is the volume constraint conveniently written as

$$g_\Omega := \frac{|\Omega| - M}{M}, \tag{2.2}$$

with $M > 0$ the volume target. Now, let us define a tracking-type shape functional of the form

$$\mathcal{J}(u) = \int_{\Gamma_N} p \cdot u\ , \tag{2.3}$$

with $p$ a given vector function in $H^{-1/2}(\Gamma_N)$. The displacement vector field $u : \mathcal{D} \to \mathbb{R}^3$ is solution to the following variational problem: Find $u \in \mathcal{U}$, such that

$$\int_{\mathcal{D}} \sigma(u) \cdot \varepsilon(\eta) = \int_{\Gamma_N} q \cdot \eta\ , \quad \forall \eta \in \mathcal{V}\ , \tag{2.4}$$

with $q \in H^{-1/2}(\Gamma_N)$ a given boundary traction and $\sigma(u) = \rho \mathbb{C} \varepsilon(u)$, where $\rho$ is a distributed parameter $\rho : \mathcal{D} \mapsto \{1, \rho_0\}$ defined as

$$\rho(x) := \begin{cases} 1, & \text{if } x \in \Omega\,, \\ \rho_0, & \text{if } x \in \mathcal{D} \setminus \Omega\,, \end{cases} \tag{2.5}$$

with $0 < \rho_0 \ll 1$ representing a very weak phase used to mimic voids. For a given smooth vector function $\varphi : \mathcal{D} \to \mathbb{R}^3$, the linearized strain tensor is written as

$$\varepsilon(\varphi) = \frac{1}{2}\left(\nabla\varphi + (\nabla\varphi)^\top\right)\ . \tag{2.6}$$

In addition, the constitutive tensor is given by

$$\mathbb{C} = 2\mu\mathbb{I} + \lambda(\mathrm{I} \otimes \mathrm{I})\ , \tag{2.7}$$

where I and $\mathbb{I}$ are the second and fourth-order identity tensors, respectively. The Lamé's coefficients $\mu$ and $\lambda$ are considered constants everywhere and given by

$$\mu = \frac{E}{2(1+\nu)} \quad \text{and} \quad \lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}\ , \tag{2.8}$$

where $E$ is the Young's modulus and $\nu$ the Poisson's ratio. The spaces $\mathcal{U}$ and $\mathcal{V}$ are defined as:

$$\mathcal{U} = \mathcal{V} := \{\varphi \in H^1(\mathcal{D}) : \varphi_{|\Gamma_D} = 0\} \, . \tag{2.9}$$

We now introduce the auxiliary vector function $v : \mathcal{D} \mapsto \mathbb{R}^3$, which is solution of the following adjoint variational problem: Find $v \in \mathcal{V}$, such that

$$\int_{\mathcal{D}} \sigma(v) \cdot \varepsilon(\eta) = \int_{\Gamma_N} p \cdot \eta \quad \forall \eta \in \mathcal{V}. \tag{2.10}$$

**Remark 1.** *Note that, by taking $p = q$ in the tracking-type shape functional* (2.3)*, it becomes the so-called structural compliance shape functional. In addition, in this particular case, the problem becomes self-adjoint in the sense that after replacing $p$ by $q$ in the right-hand side of the adjoint equation* (2.10) *and comparing it with the state equation* (2.4)*, we conclude that $v = u$ for $p = q$, provided that $\mathcal{U} = \mathcal{V}$.*

## 3 Topological derivative method

The topological derivative is a relatively new concept originally designed to deal with topology optimization. It is defined as the first term of the asymptotic expansion of a given shape functional with respect to a small parameter that measures the size of singular domain perturbations, such as holes, inclusions, source-terms and cracks. This concept can naturally be used as a steepest-descent direction in an optimization process like in any method based on the gradient of the cost functional. Therefore, the topological derivative method has been successfully applied in many different fields such as image processing, inverse problems, multi-scale material design, and mechanical modeling including damage and fracture evolution phenomena. For a comprehensive overview of the most recent developments on the topological derivative method, the reader may refer to the collection of articles in the special issue *On the topological derivative and its applications in computational engineering* (Novotny et al., 2022), for instance. For the sake of completeness, the topological derivative of the tracking-type shape functional $\mathcal{J}(u)$ from (2.3) is stated in its closed form. Since we consider here a very weak material for mimicking voids, the topological derivatives are presented in their limit cases versions. Those limit cases represent the scenarios whenever a small portion of material is either removed or added to the design domain. See, for instance, the work by Filho and Novotny (2024).

**Theorem 2.** *The topological derivative of the tracking-type shape functional $\mathcal{J}(u)$ from* (2.3)*, with respect to the nucleation of a small spherical inclusion endowed with different material property from the background governed by the contrast parameter $\gamma$, is given by Filho and Novotny (2024)*

$$D_T\mathcal{J}(x) = \mathbb{P}_\gamma \sigma(u) \cdot \varepsilon(v)(x) \quad \forall x \in \Omega, \tag{3.1}$$

*where $\mathbb{P}_\gamma$ is the fourth order isotropic polarization tensor given by*

$$\mathbb{P}_\gamma = (1-\gamma)[3\alpha_2 \mathbb{I} + (\alpha_1 - \alpha_2)\mathrm{I} \otimes \mathrm{I}], \tag{3.2}$$

*with the constant parameters $\alpha_1$ and $\alpha_2$ given by*

$$\alpha_1 = \frac{1-\nu}{3(1-\nu) - (1+\nu)(1-\gamma)} \quad \textit{and} \quad \alpha_2 = \frac{5(1-\nu)}{15(1-\nu) - (8-10\nu)(1-\gamma)} \, . \tag{3.3}$$

*We are interested in two particular limit cases, which are:*
***Case A**. Let us consider $x \in \Omega$. In this case, a small portion of material is removed from $\Omega$, namely $\gamma \to 0$. Then the topological derivative $D_T\mathcal{J}$ reads*

$$D_T\mathcal{J} = \mathbb{P}_0 \sigma(u) \cdot \varepsilon(v) \, , \tag{3.4}$$

*with the polarization tensor $\mathbb{P}_0$ written as*

$$\mathbb{P}_0 = \frac{3}{2}\frac{1-\nu}{7-5\nu}\left(10\mathbb{I} - \frac{1-5\nu}{1-2\nu}\mathrm{I} \otimes \mathrm{I}\right) \, . \tag{3.5}$$

***Case B***. *Let us consider $x \in \mathcal{D} \setminus \Omega$. In this case, a small portion of material is added within $\mathcal{D} \setminus \Omega$, namely $\gamma \to \infty$. Then the topological derivative $D_T\mathcal{J}$ is given by*

$$D_T\mathcal{J} = \mathbb{P}_\infty \sigma(u) \cdot \varepsilon(v) \,, \tag{3.6}$$

*with the polarization tensor $\mathbb{P}_\infty$ written as*

$$\mathbb{P}_\infty = -\frac{3}{2}\frac{1-\nu}{4-5\nu}\left(5\mathbb{I} + \frac{1-5\nu}{1+\nu}\mathrm{I}\otimes\mathrm{I}\right) \,. \tag{3.7}$$

In order to solve the constrained optimization problem addressed in (2.1), an external penalization method is employed herein aiming to fulfill the required volume target. More precisely, an augmented Lagrangian scheme is applied to (2.1) leading to the following equivalent unconstrained topology optimization problem (Campeão et al., 2014; Ferrer et al., 2018).

$$\underset{\Omega\subset\mathcal{D}}{\text{Minimize}}\; J^\beta(\Omega) := J(\Omega) + \alpha g_\Omega^+ + \frac{1}{2}\beta {g_\Omega^+}^2 \,, \tag{3.8}$$

where $J(\Omega) = \mathcal{J}(u)/\mathcal{J}(u_0)$, with $u$ and $u_0$ solutions to (2.4) for $\Omega \subset \mathcal{D}$ and $\Omega \equiv \mathcal{D}$, respectively. In addition, the function $g_\Omega^+$ is defined as

$$g_\Omega^+ := \max\{g_\Omega; -\alpha/\beta)\} \tag{3.9}$$

with $\alpha$ and $\beta$ positive parameters. Note that, if $g_\Omega^+ = -\alpha/\beta$, $D_T g_\Omega^+ = 0$. Otherwise, if $g_\Omega^+ = g_\Omega$, $D_T g_\Omega^+ = D_T g_\Omega$. Finally, the parameter $\beta$ is fixed and $\alpha$ is updated according to the following rule

$$\alpha \leftarrow \max\{0; \alpha + \beta g_\Omega\}. \tag{3.10}$$

# 4 Topology optimization algorithm

In this section, we provide a brief description of the topology optimization algorithm based on the topological derivative combined with a level-set domain representation method as proposed by Amstutz and Andrä (2006). The idea basically consists in achieving a local optimality condition for the minimization problem (3.8), given in terms of the topological derivative and a level-set function. Furthermore, the domain $\Omega \subset \mathcal{D}$ and the complement $\mathcal{D} \setminus \Omega$ are characterized by a level-set function $\Psi$ of the form

$$\Omega = \{x \in \mathcal{D} : \Psi(x) < 0\} \quad \text{and} \quad \mathcal{D} \setminus \Omega = \{x \in \mathcal{D} : \Psi(x) > 0\} \,, \tag{4.1}$$

where $\Psi$ vanishes on the interface between $\Omega$ and $\mathcal{D}\setminus\Omega$. A local sufficient optimality condition for problem (3.8), under the considered class of domain perturbation given by spherical inclusions, can be stated as

$$D_T J^\beta(x) > 0 \quad \forall x \in \mathcal{D} \,. \tag{4.2}$$

Let us define the topological gradient as a single function that measures the sensitivity of $J(\Omega)$ with respect to an oriented topology change as

$$G_T(x) := \begin{cases} -D_T J(x), & \text{if } \Psi(x) < 0 \,, \\ +D_T J(x), & \text{if } \Psi(x) > 0 \,. \end{cases} \tag{4.3}$$

