Comptes Rendus
Mathématique

Charles Dapogny and Florian Feppon

Shape optimization using a level set based mesh evolution method: an overview and tutorial

Volume 361 (2023), p. 1267-1332

Published online: 31 October 2023

https://doi.org/10.5802/crmath.498

Comptes Rendus
Mathématique
2023, Vol. 361, p. 1267-1332
https://doi.org/10.5802/crmath.498

Numerical analysis, Mechanics / *Analyse numérique, Mécanique*

Shape optimization using a level set based mesh evolution method: an overview and tutorial

Une stratégie d'évolution de maillage basée sur la méthode des lignes de niveaux pour l'optimisation de formes : survol et mise en pratique

Charles Dapogny[a] **and Florian Feppon**[b]

[a] Univ. Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), LJK, 38000 Grenoble, France

[b] Department of Computer Science, KU Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium

E-mails: charles.dapogny@univ-grenoble-alpes.fr, florian.feppon@kuleuven.be

Abstract. This article revolves around a recent numerical framework for shape and topology optimization, which features an exact mesh of the shape at each iteration of the process, while still leaving the room for an arbitrary evolution of the latter (including changes in its topology). In a nutshell, two complementary representations of the shape are combined: on the one hand, it is meshed exactly, which allows for precise mechanical calculations based on the finite element method; on the other hand, it is described implicitly, using the level set method, which makes it possible to track its evolution in a robust way. In the first part of this work, we overview the main aspects of this numerical strategy. After a brief presentation of some necessary background material – related to shape optimization and meshing, among others – we describe the numerical schemes involved, notably when it comes to the practice of the level set method, the remeshing algorithms, and the considered optimization solver. This strategy is illustrated with 2d and 3d numerical examples in various physical contexts. In the second part of this article, we propose a simple albeit efficient `python`-based implementation of this framework. The code is described with a fair amount of details, and it is expected that the reader can easily elaborate upon the presented examples to tackle his own problems.

Résumé. Cet article traite d'un cadre de travail récent dédié à la résolution numérique de problèmes d'optimisation de formes ; il s'illustre par une représentation exacte, maillée, de la forme à chaque itération du procédé, tout en laissant la place à une évolution arbitraire de celle-ci (y compris des changements de sa topologie). L'idée centrale de cette stratégie est de combiner deux représentations complémentaires de la forme : d'une part, celle-ci est maillée explicitement, de sorte qu'il est possible d'effectuer des calculs mécaniques précis par la méthode des éléments finis ; d'autre part, elle est décrite implicitement, par la méthode des lignes de niveaux, facilitant ainsi le suivi robuste de son évolution. Dans la première partie de ce travail, on résume les points saillants de cette stratégie numérique. Après avoir brièvement rappelé quelques notions de base – en lien, entre autres, avec l'optimisation de formes et le maillage – on décrit les schémas numériques mis en jeu, notamment pour la pratique de la méthode des lignes de niveaux, les algorithmes de remaillage, et l'algorithme d'optimisation numérique. Cette méthodologie est illustrée par des exemples numériques en deux et trois dimensions d'espace, dans différents contextes physiques. Dans la seconde partie de cet article, on propose une implémentation `python` open-source, simple mais efficace, de ce cadre de travail. Le code est détaillé de sorte que le lecteur puisse facilement intervenir dedans et le modifier pour traiter un problème de son choix.

2020 Mathematics Subject Classification. 49M41, 65K05, 65N50, 74P05, 74P10, 90C90.

Funding. The work of C.D. is partially supported by the project ANR-18-CE40-0013 SHAPO, financed by the French Agence Nationale de la Recherche (ANR)..

Manuscript received 1 December 2022, revised 16 March 2023, accepted 17 March 2023.

1. Introduction

The scarcity and the cost of resources such as raw materials and energy make it ever more necessary to tailor the design of physical devices from the early stages of design, so that they achieve their purpose with the minimum amount of constituent material and input power. For this reason, shape and topology optimization has aroused a tremendous enthusiasm in both academic and industrial communities, as a set of fully automated algorithms for predicting optimized designs with respect to physical requirements, see e.g. [13, 15, 23, 53, 80].

From the mathematical point of view, shape and topology optimization problems show up under the generic form

$$\min_{\Omega} J(\Omega) \text{ s.t. } \begin{cases} G(\Omega) = 0, \\ H(\Omega) \leq 0, \end{cases}$$

where the objective and the equality and inequality constraint functionals $J(\Omega)$, $G(\Omega)$ and $H(\Omega)$ depend on the optimized shape Ω in a possibly very intricate way: in concrete applications, they involve a taste of the physical behavior of Ω through the solution to a partial differential equation posed on Ω. For instance,

- When Ω accounts for a mechanical structure subjected to prescribed loads, $J(\Omega)$, $G(\Omega)$ and $H(\Omega)$ bring into play the elastic displacement, characterized as the solution to the linear elasticity system posed on Ω;
- When Ω represents a fluid duct, $J(\Omega)$, $G(\Omega)$ and $H(\Omega)$ depend on the velocity of the fluid, which is governed by the Stokes or the Navier–Stokes equations on Ω.

Leaving aside theoretical issues, regarding for instance the existence of an optimal solution (a purpose for which we refer to e.g. [29,67]), the numerical treatment of such problems is classically plagued by the need to reconcile two antagonistic needs. On the one hand, the evaluation of the objective and constraint functionals $J(\Omega)$, $G(\Omega)$, $H(\Omega)$ and the calculation of their sensitivities with respect to "small" variations of the design Ω require to solve one or several partial differential equations posed on Ω. This task is typically accomplished by using the finite element method, which in turn relies on a high-quality mesh $\mathscr{T}$ of Ω. On the other hand, the update of the shape Ω between successive stages of the optimization process, which is typically realized by deforming Ω according to a velocity field $\theta : \mathbb{R}^d \to \mathbb{R}^d$, is a difficult operation to translate in terms of a mesh of Ω, since the latter very likely becomes degenerate or even invalid in the course of such process.

This difficulty of finding a numerical representation of the shape which is amenable to all the operations of a shape and topology optimization workflow has been acknowledged as a genuine bottleneck since the early days of shape optimization. It is also encountered in multiple disciplines such as inverse problems, image processing, etc., see for instance Chap. 23 in [60] about this point. Various paradigms have been thought of to overcome this issue; let us notably mention two of them:

- Relaxation methods replace the parametrization of the shape and topology optimization problem by classical "black-and-white" designs Ω – or equivalently their characteristic function, taking the values 0 and 1 outside and inside Ω respectively – by "grayscale" density functions $h : D \to [0,1]$ defined on a large "hold-all" domain D (and possibly a microstructure tensor, describing the local microscopic structure of the shape). This change of perspectives can be rigorously justified by the mathematical theory of homogenization [3, 4, 74], which has inspired simplified, heuristic variants of "classical" shape optimization problems such as the popular SIMP framework in mechanical engineering, see [22, 23].
- Eulerian methods, such as the level set method [11, 86, 94, 102] or the phase-field method [24, 27, 44, 99] bring into play a fixed mesh of a large "hold-all" domain D. The sought shape Ω is defined implicitly, in terms of quantities defined on the whole domain D. For instance, the level set method features shapes defined as negative subdomains $\left\{x \in D,\ \phi(x) < 0\right\}$ of functions $\phi : D \to \mathbb{R}$, see Section 3.3.1 below for more details.

The aforementioned methods alleviate the need to track the evolution of a mesh of the optimized shape Ω, but this comes at a price: the fact that no proper mesh of Ω is available raises the need for an approximation of the partial differential equations characterizing its physical behavior, or for an advanced and often intrusive finite element method tailored for implicitly-defined domains, such as the eXtended Finite Element Method (X-FEM) [50], or the cut-FEM method [32].

In this article, we overview a recent mesh evolution framework introduced in [5–7], which allows to describe arbitrarily large deformations of the shape throughout the process, and which at the same time features an exact, well-shaped mesh of the latter. This framework leverages two complementary representations of shapes: on the one hand, they are meshed exactly, and so accurate mechanical computations can be conducted by standard finite element methods and solvers used in a "black-box" fashion. On the other hand, they are represented by means of the level set method on a larger, fixed computational domain, which makes it possible to account for arbitrarily large deformations. The core of this strategy is a set of numerical algorithms allowing to switch from one of these representations to the other, and so to consistently use that which is most relevant for every operation involved in the shape optimization workflow. Since its introduction in the context of structural mechanics, this idea has been successfully applied in a variety of more challenging physical contexts, such as fluid mechanics and fluid-structure interaction [56, 58], heat transfer [59], quantum chemistry [28], plasticity [47], fracture mechanics [46], etc.

The aim of this article is twofold. First and foremost, we describe this level set based mesh evolution strategy and its main numerical ingredients, such as the algorithms involved in the practice of the level set method, the meshing aspects of the framework, and the efficient resolution of infinite-dimensional constrained optimization problems. We conclude this presentation with a selection of numerical results obtained by application of this strategy. The second purpose of this work is in line with the multiple educational articles that have been devoted to shape and topology optimization, see e.g. [12, 76, 95] or [101] for an exhaustive list. We propose and detail a `python` implementation of this level set based mesh evolution framework which is at the same time pedagogical (and thus reasonably simple), and general enough to allow the user to easily build upon this code so as to address his personal shape optimization problems.

The remainder of this article is organized as follows. In Section 2, we present the model physical setting of our discussion, that of shape optimization of elastic structures. We introduce the chief theoretical concepts at stake, and provide a general sketch of a typical shape optimization procedure. In Section 3, we discuss the choice of an adequate representation of the shape in the perspective of accounting for its update between the various iterations of the shape optimization process. After a brief reminder of the relevant notions, we describe Lagrangian strategies, their limitations, and we present the level set method and its applications in shape and topology optimization as a potential remedy. In Section 4, we describe the level set based mesh evolution method at stake in this article, and we describe the main numerical operations involved. In Section 5, we then present several numerical results obtained with this strategy. This article ends with a fairly long appendix where a proposed open-source implementation of this framework is presented: each step of the method is carefully described, with the hope that it is easy for the reader to get into the code and elaborate upon it for its own purposes.

2. Presentation of the shape optimization framework

In this section, we introduce the theoretical framework of this article, and we discuss a few important notions. To set ideas, our discussion unfolds in the relatively simple physical setting of linearly elastic structures, which is presented in Section 2.1; some basic facts about Hadamard's boundary variation method and shape derivatives are then recalled in Section 2.2. Finally, we sketch a generic shape optimization algorithm in Section 2.3, as a support for the subsequent practical considerations.

2.1. *A model shape optimization problem in the context of structural mechanics*

Throughout this presentation, a shape is a bounded, Lipschitz domain $\Omega \subset \mathbb{R}^d$ $(d = 2, 3)$, whose boundary is the reunion of three disjoint, open regions Γ_D, Γ_N, Γ:

$$\partial\Omega = \overline{\Gamma_D} \cup \overline{\Gamma_N} \cup \overline{\Gamma}.$$

In this decomposition,

- Γ_D is a given piece of hypersurface where Ω is clamped;
- Γ_N is another given piece of hypersurface where loads $g \in L^2(\Gamma_N)^d$ are applied;
- The remaining part Γ of $\partial\Omega$ is traction free; it is the only part of $\partial\Omega$ which is subject to optimization;

see Figure 1 for an illustration.

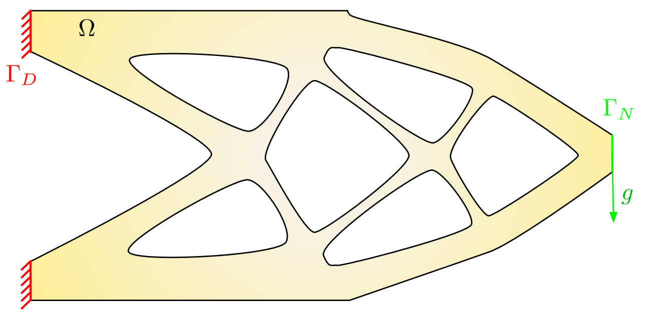

Figure 1. Physical setting of mechanical structures considered in Section 2.1.

Omitting body forces for simplicity, the displacement of Ω is the unique solution u_Ω in the space

$$H^1_{\Gamma_D}(\Omega)^d := \left\{ u \in H^1(\Omega)^d,\ u = 0 \text{ on } \Gamma_D \right\}$$

to the linearized elasticity system:

$$\begin{cases} -\operatorname{div}(Ae(u_\Omega)) = 0 & \text{in } \Omega, \\ u_\Omega = 0 & \text{on } \Gamma_D, \\ Ae(u_\Omega)n = g & \text{on } \Gamma_N, \\ Ae(u_\Omega)n = 0 & \text{on } \Gamma. \end{cases} \tag{1}$$

In this formulation, $H^1(\Omega)$ is the usual Sobolev space of functions in $L^2(\Omega)$ whose first-order derivatives are also in $L^2(\Omega)$, see [2]. We have denoted by $n : \partial\Omega \to \mathbb{R}^d$ the unit normal vector to $\partial\Omega$, pointing outward Ω, and the symmetric $d \times d$ matrix $e(u) := \frac{1}{2}(\nabla u + \nabla u^T)$ is the strain tensor associated to a displacement field $u : \Omega \to \mathbb{R}^d$. The Hooke's law A is a fourth-order tensor characterizing the physical properties of the constituent material of Ω: it relates the state of stress $\sigma(u)$ inside the structure with the strain $e(u)$ via the relation

$$\sigma(u) = Ae(u), \text{ where, for any symmetric } d \times d \text{ matrix } e, \quad Ae = 2\mu e + \lambda \operatorname{tr}(e)\mathrm{I},$$

and λ, μ are the Lamé coefficients of the material.

In this context, a generic shape optimization problem reads

$$\min_\Omega J(\Omega) \text{ s.t. } \begin{cases} G(\Omega) = 0, \\ H(\Omega) \leq 0. \end{cases} \tag{2}$$

Several important examples about the objective functional $J(\Omega)$ are:

- The compliance of Ω
$$C(\Omega) = \int_\Omega Ae(u_\Omega) : e(u_\Omega)\,\mathrm{d}x = \int_\Gamma g \cdot u_\Omega\,\mathrm{d}s, \tag{3}$$
where : is the Frobenius inner product over the set of $d \times d$ matrices. Equivalently, $C(\Omega)$ measures the total elastic energy stored inside Ω or the work done by the applied loads g;
- A least-square discrepancy criterion
$$D(\Omega) = \int_\Omega k(x)|u_\Omega - u_T(x)|^2\,\mathrm{d}x$$
between the elastic displacement of Ω and a target displacement $u_T(x)$, weighted by a function $k(x)$.
- An integral measure of the stress inside the structure
$$S(\Omega) = \int_\Omega k(x)\|\sigma(u_\Omega)\|^2\,\mathrm{d}x, \tag{4}$$
where for any $d \times d$ matrix σ, we have denoted $\|\sigma\|^2 := \sigma : \sigma$.

In the formulation (2), $G(\Omega) = (G_i(\Omega))_{i=1,\dots,p}$ and $H(\Omega) = (H_j(\Omega))_{j=1,\dots,q}$ are collections of p real-valued equality constraints and q inequality constraints, respectively. These functionals may be, for instance:

- The volume $\mathrm{Vol}(\Omega)$ or the perimeter $\mathrm{Per}(\Omega)$ of the shape,
$$\mathrm{Vol}(\Omega) = \int_\Omega \mathrm{d}x, \quad \text{and} \quad \mathrm{Per}(\Omega) = \int_{\partial\Omega} \mathrm{d}s. \tag{5}$$
- Other constraints related to the geometry of Ω (thickness, curvature radii, etc.), as imposed by the manufacturing process.

In the sequel, when it allows to simplify the exposition, we shall sometimes consider unconstrained versions of (2), of the form

$$\min_\Omega J(\Omega), \tag{6}$$

where the minimized function $J(\Omega)$ may represent e.g. a weighted sum of some of the above shape functionals.

2.2. *The boundary variation method of Hadamard*

Most numerical algorithms dedicated to problems of the form (2) rely on the derivatives of the objective and constraint functions with respect to the optimized variable. In the present context, this raises the need to account for derivatives with respect to the domain. This task can be achieved in different fashions, and we rely on the Hadamard's boundary variation method, pioneered by [64, 81], see also [13, 45, 67, 96].

In this framework, variations of a given shape Ω are considered under the form

$$\Omega_\theta := (\mathrm{Id}+\theta)(\Omega),\ \theta \in W^{1,\infty}(\mathbb{R}^d,\mathbb{R}^d),\ \|\theta\|_{W^{1,\infty}(\mathbb{R}^d,\mathbb{R}^d)} < 1, \tag{7}$$

where $W^{1,\infty}(\mathbb{R}^d,\mathbb{R}^d)$ is the Sobolev space of Lipschitz vector fields on $\mathbb{R}^d$, see [54]. Intuitively, (7) expresses the fact that Ω is deformed according to the "small" vector field θ, see Figure 2.

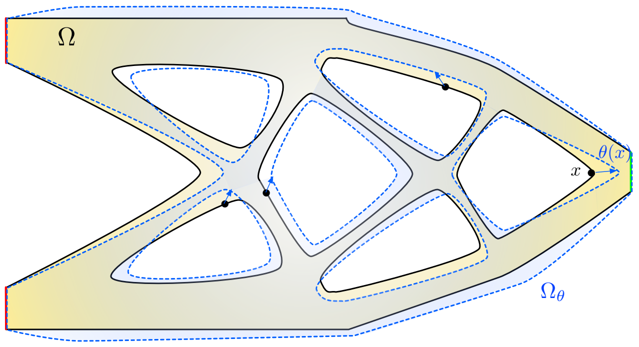

Figure 2. Deformed version Ω_θ of a shape Ω according to Hadamard's boundary variation method.

Remark 1. Often in practice, the vector fields θ featured in Hadamard's method are required to enjoy higher regularity, or to vanish on a region of space which is not subject to modifications, so that they are actually confined to a subset of $W^{1,\infty}(\mathbb{R}^d,\mathbb{R}^d)$. To keep the presentation elementary, we ignore this technicality in the following, see also Section 4.5.

One functional of the domain $F(\Omega)$ is then said to be shape differentiable at a particular shape Ω if the underlying mapping $\theta \mapsto F(\Omega_\theta)$, from $W^{1,\infty}(\mathbb{R}^d,\mathbb{R}^d)$ into $\mathbb{R}$ is Fréchet differentiable at $\theta = 0$. The corresponding derivative $\theta \mapsto F'(\Omega)(\theta)$ satisfies the following expansion:

$$F(\Omega_\theta) = F(\Omega) + F'(\Omega)(\theta) + \mathrm{o}(\theta),\ \text{where } \frac{|\mathrm{o}(\theta)|}{\|\theta\|_{W^{1,\infty}(\mathbb{R}^d,\mathbb{R}^d)}} \xrightarrow{\theta\to 0} 0. \tag{8}$$

The shape derivatives of the objective and constraint functions $J(\Omega)$, $G(\Omega)$ and $H(\Omega)$ of an optimization problem of the form (2) are handful for a variety of purposes. From the theoretical point of view, they are the building blocks of the necessary conditions for a shape Ω to be locally optimal with respect to (2); from the numerical vantage, they make it possible to calculate descent directions as vector fields $\theta : \mathbb{R}^d \to \mathbb{R}^d$ encoding deformations of Ω improving its "performance".