The topological gradient to be used as steepest descent direction of the shape functional $J^\beta(\Omega)$ is then defined as

$$G_T^\beta(x) := G_T(x) + \frac{1}{M}\max\{0; \alpha + \beta g_\Omega\}. \tag{4.4}$$

Therefore, the optimality condition (4.2) can be conveniently rewritten as follows

$$\begin{cases} G_T^\beta(x) < 0, & \text{if } \Psi(x) < 0 \,, \\ G_T^\beta(x) > 0, & \text{if } \Psi(x) > 0 \,. \end{cases} \tag{4.5}$$

Note that (4.5) is satisfied whenever the quantity $G_T^\beta$ coincides with the level-set function $\Psi$ up to a strictly positive factor, namely $\exists\, \tau > 0 : G_T^\beta = \tau\Psi$, or equivalently

$$\theta := \arccos\left[\frac{\langle G_T^\beta, \Psi\rangle_{L^2(\mathcal{D})}}{\|G_T^\beta\|_{L^2(\mathcal{D})}\|\Psi\|_{L^2(\mathcal{D})}}\right] = 0\ , \tag{4.6}$$

which shall be used as the optimality condition in the topology design algorithm, where $\theta$ is the angle between the functions $G_T^\beta$ and $\Psi$ in $L^2(\mathcal{D})$. From all the elements presented so far, we now provide a simple explanation of the topology optimization algorithm.

1. We start by choosing an initial level-set function $\Psi_0$, namely

$$\Psi_0 : \|\Psi_0\|_{L^2(\mathcal{D})} = 1\ . \tag{4.7}$$

2. For a generic iteration $i$, we compute the function $G_T^\beta$ associated with $\Psi_i$, which is denoted by $G_i^\beta$. Then, the new level-set function $\Psi_{i+1}$ is updated according to the following linear combination of the functions $G_i^\beta$ and $\Psi_i$

$$\Psi_{i+1} = \frac{1}{\sin\theta_i}\left[\sin((1-w)\theta_i)\Psi_i + \sin(w\theta_i)\frac{G_i^\beta}{\|G_i^\beta\|_{L^2(\mathcal{D})}}\right] \quad \forall i \in \mathbb{N}\ , \tag{4.8}$$

   where $\theta_i$ is the angle between $G_i^\beta$ and $\Psi_i$, and $w$ is a step size determined by a linear search performed to decrease the value of the objective function $J_i^\beta$ associated with $\Psi_i$.

3. The process continues until $\theta_i \leq \epsilon_\theta$ or $w \leq \epsilon_w$, where $\epsilon_\theta > 0$ and $\epsilon_w > 0$ are small numerical tolerances for the optimality condition and for the line-search step size $w$, respectively. Whenever the stopping criterion is fulfilled ($w \leq \epsilon_w$), adaptive mesh refinement may be performed and the iterative process is restarted on the new discretized domain. Otherwise, if the optimality condition holds true ($\theta_i \leq \epsilon_\theta$), the level-set function associated with the local optimizer $\Omega^\star$ is set as $\Psi^\star = \Psi_i$.

# 5 Parallel FreeFEM computational implementation

The topology optimization algorithm presented in Section 4 is implemented as a script in the FreeFEM EDSL. The code is available as supplementary material for the reader convenience (`elas3d.edp`). Here, we present a rich and thorough discussion of the main aspects of the implementation. Before proceeding to the implementation itself and for the sake of completeness, it is convenient to provide the reader some basic information about the support of FreeFEM to parallelism.

## 5.1 FreeFEM parallel version

The task of turning a sequential script into parallel in FreeFEM can be performed by using the parallel version of the FreeFEM program. This version offers a variety of domain decomposition methods, some of them integrated with the Portable, Extensible Toolkit for Scientific Computation (PETSc) library.[3] When using PETSc, FreeFEM is responsible for the distributed discretization of PDEs using domain partitioning, and PETSc is responsible for solving the resulting (distributed) linear systems. Since PETSc relies on MPI for inter-process communication, and some parts of the topology optimization algorithm are sequential, some additional MPI routines for data distribution and synchronization are also used. In the following we briefly discuss the use of these technologies on FreeFEM.

[3] `http://petsc.org`

### 5.1.1 PETSc

This toolkit offers a large suite of linear solvers, including parallel and sequential, direct and iterative, which are accessible to FreeFEM scripts with the use of the `load "PETSc"` statement. In this work we explore Krylov methods and associated preconditioners and their usage within FreeFEM as part of a domain decomposition procedure. In simple terms, domain decomposition aims to solve a boundary value problem by splitting it into smaller boundary value problems on subdomains. The parallel version of FreeFEM offers a variety of tools for performing domain decomposition and all the necessary data exchange between subdomains, some of them being accessible to FreeFEM scripts with the use of the `include "macro_ddm.idp"` statement. In the implementation of our topology optimization algorithm using the parallel FreeFEM version, this statement is used together with `load "PETSc"`, which means that domain decomposition is used as part of a preconditioner to a chosen Krylov method. For more details on the use of domain decomposition methods within FreeFEM, the reader may refer to the works by Jolivet et al. (2012) and Sadaka et al. (2020).

### 5.1.2 MPI

This standard addresses primarily the message-passing parallel programming model, in which data are moved between the address space of different processes by means of communication operations executed by each process on a certain *communicator*. A communicator specifies a communication scope for the communication operations; in the implementation of our topology optimization algorithm using the parallel FreeFEM version, the communication between processes is executed through the default communicator to which all processes participate (`MPICommWorld`). This communicator is used for three main operations in the sequential parts of the topology optimization algorithm: (i) *broadcast* data from one process to all the other processes; (ii) *reduce* a set of values—each one of them stored in a different process—into a single value via a function (in the case of this work, typically a sum); (iii) synchronize the processes in *barriers* so that computation only proceeds when all processes reach a same barrier.

## 5.2 Description of the parallel FreeFEM implementation

At the beginning of the implementation we load the plugins which will be used along the code. The `Cube.idp` plugin is for the 3D mesh generation with tetrahedral elements, while `mmg` and `mshmet` are used for performing adaptive mesh refinement. The implementation is organized in eight parts, described in the following subsections.

### 5.2.1 Setting material properties, applied loads and parameters

The first lines of code are dedicated to set the material properties, namely the Young modulus `E`, Poisson's ratio `nu` and the small parameter used to mimic voids `rho0`. The components of the applied external load `qx, qy` and `qz` as well as the optimization parameters and the coefficients to compute the topological derivative are also declared and assigned.

### 5.2.2 Mesh generation

FreeFEM offers a simple way to generate cubic meshes through the function `Cube` contained in the plugin `cube.idp`. The syntax for creating a 3D mesh with tetrahedral elements is the following:

```
mesh3 Th = Cube(Nxyz, Bxyz, Lxyz);
```

where `Nxyz`, `Bxyz` and `Lxyz` are arrays containing information about the grid size of the discretized cube, the position of each face of the cube and the associated boundary labels, respectively. For example, the following lines of code are all that is necessary to generate a $10 \times 10 \times 10$

cubic grid of $0.5 \times 0.5 \times 0.5\ m^3$ dimension:

```
int[int] Nxyz = [10, 10, 10];
real[int, int] Bxyz = [[0.0, 0.5], [0.0, 0.5], [0.0, 0.5]];
int[int, int] Lxyz = [[1,2],[3,4],[5,6]];
mesh3 Th = Cube(Nxyz, Bxyz, Lxyz);
```

The input parameters of `Nxyz` is the number of grid divisions for each canonical direction. In `Bxyz` the input parameters are spatial coordinates of the cube faces and `Lxyz` is used for assigning the boundary label associated with each of these faces.

### 5.2.3 Macros

This is a very powerful/useful feature of FreeFEM. The use of macros simplifies the implementation by making names shorter and also avoids runtime overheads. In simple terms, whenever a macro name is encountered by the compiler, it replaces the name by the definition of the macro. The macros declared in the source code are listed and described in Table 1.

Table 1: Description of macros

| Macro | Description |
|---|---|
| `defS(i), defV(i)` | scalar and vector field definition, respectively |
| `initS(i), initV(i)` | scalar and vector field initialization, respectively |
| `e11(u)`, $\cdots$, `e33(u)` | components of the strain tensor |
| `s11(u)`, $\cdots$, `s33(u)` | components of the stress tensor |
| `div(u)` | divergent operator |
| `trs(u)` | trace of the stress tensor |
| `se(u,v)` | inner product of stress and strain |
| `gte(u,v)` | topological gradient in the bulk phase |
| `gti(u,v)` | topological gradient in the weak phase |

The macros described in Table 1 are written in FreeFEM syntax as follows.

```
macro defV(i) [i, i#y, i#z] //EOM
macro initV(i)[i,i,i] //EOM
macro defS(i)i //EOM
macro initS(i)i //EOM
macro e11(u) (dx(u)) // EOM
macro e22(u) (dy(u#y)) // EOM
macro e33(u) (dz(u#z)) // EOM
macro e12(u) ((dy(u) + dx(u#y))/2.0) // EOM
macro e13(u) ((dz(u) + dx(u#z))/2.0) // EOM
macro e23(u) ((dz(u#y) + dy(u#z))/2.0) // EOM
macro s11(u) (rho*(la*div(u) + 2.0*mu*e11(u))) // EOM
macro s22(u) (rho*(la*div(u) + 2.0*mu*e22(u))) // EOM
macro s33(u) (rho*(la*div(u) + 2.0*mu*e33(u))) // EOM
macro s12(u) (rho*(2.0*mu*e12(u))) // EOM
macro s13(u) (rho*(2.0*mu*e13(u))) // EOM
macro s23(u) (rho*(2.0*mu*e23(u))) // EOM
macro div(u) (dx(u) + dy(u#y) + dz(u#z)) // EOM
macro trs(u) (s11(u) + s22(u) + s33(u)) // EOM
macro se(u,v) (s11(u)*e11(v) + s22(u)*e22(v) + s33(u)*e33(v) +
```

```
      2.0*(s12(u)*e12(v) + s13(u)*e13(v) + s23(u)*e23(v))) // EOM
macro gte(u) (coef1e*se(u,u) + coef2e*trs(u)*div(u)) // EOM
macro gti(u) (coef1i*se(u,u) + coef2i*trs(u)*div(u)) // EOM
```

### 5.2.4 Domain decomposition and matrices declaration

In this part of the code, the domain decomposition is performed and the parallel matrices for solving the elasticity system and the topological derivative smoothing problem are declared and distributed among the processes. The function `buildDmesh` (where `D` stands for distributed) made available by the `include "macro_ddm.idp"` statement is used for the domain partitioning. The domain is decomposed into the number of MPI processes. Since we are dealing with vector (for the displacement field) and scalar (for the topological derivative field) finite element spaces, the matrices associated with the elasticity system `A` and the filtering problem `F` must be declared as:

```
Mat A, F;
{
   macro def(i)defV(i) //EOM
   macro init(i) initV(i) //EOM
   createMat(ThL, A, [P1, P1, P1]);
}
{
   macro def(i)defS(i) //EOM
   macro init(i) initS(i) //EOM
   createMat(ThL, F, P1);
}
```

Note that the PETSc parallel matrices `A` and `F` must be declared outside the brackets. In addition, the parallel matrices are distributed through the interaction between FreeFEM and PETSc with the help of the `createMat` routine from `macro_ddm.idp`.

### 5.2.5 Finite element spaces and functions

Finite element spaces as well as the associated finite element functions are declared in this section of the implementation. The scalar finite element spaces `Sh0G` and `Sh1G` are declared for the approximated topological derivative field and level-set function, respectively. In addition, the vector finite element space `Vh1G` is stated for the approximated global displacement field `u`. Note that all these spaces are defined in the discretized computational global domain `ThG`. Similarly, for the local computational domain `ThL`, the scalar finite element space `Sh1L` and the vector finite element space `Vh1L` are declared for smoothing the topological derivative field and solving the elasticity system in a parallel fashion. Table 2 presents the FreeFEM syntax for declaring the finite element spaces mentioned above and used in the present topology optimization code. Lastly, the finite element functions associated with each finite element space are summarized in Table 3.

### 5.2.6 The elasticity problem

In order to implement a boundary value problem in FreeFEM, its variational formulation must be known. In this sense, let us recall the elasticity problem given by (2.4) with the appropriate spaces (2.9). Then, with the help of the `macro se` for the stress and strain inner product, the elasticity variational problem (2.4) is easily translated into the FreeFEM language as:

Table 2: Description of the finite element spaces

| Finite element space | Finite element type |
|---|---|
| `fespace Sh0G(ThG,P0);` | `P0`: piecewise constant discontinuous finite element |
| `fespace Sh1G(ThG,P1);`<br>`fespace Sh1L(ThL,P1);`<br>`fespace Vh1G(ThG,[P1,P1,P1]);`<br>`fespace Vh1L(ThL,[P1,P1,P1]);` | `P1`: piecewise linear continuous finite element |

Table 3: Description of the global finite element functions

| Finite element space | Finite element function | Description |
|---|---|---|
| `Sh0G` | `chi` | P0 characteristic function |
| | `rho` | P0 material distribution |
| | `gt` | P0 topological gradient |
| `Sh1G` | `psi` | P1 level-set function |
| | `psiold` | level-set function from previous iteration |
| | `pchi` | P1 characteristic function |
| | `prho` | P1 material distribution |
| | `pgt` | P1 topological gradient |
| `Vh1G` | `u,v` | P1 vector displacement field |

```
varf elast(defV(uL),defV(etaL)) = int3d(ThL)(se(uL,etaL))
   + int2d(ThL,2)(qinx*etaL + qiny*etaLy + qinz*etaLz)
   + int2d(ThL,3)(qoutx*etaL + qouty*etaLy + qoutz*etaLz) + bc(uL);
```

Note that the elasticity system is to be solved on the local computational domain `ThL`, and `uL` is the local displacement field. In addition, the boundary conditions are imposed with the help of `macro bc(uL)`. Finally, we can also observe here the resemblance of the computational language and the mathematical formulation of the problem of interest, a remarkable feature of FreeFEM.

### 5.2.7 Smoothing of the topological derivative field

The topological derivative field of the tracking-type shape functional is evaluated by considering the limit cases presented in Section 3. The theoretical results are summarized through Theorem 2, which are given by the expressions (3.4) and (3.6) for Cases A and B, respectively. The associated topological gradient from (4.3) for Case A is evaluated in FreeFEM as `gte(u,v)`, where the coefficients `coef1e` and `coef2e` are given by:

```
coef0e = (3.0/2.0)*((1.0 - nu)/(5.0*nu - 7.0));
coef1e = coef0e*10.0;
coef2e = coef0e*((5.0*nu - 1.0)/(1.0 - 2.0*nu));
```

assigning the components of the polarization tensor $\mathbb{P}_0$ given in (3.5). Analogously, the topological gradient (4.3) for Case B is evaluated in FreeFEM as `gti(u,v)`, with the coefficients `coef1i` and `coef2i` written as:

```
coef0i = (3.0/2.0)*((1.0 - nu)/(5.0*nu - 4.0));
coef1i = 5.0*coef0i;
coef2i = coef0i*((1.0 - 5.0*nu)/(1.0 + nu));
```

assigning the components of the polarization tensor $\mathbb{P}_\infty$ given in (3.5).

According to Theorem 2, the topological derivative of the shape functional $\mathcal{J}(u)$ defined in (2.3) is written in terms of the gradient of the direct $u$ and adjoint $v$ displacement fields. Since the fields $u$ and $v$ are approximated by the vector finite element space `Vh1G` of `P1` elements, the topological derivative is then defined in the approximate finite element space `Sh0G` of `P0` elements. Both the topological derivative and the level-set function must be defined on the same finite element space for a more robust implementation from a numerical point of view. Since FreeFEM interpolation operator from a finite element space to another is only suitable for continuous finite functions, we adopt the same strategy as in Oliver et al. (2019) and Filho et al. (2023) to deal with the projection of the topological derivative into the space `Sh1G`. The idea consists in applying a regularization procedure over the topological derivative discontinuous field. The regularized topological derivative field is then solution of the following variational problem: Find $\varphi \in \mathcal{P}$, such that

$$\int_{\mathcal{D}} (\texttt{eps}^2\, \nabla\varphi \cdot \nabla\eta + \varphi\eta) = \int_{\mathcal{D}} G_T^\beta\, \eta\,, \quad \forall \eta \in \mathcal{Q}\,, \tag{5.1}$$

where $\mathcal{P} = \mathcal{Q} := H^1(\mathcal{D})$. The variational problem given in (5.1) is written in the FreeFEM language as:

```
varf filter(pgtL,etaSL)
   = int3d(ThL)((eps^2)*(dx(pgtL)*dx(etaSL) + dy(pgtL)*dy(etaSL)
   + dz(pgtL)*dz(etaSL)) + (pgtL*etaSL)) + int3d(ThL)(gt*etaSL);
```

where `eps` is a parameter associated with the smallest edge size of the global mesh `ThG`. Once again, the problem is solved on the local subdomain `ThL`. Even though the use of such a filtering procedure may be seen as a drawback from the computational point of view (as it requires solving an additional partial differential equation), some control over the complexity of the optimal topology can be achieved through the small parameter `eps`. Typically, small `eps` values lead to a final topology of high complexity, which tends to decrease for larger values of the filtering parameter.