Like those introduced in Section 2.1, the functionals of interest in concrete applications usually depend on the shape Ω in a quite intricate way, via a "state" u_Ω, characterized as the solution to a "physical" partial differential equation posed on Ω. Nevertheless, their shape derivatives can be calculated thanks to the adjoint method, pertaining to the more general field of optimal control. This stake is conceptual and by no means trivial; however, it is well-understood in the literature, and we do not expand on the subject, referring to [77], or [91] for a comprehensive introduction to this method; see also the recent review [8] in the more specific context of shape optimization. Let us simply recall that the shape derivatives of such functionals usually involve the function u_Ω as well as an "adjoint state" p_Ω, satisfying a partial differential equation very similar to that for u_Ω, with a different right-hand side. Here are a few examples, working under mild regularity assumptions on the shape Ω or the state u_Ω, which are omitted for brevity:

- The volume $\text{Vol}(\Omega)$ has the shape derivative
$$\text{Vol}'(\Omega)(\theta) = \int_{\partial\Omega} \theta\cdot n \,\mathrm{d}s.$$
- The shape derivative of the compliance $C(\Omega)$ given by (3) reads
$$C'(\Omega)(\theta) = -\int_\Gamma Ae(u_\Omega) : e(u_\Omega)\,\theta\cdot n\,\mathrm{d}s.$$
- The stress functional $S(\Omega)$ in (4) has the shape derivative
$$S'(\Omega)(\theta) = \int_\Gamma \Big(k(x)\|\sigma(u_\Omega)\|^2 + Ae(u_\Omega) : e(p_\Omega)\Big)\,\theta\cdot n\,\mathrm{d}s, \tag{9}$$
where the adjoint state p_Ω is defined as the solution in $H^1_{\Gamma_D}(\Omega)^d$ to the equation
$$\begin{cases} -\operatorname{div}(Ae(p_\Omega)) = \operatorname{div}(k(x)A\sigma(u_\Omega)) & \text{in } \Omega,\\ p_\Omega = 0 & \text{on } \Gamma_D,\\ Ae(p_\Omega)n = 0 & \text{on } \Gamma\cup\Gamma_N. \end{cases} \tag{10}$$

The above expressions exemplify a quite general phenomenon. Shape derivatives are naturally calculated in volume form, i.e. their expressions are made of volume integrals on Ω, involving u_Ω, θ, $\nabla\theta$, etc.; under suitable regularity assumptions on the shape Ω, the state u_Ω and the possible adjoint state p_Ω, they can often be given a surface expression of the type:
$$J'(\Omega)(\theta) = \int_\Gamma v_\Omega\,\theta\cdot n\,\mathrm{d}s, \tag{11}$$
where $v_\Omega : \Gamma \to \mathbb{R}$ is a scalar field whose expression depends on $J(\Omega)$, u_Ω and p_Ω. In particular, such a shape derivative $J'(\Omega)(\theta)$ only depends on the values of the normal component $\theta\cdot n$ of the deformation θ on the boundary $\partial\Omega$, in agreement with the so-called *Structure theorem* for shape derivatives, see e.g. [67, Section 5.9] or [45, Chapter 9, Section 3.4].

Surface expressions (11) for shape derivatives are convenient for a variety of purposes; for instance, when the unconstrained minimization problem (6) of $J(\Omega)$ is considered, a descent direction θ is easily obtained by imposing that
$$\theta = -v_\Omega n \text{ on } \Gamma, \text{ so that } J'(\Omega)(\theta) = -\int_\Gamma v_\Omega^2\,\mathrm{d}s < 0. \tag{12}$$
On the other hand, however more difficult to exploit, volume expressions may lend themselves to more accurate numerical discretization, see [69].

Remark 2. Variations of the domain of a different nature from (7) are possible, leading to as many different notions of "derivative with respect to the domain", or, more accurately, of asymptotic expansion with respect to perturbations of the domain. Notably, it is possible to account for variations of a shape Ω of the form $\Omega_{x_0,r} := \Omega\setminus\overline{B(x_0,r)}$, where x_0 is a given point inside Ω and $r > 0$. This leads to expansions of the form
$$J(\Omega_{x_0,r}) = J(\Omega) + r^d dJ_T(x_0) + \mathrm{o}(r^d),$$
where $dJ_T(x_0)$ is called the topological derivative of J at x_0 and measures the sensitivity of J with respect to the nucleation of an infinitesimally small hole inside Ω. We refer to [84] about this concept, and to [9, 14] about different ways of using it in the context of shape and topology optimization; see also Appendix A.10 for an implementation.

Let us also evoke the existence of more exotic "topological ligament expansions", evaluating the sensitivity of a shape with respect to the addition of a thin ligament, see [38, 73, 82].

2.3. *An abstract shape optimization algorithm*

The notion of shape derivative underlies a wide range of algorithms for dealing with shape optimization problems of the form (2); in broad outline, each iteration in their implementation can be decomposed into three main stages:

(1) The physical quantities attached to the current shape Ω^n (the elastic displacement u_{Ω^n} and the adjoint state p_{Ω^n} in the context of Section 2.1) are computed by solving the corresponding partial differential equations. For instance, this can be conveniently realized thanks to a finite element solver, if a mesh of the shape Ω^n is available.

(2) A decent direction θ^n for the problem (2) is inferred: θ^n is a vector field such that the deformed version $(\mathrm{Id}+\tau^n\theta^n)(\Omega^n)$ of Ω^n along θ^n for a "small enough" time step τ^n has "improved" performance over Ω^n. For instance, when the constraint-free problem (6) is considered, θ^n is simply a vector field such that $J'(\Omega^n)(\theta^n) < 0$. When a more general constrained optimization problem of the form (2) is considered, θ^n is calculated from the knowledge of the shape derivatives $J'(\Omega)$, $G'(\Omega)$, $H'(\Omega)$ owing to a constrained optimization algorithm.

(3) The shape Ω^n is updated into $\Omega^{n+1} := (\mathrm{Id}+\tau^n\theta^n)(\Omega^n)$.

This generic procedure is summarized in Algorithm 1.

Algorithm 1 Generic shape gradient algorithm.

```
**Initialization:** Initial shape $\Omega^0$.
**for** $n = 0, \dots,$ until convergence **do**
    (1) Calculate the solution $u_{\Omega^n}$ (resp. $p_{\Omega^n}$) to the state (resp. adjoint) equation posed
        in $\Omega^n$.
    (2) From the theoretical formulas for the shape derivatives $J'(\Omega)(\theta)$, $G'(\Omega)(\theta)$ and
        $H'(\Omega)(\theta)$, infer a descent direction $\theta^n$ from $\Omega^n$ for the shape optimization prob-
        lem (2).
    (3) Deform $\Omega^n$ according to $\theta^n$ for a small descent step $\tau^n > 0$, so that the new shape
            $\Omega^{n+1} := (\mathrm{Id}+\tau^n\theta^n)(\Omega^n)$
        is "better" than the previous one in view of (2).
**end for**
**return** $\Omega^n$
```

One critical issue lies in the difficulty of finding a numerical description of the shape and its evolution which is appropriate to each of these three stages. For instance, choosing to represent the shape with a computational mesh conveniently allows to carry out the mechanical analyses implied by the first stage. Unfortunately, such decision makes it notoriously difficult to realize the update of the shape in the third stage in a robust way, all the more so as when large deformations (not to say topological changes) are at stake. This dilemma is faced by all implementations of Algorithm 1, and we focus specifically on this point from the next Section 3.

3. Meshing aspects of shape optimization

The present section is dedicated to the numerical representation of the shape in the perspective of realizing its update between two successive steps of a shape optimization procedure. At first, we briefly recall a few basic definitions and important facts about meshing in Section 3.1, referring for instance to the books [25, 26, 60] for more in-depth presentations. We then present the early "Lagrangian" strategies for shape optimization, emphasizing on their inherent limitations related with meshing aspects. Finally, we describe the level set method as one attempt to circumvent them, and as a cornerstone of the numerical framework discussed in this article.

3.1. *Basic notions about meshes*

Let Ω be a polygonal domain; a simplicial mesh of Ω is a collection $\mathscr{T} = \{T_i\}_{i=1,\dots,N}$ of open simplices (i.e. triangles in 2d, tetrahedra in 3d) which constitute a covering of Ω, that is:

$$\overline{\Omega} = \bigcup_{i=1}^{N} \overline{T_i}.$$

In addition, one usually demands that

- $\mathscr{T}$ should be *valid*, in the sense that the T_i are mutually disjoint: $T_i \cap T_j = \emptyset$ when $i \neq j$.
- $\mathscr{T}$ should be *conforming*: for $i \neq j$ the intersection $\overline{T_i} \cap \overline{T_j}$ is either a vertex, an edge, or (in 3d) a face of $\mathscr{T}$.

The volume mesh $\mathscr{T}$ of Ω additionally bears the information of a *surface mesh* $\mathscr{S}_{\mathscr{T}}$, that is, a mesh composed of edges in 2d, triangles in 3d, accounting for the boundary $\partial\Omega$ and for internal interfaces, delimiting distinct material regions within Ω.

Numerous methods are available when it comes to creating such a mesh $\mathscr{T}$, depending on how the information related to Ω is supplied. Often, a surface mesh $\mathscr{S}$ of the boundary $\partial\Omega$ is provided, which has for instance been constructed thanks to a CAD software; the interior of Ω is then filled with simplices agreeing with the surface elements of $\mathscr{S}$. The most efficient procedures to fulfill this goal are the constrained Delaunay algorithm, or advancing front strategies, [25, 26, 60]. Without entering into details, let us stress that, in spite of its importance and the attention that has been brought to its analysis for several decades, this task remains delicate.

One crucial aspect of meshes is their *quality*, a notion which actually takes on two quite different natures, both illustrated on Figure 3.

- The *finite element quality*. The accuracy of most numerical simulations – conducted with e.g. the finite element method – is strongly dependent on how close the elements of the computational mesh $\mathscr{T}$ are from being regular, see for instance [36] about this classical issue. In practice, a quality factor $\mathscr{Q}(T)$ is used to evaluate the aspect ratio of each simplex $T \in \mathscr{T}$, with the meaning that $\mathscr{Q}(T) \approx 1$ when T is close to regular, and $\mathscr{Q}(T) \approx 0$ when it is nearly degenerate. One popular criterion used in the literature is:
$$\mathscr{Q}(T) = \alpha \frac{\text{Vol}(T)}{\left(\sum_{i=1}^{d(d+1)/2} |e_i|^2\right)^{\frac{d}{2}}},$$
where the e_i are the edges of T, and α is a normalization factor.
- The *geometric quality*. Often in practice, the considered domain Ω (or the internal regions within Ω) is not polygonal; the surface triangulation $\mathscr{S}_{\mathscr{T}}$ is only an approximation of $\partial\Omega$, and it is crucial to ensure that this approximation is accurate enough. This may for instance be expressed by demanding that the Hausdorff distance $d^H(\mathscr{S}_{\mathscr{T}}, \partial\Omega)$ between $\mathscr{S}_{\mathscr{T}}$ and the "ideal", continuous boundary $\partial\Omega$ be smaller than a user-defined tolerance ε.

3.2. *"Geometric" shape optimization*

One early attempt to implement the abstract program of Algorithm 1 follows an intuitive "Lagrangian" strategy, see e.g. [80] or [89]. At each iteration n, the shape Ω^n is explicitly represented

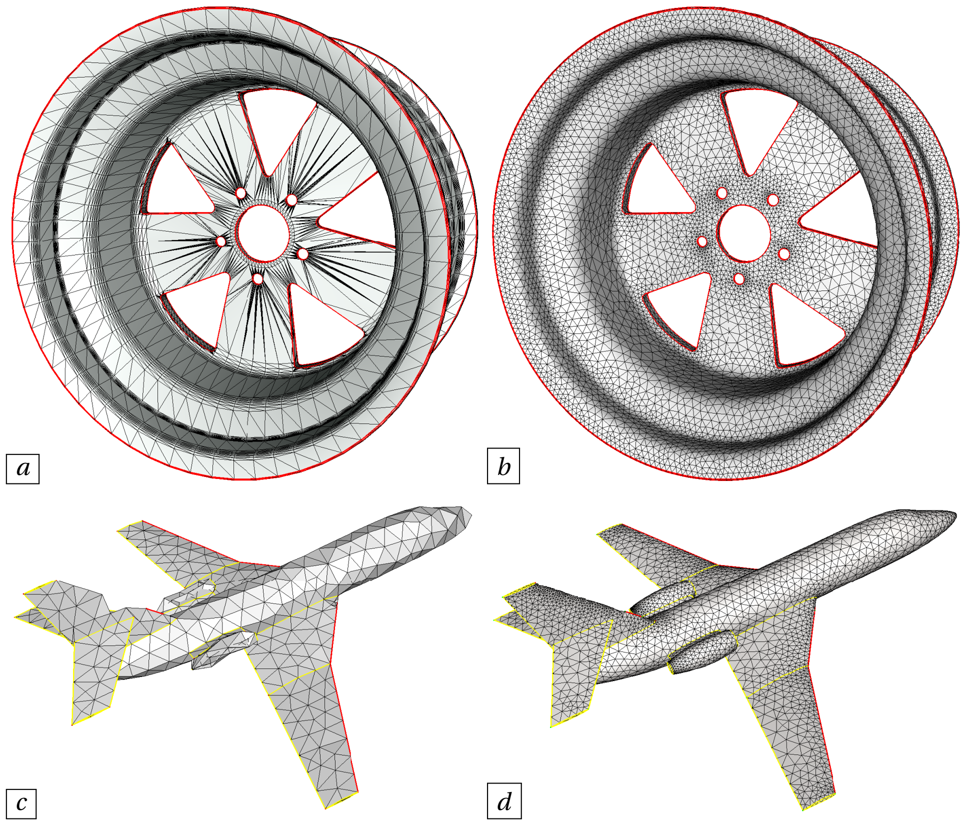

Figure 3. (a) Ill-shaped mesh of a wheel; (b) high quality mesh of the same geometry; (c) mesh accounting for a poor geometric approximation of a plane; (d) fine geometric approximation of the same plane.

by means of a computational mesh $\mathscr{T}^n$. This choice is particularly convenient for the first stage of Algorithm 1, since the solution u_{Ω^n} to the linear elasticity system (1) (and likewise, that p_{Ω^n} to the adjoint system) can be accurately computed thanks to the finite element method; in principle, any commercial solver can be used in a non intrusive way to this end. The calculation of a descent direction θ^n for the shape optimization problem (2), which is the second stage of Algorithm 1, is easily conducted from these data.

The major difficulty posed by this framework is related to the final stage of Algorithm 1, which is about passing from (the mesh $\mathscr{T}^n$ of) Ω^n to (a mesh $\mathscr{T}^{n+1}$ of) $\Omega^{n+1} := (\mathrm{Id} + \tau^n\theta^n)(\Omega^n)$. Certainly, one mesh $\mathscr{T}^{n+1}$ of Ω^{n+1} can be obtained by relocating the vertices of $\mathscr{T}^n$ according to the rule

$$\forall \text{ vertex } x \text{ of } \mathscr{T}, \quad x \longmapsto x + \tau^n\theta^n(x), \tag{13}$$

while keeping the connectivities of the mesh unchanged, see Figure 4. This simple practice unfortunately suffers from serious limitations, since some elements of the mesh are prone to become seriously ill-shaped in the process, not to say downright invalid, see Figure 4 (c).

Admittedly, such an update of the computational mesh can be conducted in a relatively efficient manner thanks to a number of heuristics. For instance,

- One may extend the displacement field θ^n from the boundary $\partial\Omega^n$ to the interior vertices by solving an elasticity system, with the hope that the extended field induces little

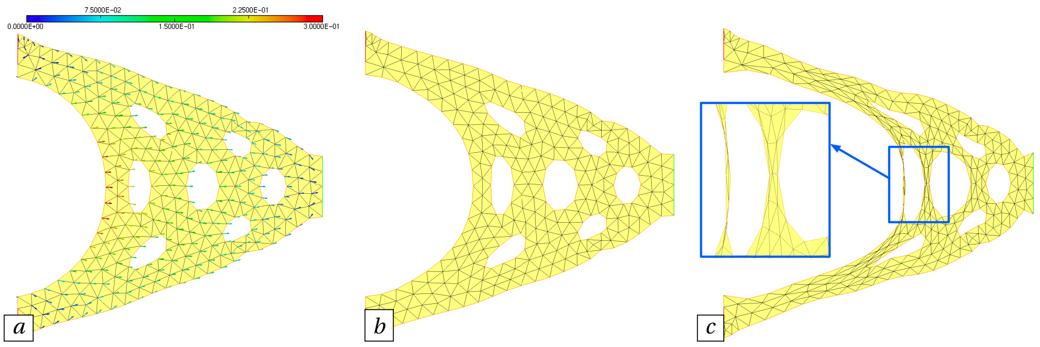

Figure 4. (a) Mesh $\mathscr{T}^n$ of the shape Ω^n with the descent direction θ^n discretized at its vertices; (b) updated mesh $\mathscr{T}^{n+1}$ of $\Omega^{n+1} := (\mathrm{Id}+\tau^n\theta^n)(\Omega^n)$ obtained by using the rule (13) with a "small enough" time step τ^n; (c) invalid mesh $\mathscr{T}^{n+1}$ when the chosen time step τ^n is "too large".

compression of the mesh elements. Large mesh displacements have been realized based on this idea in [17].

- The displacement (13) of the vertices of $\mathscr{T}^n$ can be realized within several sub-stages intertwined with remeshing operations, whose aim is to improve the quality of elements and thereby to postpone the onset of degenerate or invalid elements, insofar as possible, see for instance [20, 65] in this direction.

The so-called Deformable Simplicial Complex (DSC) method, which leverages such ideas together with further heuristics for coping with topological changes, has recently achieved impressive motions of shapes in the course of the shape and topology optimization process, see [34,35].

3.3. *Level set methods for shape and topology optimization*

The level set method is a general paradigm for tracking dramatic evolutions of domains or interfaces, which may even feature topological changes. Since its inception in [87] in the context of curvature-driven interface motion, it has proved to be a very efficient and robust framework for fluid simulations and image processing, to name just a few applications. We outline the basic stakes of this method in Section 3.3.1, referring to [62, 85, 93] for more exhaustive presentations. The use of this method in the context of shape optimization, as an attempt to overcome the aforementioned meshing issues raised by the update of the shape, is then discussed in the next Section 3.3.2.