### 5.2.8 Optimization process

Finally, the last part of the code consists in implementing the topology optimization algorithm described in Section 4. As previously mentioned, except for the algebraic solving of the elasticity and topological derivative filtering problems, the algorithm is performed sequentially on a single process. In simple terms, for a generic iteration $i$, we compute the associated topological gradient $G_T^i$ and the angle $\theta_i$ according to (4.3) and (4.6), respectively. The next step is to perform the line search for the step size $w$ in order to decrease the value of the shape functional $J^\beta(\Omega)$. The level-set function is updated according to expression (4.8). The shape functional is re-evaluated and we make $w \leftarrow w/2$. This line search procedure is repeated until `jBeta - jBetaold < numtol` or `w < tolw` is fulfilled, with `numtol` a small numerical tolerance.

```
if(mpirank == 0){
   iter = iter + 1;
   if(problemType == 0)
      v[]= u[];
```

```
   gt = chi*gte(u,v) + (1.0 - chi)*gti(u,v);
}
broadcast(processor(0), iter); broadcast(processor(0), gt[]);
F = filter(Sh1L, Sh1L); bf[] = filter(0, Sh1L);
set(F, sparams = petscArgsF);
pgtL[] = F^-1*bf[];
IdxScalar = restrict(Sh1L, Sh1G, N2O);
changeNumbering(F, pgtL[], locSolS);
changeNumbering(F, pgtL[], locSolS, inverse = true);
pgtR[](IdxScalar) = pgtL[];
mpiAllReduce(pgtR[], pgt[], mpiCommWorld, mpiSUM);
if(mpirank == 0){
   gtV = max(0.0, (alpha + beta*jV));
   pgt = pgt/jD + gtV/M;
   gtnorm = sqrt(int3d(ThG)(pgt^2)); pgt = pgt/gtnorm;
   cosin = max(min(int3d(ThG)(pgt*psi), 1.0), -1.0);
   theta = max(real(acos(cosin)), 1.0e-4);
   jBetaold = jBeta; psiold = psi; jOmegaold = jOmega;
   jBeta = jBeta + 1.0;
   w = min(1.0, 1.5*w);
}
broadcast(processor(0), jBeta); broadcast(processor(0), jBetaold);
broadcast(processor(0), w); mpiBarrier(mpiCommWorld);
while(jBeta - jBetaold > numtol){
   if(w < tolw/2.0){
      psi = psiold; jBeta = jBetaold; jOmega = jOmegaold;
      break;
   }
   mpiBarrier(mpiCommWorld);
   if(mpirank == 0){
      psi = (sin((1.0 - w)*theta)*psiold + sin(w*theta)*pgt)/sin(theta);
      psinorm = sqrt(int3d(ThG)(psi^2));
      psi = psi/psinorm;
      prho = (psi < 0) + rho0*(psi >= 0); rho = prho;
   }
   broadcast(processor(0), rho[]);
   A = elast(Vh1L, Vh1L, tgv = -2); b[] = elast(0, Vh1L, tgv = -2);
   set(A, sparams = petscArgs, bs = 3, nearnullspace = Rb);
   uL[] = A^-1*b[];
   IdxVector = restrict(Vh1L, Vh1G, N2O);
   changeNumbering(A, uL[], locSol);
   changeNumbering(A, uL[], locSol, inverse = true);
   uR[](IdxVector) = uL[];
   mpiAllReduce(uR[], u[], mpiCommWorld, mpiSUM);
   if(mpirank == 0){
      jOmega = int2d(ThG,2)(qinx*u + qiny*uy + qinz*uz) +
         int2d(ThG,3)(kappa*(qoutx*u + qouty*uy + qoutz*uz));
      pchi = (psi < 0); chi = pchi; VolOmega = int3d(ThG)(chi);
      jV = (VolOmega - M)/M;
      gp = max(jV,-alpha/beta);
      jBeta = jOmega/jD + alpha*gp + 0.5*beta*gp^2;
      w = w/2.0;
   }
```

```
    broadcast(processor(0), jBeta); broadcast(processor(0), w);
    broadcast(processor(0), theta);
}
    if(problemType == 1){
       Aadj = adjoint(Vh1L, Vh1L, tgv = -2);
       badj[] = adjoint(0, Vh1L, tgv = -2);
       set(Aadj, sparams = petscArgs, bs = 3, nearnullspace = Rb);
       vL[] = Aadj^-1*badj[];
       IdxVector = restrict(Vh1L, Vh1G, N2O);
       changeNumbering(Aadj, vL[], locSol);
       changeNumbering(Aadj, vL[], locSol, inverse = true);
       vR[](IdxVector) = vL[];
       mpiAllReduce(vR[], v[], mpiCommWorld, mpiSUM);
    }
    w = 2.0*w;
    if(mpirank == 0){
       if(abs(jV) > tolM){
          if(jV > 0)
             alpha = alpha + tau/beta*(max(0.0, alpha + beta*jV) - alpha);
          else
             alpha = max(0.0, alpha + beta*jV);
          gp = max(jV,-alpha/beta);
          jBeta = jOmega/jD + alpha*gp + 0.5*beta*gp^2;
       }
    }
    broadcast(processor(0), jBeta);
    mpiBarrier(mpiCommWorld);
```

Since the problems for filtering the topological derivative field and solving the elasticity system are performed in a parallel fashion, it is necessary to collect the solutions associated to each local subdomain into the global mesh. Some special attention is required in this step due to the existence of ghost elements in the overlapping regions between subdomains. The macro `ThLN2O` plays a fundamental role in this procedure as it preserves the connectivity between the local subdomains and the global mesh. Briefly, the summation of the local solutions over the global mesh is performed without any issues once the routine `changeNumbering` sets the solution to zero on overlapping regions which are not locally owned by the process, i.e. there are not duplicated values on ghost elements. In addition, in line `set(A, sparams = petscArgs);` the PETSc solver options for the elasticity system are set as:

```
string petscArgs = "-pc_type gamg -ksp_type gmres -ksp_pc_side right
-ksp_converged_reason";
```

where we have chosen a generalized minimal residual method (GMRES) iterative method together with an algebraic multigrid preconditioner, based on the recommendation from the PETSc documentation.[4]. For the problem of filtering the topological derivative field, the PETSc solver options are set through `set(F, sparams = petscArgsF);` with

[4] `https://petsc.org/main/manual/performance/#tips-for-efficient-use-of-linear-solvers`

```
string petscArgsF = "-pc_type gamg -ksp_type gmres -ksp_converged_reason";
```

If at some iteration, the line-search step size is smaller than the tolerance `tolw`, an adaptive mesh refinement is sequentially performed and the iterative process is carried on with the new mesh. The use of adaptive mesh refinement aims not only to enhance the resolution of the boundary representation but also to reduce the high computational cost of structured mesh refinements over the hold-all domain. The mesh adaptation procedure basically consists in refining the mesh only in the region adjacent to the design domain $\Omega$ obtained in the previous iteration, while the weak region $\mathcal{D} \setminus \Omega$ remains with coarse elements. For such, we rely on the FreeFEM interface to the open-source library `Mmg`. First `mshmet` is used to save the metric `met` for the mesh `ThG` and characteristic function `chi`. Then `mmg3d` is called in order to read the metric and generate the new adapted mesh.

```
if(mpirank == 0){
   real[int] met = mshmet(ThG, chi, hmin = Hmin, hmax = Hmax, nbregul = 20);
   ThL = mmg3d(ThG, metric=met, hausd = 0.008, hgrad = 1.0);
   ThG = ThL;
}
broadcast(processor(0), ThL); broadcast(processor(0), ThG);
```

Finally, the input parameters for the `mshmet` and `mmg3d` routines are summarized as follows:

- `metric`: array to set or get metric data information;
- `hmin`, `hmax`: minimum and maximum edge size, respectively;
- `hausd`: Hausdorff parameter to control boundary approximation, set as 0.01 by default;
- `hgrad`: parameter to set the gradation level, which controls the ratio between two adjacent edges. With a gradation of $h$, two adjacent edges $l_1$ and $l_2$ must respect that $1/h \leq l_1/l_2 \leq h$.

# 6 Numerical experiments

In this section, two benchmark examples are proposed in order to show the effectiveness of the topology optimization algorithm. The first example is given by a structural stiffness maximization problem, whereas the second one consists in the synthesis of a compliant mechanism. In both cases, the material properties are set as $E = 1$ and $\nu = 0.2$ for the Young modulus and Poisson's ratio, respectively, and the contrast phase parameter is set as $\rho_0 = 1.0 \times 10^{-2}$. The linear penalization parameter $\alpha$ from the augmented Lagrangian method is initialized as $\alpha = 0$ and the quadratic one is set as $\beta = 0.2$ for both examples. The following boundary labels were adopted in order to impose the desired boundary conditions:

1. Homogeneous Dirichlet boundary condition;
2. Non-homogeneous Neumann boundary condition ($\Gamma_{in}$);
3. Non-homogeneous Neumann boundary condition ($\Gamma_{out}$);
4. Homogeneous Neumann boundary condition;
5. symmetry boundary condition on plane $x = 0$;

6. symmetry boundary condition on plane $y = 0$;

7. symmetry boundary condition on plane $z = 0$.

An important remark about FreeFEM in 3D is that all boundary conditions are taken on a surface rather than on degrees of freedom. In addition, the flag `ProblemType = 0` is used to assign the structural stiffness maximization application whereas `ProblemType = 1` refers to the problem of designing compliant mechanisms. It is worth mentioning that this flag is introduced in the code just to avoid the unnecessary calculation of the adjoint state in the case of compliance minimization (see Remark 1). Otherwise, the code flux would be unique, since all we have to do is set $p$ from (2.3) accordingly, which will be explained in detail in what follows.