3.3.1. *A brief reminder about the level set method*

Let $D \subset \mathbb{R}^d$ be a fixed computational domain. The level set method is a general philosophy whereby an arbitrary shape $\Omega \subset D$ can be equivalently described as the negative subdomain of a scalar "level set" function $\phi : D \to \mathbb{R}$, i.e.:

$$\begin{cases} \phi(x) < 0 & \text{if } x \in \Omega, \\ \phi(x) = 0 & \text{if } x \in \partial\Omega, \\ \phi(x) > 0 & \text{if } x \in D \setminus \overline{\Omega}, \end{cases} \tag{14}$$

see Figure 5 for an illustration.

This representation conveniently allows to reformulate the motion of a shape $\Omega(t)$ over a time period $(0, T)$, with respect to a velocity field $V(t, x)$, in terms of an associated level set function

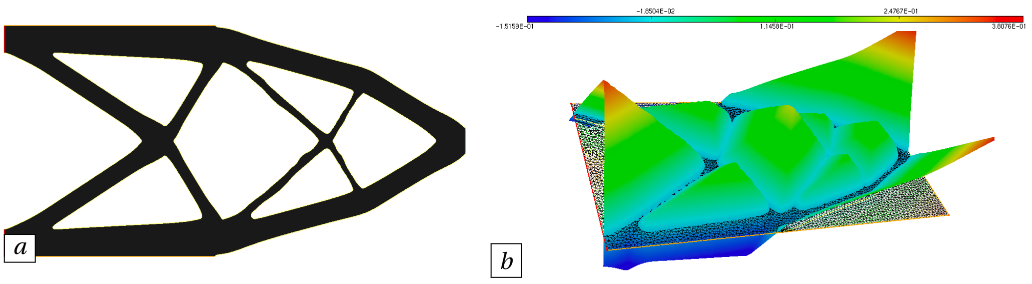

Figure 5. (a) One shape $\Omega \subset \mathbb{R}^2$; (b) graph of an associated level set function ϕ defined on (a mesh of) a larger domain D.

$\phi(t,\cdot)$ (i.e. (14) holds at every time $t > 0$). Formally, the latter satisfies the following “advection-like” equation:

$$\begin{cases} \dfrac{\partial \phi}{\partial t}(t,x) + V(t,x)\cdot\nabla\phi(t,x) = 0 & \text{for } t \in (0,T),\ x \in \mathbb{R}^d, \\ \phi(0,x) = \phi_0(x) & \text{for } x \in \mathbb{R}^d, \end{cases} \tag{15}$$

where ϕ_0 is a level set function for the initial shape $\Omega(0)$. Hence, the intricate geometric evolution problem of $\Omega(t)$ translates into the partial differential equation (15) on the fixed domain D.

From the practical vantage, the computational domain D is equipped with a fixed mesh $\mathscr{T}$, for instance a simplicial mesh, or a finite difference grid. The level set function $\phi(t,x)$ and the velocity field $V(t,x)$ are discretized in time and at the vertices of the mesh $\mathscr{T}$. The evolution equation (15) can then be solved efficiently thanks to an adapted numerical scheme, see for instance Section 4.3.

Remark 3. The above theoretical framework does not rely on any assumption about the nature of the level set function ϕ chosen to represent the shape $\Omega \subset D$. Although, in principle, any function satisfying (14) could be used, it is well-known [33] that the numerical stability of the level set method is significantly improved when ϕ is the signed distance function d_Ω to Ω. The latter is defined by:

$$d_\Omega(x) = \begin{cases} -d(x,\partial\Omega) & \text{if } x \in \Omega, \\ 0 & \text{if } x \in \partial\Omega, \\ d(x,\partial\Omega) & \text{if } x \in D \setminus \overline{\Omega}, \end{cases} \tag{16}$$

where

$$d(x,\partial\Omega) := \min_{p\in\partial\Omega} |x-p| \tag{17}$$

is the usual Euclidean distance from x to $\partial\Omega$. Among others, d_Ω enjoys the desirable “Eikonal” property, whereby its gradient has unit norm wherever it is defined:

$$|\nabla d_\Omega(x)| = 1 \ \text{ for a.e. } x \in D, \tag{18}$$

which expresses a regular spacing of its level sets. Aside from its relevance in the context of the level set method, the signed distance function d_Ω enjoys multiple interesting properties related to the geometry of Ω. Hence, the calculation of the signed distance function to a shape is a topic of interest on its own, see e.g. [10] about the modeling of thickness constraints in shape optimization using distance functions.

3.3.2. *Application of the level set method to shape and topology optimization*

The application of the level set method in the context of shape and topology optimization was originally proposed in [11, 86, 94, 102]. It brings into play a computational domain D equipped with a fixed mesh $\mathscr{T}$ (e.g. simplicial or Cartesian). At any time t, the shape $\Omega(t) \subset D$ is represented by a level set function $\phi(t,\cdot): D \to \mathbb{R}$, discretized, e.g., at the vertices of $\mathscr{T}$. The update of the shape between any two iterations n, $(n+1)$ of the optimization process is efficiently achieved by solving the equation (15) for the evolution of $\phi(t,x)$. In the present context, the velocity field $V(t,x)$ is the descent direction $\theta^n(x)$ for the shape optimization problem (2), which depends on the elastic displacement u_{Ω^n} and the adjoint state p_{Ω^n}.

The bottleneck of this approach lies in the calculation of u_{Ω^n} and p_{Ω^n}. Indeed, at a given iteration n of the process (whose reference in notation is omitted for brevity), no mesh of Ω is available, as Ω is solely known via the function ϕ, defined on (the mesh of) the larger domain D. One idea to conduct this calculation leverages a so-called "fictitious domain approach": the displacement u_Ω is approximated by the solution u_ε to an equation posed on the total domain D. In the present context of linear elasticity, where the traction-free part of shapes is optimized, the void region $D \setminus \overline{\Omega}$ is filled with a very soft "ersatz" material, with Hooke's law εA, $\varepsilon \ll 1$, so that an approximate counterpart to (1) is given by the following equation posed on the fixed domain D:

$$\begin{cases} -\operatorname{div}(A_\varepsilon e(u_\varepsilon)) = 0 & \text{in } D, \\ u_\varepsilon = 0 & \text{on } \Gamma_D, \\ A_\varepsilon e(u_\varepsilon) n = g & \text{on } \Gamma_N, \\ A_\varepsilon e(u_\varepsilon) n = 0 & \text{on } \Gamma, \end{cases} \quad \text{where} \quad A_\varepsilon(x) = \begin{cases} A & \text{if } x \in \Omega, \\ \varepsilon A & \text{if } x \in D \setminus \Omega. \end{cases} \tag{19}$$

see for instance [37] for a justification of this procedure. In practice, the tensor A_ε is easily calculated from the knowledge of the level set function ϕ for Ω.

In a different spirit, advanced finite element techniques, featuring enriched basis functions, have been employed to ensure a more accurate approximation of the displacement u_Ω in such a context where only a fixed mesh of a large computational domain is available; see [50] about the use of the eXtended Finite Element Method (XFEM) and [32] about that of the cut Finite Element Method (cutFEM). Let us point out that both approaches to cope with the absence of a body-fitted mesh for Ω are intrusive, insofar as they do not lend themselves to a black-box use of a commercial finite element solver.

A typical implementation of the level set method for the shape optimization problem (2) is sketched in Algorithm 2. Note that we deliberately omit the details of Step (2), about the practical calculation of a descent direction from the derivatives of the shape functionals at stake, as it will be the focus of the later Section 4.6.

Algorithm 2 The level set method for shape and topology optimization.

```
Initialization:
    • Mesh $\mathscr{T}$ of the computational domain $D$;
    • Level set function $\phi^0 : D \to \mathbb{R}$ representing the initial shape $\Omega^0$.
for $n = 0, \dots,$ until convergence do
    (1) Calculate an approximate version of the elastic displacement $u_{\Omega^n}$ and of the adjoint
        state $p_{\Omega^n}$ on $\mathscr{T}$ by solving the ersatz material problem (19).
    (2) From the theoretical formulas for the shape derivatives $J'(\Omega)(\theta)$, $G'(\Omega)(\theta)$ and
        $H'(\Omega)(\theta)$, infer a descent direction $\theta^n : D \to \mathbb{R}^d$ for (2) according to the selected con-
        strained optimization algorithm.
    (3) Solve the level set advection equation (15) on $\mathscr{T}$ with (time-independent) velocity
        $V(t,x) = \theta^n(x)$, final time $T = \tau^n$ and initial datum $\phi_0 = \phi^n$; a level set function $\phi^{n+1}$
        for the new shape $\Omega^{n+1}$ is obtained.
end for
return Level set function $\phi^n$ for the optimized design $\Omega^n$.
```

Summarizing, the level set method conveniently alleviates meshing issues and allows to describe dramatic evolutions of the optimized shape Ω, including changes in its topology. Unfortunately, it raises the issue of solving the mechanical equations posed on Ω: although the aforementioned fictitious domain approaches (notably, the ersatz material method) and extended finite element methods work reasonably well in the context of linear elasticity, corresponding strategies are not readily available in more challenging physical contexts, such as that of fluid-structure interactions.

The level set based mesh evolution strategy, presented in the next section, is an increment over this level set method for shape and topology optimization: it enjoys all the assets of the latter, and additionally features an exact mesh of the shape at each stage of the process.

4. The level-set based mesh evolution method for shape and topology optimization

We now turn to the numerical framework introduced in [5–7] as an attempt to overcome the individual difficulties posed by Lagrangian and level set methods when tracking the evolution of the shape during the optimization process. To set ideas, the presentation unfolds in the structural optimization context of Section 2.1, where one aims to solve a problem of the form (2). After sketching the main stages of the method in Section 4.1, we overview its main ingredients in the subsequent sections.

4.1. *General description of the method*

As the name suggests, the level set based mesh evolution method at stake in this article combines the meshed and level set representations of shapes discussed in Sections 3.2 and 3.3. Efficient algorithms make it possible to switch from one to the other, so that the most convenient of them with respect to the ongoing operation can be used.

Let D be a large "hold-all" domain containing all the considered shapes Ω. At each iteration n of the process, D is endowed with a valid and conforming mesh $\mathscr{T}^n$, which is modified from one iteration to the other so that the current shape Ω^n is explicitly discretized. More precisely, $\mathscr{T}^n$ is consistently made of two complementary submeshes $\mathscr{T}^n_{\text{int}}$, $\mathscr{T}^n_{\text{ext}}$ such that $\mathscr{T}^n_{\text{int}}$ is a valid, conforming mesh of Ω^n and $\mathscr{T}^n_{\text{ext}}$ is a valid, conforming mesh of the exterior part $D \setminus \overline{\Omega^n}$. Thus, two descriptions of Ω^n are available:

- *A meshed representation.* Ω^n is explicitly represented by the submesh $\mathscr{T}^n_{\text{int}}$ of $\mathscr{T}^n$, see Figure 6 (a).
- *A level set representation.* Ω^n is implicitly defined by a level set function ϕ^n, defined on the whole mesh $\mathscr{T}^n$ of D, see Figure 6 (b).

On the one hand, the meshed representation is handful when it comes to solving partial differential equations on Ω^n, such as that (1) for the elastic displacement u_{Ω^n}; on the other hand, the level set description is adequate for conducting the update of the shape Ω^n into the next iterate Ω^{n+1} in a robust way.

As we have mentioned, this strategy crucially hinges on efficient numerical methods for passing from one of these descriptions to the other, and notably:

- An algorithm for generating one level set function $\phi : D \to \mathbb{R}$ for a shape $\Omega \subset D$ which is explicitly discretized inside a mesh $\mathscr{T}$ of the computational domain D;
- An algorithm which assumes the datum of a level set function $\phi : D \to \mathbb{R}$ defined on a mesh $\mathscr{T}$ of D, associated to a shape $\Omega \subset D$, and creates a new mesh $\widetilde{\mathscr{T}}$ of D in which Ω is explicitly discretized.

This method is outlined in Algorithm 3 in the context of the model shape optimization problem of Section 2. Its main steps are illustrated on Figure 6; as they constitute operations of general interest, we present them in a self-contained way in the next sections. The computation of the signed distance function to a shape is discussed in Section 4.2, and the resolution of the level set advection equation is presented in Section 4.3. The aspects of the method concerned with remeshing are addressed in Section 4.4; we then broach the versatile Hilbertian extension and regularization procedure in Section 4.5, before finally describing the numerical constrained optimization algorithm in Section 4.6.

Algorithm 3 The level set based mesh evolution method.

```
Initialization: Mesh $\mathscr{T}^0$ of the computational domain $D$, enclosing a mesh $\mathscr{T}^0_{\text{int}}$ of $\Omega^0$ as a submesh.
for $n = 0, \dots,$ until convergence do
    (1) Calculate the signed distance function $\phi^n = d_{\Omega^n}$ to $\Omega^n$ at the vertices of $\mathscr{T}^n$.
    (2) Calculate the displacement $u_{\Omega^n}$ and the adjoint state $p_{\Omega^n}$ by solving the partial differential equation (1) on the interior part $\mathscr{T}^n_{\text{int}}$ of $\mathscr{T}^n$.
    (3) Use the Hilbertian procedure to calculate gradients $\theta^n_J, \theta^n_G$ and $\theta^n_H$ for $J(\Omega)$, $G(\Omega)$ and $H(\Omega)$ at $\Omega = \Omega^n$ on the whole mesh $\mathscr{T}^n$.
    (4) Infer a descent direction $\theta^n$ on $\mathscr{T}^n$ for (2) thanks to a constrained optimization algorithm.
    (5) Choose a small enough time step $\tau^n$ and calculate a level set function $\widetilde{\phi}^{n+1}$ for the new shape $\Omega^{n+1} := (\text{Id} + \tau^n\theta^n)(\Omega^n)$ on the mesh $\mathscr{T}^n$ by solving the level set evolution equation (15) over $(0, \tau^n)$ with velocity field $V(t,x) = \theta^n(x)$ and initial condition $\phi_0 = \phi^n$.
    (6) Create a new mesh $\mathscr{T}^{n+1}$ of $D$ where $\Omega^{n+1}$ is explicitly discretized, from the datum of the level set function $\widetilde{\phi}^{n+1}$ on $\mathscr{T}^n$.
end for
return Mesh $\mathscr{T}^n$ of $D$ where $\Omega^n$ is discretized as a submesh $\mathscr{T}^n_{\text{int}}$.
```

Remark 4. Appealing features of this approach are its modularity and non intrusiveness: the aforementioned steps can be carried out with independent numerical codes. In particular, the resolution of the state (and adjoint) equation for the elastic displacement does not demand any exotic treatment; it can be realized with any solver, without entering into its implementation details.

4.2. *Calculation of the signed distance function*

This section deals with the construction of one particular level set function ϕ for a shape $\Omega \subset D$, namely the signed distance function d_Ω in (16), out of a meshed representation of the latter (Step (1) in Algorithm 3), see the discussion in Remark 3.

Let the computational domain D be endowed with a simplicial mesh $\mathscr{T}$ and let $\Omega \subset D$ be a shape; we wish to calculate the values $\phi(x) = d_\Omega(x)$ at all vertices $x \in \mathscr{T}$. Note that in the shape optimization workflow of Section 4.1, Ω is supplied as an explicit submesh $\mathscr{T}_{\text{int}}$ of $\mathscr{T}$, but for the purpose of this section, it could actually be given under a different format, for instance, via a proper mesh $\mathscr{T}_\Omega$ which is not necessarily a part of $\mathscr{T}$.

The numerical calculation of d_Ω can be conducted in a variety of manners. "Geometric" algorithms involve an exhaustive calculation of the distance $d(x, \partial\Omega)$ from x to $\partial\Omega$ at every vertex x of $\mathscr{T}$, defined as the minimum value featured in (17). Although this operation can be made

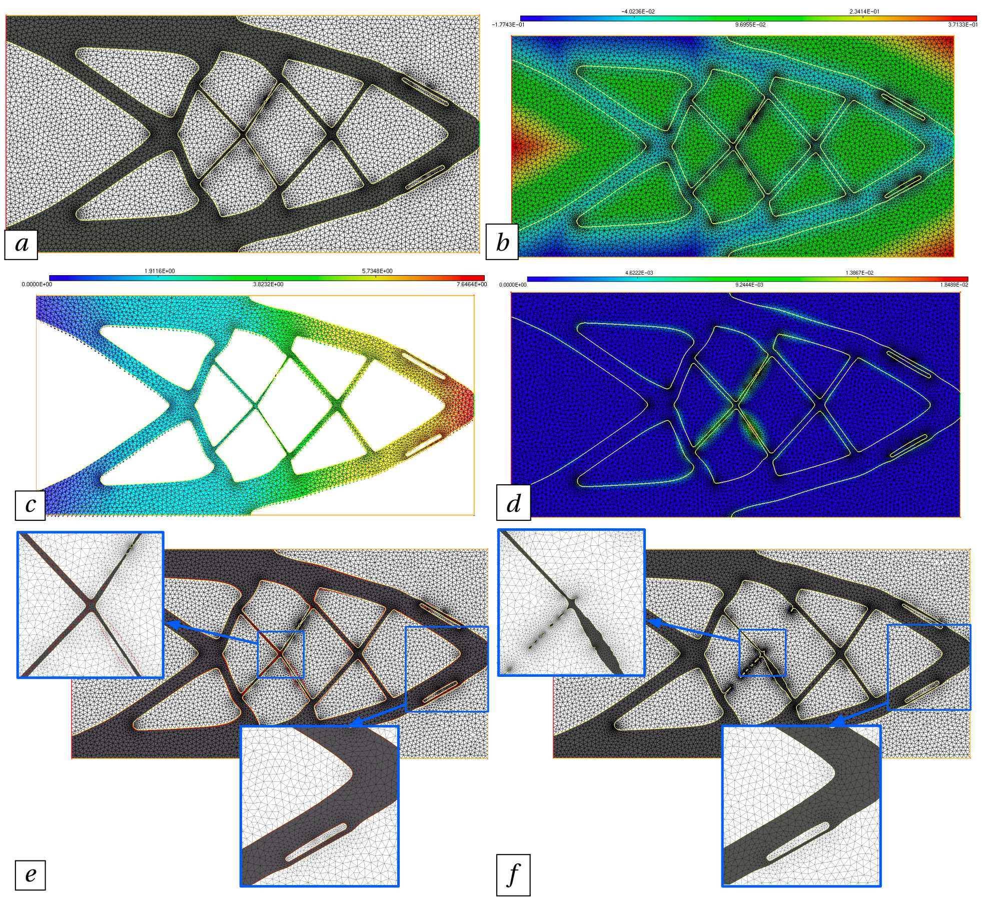

Figure 6. Illustration of the main stages of the mesh evolution method sketched in Section 4.1. (a) Mesh $\mathscr{T}^n$ of the computational domain D; the submesh $\mathscr{T}^n_{\text{int}}$ of Ω^n consists of the black elements, and that $\mathscr{T}^n_{\text{ext}}$ of $D \setminus \overline{\Omega^n}$ is made of the white elements; (b) isovalues of the signed distance function ϕ^n to Ω^n, calculated on $\mathscr{T}^n$; (c) solution u_{Ω^n} to the elasticity system on the mesh $\mathscr{T}^n_{\text{int}}$ of Ω^n; (d) descent direction θ^n on the whole mesh $\mathscr{T}^n$; (e) level set function $\widetilde{\phi}^{n+1}$ on the mesh $\mathscr{T}^n$; the 0 level set of $\widetilde{\phi}^{n+1}$ is depicted in red; (f) new mesh $\mathscr{T}^{n+1}$ with the new shape Ω^{n+1} enclosed as the submesh $\mathscr{T}^{n+1}_{\text{int}}$ (made of the black elements).

efficient owing to a number of heuristics – see e.g. [79, 100] and the references therein – we rather rely on "propagation algorithms", which comprise two steps:

(1) The function ϕ is initialized with (an approximation of) the exact value of d_Ω at the vertices of $\mathscr{T}$ which are "close" to $\partial\Omega$ (for instance, at the vertices of the simplices $T \in \mathscr{T}$ intersecting $\partial\Omega$), and with large, positive or negative values elsewhere. This stage is elementary; however, depending on the input format and of the complexity of the geometry of Ω, it may prove tedious from the implementation viewpoint, and time-consuming in practice, see e.g. [40].