Finally, as already mentioned, the proposed code can be executed in any standard multi-core PCs as well as in supercomputer architectures. In particular, the numerical examples were executed in both the Santos Dumont (SDumont) supercomputing facility at LNCC in Brazil and in a Windows Workstation with an Intel Xeon Silver 4216 processor with a clock frequency of 2.10 GHz having 16 cores in total and 96 Gb of memory. However, the results reported in this section were obtained by using this last architecture with 10 MPI processes, allowing for timing comparison to other implementations.

### 6.1 Example 1

Minimizing the structural flexibility under volume constraint is one of the most studied problems in topology optimization. In order to deal with such a problem, we set $p = q$ in (2.10). Therefore, in this particular case, $\mathcal{J}(u)$ becomes the so-called compliance shape functional, namely

$$\mathcal{J}(u) = \int_{\Gamma_N} q \cdot u \, , \tag{6.1}$$

with $u$ solution of (2.4) and $q$ a given surface traction on the boundary $\Gamma_N$. Moreover, according to Remark 1, it follows immediately that the adjoint state $v$, solution of (2.10), can be obtained as $v = u$.

In this first example, the initial cubic hold-all domain $\mathcal{D}$ has four roller supports at the bottom face and is subject to a surface traction $q = (0, 0, -1)$ distributed in a small square region centered at the top face. The roller supports are also square. In addition, for symmetry reasons, only one fourth of the hold-all domain $\mathcal{D}$ is discretized, as shown in Figure 2. See the

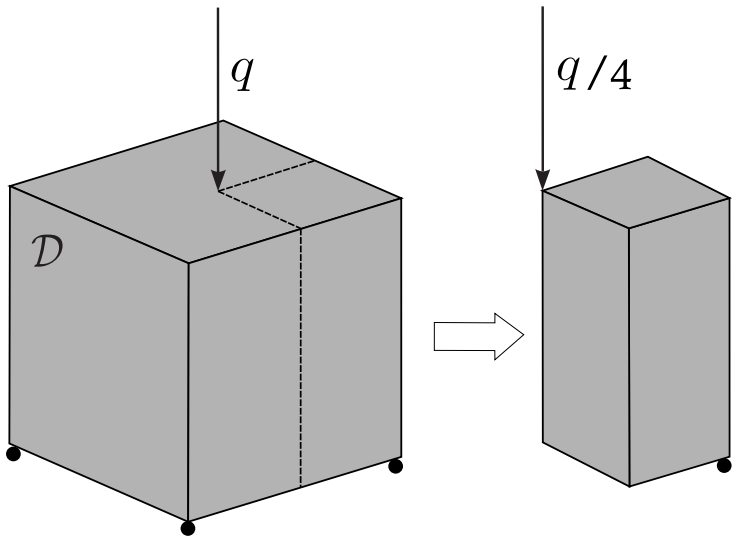
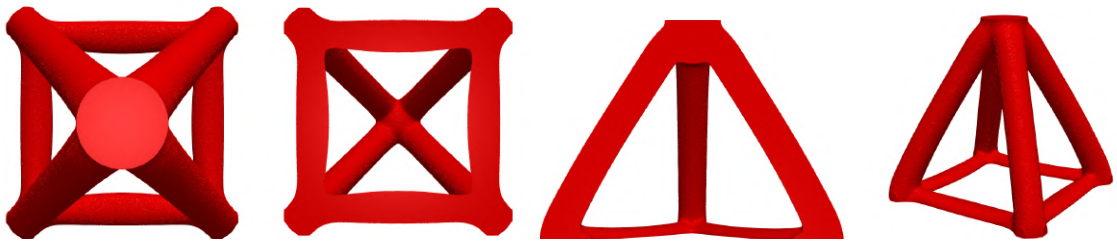



Figure 2: Example 1: Hold-all and computational initial domains.

`Mesh generation` section in the source code available as supplementary material (`elas3d.edp`) for all the steps on how to generate the initial discretized domain and for imposing the boundary

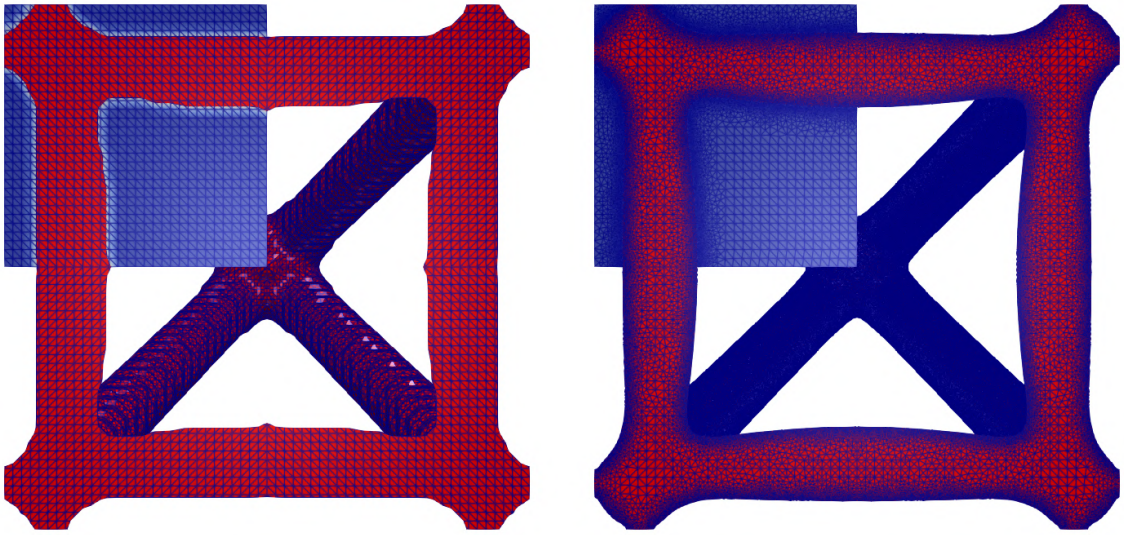
conditions considered in this example. The volume target is set as 10% of the hold-all domain. The optimal topology was obtained after 18 iterations and three adaptive refinements and is represented in Figure 3. More precisely, the initial mesh contains 324,000 elements and 58,621 nodes, whereas the final adapted mesh contains 11,265,428 elements and 1,891,084 nodes, see Figure 4. A bottom view of the final topology is shown in Figure 4(b) in order to depict the



(a) top view (b) bottom view (c) diagonal cut view (d) perspective view

Figure 3: Example 1: Optimal topology.

adapted mesh associated with the optimal topology. The weak region (in blue) remains with coarse elements whereas the interface between the solid (in red) and weak material has fine elements to enhance the boundary representation. In addition, Table 4 presents $i$th iteration



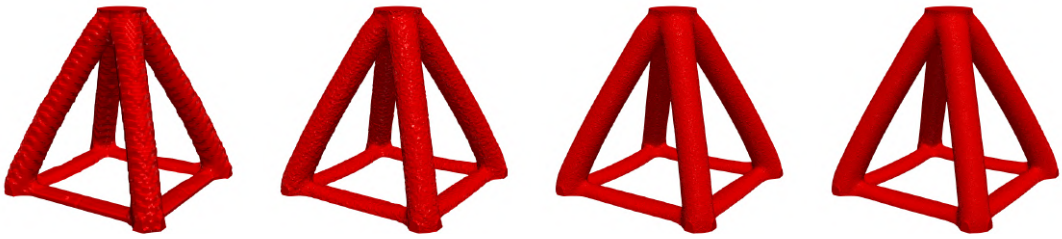
(a) (b)

Figure 4: Example 1: Illustration of the adaptive mesh refinement: topologies associated with the initial (a) and final (b) meshes.

in which the optimality condition is fulfilled together with the associated value of the $\theta$ angle, line-search step size $w$ and volume constraint $g_\Omega$, whereas Figure 5 shows the topology associated with those iterations. Note that, the initial coarse mesh provides a good direction for the optimal topology but with poor boundary representation. In this sense, the successive mesh adaptation procedures aim to refine the mesh over the boundary of the topology, as shown in Figure 4, enhancing resolution of the boundary representation of the optimal topology avoiding the high computational cost of structured mesh refinement over the entire hold-all domain.

Table 4: Example 1: Optimality condition values

| iteration $i$ | $\theta$ | $w$ | $g_\Omega$ |
|---|---|---|---|
| 14 | 0.89° | 1.00 | 3.20% |
| 16 | 0.53° | 1.00 | 1.66% |
| 17 | 0.77° | 1.00 | 1.31% |
| 18 | 0.53° | 1.00 | 0.98% |



(a) iteration 14 (b) iteration 16 (c) iteration 17 (d) iteration 18

Figure 5: Example 1: Topologies associated with initial and adapted meshes.