(2) The calculation of the signed distance $\phi(x) = d_\Omega(x)$ is realized from the vertices $x \in \mathscr{T}$ closest to $\partial\Omega$ to farther ones, by relying on a discretization of the Eikonal equation (18).

This purpose is greatly simplified when the computational mesh $\mathscr{T}$ is a Cartesian grid, since high-order finite different schemes are available.

The most popular algorithm in this second category is certainly the fast marching method, see [92] for an overview and [72] for an adaptation to the case where the computational mesh is simplicial; let us also mention the fast sweeping method [104].

In the present work, we rely on an algorithm based on the properties of the so-called redistancing equation, which is described in [40] and is available in the open-source library `mshdist`[1].

4.3. *Advection of the level set function*

In this section, we turn to the numerical resolution of the equation (15) accounting for the evolution of the shape.

The generic situation, occurring at each iteration n of the process described in Section 4.1, is the following: a level set function $\phi = \phi^n : D \to \mathbb{R}$ accounting for the current shape $\Omega = \Omega^n$ is supplied via its values at the vertices of a simplicial mesh $\mathscr{T} = \mathscr{T}^n$ of the computational domain D. Analogously, the velocity field $V : D \to \mathbb{R}^d$ driving the evolution of the shape, which is the descent direction θ^n at the current iteration, is supplied at the vertices of $\mathscr{T}$. We aim to compute the solution $\psi(t,x)$ to the following advection equation:

$$\begin{cases} \dfrac{\partial \psi}{\partial t}(t,x) + V(x)\cdot\nabla\psi(t,x) = 0 & \text{for } t \in (0,T),\ x \in D, \\ \psi(0,x) = \phi(x) & \text{for } x \in D, \end{cases} \tag{20}$$

and notably its values $\psi(T,x)$ at the final time $t = T$, which stands for the time step τ^n.

Numerical methods for the resolution of (20) are manifold when the computational support is a Cartesian grid, which allows for the use of high-order finite difference methods. This issue is however a little less classical and deserves a few comments in our context where it is a simplicial mesh. We rely on the method of characteristics proposed in [90], which is close in spirit to the semi-Lagrangian scheme developed in [97]; see alternatively [52] about the use of discontinuous Galerkin methods, and [1,21] for the construction of numerical schemes for more general Hamilton–Jacobi equations on simplicial meshes.

The method of characteristics is based on the analytical formula for the solution to (20). The latter is expressed in terms of the characteristic curves $t \mapsto X(t,t_0,x)$ of the velocity field $V(x)$, emerging from an arbitrary point $x \in D$ at a time t_0, defined as the solution to the ordinary differential equation:

$$\begin{cases} \dfrac{\mathrm{d}}{\mathrm{d}t} X(t,t_0,x) = V(X(t,t_0,x)) & \text{for } t \in \mathbb{R}, \\ X(t_0,t_0,x) = x. \end{cases} \tag{21}$$

Intuitively, $t \mapsto X(t,t_0,x)$ is the trajectory of a particle located in x at time $t = t_0$, which is transported according to the velocity field $V(x)$. The exact solution to (20) is then given by:

$$\forall\ t \in (0,T),\ x \in D, \quad \psi(t,x) = \phi(X(0,t,x)), \tag{22}$$

which expresses the natural fact that the value of ψ at time t and point x is the value of the initial datum ϕ at the initial position $X(0,t,x)$ of the particle lying in x at time t.

One simple means to exploit the closed-form formula (22) is to directly discretize it: for each vertex $x \in \mathscr{T}$, the ordinary differential equation (21) is solved for the "backward" characteristic curve $t \mapsto X(t,T,x)$, for instance by a Runge–Kutta 4 scheme; ϕ is then evaluated at the "foot" $X(0,T,x)$ of this characteristic line.

[1] `https://github.com/ISCDtoolbox/Mshdist`

Remark 5. In practice, the origin $X(0,T,x)$ of the characteristic curve passing through x at $t=T$ may lie outside D. This notably happens when the velocity field $V(x)$ is pointing inward D on at least one portion of the boundary ∂D. In such a case, the information supplied in (20) is not sufficient to guarantee the well-posedness of this equation (as one would have to add a boundary condition on the "entrant" part of ∂D); in numerical practice, it is customary to endow $\psi(T,x)$ with a consistent value by extrapolating the value of ϕ (and the characteristic curve $t \mapsto X(t,T,x)$) outside D.

An open-source implementation of this algorithm is proposed in the `advection`[2] code.

4.4. *Using remeshing to pass from a level set to a meshed representation of a shape*

This section is devoted to the delicate operation of passing from a level set to a meshed representation of a shape Ω, which is Step (6) in Algorithm 3. Let us consider the following generic situation: the computational domain D is equipped with a simplicial mesh $\mathscr{T}$, and a level set function $\phi : D \to \mathbb{R}$ for a shape $\Omega \subset D$ is provided via its values at the vertices of $\mathscr{T}$; we aim to construct a new mesh $\widetilde{\mathscr{T}}$ of D in which Ω appears as a submesh, see Section 4.1.

This task is achieved in two steps:

(1) The 0 level set of ϕ is explicitly discretized into $\mathscr{T}$. This crude procedure yields a valid, conforming mesh $\mathscr{T}_{\text{temp}}$ of D, in which Ω exists as a submesh; unfortunately, $\mathscr{T}_{\text{temp}}$ has poor finite element quality, and the interior submesh is a poor geometric approximation of the shape Ω; see Figure 7 (b).
(2) Local remeshing operations are applied to modify $\mathscr{T}_{\text{temp}}$ into a new, high-quality mesh $\widetilde{\mathscr{T}}$ of D in which Ω is still explicitly discretized, see Figure 7 (c).

These two steps are presented with a little more details in the next Sections 4.4.1 and 4.4.2, respectively. They are implemented in the general purpose open-source library `mmg`[3] devoted to simplicial remeshing; we refer to [39] for a more exhaustive presentation of the latter, and to [18] for recent developments on the subject.

Throughout the rest of this section, for notational simplicity, the mesh of D will be systematically denoted by $\mathscr{T}$, although it is constantly subject to modifications.

4.4.1. *Explicit discretization of the* 0 *level set of* ϕ *in the mesh* $\mathscr{T}$

From a simplicial mesh $\mathscr{T}$ of D and a level set function $\phi : D \to \mathbb{R}$ for a subdomain $\Omega \subset D$, supplied at the vertices of $\mathscr{T}$, we aim to construct a new, valid but possibly ill-shaped mesh of D in which Ω is explicitly discretized.

To achieve this, we rely on a simplicial variant of the well-known marching cubes algorithm [78], namely the marching triangles (in 2d) or tetrahedra (in 3d) algorithm [48]. Briefly, we first identify all the elements $T \in \mathscr{T}$ which intersect the 0 level set of ϕ, by looking at the signs of ϕ at their vertices. After linear interpolation of ϕ inside any of these elements T, the intersection $\partial\Omega \cap T$ is a portion of line (in 2d) or plane (in 3d) which is completely determined by the values of ϕ at the vertices of T. Then, using a pre-defined pattern, T is split into several sub-triangles (in 2d) or sub-tetrahedra (in 3d), so that $\partial\Omega \cap T$ explicitly appears in the resulting mesh, see Figure 8.

The mesh $\mathscr{T}$ resulting from this rough operation is valid, conforming, and it encloses two submeshes $\mathscr{T}_{\text{int}}$ and $\mathscr{T}_{\text{ext}}$ of Ω and $D \setminus \overline{\Omega}$, respectively. Unfortunately, it generally contains elements with very bad quality, and it accounts for a poor geometric approximation of $\partial\Omega$.

[2]`https://github.com/ISCDtoolbox/Advection`
[3]`https://github.com/MmgTools/mmg`

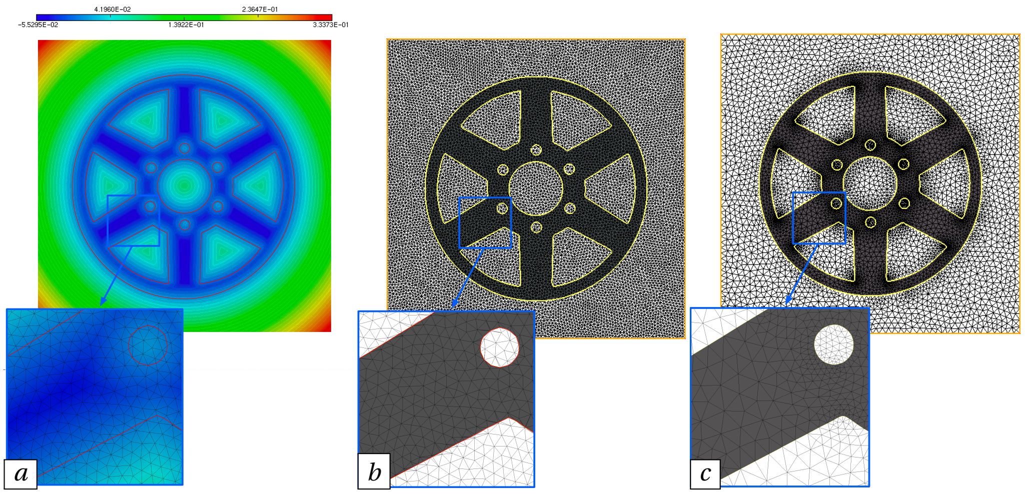

Figure 7. (a) Isovalues of a level set function discretized at the vertices of a mesh $\mathscr{T}$ of a computational domain D; (b) Ill-shaped mesh $\mathscr{T}_{\text{temp}}$ resulting from the rough discretization of the 0 level set of ϕ into $\mathscr{T}$; (c) High-quality mesh $\widetilde{\mathscr{T}}$ obtained from $\mathscr{T}_{\text{temp}}$ by remeshing.

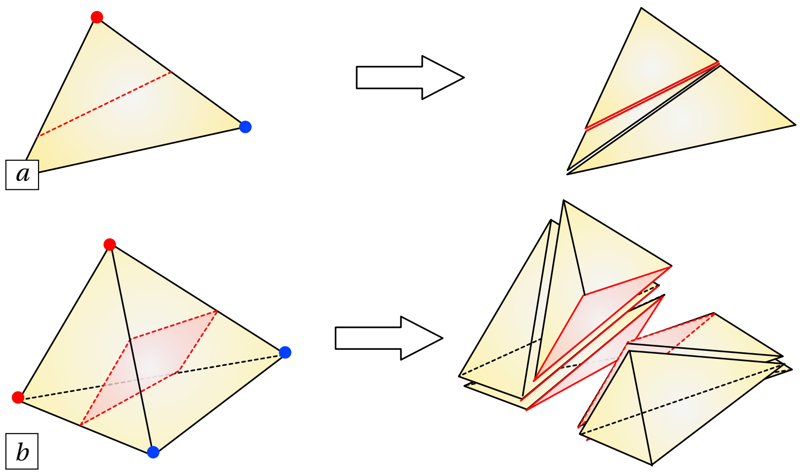

Figure 8. (a) One pattern for splitting a triangle, based on the values of ϕ at its vertices (ϕ is positive at the red vertex, negative at the blue ones); (b) one possible pattern for splitting a tetrahedron.

4.4.2. *Improvement of the quality of* $\mathscr{T}$ *by local remeshing operations*

The algorithm of the previous section has produced a valid and conforming mesh $\mathscr{T}$ of the domain D, in which the considered shape $\Omega \subset D$ explicitly appears as a submesh. Unfortunately, $\mathscr{T}$ has bad quality, in both senses introduced in Section 3.1: it contains nearly degenerate elements, and it accounts for a poor geometric approximation of the boundary of D and of its internal interfaces, in particular of the boundary $\partial\Omega$ of the shape, see again Figure 7 (b). We wish to modify $\mathscr{T}$ by repeatedly applying local remeshing operations, so as to improve both features.

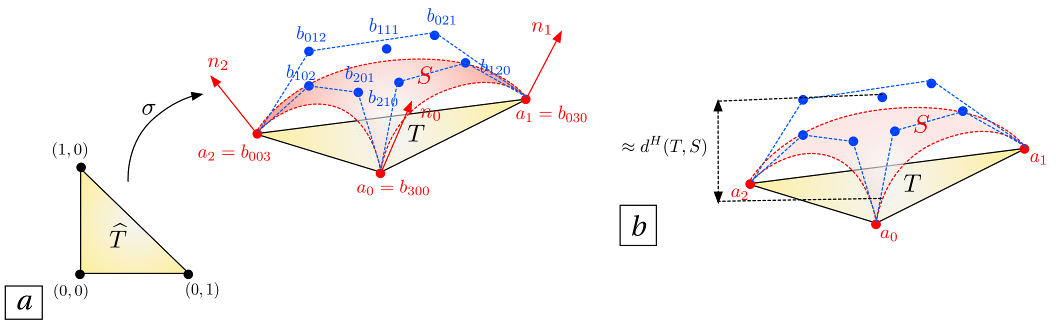

Figure 9. (a) Creation of a cubic Bézier patch for the region of $\partial\Omega$ associated to a surface triangle $T \in \mathscr{S}_{\mathscr{T}}$; (b) measurement of the gap between the continuous and discrete domains from this local patch.

The presentation of this section focuses on the case of three space dimensions, since it contains numerous specificities, and it is on any aspect more involved than its 2d counterpart.

One immediate issue posed by the desire to better approximate the domains D and Ω is that no continuous description is provided: the only available information about D or Ω is the supplied discrete mesh $\mathscr{T}$ itself. To cope with this difficulty, we first reconstruct local geometric information about D and Ω from the discrete datum of $\mathscr{T}$; this information will be updated throughout the remeshing process. More precisely, normal vectors $n(x)$ are calculated at the vertices x of the surface mesh $\mathscr{S}_{\mathscr{T}}$, for instance as weighted sums of the normal vectors to the nearby surface triangles. Besides, remarkable geometric entities such as sharp edges and corner points are identified. This information allows to infer a piece of the "ideal" continuous surface ∂D or $\partial\Omega$ associated to a given surface triangle $T \in \mathscr{S}_{\mathscr{T}}$: a cubic Bézier parametrization $\sigma : \widehat{T} \to \mathbb{R}^3$ (defined on the reference simplex $\widehat{T} \subset \mathbb{R}^2$) is calculated, which passes through the three vertices a_0, a_1, a_2 of T and fits the attached normal vectors n_0, n_1, n_2, see Figure 9 (a). This local continuous model is helpful for a variety of purposes, in particular when it comes to measuring "how far" $\mathscr{T}$ stands from the continuous domains D and Ω, as depicted in Figure 9 (b).

Another key component of the remeshing strategy is the construction and the use of a size map $h : D \to \mathbb{R}$, which encodes at each vertex $x \in \mathscr{T}$ the desired size for edges surrounding x. This value is calculated from user-defined requirements, such as lower and upper bounds for the size of edges in $\mathscr{T}$, or a more detailed local size prescription stemming from a priori or a posteriori error estimates related to the finite element method. Additionally, when x is a surface vertex, the calculation of $h(x)$ has to take into account the needed local size to comply with the geometric approximation requirements.

The remeshing procedure then starts, properly speaking. The elements of $\mathscr{T}$ are travelled repeatedly; the action of four local remeshing operators is simulated, and their outcome is effectively retained if it shows an improved mesh quality. These operators are quite commonly used in remeshing practice; their action is exemplified on Figure 10 in the case of two space dimensions, and it can be summarized as follows:

- *Edge split.* When an edge pq in the mesh is "too long" (with respect to the size map h), it is split into two edges pm and qm after introduction of a new point m in $\mathscr{T}$; the tetrahedra that were formerly in the shell of pq – i.e. that were sharing pq as an edge – are appropriately reconnected.
- *Edge collapse.* When an edge pq is "too short", q is merged with p; the tetrahedra in the shell of pq are suppressed, and the other elements formerly connected to q are suitably updated.

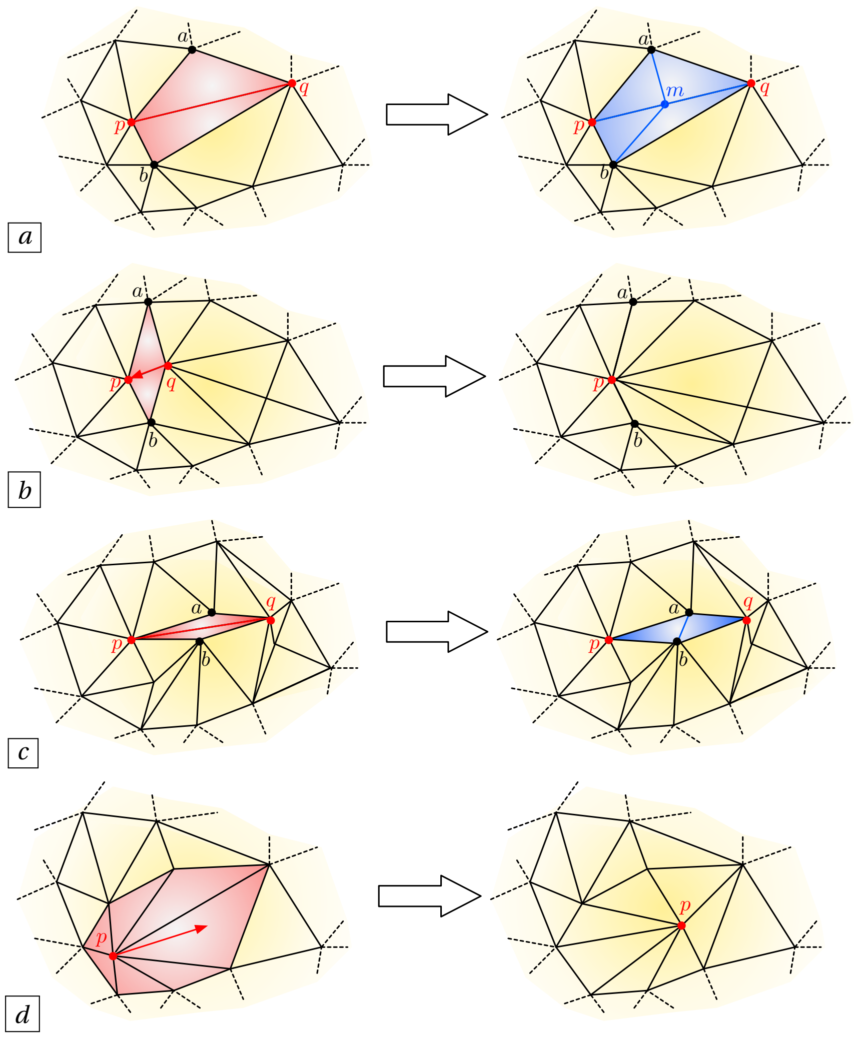

Figure 10. Two-dimensional illustration of the four remeshing operations described in Section 4.4.2. (a) Split of the edge pq: the midpoint m is introduced, and the two red triangles are replaced by the four blue triangles; (b) Collapse of vertex q onto p; the “shell” of pq, made of the two red triangles, is removed; (c) Swap of the edge pq; the two red triangles are replaced by the two blue triangles; (d) Relocation of the vertex p, while maintaining the connectivities of the mesh.

- *Edge swap.* One edge pq in the mesh is suppressed and the elements in the shell of pq are reconnected so that the mesh remains valid and conforming.
- *Vertex relocation.* One vertex p of the mesh is assigned to a new position, while all the mesh connectivities are unaltered.

Each operator exists under two versions, dedicated to surface and internal configurations, respectively. For instance, when a boundary edge pq is split, the new vertex m ought to be inserted directly on the curved segment on ∂D or $\partial\Omega$ which is associated to pq via the local surface model σ discussed above; on the contrary, when pq is an internal edge of $\mathscr{T}$, m is simply introduced at the center of pq.

4.5. *Extension - regularization of shape derivatives via the Hilbertian method*

The shape derivatives $J'(\Omega)(\theta)$, $G'(\Omega)(\theta)$ and $H'(\Omega)(\theta)$ of the objective and constraint functions $J(\Omega)$, $G(\Omega)$ and $H(\Omega)$ are continuous linear forms on $W^{1,\infty}(\mathbb{R}^d,\mathbb{R}^d)$, which is a Banach space. The nature and structure of these derivatives do not lend themselves to a simple numerical treatment.

This fact is conveniently illustrated by the unconstrained shape optimization problem (6). As we have mentioned, when the shape derivative of $J(\Omega)$ is known under the structure (11), a descent direction is immediately revealed as $\theta = -v_\Omega n$, see (12). However appealing, this choice is awkward, since this formula only characterizes the values of θ on the boundary $\partial\Omega$, while we have seen that for a variey of purposes, including the practice of Lagrangian mesh update strategies as in Section 3.2, or that of the level set method described in Section 3.3, θ ought to be "adequately" defined on the total computational domain D. Moreover, in practice, the choice (12) results in a descent direction which lacks smoothness; this often causes the shape to develop numerical artifacts in the course of the optimization, see [80] for a discussion about this feature.

Returning to the treatment of a general problem of the form (2), many efficient infinite-dimensional optimization algorithms leverage a Hilbertian structure when it comes to calculating descent directions, see for instance the algorithm described in the next Section 4.6.

One handful practice to circumvent these difficulties, with countless additional benefits, is the so-called Hilbertian extension and regularization method, which consists in endowing the space of deformations θ with the structure of a Hilbert space, composed of vector fields obeying certain user-specified properties, such as their smoothness, their domain of definition, or imposed values on particular regions of space (e.g. $\theta = 0$ where shapes are not subject to optimization). We refer to [16, 31, 43] about this idea, see also [8] for a summary.

The key idea is to introduce a Hilbert space V, with inner product $a(\cdot,\cdot)$, which is continuously embedded in $W^{1,\infty}(\mathbb{R}^d,\mathbb{R}^d)$. Thus, the derivative $J'(\Omega)(\theta)$ of a function of the domain $J(\Omega)$ induces a continuous linear form on V; according to the Riesz representation theorem, we may represent the latter by the element $\theta_J \in V$ defined by:

$$\forall\, w \in V, \quad a(\theta_J, w) = J'(\Omega)(w). \tag{23}$$

This element $\theta_J \in V$ is the gradient associated to the derivative $\theta \mapsto J'(\Omega)(\theta)$ via the inner product $a(\cdot,\cdot)$. As expected, $-\theta_J$ is a descent direction for $J(\Omega)$ since

$$J'(\Omega)(-\theta_J) = -a(\theta_J, \theta_J) \leq 0,$$

where the latter quantity vanishes if and only if Ω is already a critical shape for $J(\Omega)$.

Multiple possibilities are available as regards the choice of the space V and its inner product $a(\cdot,\cdot)$, see again [31]. For instance, one natural choice is:

$$V = H^m(D)^d, \quad \text{with the usual inner product } a(u,v) := \sum_{|\alpha|\leq m} \int_D \partial^\alpha u \cdot \partial^\alpha v \,\mathrm{d}x,$$

where the index m is chosen large enough so that the continuous inclusion $V \subset W^{1,\infty}(\mathbb{R}^d,\mathbb{R}^d)$ holds by virtue of the Sobolev embedding theorem [2]. The gradient $\theta_J \in V$ produced by the identification problem (23) is thus a vector field on the whole computational domain D (and not only on the boundary $\partial\Omega$ of the shape), and it enjoys the higher regularity of a vector field in $H^m(D)$.

Another popular practice, albeit formal, is that retained in this article: only the normal component of the velocity field θ is extended and regularized. More precisely, we consider the Hilbert space

$$V = H^1(D), \quad \text{with inner product } a(u,v) = \alpha^2 \int_D \nabla u \cdot \nabla v \,\mathrm{d}x + \int_D uv \,\mathrm{d}x, \tag{24}$$

where $\alpha > 0$ is a user-defined parameter, and we solve the following identification problem:

$$\text{Search for } v \in V \text{ s.t. } \forall\, w \in V, \quad a(v,w) = J'(\Omega)(wn), \tag{25}$$

where n stands for an extension of the unit normal vector to $\partial\Omega$. We then take $\theta_J = vn$ as the "gradient" of $J(\Omega)$, which is defined on the total computational domain D. Although not rigorous, since $H^1(D)^d$ is not embedded into $W^{1,\infty}(\mathbb{R}^d,\mathbb{R}^d)$, this practice yields good results; intuitively, the identification problem (24) and (25) amounts to smoothing of the "$L^2(\partial\Omega)$ gradient" v_Ω of $J(\Omega)$ featured in (11) over a thickness α around $\partial\Omega$.

4.6. *A null space algorithm for constrained optimization*

Only relatively few of the numerous constrained optimization algorithms (see e.g. [83] for an overview) lend themselves to an efficient treatment of infinite-dimensional problems, and in particular shape optimization problems. Let us nevertheless mention the article [98] introducing the popular Method of Moving Asymptotes (MMA) in the context of density-based topology optimization, or that [49] about the adaptation of the Sequential Linear Programming (SLP) method to the shape optimization context. In this section, we present a numerical algorithm for dealing with constrained optimization which is particularly well-suited for our applications; it is presented in detail in [57][4], see also the article [19] where a similar idea was developed independently. For simplicity, we restrict ourselves to a heuristic presentation in the case of an optimization problem featuring only equality constraints, although the method allows to treat an arbitrary number of equality and inequality constraints.

We consider a shape optimization problem of the form

$$\min_{\Omega} J(\Omega) \text{ s.t. } G(\Omega) = 0, \tag{26}$$

where $G(\Omega) = (G_i(\Omega))_{i=1,\dots,p} \in \mathbb{R}^p$ is a collection of p real-valued equality constraint functionals.

Let us place ourselves at a given iteration n of the execution of Algorithm 3, dropping all superscripts referring to the latter for notational simplicity. The current shape is denoted by Ω, and we assume that a Hilbert space V and an inner product $a(\cdot,\cdot)$ have been chosen for the considered deformation fields θ according to the Hilbertian method of Section 4.5. As discussed in there, the shape derivatives $J'(\Omega)$ and $G_i'(\Omega)$ are accounted for by gradients θ_J, $\theta_{G_i} \in V$, $i = 1,\dots,p$, that is:

$$\forall\, \xi \in V, \quad J'(\Omega)(\xi) = a(\theta_J, \xi), \text{ and } G_i'(\Omega)(\xi) = a(\theta_{G_i}, \xi).$$

The next iterate $(\mathrm{Id} + \tau\theta)(\Omega)$ in the solution of the shape optimization problem (26) is obtained from a deformation θ which is sought as a linear combination of θ_J and the θ_{G_i}; we may write the latter under the form

$$\theta = -\left(\alpha_J \xi_J + \alpha_G \xi_G\right), \quad \text{where } \begin{cases} \xi_J = \theta_J - \sum_{i=1}^p \lambda_i \theta_{G_i}, \\ \xi_G = \sum_{i=1}^p \beta_i \theta_{G_i}, \end{cases} \tag{27}$$

and the coefficients λ_i, β_i are characterized by the following requirements:

- The vector field $\xi_J \in V$ is a descent direction for $J(\Omega)$ lying in the null-space of the constraint functional $G(\Omega)$, i.e.

$$\forall\, i = 1,\dots,p, \quad G_i'(\Omega)(\xi_J) = a(\theta_{G_i}, \xi_J) = 0.$$

[4]An open source implementation is available at the address:

`https://gitlab.com/florian.feppon/null-space-optimizer`

This property rewrites under the form of a $p \times p$ matrix system for the coefficients $\lambda = (\lambda_i)_{i=1,\dots,p}$:

$$S\lambda = b, \quad \text{where } S_{ij} := a(\theta_{G_i}, \theta_{G_j}), \text{ and } b_i := a(\theta_{G_i}, \theta_J).$$

- The contribution $\xi_G \in V$, which is then the only one able to modify the value of the constraint functional, should ensure a reduction in the absolute value of $G(\Omega)$ by a unit:

  $$G((\mathrm{Id} + \tau\theta)(\Omega)) = (1 - \alpha_G \tau) G(\Omega).$$

  This requires that $|1 - \alpha_G \tau| < 1$, that is $\alpha_G < \frac{2}{\tau}$. Using the asymptotic expansion (8), one obtains, at first order:

  $$\forall\, i = 1, \dots, p, \quad \sum_{j=1}^{p} \beta_j G_i'(\Omega)(\theta_{G_j}) = -G_i(\Omega),$$

  which leads to the $p \times p$ matrix system:

  $$S\beta = -c, \text{ where } c_i := -G_i(\Omega).$$

Intuitively, the weights α_J and α_G in (27) encode the relative rates at which the algorithm imposes the reduction in the value of the objective function and the fulfillment of the constraints.

The practical implementation of this strategy relies on the following adaptations of these considerations:

- The coefficients α_J and α_G in (27) are actually adapted from one iteration to the other, in order to control the maximum amplitude $\|\theta\|_{L^\infty(D)^d}$ of the descent direction θ; we set:

  $$\alpha_J = \begin{cases} \dfrac{A_J h}{\|\xi_J\|_{L^\infty(D)^d}} & \text{if the iteration number } n \text{ is } \le n_0, \\[2ex] \dfrac{A_J h}{\max(\|\xi_J\|_{L^\infty(D)^d}, \Xi)} & \text{otherwise,} \end{cases}$$
  $$\text{and} \quad \alpha_G = \min\left(\frac{A_G h}{\|\xi_G\|_{L^\infty(D)^d}}, 2 \right),$$

  where h is the average size of an edge in the mesh, and A_J, A_G are fixed ratios accounting for the largest displacement (relatively to the mesh size) of the boundary $\partial\Omega$ induced by the requirements to reduce the value of the objective function and to satisfy the constraints, respectively.

  In the above formula, a given number of iterations n_0 is introduced, and the normalization of ξ_J expressed in the definition of α_J is only applied during the first n_0 iterations of the optimization process; when $n > n_0$, the normalization factor is replaced by the maximum between the norm $\|\xi_J\|_{L^\infty(D)^d}$ of the current direction ξ_J, and its value Ξ at iteration n_0. This leaves the room for the contribution $\alpha_J \xi_J$, which is in charge of making the value of $J(\Omega)$ decrease, to tend to 0 as n grows, and thus to avoid oscillations of the algorithm in the final iterations of the method.
- A merit function is used to appraise the size of the step Δt during which the direction θ in (27) allows to improve Ω with respect to the problem (26). More precisely, we introduce the augmented Lagrangian-like shape functional

  $$M(\Omega) := \alpha_J \left(J(\Omega) - \sum_{i=1}^{p} \lambda_i G_i(\Omega) \right) + \frac{\alpha_G}{2} S^{-1} G(\Omega) \cdot G(\Omega), \tag{28}$$

  whose shape gradient (with respect to the inner product $a(\cdot,\cdot)$) is precisely the deformation field θ in (27). The time step Δt must then be chosen in such a way that $M((\mathrm{Id} + \Delta t\theta)(\Omega)) < M(\Omega)$, at least up to a certain tolerance.

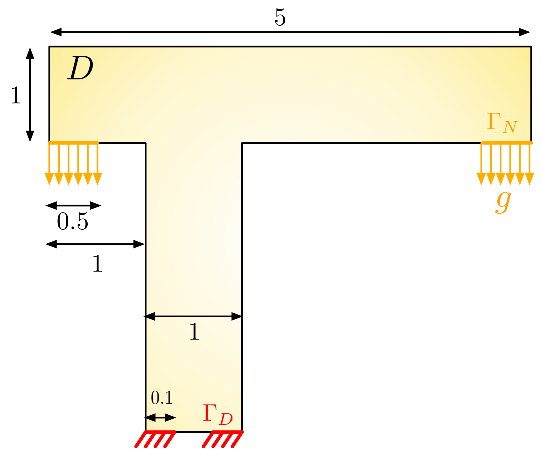

Figure 11. Setting of the 2d crane optimization example of Section 5.1.

Remark 6. One interesting feature of this algorithm is that it does not involve many user-defined parameters which may prove difficult to tune: only A_J and A_G have to be specified, whose physical meaning is quite clear. The ratio A_J/A_G allows to control the pace at which constraints become satisfied.

5. Numerical illustrations

In this section, we present several numerical examples illustrating the main features of our shape optimization framework, in physical situations where a body-fitted approach is particularly desirable. The first Section 5.1 arises in the exact physical setting of 2d linearly elastic structures introduced in Section 2.1. We then turn to more complex physical situations; in Section 5.2, we consider the optimization of a two-dimensional thermoelastic device; in Section 5.3, we deal with the three-dimensional optimization of an aerodynamic profile, and finally, in Section 5.4, we consider the optimization of a three-dimensional thermal-fluid heat exchanger.

5.1. *Optimization of the shape of a two-dimensional crane*

Our first example deals with the optimal design of a 2d crane, as depicted on Figure 11. The considered shapes Ω are enclosed in a computational box D with size 5×5; they are clamped on two small regions Γ_D near the corners of their lower part, and uniform traction loads $g = (0,-1)$ are applied on the two regions Γ_N located at the left and right ends of their upper part, accounting for the counterweight of the crane and the weight of the lifted object, respectively.

The shapes are made of a linearly elastic material with normalized Lamé parameters $\lambda = 0.5769$ and $\mu = 0.3846$. As presented in Section 2.1, the physical behavior of the shape in this situation is described by the linearized elasticity system (1), and we solve the problem

$$\min_{\Omega} C(\Omega) \text{ s.t. } \mathrm{Vol}(\Omega) = V_T,$$

where $C(\Omega)$ stands for the compliance (3) of Ω, and where the target value V_T for the volume $\mathrm{Vol}(\Omega)$ is set to $V_T = 2.5$.

Several intermediate shapes are represented on Figure 12, exemplifying in particular the multiple topological changes undergone by the shape in the course of its evolution. The histories of the values of $C(\Omega$ and $\mathrm{Vol}(\Omega)$ are reported in Figure 13.

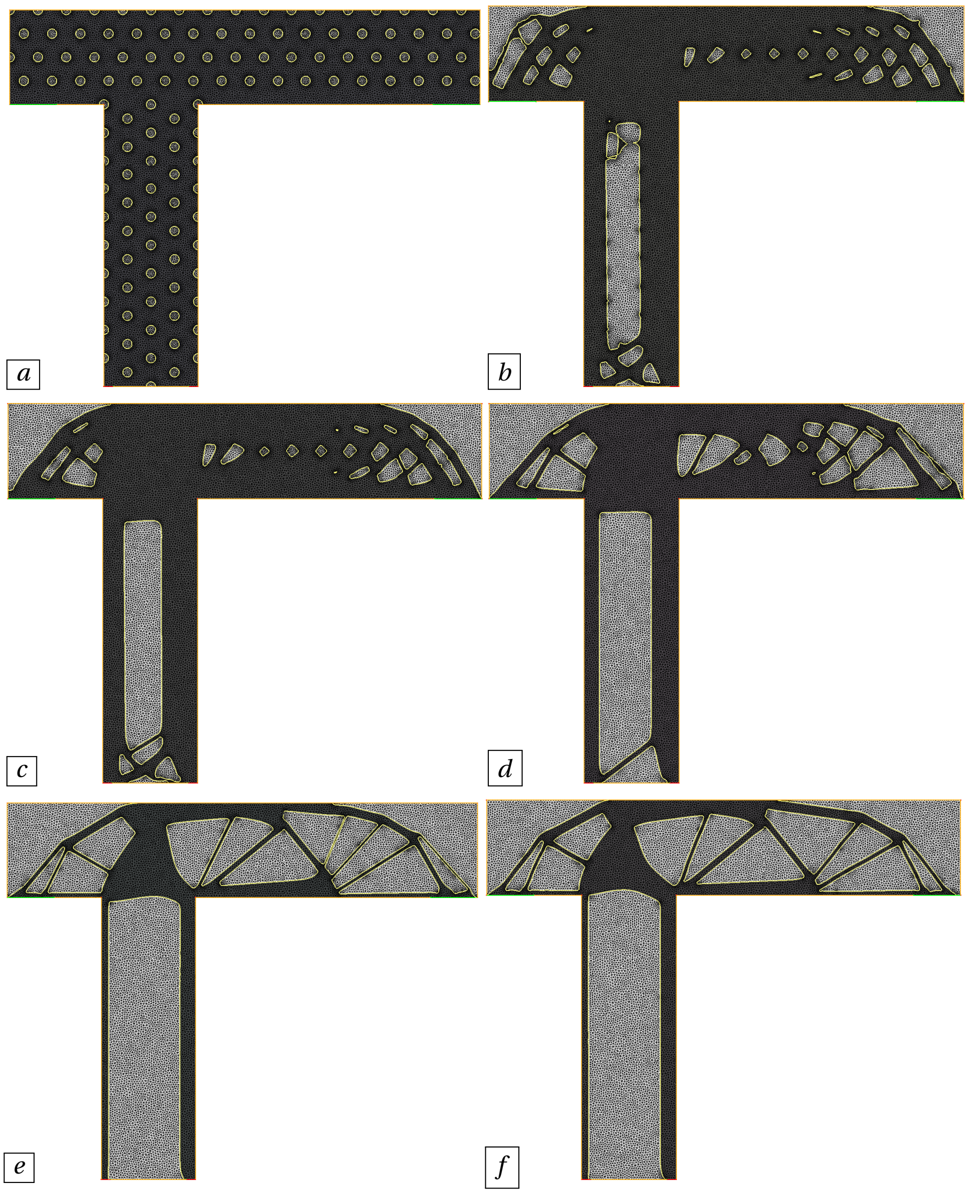

Figure 12. (From (a) to (f)) Intermediate shapes obtained at iterations 0, 16, 25, 50, 100 and 200 of the crane optimization test case of Section 5.1

5.2. *Minimum compliance problem in thermoelasticity*

In this section, we illustrate the application of the level set based mesh evolution methodology to the minimization of the compliance of a thermoelastic structure. This test case is partially reproduced from the articles [57, 103]. The considered shapes Ω are contained in the fixed computational domain $D = (0,2) \times (0,1)$; they are clamped on the left and right sides of D, whose reunion is denoted by Γ_D, and they are subjected to a traction load applied on a small region Γ_N with size 0.0125 centered at the middle of the bottom boundary, see Figure 14.