Figure 6 depicts the cumulative execution time of the optimization iterations obtained from the standard multi-core PC specified at the beginning of this section. Some comments are relevant regarding the computational efficiency of the FreeFEM parallel framework proposed here. As previously mentioned, parallel computing is performed for solving the PDEs posed in three spatial dimensions, whereas other procedures are sequentially executed. Thus, such sequential operations tend to become computational bottlenecks, since they typically take the same time despite the number of processes. In particular, the mesh adaptation procedure is performed here in a sequential fashion with the help of `mmg`, and may be an important bottleneck in case the user chooses to apply several adaptive mesh refinement steps. In this specific example, the time spent to perform the three adaptive mesh refinements is around 7 minutes, which is approximately 1/6 of the total execution time.

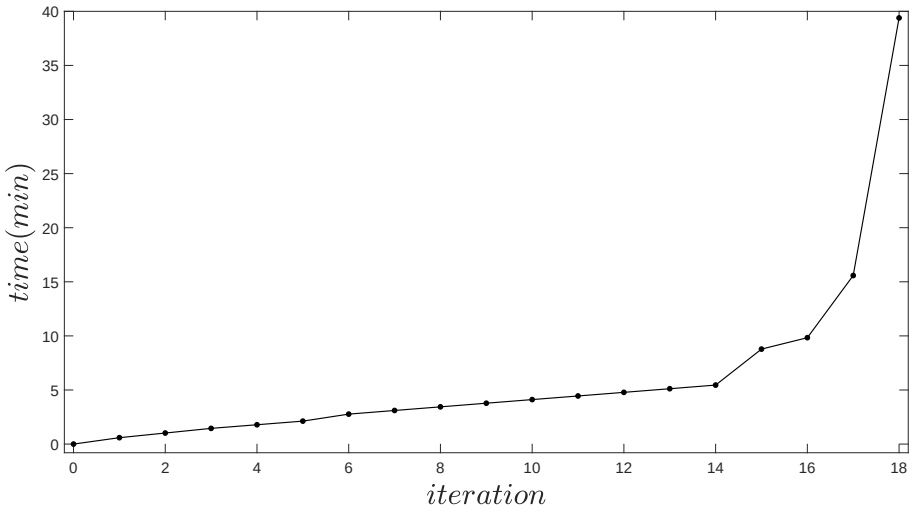



Figure 6: Example 1: Computational time across the 18 iterations of the optimization process.

Finally, to have a better idea of the impact of the sequential operations on the overall computational performance, we conducted a strong scaling analysis of the optimization iterations for 1, 2, 4, 8, and 16 MPI processes in the Windows workstation, as shown in Figure 7. We can observe an almost linear speed-up from 1 to 4 MPI processes and smaller gains from 8 to 16 processes, which is mostly due to the sequential operations in general, and the mesh refinement

in particular. In this sense, it is important to mention that the proposed implementation can certainly be optimized from a computational efficiency point of view through the parallelization of the sequential parts of the code. However, this would exceed the main purpose of this work, which is to pedagogically provide a simple parallel FreeFEM framework for topology optimization.

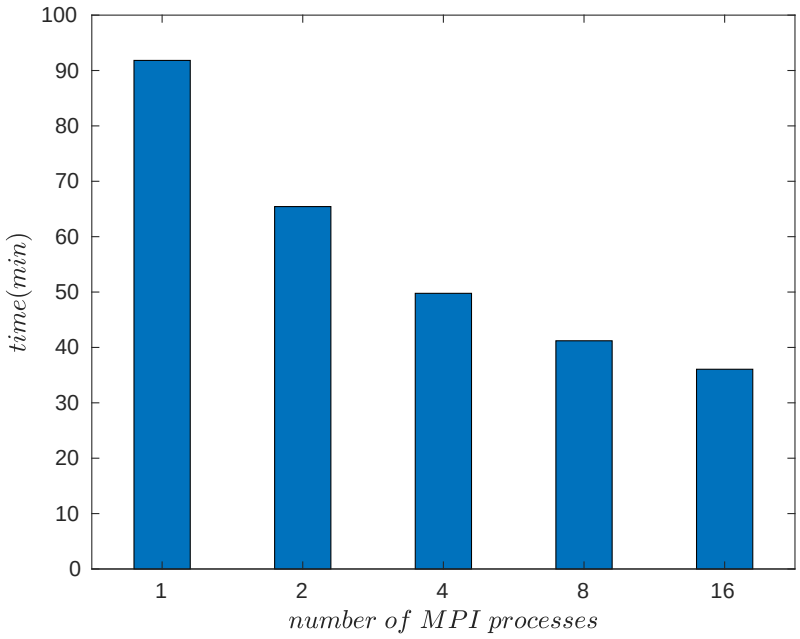



Figure 7: Example 1: Execution time comparison for different of MPI processes.

### 6.2 Example 2

Now, let us consider the problem of topology design of compliant mechanisms. In simple terms, compliant mechanisms are single-piece (monolithic) flexible structures which convert elementary inputs into complex movements by amplifying and changing their direction (Burns and Crossley, 1964; Sigmund, 1997). These mechanisms are typically easy to fabricate and miniaturize and they also have no need for lubrication.

In order to deal with the synthesis of compliant mechanisms, let us adapt the problem stated in Section 2. Firstly, the boundary $\Gamma_N$ is split into two mutually disjoint parts $\Gamma_{in}$ and $\Gamma_{out}$, such that $\Gamma_N = \Gamma_{in} \cup \Gamma_{out}$. Then, we define $q = q_{in}$ on $\Gamma_{in}$ and $q = q_{out}$ on $\Gamma_{out}$. Therefore, the variational problem (2.4) can be rewritten as

$$u \in \mathcal{U} : \int_\Omega \sigma(u) \cdot \varepsilon(u) = \int_{\Gamma_{in}} q_{in} \cdot \eta + \int_{\Gamma_{out}} q_{out} \cdot \eta \quad \forall \eta \in \mathcal{V}. \tag{6.2}$$

In addition, we set $p = q_{in}$ on $\Gamma_{in}$ and $p = \kappa q_{out}$ on $\Gamma_{out}$. Thus, the shape functional (2.3) reads

$$\mathcal{J}(u) = \int_{\Gamma_{in}} q_{in} \cdot u + \kappa \int_{\Gamma_{out}} q_{out} \cdot u \ , \tag{6.3}$$

and the associated adjoint system (2.10) may be stated as

$$v \in \mathcal{V} : \int_\Omega \sigma(v) \cdot \varepsilon(u) = \int_{\Gamma_{in}} q_{in} \cdot \eta + \kappa \int_{\Gamma_{out}} q_{out} \cdot \eta \quad \forall \eta \in \mathcal{V}. \tag{6.4}$$

where $\kappa > 0$ is a weight parameter. The reader may refer to the paper by Lopes and Novotny (2016) for more details regarding the formulation adopted herein.

Let us now present the numerical example of an inverter mechanism. The hold-all domain which also represents the initial guess is given by a cube clamped on a small area on the bottom

corners, whereas loads $q_{in} = (0, 0, 2)$ and $q_{out} = (0, 0, 1)$ are applied at center of bottom and top faces, respectively. Once again, for symmetry reasons only one fourth of the hold-all domain needs to be discretized. See the sketch in Figure 8. In addition, the weight parameter is set as $k = 10$ and the volume target is set as 10% of the hold-all domain.

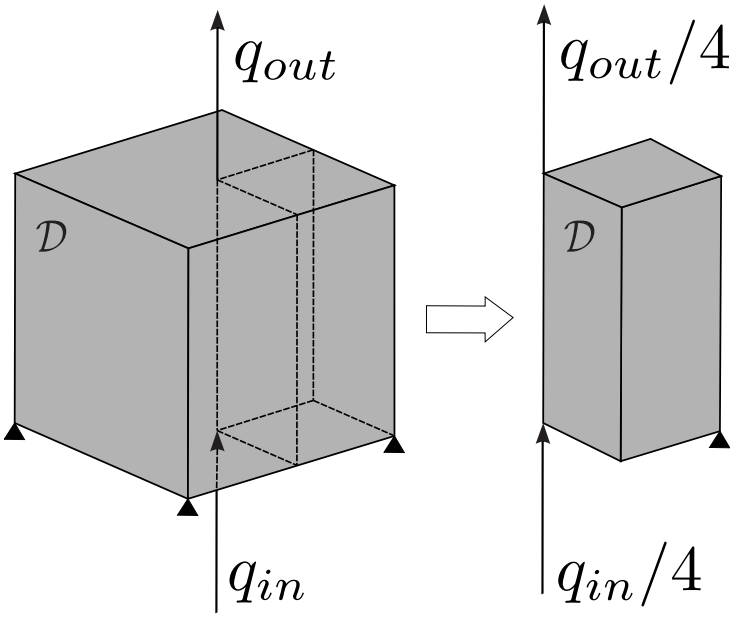
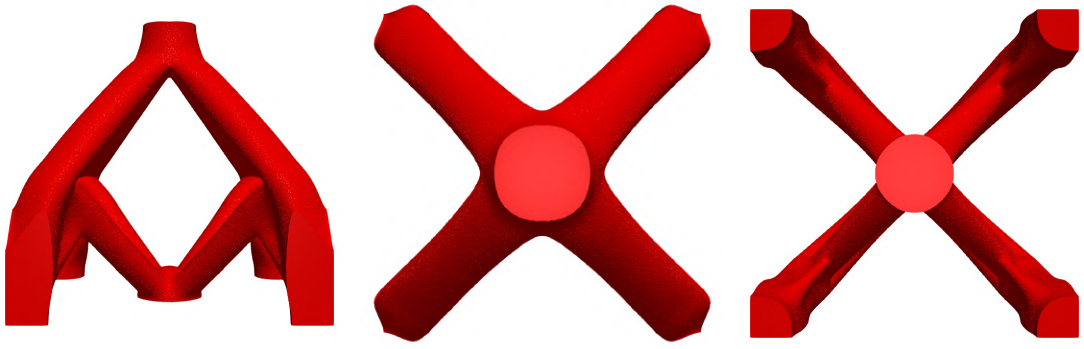



Figure 8: Example 2: Hold-all and computational initial domains.