The shape Ω is filled by a thermoelastic material, characterized by the Lamé parameters

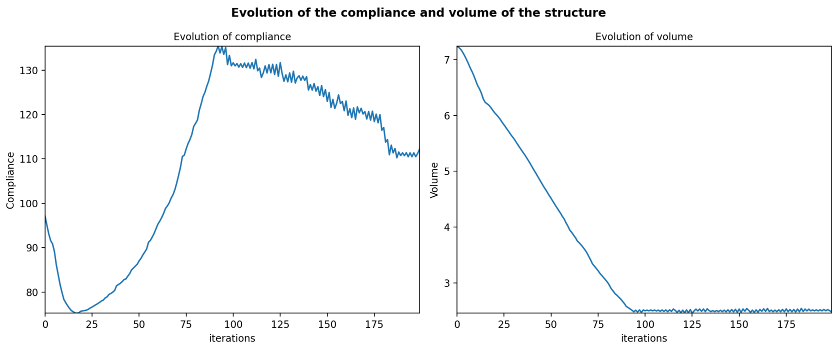

Figure 13. Evolution of the compliance and of the volume in the crane optimization example of Section 5.1.

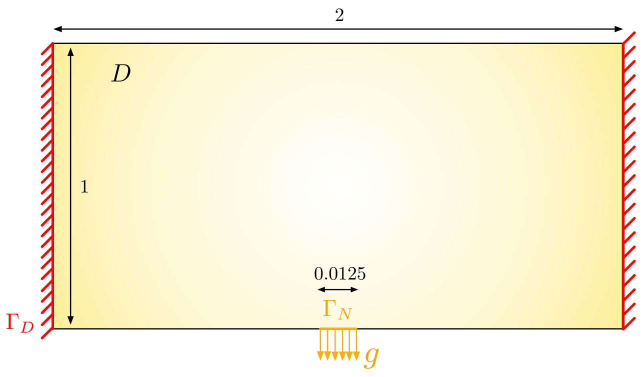

Figure 14. Physical setting of the compliance minimization problem of a thermoelastic structure considered in Section 5.2, issued from [103].

$\lambda = 11510$, $\mu = 7673$, and the thermal expansion coefficient $\alpha = 0.77$ with reference temperature $T_{\text{ref}} = 0$. The quantities α and T_{ref} express that the structure experiences an additional stress induced by thermal dilation when the temperature T inside the material is larger than T_{ref}. In this example, the constant temperature field $T = 5$ is imposed to the structure, causing expansion of the structure.

The displacement u_Ω of the shape in this setting is the solution to the following linear thermoelasticity system:

$$\left\{\begin{aligned} -\operatorname{div}(\sigma(u_\Omega, T)) &= 0 && \text{in } \Omega \\ u_\Omega &= 0 && \text{on } \Gamma_D \\ \sigma(u_\Omega, T)n &= g && \text{on } \Gamma_N \\ \sigma(u_\Omega, T)n &= 0 && \text{on } \Gamma, \end{aligned}\right. \tag{29}$$

where the stress tensor $\sigma(u,T)$ reads

$$\sigma(u,T) = Ae(u) - \alpha(T - T_{\text{ref}})\mathrm{I} \quad \text{with} \quad Ae = 2\mu e + \lambda \operatorname{tr}(e)\mathrm{I}. \tag{30}$$

We solve the following minimum compliance minimization problem:

$$\min_{\Omega \subset D} \quad J(\Omega) := \int_\Omega Ae(u_\Omega) : e(u_\Omega)\,\mathrm{d}x \quad \text{s.t.} \quad \operatorname{Vol}(\Omega) \le V_T,$$

where the target volume is $V_T = 0.7$.

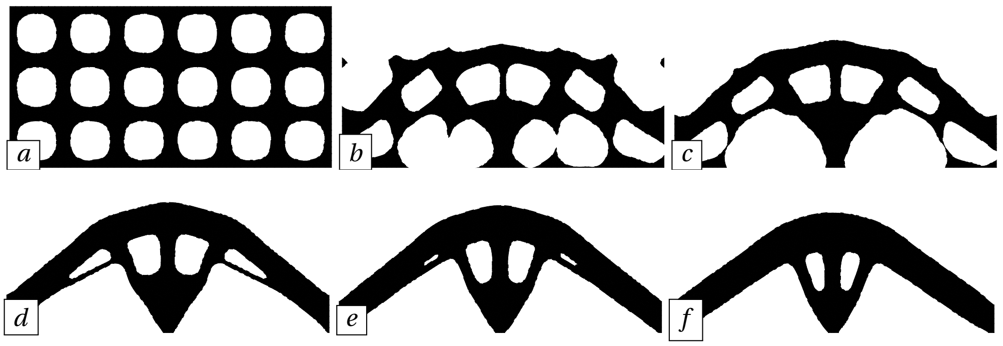

Figure 15. (From (a) to (f)) Intermediate shapes obtained at iterations 0, 5, 10, 30, 50 and 100 of the thermoelastic test case of Section 5.2

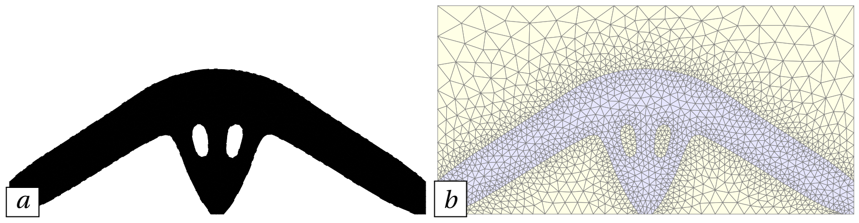

Figure 16. (a) Optimized design (iteration 200) in the thermoelastic test case of Section 5.2, (b) associated mesh $\mathscr{T}$ of the computational domain D.

The shape derivative of the compliance in the present context has been calculated in [57, 103], and the formulas are omitted for brevity. The optimization results are shown on Figures 15 and 16, which include respectively plots of a few intermediate shapes, and the mesh of the optimized design obtained at convergence.

5.3. *Lift–drag topology optimization for three-dimensional aerodynamic design*

This section presents an application of the level set based mesh evolution method in the context of 3d aerodynamic design. We aim to optimize the shape of a flying solid obstacle immersed into an incoming fluid flow, so that it generate the largest possible lift force while keeping friction forces small. This classical industrial problem, which is commonly referred to as "lift-drag" optimization in the literature, has been mostly addressed in the context where the optimized design is described by a set of CAD parameters, see for instance [51, 63, 70, 71, 75, 80, 88]. The test case presented in this section is a variation from that discussed in [58, Section 5.3].

The physical setting is that depicted on Figure 17: the fixed computational domain is the cube $D = (0,1)^3$. It is filled by a fluid, entering from the left-hand boundary Γ_{in} with a velocity $v_0 = (1,0,0)$, and exiting with a stress-free boundary condition at the opposite face Γ_{out}. The domain is divided into a fluid part Ω, denoted by Ω_f, and a solid obstacle $\Omega_s := D \setminus \overline{\Omega_f}$ which is such that $\Omega_s \Subset D$. The boundary of the solid obstacle is denoted by $\Gamma = \partial\Omega_s$. The physical state of the fluid is determined by its velocity field v_Ω and pressure field p_Ω which are characterized as

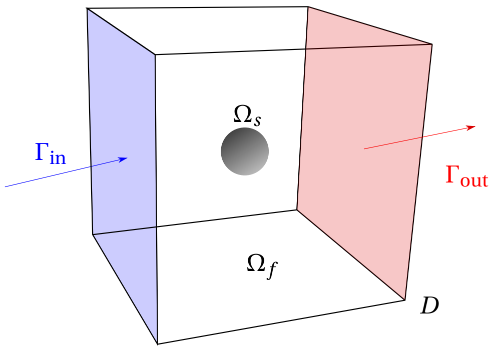

Figure 17. Physical setting of the lift-drag optimal design problem considered in Section 5.3.

the solutions to the system of incompressible steady-state Navier–Stokes equations:

$$\begin{cases} -\operatorname{div}(\sigma_f(v_\Omega, p_\Omega)) + \rho \nabla v_\Omega \, v_\Omega = 0 & \text{in } \Omega_f \\ \operatorname{div}(v_\Omega) = 0 & \text{in } \Omega_f \\ v_\Omega = v_0 & \text{on } \Gamma_{\rm in} \\ \sigma_f(v_\Omega, p_\Omega) n = 0 & \text{on } \Gamma_{\rm out} \\ v_\Omega = 0 & \text{on } \Gamma \\ v_\Omega \cdot n = 0 & \text{on } \partial D \setminus (\overline{\Gamma_{\rm in}} \cup \overline{\Gamma_{\rm out}}), \end{cases} \tag{31}$$

where ν is the fluid viscosity, ρ its density and where $\sigma_f(v,p)$ denotes the fluid stress tensor which is related to the rate of strain tensor $e(v) := \frac{1}{2}(\nabla v + \nabla v^T)$ via Newton's law:

$$\sigma(v,p) := 2\nu e(v) - p\mathrm{I},$$

and where n denotes the unit normal vector to $\partial\Omega_f$, pointing outward Ω_f.

The considered problem is to maximize the lift generated by the obstacle Ω_s (which is the force allowing it to fly), under an upper bound constraint on the drag (i.e. the reaction force of the fluid medium on Ω_s), so that the obstacle remains aerodynamic. In addition, the volume $\mathrm{Vol}(\Omega_s)$ and the center of mass $X(\Omega_s) \in \mathbb{R}^3$ of the obstacle are prescribed. The corresponding optimization program reads:

$$\begin{aligned} \min_{\Omega} \quad & -\mathrm{Lift}(\Omega) \\ \text{s.t.} \quad & \begin{cases} \mathrm{Drag}(\Omega) \leqslant \mathrm{Drag}_0 \\ \mathrm{Vol}(\Omega_s) = V_T \\ X(\Omega_s) := \dfrac{1}{|\Omega_s|}\displaystyle\int_{\Omega_s} x \, \mathrm{d}x = x_0, \end{cases} \end{aligned} \tag{32}$$

where the lift functional $\mathrm{Lift}(\Omega)$ accounts for the vertical force exerted by the fluid on the solid obstacle:

$$\mathrm{Lift}(\Omega) := -\int_\Gamma \varepsilon_y \cdot \sigma_f(v_\Omega, p_\Omega) n \, \mathrm{d}s,$$

where $\varepsilon_y = (0,1,0)$. The Drag functional $\mathrm{Drag}(\Omega)$ is defined by

$$\mathrm{Drag}(\Omega) := \int_{\Omega_f} \sigma_f(v_\Omega, p_\Omega) : \nabla v_\Omega \, \mathrm{d}x = \int_{\Omega_f} 2\nu e(v_\Omega) : e(v_\Omega) \, \mathrm{d}x.$$

The computation of the shape derivatives of these functionals is fairly classical and can be found e.g. in [41, 55, 68]. The parameters considered for the present test case are set to $\nu = 1/200$, $\rho = 1$, corresponding to a Reynolds number of the order of $\mathrm{Re} \simeq 200$, $V_T := 0.03$ and $x_0 = 0$.

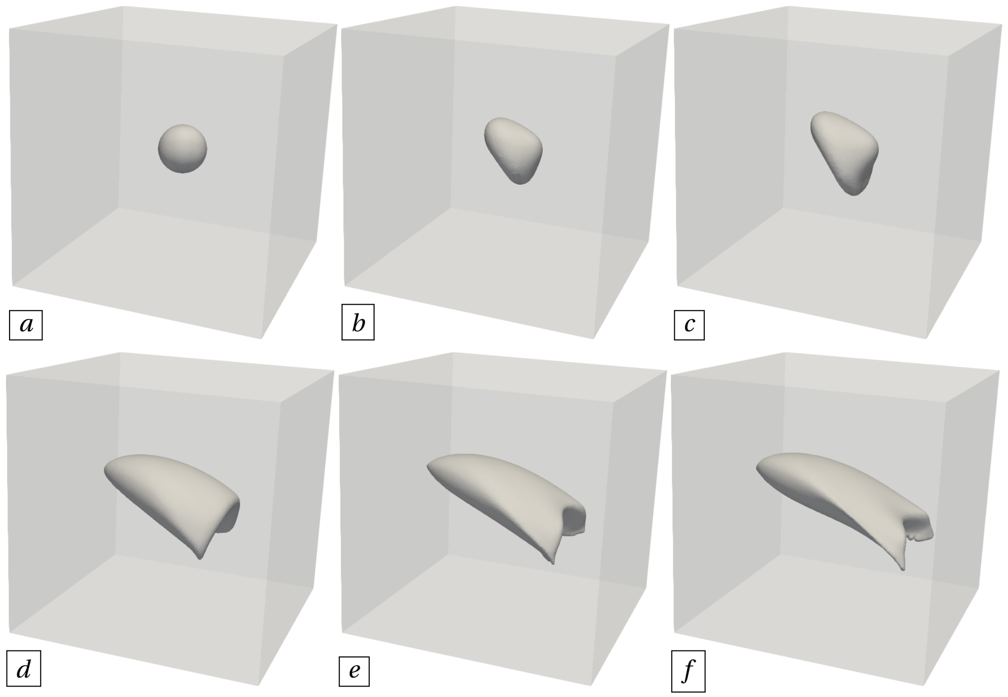

Figure 18. (From (a) to (f)) Intermediate designs obtained at iterations 0, 5, 10, 50, 100 and 200 of the optimization process for the lift-drag test case of Section 5.3.

The maximum value allowed for the drag is set to $\text{Drag}_0 := 0.078405$, corresponding to 1.5 times the minimum possible drag value for the prescribed solid volume (obtained from another minimization).

This computationally intense problem is solved with parallel computing using 20 CPUs and the methodology described in [58]. The typical edge length imposed on the fluid-solid interface equals 0.01. The final mesh features 154,982 vertices and 846,138 tetrahedra, including 134,637 vertices and 700,520 tetrahedra in the fluid domain.

A few intermediate shapes obtained during the optimization procedure are shown on Figure 18. The optimized design, its mesh as well as its surrounding velocity field can be appraised on Figures 19 and 20. Despite the low Reynolds number which makes this study quite unrealistic with respect to concrete aeronautic applications, is it quite remarkable that it is able to predict a design whose longitudinal profile recalls that of an airfoil, and whose "V-shape" at the back reminds that of airplanes.

5.4. *Optimal design of a three-dimensional fluid heating device*

The numerical example of this section is reproduced from [58, Section 5.5]; it deals with the optimal design of a heating device whose purpose is to warm up a cold fluid under the coupled effects of convection and conduction.

The situation is that depicted on Figure 21. The computational domain $D \subset \mathbb{R}^3$ is the box $[0,2] \times [0,1] \times [0,1]$. It is divided into a fluid phase $\Omega \subset D$, denoted by Ω_f, and a solid phase $\Omega_s = D \setminus \overline{\Omega}$, with respective thermal conductivities k_f and k_s; these are separated by the interface Γ. A "cold" fluid with density ρ and viscosity ν enters from the left side of D through a disk-shaped inlet boundary $\partial\Omega_f^{\text{in}}$ of radius 0.1 with a unit parabolic velocity profile $v = v_0$ and a prescribed

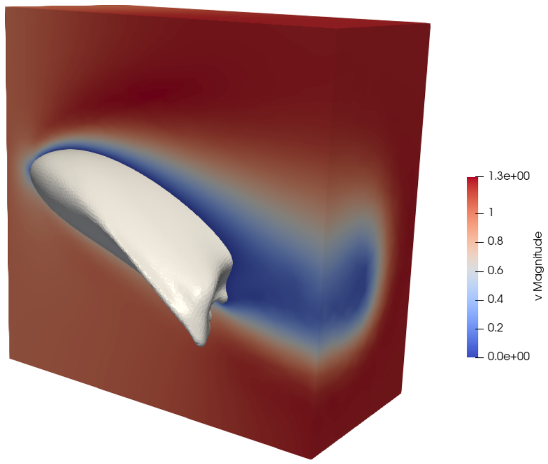

Figure 19. Optimized design in the lift-drag test case of Section 5.3, with a cut view of the magnitude of the surrounding velocity field v_Ω.

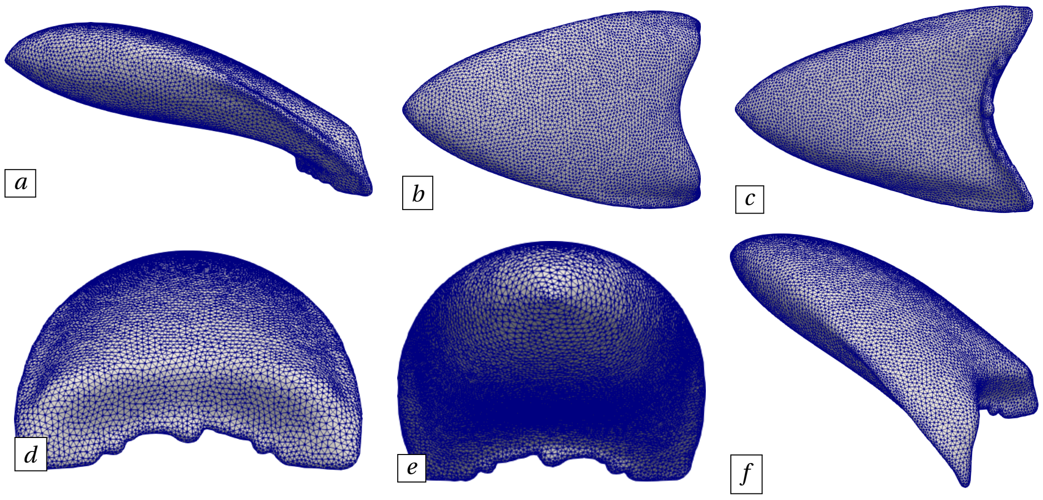

Figure 20. Different views of the optimized design obtained in the left-drag test case of Section 5.3; (a) side view, (b) bottom view, (c) top view, (d) back view, (e) front view and (f) perspective view.

temperature $T = 0$. The fluid absorbs the heat diffused from a "hot" stripe $\partial\Omega_T^D$ at the middle of ∂D which is maintained at the constant temperature $T = 10$, before exiting through a similar disk-shaped outlet boundary $\partial\Omega_f^{\text{out}}$ at the right side of ∂D with a stress-free boundary condition. No heat is lost through the remaining parts of ∂D which are considered adiabatic.