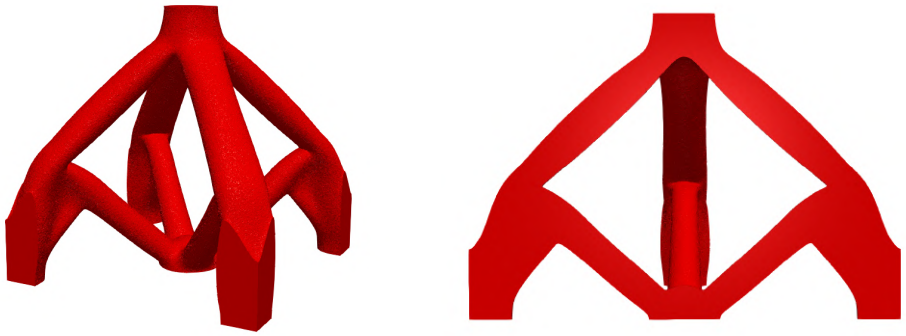
The optimal topology is obtained after 25 iterations and three adaptive mesh refinements. More precisely, the initial mesh contains 96,000 elements and 18,081 nodes, whereas the final adapted mesh contains 4,904,484 elements and 824,942 nodes. The final topology is shown in Figure 9. In addition, Figure 10 shows an amplified deformation of the optimal design, ratifying the mechanism functionality. Finally, Table 5 presents the $i$th iteration in which the optimality condition is fulfilled together the associated value of the $\theta$ angle, line-search step size $w$ and volume constraint $g_\Omega$.

Table 5: Example 2: Optimality condition values

| iteration $i$ | $\theta$ | $w$ | $g_\Omega$ |
|---|---|---|---|
| 19 | 0.98° | 1.00 | 0.05% |
| 21 | 0.87° | 1.00 | 0.29% |
| 24 | 0.46° | 0.75 | 0.03% |
| 25 | 0.79° | 1.00 | 0.15% |

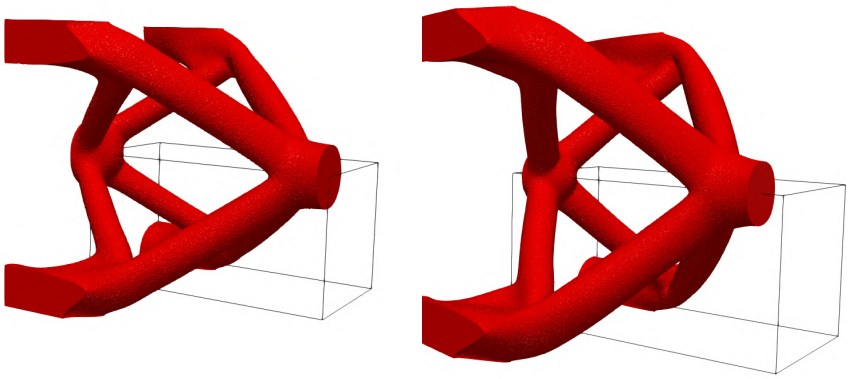


(a) side view (b) top view (c) bottom view



(d) isometric view (e) diagonal cut view

Figure 9: Example 2: Optimal topology.

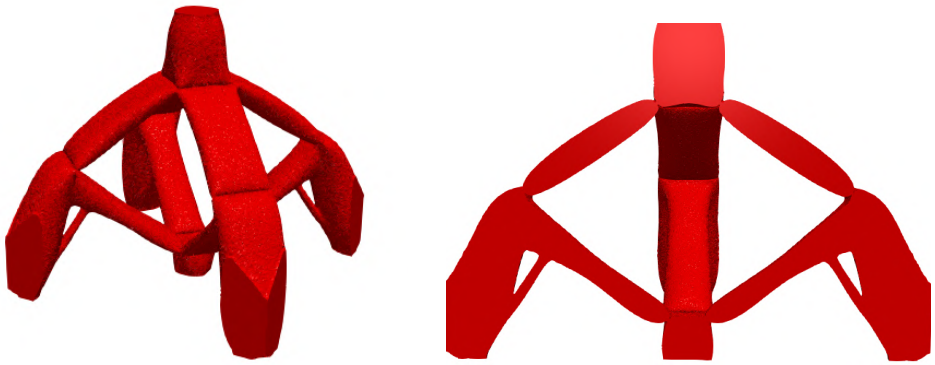


(a) undeformed configuration (b) deformed configuration

Figure 10: Example 2: Undeformed and deformed configurations of the optimal design.

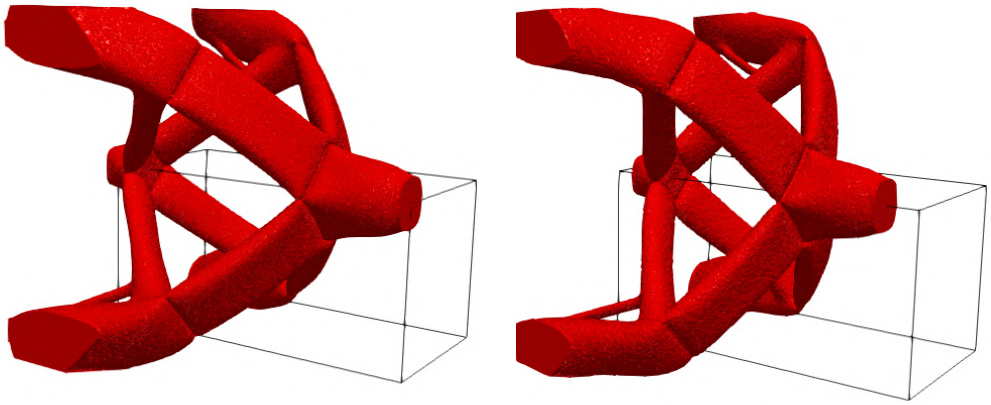
Finally, the reader is welcome to make changes in some of the parameters of the code in order to understand the role played by each of them. For example, we may obtain an inverter mechanism with better efficiency by setting the contrast `rho0 = 1.0e-3`. However, this comes at the expense of some convergence issues, which suggests that a finer mesh and more iterations are required. The result is presented in Figure 11, which was obtained after 99 iterations for a mesh generation parameter `n = 2` and three adaptive mesh refinements, analogously to the previous examples. In particular, the imposed volume constraint is fulfilled, $g_\Omega = 0.68\%$, and the optimality condition is given by $\theta = 6.29°$, which is considered acceptable for this type of problem (Amstutz and Andrä, 2006). Figure 12 presents an amplified deformation of the

optimal design, showing the mechanism functionality.



(a) isometric view (b) diagonal cut view

Figure 11: Example 2: Optimal topology for `rho0 = 1.0e-3`.



(a) undeformed configuration (b) deformed configuration

Figure 12: Example 2: Undeformed and deformed configurations of the optimal design for `rho0 = 1.0e-3`.

## 7 Conclusions

In this work, a simple and effective parallel FreeFEM implementation for 3D structural optimization was presented. The topology optimization algorithm relies on the topological derivative concept together with a level-set domain representation method. For the sake of completeness, all the background knowledge necessary for implementing the topology optimization algorithm was provided. Some remarks about parallel FreeFEM were also presented and the complete computational implementation was provided. Two benchmark examples highlighted the effectiveness of the proposed approach and the role played by adaptive mesh refinement in enhancing the boundary representation resolution with a relatively low computational cost. Finally, it is worth mentioning that the parallel FreeFEM framework provided here can be used/adapted by students or newcomers to the field as a starting point for more complex and advanced problems in three spatial dimensions, regardless of the adopted optimization approach. In particular, the resulting code can be converted into a SIMP-based topology design algorithm which, in this context, combines the density-based approach with a level-set domain representation method. Actually, the sensitivity analysis based on the SIMP method is written as (Bendsøe and Sigmund,

2003)

$$\frac{d\mathcal{J}}{d\rho_e} = -p\rho_e^{p-1}[\mathbf{v}_e]^\top[\mathbf{k}_e][\mathbf{u}_e] \,, \tag{7.1}$$

where $\rho_e$ is the density, $\mathbf{u}_e$ is the direct displacement, $\mathbf{v}_e$ is the adjoint displacement and $\mathbf{k}_e$ is the stiffness matrix, all of them associated with the $e$th finite element. In addition, $p$ is the SIMP penalty factor. Now, we need to replace the topological gradient (4.3) by the SIMP sensitivity (7.1). Thanks to the FreeFEM feature of macros, the SIMP-based topology design algorithm can be devised simply by redefining the macros `gte(u,v)` and `gti(u,v)` from Table 1 as follows

```
macro gte(u,v) (-p*rho^(p-2.0)*se(u,v)) // EOM
macro gti(u,v) (-p*rho^(p-2.0)*se(u,v)) // EOM
```

The power `(p-2)` comes out from the fact that `se(u,v)` is already multiplied by `rho`, as can be seen in Table 1. By setting

```
real p = 2.0;
```

we obtain exactly the same results as the ones reported in Figures 3 and 9, for instance. However, for a Poisson ratio different from $\nu = 0.2$, the reader/user has to find the optimal $p$ numerically. The same phenomenon is observed in two spatial dimensions, where the optimal SIMP penalty factor is given by $p = 3$ for the particular case in which $\nu = 1/3$ or $\nu = 1/4$, assuming either plane stress or plane strain states, respectively. For a deeper discussion concerning the relation between topological derivatives and material interpolation schemes in topology optimization, the reader may refer to the papers by Amstutz (2011) and Ferrer (2019). Finally, the complete 3D topology optimization FreeFEM code is available as supplementary material (`elas3d.edp`) for the reader convenience.