In this context, we aim to optimize the shape $\Omega_f = \Omega$ of the fluid phase, and thereby the repartition of Ω_f and Ω_s within D, in order to maximize the total heat carried by the fluid, under constraints on the static pressure loss through the device and on the volume of Ω_f.

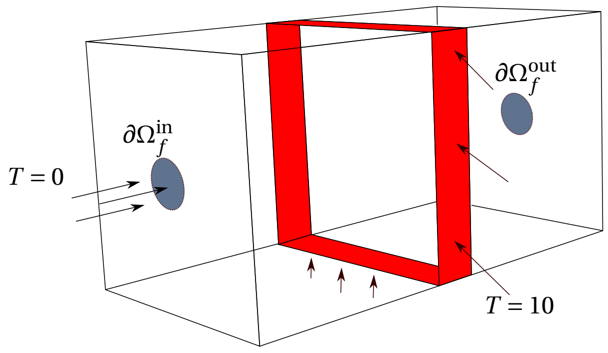
$\partial\Omega_f^{\text{out}}$
$\partial\Omega_f^{\text{in}}$
$T = 0$
$T = 10$

Figure 21. Setting of the optimal heat transfer problem of Section 5.4. A "cold" fluid flows from the left to the right blue disk-shaped regions and captures the heat communicated by the red stripe $\partial\Omega_T^D$ where the "hot" temperature $T = 10$ is imposed.

Mathematically, the following optimization program is considered, see [55, 58]:

$$\begin{aligned} &\min_{\Omega} \quad J(\Omega) := -\int_{\Omega_f} \rho c_p v_\Omega \cdot \nabla T_\Omega \, \mathrm{d}x, \\ &\text{s.t.} \quad \begin{cases} \mathrm{DP}(\Omega) := \displaystyle\int_{\partial\Omega_f^{\text{in}}} p_\Omega \, \mathrm{d}s - \int_{\partial\Omega_f^{\text{out}}} p_\Omega \, \mathrm{d}s \leqslant \mathrm{DP}_T, \\ \mathrm{Vol}(\Omega_f) = V_T. \end{cases} \end{aligned} \tag{33}$$

Here, the velocity v_Ω and the pressure p_Ω of the fluid are governed by the static, incompressible Navier–Stokes equations:

$$\begin{cases} -\operatorname{div}(\sigma_f(v_\Omega, p_\Omega)) + \rho \nabla v_\Omega \, v_\Omega = 0 & \text{in } \Omega_f, \\ \operatorname{div}(v_\Omega) = 0 & \text{in } \Omega_f, \\ v_\Omega = v_0 & \text{on } \partial\Omega_f^{\text{in}}, \\ \sigma_f(v_\Omega, p_\Omega) n = 0 & \text{on } \partial\Omega_f^{\text{out}}, \\ v_\Omega = 0 & \text{on } \Gamma. \end{cases} \tag{34}$$

where $\sigma_f(v, p)$ is the stress tensor within a fluid with velocity v and pressure p, which is related to the rate of strain tensor $e(v) := \frac{1}{2}(\nabla v + \nabla v^T)$ via the Newton's law:

$$\sigma_f(v, p) := 2\nu e(v) - p\mathrm{I}.$$

The temperature field T_Ω inside D depends on the velocity v_Ω of the fluid via the static convection-diffusion equation:

$$\begin{cases} -\operatorname{div}(k_f \nabla T_\Omega) + \rho c_p v_\Omega \cdot \nabla T_\Omega = 0 & \text{in } \Omega_f, \\ -\operatorname{div}(k_s \nabla T_\Omega) = 0 & \text{in } \Omega_s, \\ T_\Omega = 0 & \text{on } \partial\Omega_f^{\text{in}}, \\ T_\Omega = 10 & \text{on } \partial\Omega_T^D, \\ -\dfrac{\partial T_\Omega}{\partial n} = 0 & \text{on other boundaries}, \\ T_\Omega \text{ is continuous} & \text{on } \Gamma, \\ \text{The flux } -k\dfrac{\partial T_\Omega}{\partial n} \text{ is continuous} & \text{on } \Gamma, \end{cases} \tag{35}$$

where c_p is the thermal capacity of the fluid.

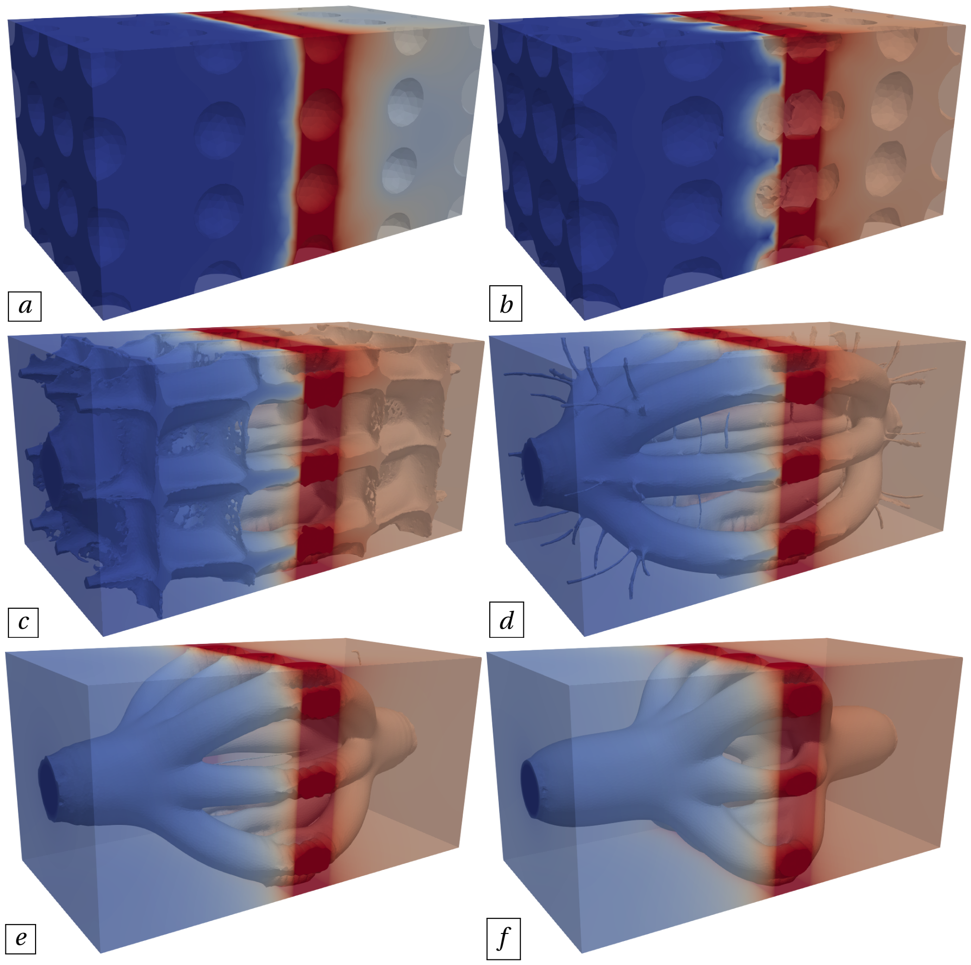

Figure 22. (From (a) to (f)) Iterations 0, 5, 15, 25, 50 and 152 of the optimization process of a heating device considered in Section 5.4.

The values of the physical parameters entering the formulation (31), (33) and (35) are chosen as follows. The density and the viscosity of the fluid are respectively $\rho = 10$ and $\nu = 0.167$, so that the Reynolds number of the flow equals $\mathrm{Re} \simeq 60$. Its thermal conductivity and capacity coefficients are respectively set to $k_f = 1$ and $c_p = 500$; hence, the Péclet number Pe equals 5,000. The thermal conductivity of the solid phase is $k_s = 10$. The imposed upper bound on the pressure drop in (33) is set to $\mathrm{DP}_T = 0.85$, while the target volume for the fluid phase equals 20% of the total volume: $V_T = 0.2\,\mathrm{Vol}(D)$.

The numerical realization of this program relies on the application of the level set based mesh evolution Algorithm 3, where the formulas for the derivatives of the shape functionals at play have been established in [56]. The initial repartition of the phases Ω_f, Ω_s features islands of solid spherical inclusions, see Figure 22 (a). A few intermediate shapes arising in the course of the optimization process can be seen on Figure 22, illustrating the dramatic topological changes undergone by the system. The resulting optimized design is displayed on Figure 23. Interestingly, the latter features thin inclusions of fluid attached to the main pipes so as to take advantage of the insulating effect induced by the low thermal conductivity of the fluid with respect to that of the solid.

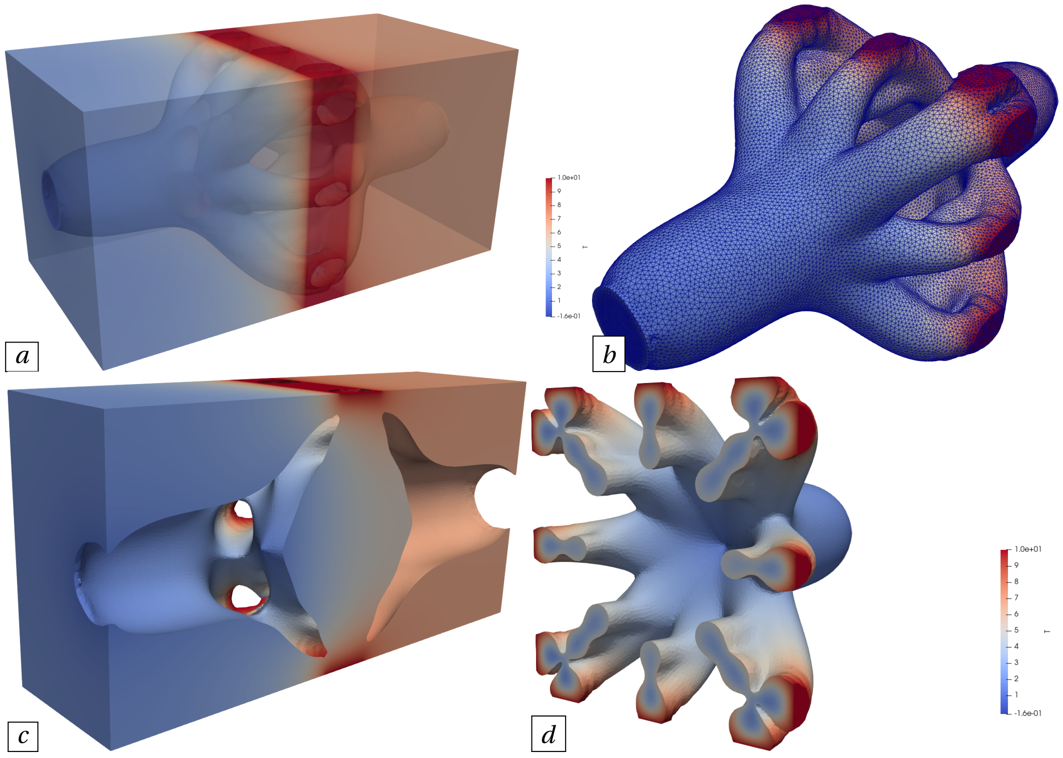

Figure 23. (a) Optimized heating device obtained in the example of Section 5.4; (b) mesh of the fluid component of the final design; (c) sectional view of the optimized solid domain (d) sectional view of the optimized fluid domain. In all the pictures, the colors correspond to the temperature profile.

The sizes of the meshes of D considered for this test-case range from approximately 22,000 to 278,100 vertices. The numerical resolutions of the Navier–Stokes and the advection-diffusion equations (31) and (35) are efficiently achieved by combining the finite element method with domain decomposition techniques on 30 processes. The total computation takes approximately 2 days.

6. Conclusion and perspectives

In this article, we have presented a numerical framework for shape and topology optimization which combines the level set method with remeshing techniques to reconcile the antagonistic needs for accurate mechanical computations on the shape and for a robust description of its evolution. This numerical strategy has been successfully applied in a wide variety of physical contexts, including those of linearly elastic structures, fluid-structure interacting devices, heat exchangers, etc. It could also be used fruitfully beyond the realm of shape optimization, and notably in the simulation of multi-phase problems (for instance, in the study of multi-phase flows), or in the solution of inverse problems implying the detection or reconstruction of objects.

Beyond ever more complicated physical situations, we are currently working on extending this method to deal with the optimization of the shape and topology of regions embedded in a fixed surface. This would allow, for instance, to optimize the regions of the boundary of a domain bearing various types of boundary conditions, in the spirit of our previous work [42]. We also wish to extend this methodology to handle the evolution of open curves in 2d, and open surfaces in 3d, which would make it possible to optimize shell structures.

Acknowledgements

The body of work presented in this article owes much to the diverse expertise of many colleagues: in particular, we are very grateful to M. Albertelli, F. Bordeu, J. Cortial, C. Dobrzynski, A. Froehly, G. Michailidis. The authors are especially indebted to G. Allaire and P. Frey, who were at the inception of this project, and made it thrive with their key insight and relentless suggestions.

Appendix A. Manual of the companion code

In this appendix, we describe the `python`-based implementation of the level set mesh evolution method accompanying this article. Our aim is to provide a code which meets a compromise between simplicity, re-usability and computational efficiency: we hope that it is simple enough so that a user can easily get acquainted with its main features, and at the same time general and modular enough so that it can be modified with a minimum amount of effort to address a whole gamut of new and more challenging applications.

The source code can be downloaded from the `Github` repository

`https://github.com/dapogny/sotuto`

which is organized in several folders: the one named `./base` contains the material devoted to the benchmark 2d cantilever test case; it serves as support for this presentation. The other folders are as many variations around this main theme, which are briefly presented in Appendix A.10.

This manual is organized as follows. After a brief description of a guiding test case in Appendix A.1, we provide in Appendix A.2 the information for installing the needed external libraries in our framework. We then present the overall architecture of the program in Appendix A.3. In Appendix A.4, we outline the practical shape optimization strategy realizing the abstract Algorithm 3, and we introduce the functions corresponding to its main stages. These functions are detailed in the subsequent sections, with a particular emphasis on the parts that the user may have to modify in order to implement his own examples. Note that, for clarity, we have allowed some redundancy between the various sections of this manual, as it is not necessarily meant to be read in linear fashion.

A.1. *Description of the 2d cantilever test-case*

Our model example arises in the linearized elasticity setting described in Section 2.1. The considered shape Ω represents a cantilever beam; Ω is comprised in a box D with size 2×1, it is clamped on the left-hand side Γ_D of ∂D, and traction loads $g = (0, -1)$ are applied on a given region Γ_N near the center of its right-hand side, see Figure 24. In this context, we aim to solve the problem

$$\min_{\Omega} C(\Omega), \quad \text{s.t.} \quad \text{Vol}(\Omega) = V_T, \tag{36}$$

where the compliance $C(\Omega)$ and the volume $\text{Vol}(\Omega)$ of a shape Ω have been defined in Section 2.1 as:

$$C(\Omega) = \int_{\Omega} Ae(u_{\Omega}) : e(u_{\Omega})\,\mathrm{d}x \quad \text{and} \quad \text{Vol}(\Omega) = \int_{\Omega} \mathrm{d}x,$$

and where V_T is a volume target. The elastic displacement u_{Ω} involved in the definition of $C(\Omega)$ is the unique solution in $H^1_{\Gamma_D}(\Omega)^d$ to the linear elasticity system (1), whose variational formulation reads:

$$\forall\, v \in H^1_{\Gamma_D}(\Omega)^d, \quad \int_{\Omega} Ae(u_{\Omega}) : e(v)\,\mathrm{d}x = \int_{\Gamma_N} g \cdot v\,\mathrm{d}s. \tag{37}$$

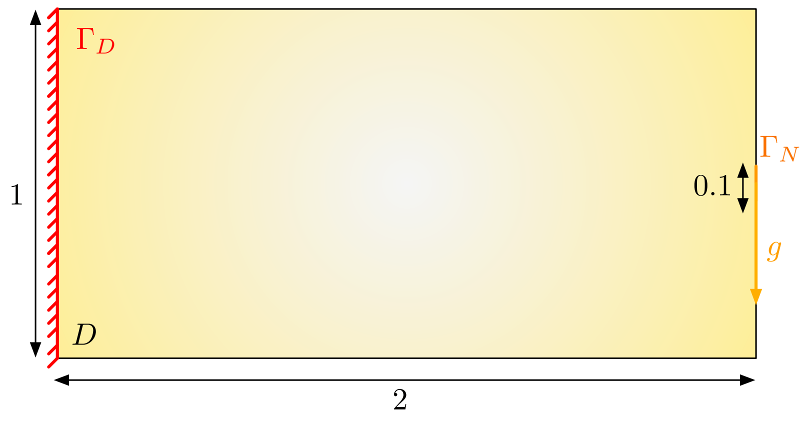

Figure 24. Setting of the 2d cantilever test-case considered in Appendix A.

For the convenience of the reader, we recall from Section 2.2 the expressions of the shape derivatives of $C(\Omega)$ and $\mathrm{Vol}(\Omega)$:

$$C'(\Omega)(\theta) = -\int_\Gamma Ae(u_\Omega) : e(u_\Omega)\,\theta\cdot n\,\mathrm{d}s, \quad \text{and} \quad \mathrm{Vol}'(\Omega)(\theta) = \int_\Gamma \theta\cdot n\,\mathrm{d}s. \tag{38}$$

A.2. *Getting started: download and installation of the files*

As a preliminary stage, the use of our program requires the installation of four open-source libraries, written in C or C++:

- The library `mshdist` [40] performs the computation of the signed distance function to an input domain at the vertices of a simplicial computational mesh, as detailed in Section 4.2. It can be freely downloaded from the repository

  `https://github.com/ISCDtoolbox/Mshdist`

  and compiled by following the instructions supplied at this address.
- The library `advection` [30] solves the advection equation (15) governing the evolution of a level set function on a simplicial computational mesh in 2d and 3d, see Section 4.3. It is available at the following link:

  `https://github.com/ISCDtoolbox/Advection`

  where instructions are given to compile the library.
- The library `mmg` [39] is in charge of the remeshing operations of the strategy, as presented in Section 4.4. It can be downloaded through the webpage of the project

  `https://www.mmgtools.org/`

  where a series of tutorials is available to exemplify the main features of this program, see also the `github` link

  `https://github.com/MmgTools/mmg`
- The software `FreeFem` [66] is a numerical environment where partial differential equations can be solved in a few lines of commands through the input of their variational formulation. More generally, it conveniently and efficiently allows to handle finite element

spaces and functions thanks to an intuitive C++ based pseudo-language. The latest version of Freefem can be downloaded from the webpage[5]:

https://freefem.org/

which also contains an exhaustive documentation, and multiple worked examples.