## Acknowledgments

The authors acknowledge the National Laboratory for Scientific Computing (LNCC/MCTI, Brazil) for providing HPC resources of the SDumont supercomputer, which have contributed to the research results reported within this paper. URL: `http://sdumont.lncc.br`. This research was partly supported by CNPq (Brazilian Research Council), CAPES (Brazilian Higher Education Staff Training Agency) and FAPERJ (Research Foundation of the State of Rio de Janeiro). These financial supports are gratefully acknowledged. Finally, we would like to thank Prof. Ole Sigmund for the helpful suggestions.

## Funding

This research was partly supported by CNPq (Brazilian Research Council), CAPES (Brazilian Higher Education Staff Training Agency) and FAPERJ (Research Foundation of the State of Rio de Janeiro). These supports are gratefully acknowledged.

## Conflict of interest

The authors declare that they have no conflict of interest.

## Replication of results

The authors state that all the data necessary to replicate the results are presented in the manuscript.

## References


G. Allaire. *Shape optimization by the homogenization method*, volume 146 of *Applied Mathematical Sciences*. Springer-Verlag, New York, 2002.

G. Allaire and O. Pantz. Structural optimization with FreeFEM++. *Structural and Multidisciplinary Optimization*, 32:173–181, 2006.

G. Allaire, C. Dapogny, and F. Jouve. *Handbook of Numerical Analysis, Geometric Partial Differential Equations – Part 2*, volume XXII, chapter Shape and topology optimization, pages 1–132. Elsevier, Amsterdam, NL, 2021.

S. Amstutz. Connections between topological sensitivity analysis and material interpolation schemes in topology optimization. *Structural and Multidisciplinary Optimization*, 43(6):755–765, 2011.

S. Amstutz and H. Andrä. A new algorithm for topology optimization using a level-set method. *Journal of Computational Physics*, 216(2):573–588, 2006.

E. Andreassen, A. Clausen, M. Schevenels, B. S. Lazarov, and O. Sigmund. Efficient topology optimization in MATLAB using 88 lines of code. *Structural and Multidisciplinary Optimization*, 43:1–16, 2011.

M. P. Bendsøe and N. Kikuchi. Generating optimal topologies in structural design using an homogenization method. *Computer Methods in Applied Mechanics and Engineering*, 71(2): 197–224, 1988.

M. P. Bendsøe and O. Sigmund. *Topology optimization. Theory, methods and applications*. Springer-Verlag, Berlin, 2003.

R. H. Burns and F. R. E. Crossley. Kinetostatic synthesis of flexible link mechanisms. *ASME-Paper*, 68(36), 1964.

D.E. Campeão, S.M. Giusti, and A.A. Novotny. Topology design of plates consedering different volume control methods. *Engineering Computations*, 31(5):826–842, 2014.

V. J. Challis. A discrete level-set topology optimization code written in Matlab. *Structural and Multidisciplinary Optimization*, 41:453–464, 2010.

Y. Cui, T. Takahashi, and T. Matsumoto. An exact volume constraint method for topology optimization via reaction–diffusion equation. *Computers & Structures*, 280:106986, 2023.

A. Ferrer. SIMP-ALL: A generalized SIMP method based on the topological derivative concept. *International Journal for Numerical Methods in Engineering*, 120:361–381, 2019.

A. Ferrer, J. C. Cante, J. A. Hernández, and J. Oliver. Two-scale topology optimization in computational material design: An integrated approach. *International Journal for Numerical Methods in Engineering*, 114:232–254, 2018.

J.M.M. Luz Filho and A.A. Novotny. Topology optimization of three-dimensional structures subject to self-weight loading. *Engineering Computations*, 41(2):307–332, 2024.

J.M.M. Luz Filho, R. Mattoso, and L. Fernandez. A freefem code for topological derivative-based structural optimization. *Structural and Multidisciplinary Optimization*, 66:74, 2023.

P. Gangl, K. Sturm, M. Neunteufel, and J. Schöberl. Fully and semi-automated shape differentiation in ngsolve. *Structural and Multidisciplinary Optimization*, 63(6):1579–1607, 2021.

F. Hecht. New development in freefem++. *Journal of Numerical Mathematics*, 20(3–4):251–265, 2012.

P. Jolivet, V. Dolean, F. Hecht, C. Prud'Homme, and N. Spillance. High performance domain decomposition methods on massively parallel architectures with freefem++. *Journal of Numerical Mathematics*, 20(3–4):287–302, 2012.

C. Kim, M. Jung, T. Yamada, S. Nishiwaki, and J. Yoo. FreeFEM++ code for reaction-diffusion equation-based topology optimization: for high resolution boundary representation using adaptive mesh refinement. *Structural and Multidisciplinary Optimization*, 62:439–455, 2020.

A. Laurain. A level set-based structural optimization code using FEniCS. *Structural and Multidisciplinary Optimization*, 58:1311–1334, 2018.

H. Li, T. Yamada, P. Jolivet, K. Furuta, T. Kondoh, K. Izui, and S. Nishiwaki. Full-scale 3d structural topology optimization using adaptive mesh refinement based on the level-set method. *Finite Elements in Analysis and Design*, 194:103561, 2021.

K. Liu and Andrés Tovar. An efficient 3D topology optimization code written in Matlab. *Structural and Multidisciplinary Optimization*, 50:1175–1196, 2014.

Z. Liu, J. G. Korvink, and R. Huang. Structure topology optimization: fully coupled level set method via FEMLAB. *Structural and Multidisciplinary Optimization*, 29:407–417, 2005.

C.G. Lopes and A.A. Novotny. Topology design of compliant mechanisms with stress constraints based on the topological derivative concept. *Structural and Multidisciplinary Optimization*, 54(4):737–746, 2016.

A.A. Novotny and J. Sokołowski. *An introduction to the topological derivative method*. Springer Briefs in Mathematics. Springer Nature Switzerland, 2020. doi: 10.1007/978-3-030-36915-6.

A.A. Novotny, S.M. Giusti, and S. Amstutz. Guest Editorial: On the topological derivative method and its applications in computational engineering. *Engineering Computations*, 39(1): 1–2, 2022.

J. Oliver, D. Yago, J. Cante, and O. Lloberas-Valls. Variational approach to relaxed topological optimization: closed form solutions for structural problems in a sequential pseudo-time framework. *Computer Methods in Applied Mechanics and Engineering*, 355(1):779–819, 2019.

S. Osher and J. A. Sethian. Front propagating with curvature dependent speed: algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79(1):12–49, 1988.

M. Otomori, T. Yamada, K. Izui, and S. Nishiwaki. Matlab code for a level set-based topology optimization method using a reaction diffusion equation. *Structural and Multidisciplinary Optimization*, 51(5):1159–1172, 2015.

G. Sadaka, A. Rakotondrandisa, P.H. Tournier, F. Luddens, C. Lothodé, and I. Danaila. Parallel finite-element codes for the simulation of two-dimensional and three-dimensional solid-liquid phase-change systems with natural convection. *Computer Physics Communications*, 257(107492), 2020.

O. Sigmund. On the design of compliant mechanisms using topology optimization. *Mechanics of Structures and Machines: An International Journal*, 25(4):493–524, 1997.

O. Sigmund. A 99 line topology optimization code written in Matlab. *Structural and Multidisciplinary Optimization*, 21:120–127, 2001.

O. Sigmund and K. Maute. Topology optimization approaches. *Structural and Multidisciplinary Optimization*, 48:1031–1055, 2013.

O. Sigmund and J. Petersson. Numerical instabilities in topology optimization: a survey on procedures dealing with checkerboards, mesh-dependencies and local minima. *Structural and Multidisciplinary Optimization*, 16:68–75, 1998.

J. Sokołowski and J. P. Zolésio. *Introduction to shape optimization - shape sensitivity analysis*. Springer-Verlag, Berlin, Germany, 1992.

N. P. Van Dijk, K. Maute, M. Langelaar, and F. Van Keulen. Level-set methods for structural topology optimization: a review. *Structural and Multidisciplinary Optimization*, 48:437–472, 2013.

C. Wang, Z. Zhao, M. Zhou, O. Sigmund, and X. S. Zhang. A comprehensive review of educational articles on structural and multidisciplinary optimization. *Structural and Multidisciplinary Optimization*, 64:2827–2880, 2021.