In addition to these four required codes, we recommend the installation of the open-source vizualization software medit, provided at the following address:

https://github.com/ISCDtoolbox/Medit

where a short manual is also available[6].

Our numerical implementation of the level set based mesh evolution method relies on a main frame written in python, supplemented by FreeFem scripts dedicated to the operations involving finite element computations. The calls to the four aforementioned external libraries are encapsulated in python functions which merely execute unix command lines via subprocesses[7]. These interface functions are not supposed to be modified by the user, and we limit ourselves with presenting their syntax in Appendix A.7 below. Their proper functioning requires that, after installation of the libraries mshdist, advection, mmg and FreeFem, the user set the paths of the corresponding executables in the **path.py** file, as described in Appendix A.5.1. The communications between the main python program and these codes (such as the exchange of input parameters or output results) are realized via an exchange file referred to as path.EXCHFILE in the python program; this mechanism should be invisible to the user.

Remark 7. In principle, there is no obstruction in replacing FreeFem++ with another, e.g. commercial finite element solver in the proposed implementation. Essentially, this task would require minor adaptations to convert the meshes processed by mmg (and the other aforementioned libraries) into a suitable format for the latter.

A.3. *Global architecture of the program*

The source code in the ./base folder is made of files of three different types:

- The .py files are python modules;
- The .edp files are FreeFem scripts, which are executed by calling FreeFem from python functions;
- The .idp files are also processed by FreeFem; they contain general information and routines shared by most of the .edp scripts.

Two of the above files are dedicated to the input of user-defined parameters:

- The module **path.py** contains global variables and the paths to the directories where input and output objects are stored, to the executables, etc. For instance, path.EXCHFILE is the address of the aforementioned exchange file between python and FreeFem modules, the file path.HISTO contains the data (values of the compliance and volume) of the successive shapes produced during the optimization process, path.FREEFEM is the command line for calling FreeFem, etc. This file also contains the definition of shorthands used throughout the program: path.step(n,"mesh") is the string of characters

[5]A python package interfacing FreeFem is available at the following link: https://gitlab.com/florian.feppon/pyfreefem

[6]The python package PyMedit can be used to conveniently handle and vizualize the mesh structures processed by medit: https://gitlab.com/florian.feppon/pymedit

[7]These command lines are tailored to Linux and MacOs environments, and some adaptations may be necessary for systems relying on Windows.

containing the name of the mesh of D at the n^{th} iteration, `path.step(n,"u.sol")` refers to the file containing the elastic displacement, etc.

The contents of this file are more thoroughly described in Appendix A.5.1.

- The file `macros.idp` contains global variables and macros used in `FreeFem` scripts; it is described in Appendix A.5.2.

The main program is executed by calling the file **main.py**, which orchestrates the functions from the other `python` modules. These modules are:

- The module **inout.py** contains the functions for reading and printing data from and to external files;
- The module **inigeom.py** is devoted to the creation of the initial shape, see Appendix A.6 for its description;
- The module **mechtools.py** gathers the functions in charge of the mechanical calculations of the framework, and notably the resolution of the linear elasticity system (37), the calculation of the compliance of shapes, that of a descent direction for the problem (36), etc. These are described in Appendices A.8 and A.9;
- The module **lstools.py** contains the functions dedicated to the treatment of level set functions, and notably the interface functions with the `C` libraries `mshdist` and `advection`. The practical use of these functions is described in Appendix A.7.
- The module **mshtools.py** contains the interface functions with the library `mmg`, see Appendix A.7;

The execution of our program mainly results in the production of data files of two types, written in the common formats used by `mshdist`, `advect`, `FreeFem`, `mmg` and `medit`:

- `.mesh` files contain meshes,
- `.sol` files contain (scalar- or vector-valued) $\mathbb{P}_1$ finite element functions given by their values at the vertices of a companion mesh.

For brevity, and since the structure of such files is not needed for our purpose, we do not describe them in the present manual, referring for this matter to the documentation of any of the codes mentioned above.

A.4. *Overview of the global strategy: description of the file* `main.py`

The code contained in the folder `./base` is executed by entering the following command line from the root folder.

```
python ./base/main.py
```

In the present section, we describe in some detail how the file `main.py` implements the strategy of Algorithm 3; the outline of this practical realization is provided in Algorithm 4.

A.4.1. *Initialization*

The program starts with the creation of the folder `./res/`, where all the mesh and solution files produced during the execution will be saved. Meanwhile, the files `path.EXCHFILE`, `path.HISTO` are created, as well as other exchanges files whose descriptions are not needed by the user.

```
# Initialize folders and exchange files
inout.iniWF()
```

Then, the calls to the four external libraries `FreeFem`, `mshdist`, `advection` and `mmg` are tested.

Algorithm 4 Practical realization of the level set based mesh evolution method.

```
Initialization:
    • Function inout.iniWF: Initialization of folders and exchange files.
    • (Optional) Function inout.testLib: Test of the links with python functions and external C libraries.
    • Function inigeom.iniGeom: Creation of the initial mesh $\mathscr{T}^0$ of $D$.
    • Function mechtools.elasticity: Calculation of $u_{\Omega^0}$.
    • Functions mechtools.compliance and mechtools.volume: Calculation of $C(\Omega^0)$ and
      $\text{Vol}(\Omega^0)$.
for $n = 0, \dots,$ path.MAXIT $- 1$ do
    (1) Function lstools.mshdist: Calculation of the signed distance function $\phi^n$ on $\mathscr{T}^n$.
    (2) Functions mechtools.gradCp and mechtools.gradV: Calculation of the gradients
        of $C(\Omega)$ and $\text{Vol}(\Omega)$ on $\mathscr{T}^n$.
    (3) Functions mechtools.descent: Calculation of a descent direction for (36).
    (4) Functions mechtools.merit: Calculation of the merit of $\Omega^n$.
    (5) Line search: for $k = 0, \dots,$ path.MAXITLS $- 1$ do
            (a) Function lstools.advect: Resolution of the level set advection equation, for
                the new function $\phi^{n+1}$ on the old mesh $\mathscr{T}^n$ of $D$.
            (b) Function mshtools.mmg: Creation of the new mesh $\mathscr{T}^{n+1}$ of $D$, where the 0
                level set of $\phi^{n+1}$ is explicitly discretized.
            (c) Function mechtools.elasticity: Calculation of $u_{\Omega^{n+1}}$ on $\mathscr{T}^{n+1}$.
            (d) Functions mechtools.compliance, mechtools.volume and mech-
                tools.merit: Calculation of $C(\Omega^{n+1})$, $\text{Vol}(\Omega^{n+1})$ and $M(\Omega^{n+1})$.
            (e) Decision: Acceptation / Rejection of the new iterate.
        end for
end for
return Mesh $\mathscr{T}^n$ of $D$ where $\Omega^n$ is discretized as a submesh $\mathscr{T}^n_{\text{int}}$.
```

```python
# Test the links with external C libraries
inout.testLib()
```

This should result with the following terminal output.

```
FreeFem installation working.
Mshdist installation working.
Advect installation working.
Mmg installation working.
All external libraries working.
```

Of course, once the user is confident with his settings, this testing stage can be deactivated by simply removing this instruction.

The next call to the function `iniGeom` from the module **inigeom.py**, which is described in Appendix A.6, is meant to create the initial mesh $\mathscr{T}^0$ of D, equipped with the boundary conditions of the test-case, and featuring an explicit discretization of the initial shape Ω^0 as a submesh $\mathscr{T}^0_{\text{int}}$.

```python
# Creation of the initial mesh
inigeom.iniGeom(path.step(0,"mesh"),path.step(0,"phi.sol"))
```

The resulting mesh file `step.0.mesh` is stored in the folder `path.RES`; it is depicted in Figure 25 (a), where the interior submesh $\mathscr{T}^0_{\text{int}}$ is that made of the black triangles (identified by the reference `path.REFINT`), while the exterior submesh $\mathscr{T}^0_{\text{ext}}$ is the collection of white triangles (with reference `path.REFEXT`).

The linear elasticity system (37) for the elastic displacement u_{Ω^0} of Ω^0 is then solved on the interior submesh $\mathscr{T}^0_{\text{int}}$ thanks to the function `elasticity` from the **mechtools.py** module.

```
# Resolution of the state equation
mechtools.elasticity(path.step(0,"mesh"),path.step(0,"u.sol"))
```

This function is described in Appendix A.8 below.

Finally, the values $C(\Omega^0)$ and $\text{Vol}(\Omega^0)$ of the compliance and the volume of Ω^0 are calculated owing to the following commands:

```
# Calculation of the compliance and the volume of the shape
newCp  = mechtools.compliance(path.step(0,"mesh"),
                              path.step(0,"u.sol"))
newvol = mechtools.volume(path.step(0,"mesh"))
```

The functions `compliance` and `volume` from the module **mechtools.py** are detailed in Appendix A.8.

A.4.2. *Main optimization loop*

At the beginning of each iteration n, ranging from 0 to `path.MAXIT` -1, a few shorthands are defined, such as for instance:

- `curmesh` is the address `path.step(n,"mesh")` (corresponding, by default, to the file `step.n.mesh` in the folder `path.RES`) for the mesh $\mathscr{T}^n$ of $D$;
- `curphi` is the address `path.step(n,"phi.sol")` (corresponding, by default, to the file `step.n.phi.sol` in `path.RES`) for the level set function $\phi^n : D \to \mathbb{R}$.

At this moment, the following objects are available:

- The mesh $\mathscr{T}^n$ of the domain $D$, called `curmesh`; the shape $\Omega^n$ (resp. the exterior part $D \setminus \overline{\Omega^n}$) is represented by the submesh $\mathscr{T}^n_{\text{int}}$ (resp. $\mathscr{T}^n_{\text{ext}}$), whose elements are the triangles with label `path.REFINT` (resp. `path.REFEXT`).
- The displacement u_{Ω^n}, solution to the linear elasticity equation (37)) on $\Omega^n$, is called `curu`. It is encoded as a $\mathbb{P}_1$ finite element function on the whole mesh `curmesh`, with the convention that only its values in the interior part of `curmesh` are to be considered.
- The respective values `newCp`, `newvol` of the compliance and volume $C(\Omega^n)$ and $\text{Vol}(\Omega^n)$ of the current shape $\Omega^n$.

The nth iteration starts with the calculation of the signed distance function $\phi^n = d_{\Omega^n}$ to $\Omega^n$ at the vertices of the mesh $\mathscr{T}^n$. This relies on the function `mshdist` from the module **lstools.py**:

```
# Generation of a level set function for $\Omega^n$ on $D$
lstools.mshdist(curmesh,curphi)
```

The use of this function is described in Appendix A.7.

The shape gradients of the objective and constraint functionals $C(\Omega)$ and $\text{Vol}(\Omega)$ are then calculated. This is realized thanks to the functions `gradCp` and `gradV` from the module **mechtools.py**, which rely in particular on the Hilbertian extension-regularization procedure presented in Section 4.5.

```
# Calculation of the gradients of compliance and volume
mechtools.gradCp(curmesh,curu,curCPgrad)
mechtools.gradV(curmesh,curVgrad)
```

These functions are described in Appendix A.9.1.

A descent direction for the optimization problem (36) is then inferred from these gradients, following the constrained optimization strategy presented in Section 4.6.

```
# Calculation of a descent direction
mechtools.descent(curmesh,curphi,curCp,curCpgrad,curvol,
                                                curVgrad,curgrad)
```

This function is described in Appendix A.9.2. It takes as arguments

- A string of characters `curmesh` for the name of the current mesh of D;
- The name `curphi` of the level set function for the shape $\Omega^n$;
- The values `curCp`, `curvol` of the objective (compliance) and constraint (volume) functionals;
- The respective names `curCpgrad`, `curVgrad` for the gradients of these functionals;
- The name `curgrad` for the file where the calculated descent direction should be saved.

Remark 8. The descent direction produced by this function is normalized, so that its L^∞ norm equals the typical size `path.MESHSIZ` of an element in the mesh, see Appendix A.9.2 about this practice. Hence, a motion of the shape $\Omega^n$ in the direction of `curgrad` for a pseudo-time step $\tau^n = 1$ corresponds to a maximum amplitude `path.MESHSIZ` for a motion of a point of the shape.

The above function `descent` prints in the file `path.EXCHFILE` the values of the coefficients λ, S (which boils down to a scalar value in the present example), α_J, α_G used in the optimization strategy, as they are needed in the evaluation of the merit function $M(\Omega)$, see (28).

This value of $M(\Omega^n)$ is then calculated, thanks to the following function, which is described in Appendix A.9.3.

```
# Evaluation of the merit of $\Omega^n$
merit = mechtools.merit(curCp,curvol)
```

A line search procedure then begins, under the form of a nested loop indexed by k; for each substep, the following operations are carried out.

(1) The motion of the current shape $\Omega^n$ given by `curmesh` and `curphi` in the direction `curgrad` is realized, for a certain pseudo-time `coef`. Because of the chosen normalization for `curgrad` (see Remark 8), the maximum displacement of a point of the shape equals `coef` $*$ `path.MESHSIZ`.

```
# Advection of the level set function
print("     Level Set advection")
lstools.advect(curmesh,curphi,curgrad,coef,newphi)
```

The use of the function **lstools**.`advect` is explained in Appendix A.7. The level set function ϕ^{n+1} for the new shape Ω^{n+1} resulting from this operation, stored in the file `newphi`, is still attached to the mesh `curmesh`.

(2) The 0 level set ϕ^{n+1}, named `newphi`, is explicitly discretized in the mesh `curmesh` by calling `mmg`.

```
# Creation of a mesh associated to the new shape
print("    Local remeshing")
retmmg = mshtools.mmg2d(curmesh,1,newphi,path.HMIN,
                path.HMAX,path.HAUSD,path.HGRAD,1,newmesh)
```

The function **mshtools**.mmg2d in charge of the interface between the library mmg and our python framework is described in Appendix A.7.

(3) The elastic displacement $u_{\Omega^{n+1}}$ is calculated on the new shape defined by the mesh $\mathscr{T}^{n+1}$ of D, named newmesh, or more accurately, on the submesh $\mathscr{T}^{n+1}_{\text{int}}$ of $\mathscr{T}^{n+1}$ made of the triangles with reference path.REFINT. The corresponding values of the compliance and volume functionals, and then the new value of the merit are evaluated.

These operations are executed unless mmg has accidentally failed, in which case the result of the current iteration of the inner, line search loop is immediately rejected (see below).

```
if ( retmmg ) :
  # Resolution of the state equation on the new shape
  print("    Resolution of linearized elasticity system")
  mechtools.elasticity(newmesh,newu)

  # Calculation of the new values of compliance and volume
  print("    Evaluation of the new merit")
  newCp  = mechtools.compliance(newmesh,newu)
  newvol = mechtools.volume(newmesh)
  newmerit = mechtools.merit(newCp,newvol)
```

The final step of the inner line-search loop is the decision stage: the new shape newmesh is accepted if the corresponding value newmerit of the merit function is less than that of the previous shape curmesh (up to some tolerance), or if a maximum number of iterations has been made in the line search procedure with this descent direction. Otherwise, the new iterate is refused; the algorithm goes back to the beginning of the line search procedure, with a reduced value of the “time-step” coef.

```
# Accept iteration: break and increase slightly the "time step"
if ( retmmg and ( newmerit < merit + path.TOL*abs(merit) )
                       or ( k == 2 )
                       or ( coef < path.MINCOEF ) ) :
  coef = min(path.MAXCOEF,1.1*coef)
  print(" Iteration {} -
                subiteration {} accepted\n".format(n,k))
  break
# Reject iteration: go to start of line search
#                       with a decreased "time step"
else :
  print(" Iteration {} - subiteration {} rejected".format(n,k))
  proc = subprocess.Popen(["rm {nmesh}".format(nmesh=newmesh)],
                                            shell=True)
  proc.wait()
  coef = max(path.MINCOEF,0.6*coef)
```

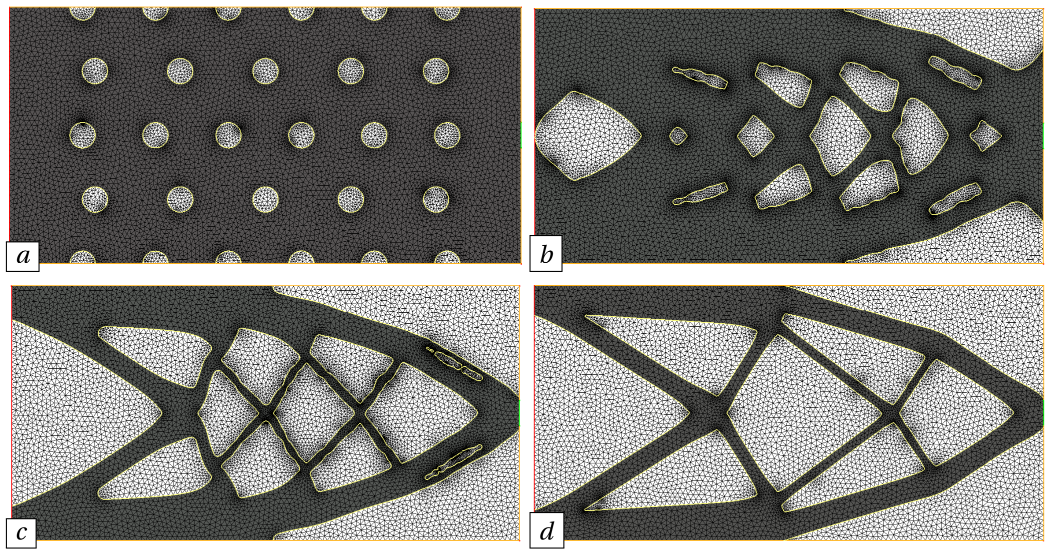

Figure 25. Iterations (a) 0, (b) 20, (c) 50 and (d) 150 (final) of the optimization process in the cantilever test-case discussed in Appendix A.

A.4.3. *Outputs*

At each iteration $n = 0, \dots,$ `path.MAXIT`-1 of the main loop of our optimization procedure, the following files are saved in the `path.RES` repository:

- The current mesh $\mathscr{T}^n$ of D, referred to via the shortcut `curmesh`, is saved as the file `path.step(n,"mesh")` (which is, unless specified otherwise by the user, the file `step.n.mesh` in the folder `path.RES`);
- The level set function ϕ^n for the shape Ω^n, called `curphi`, is saved as the file `path.step(n,"phi.sol")`;
- The solution u_{Ω^n} to the elasticity system associated to Ω^n is saved in the file `path.step(n,"u.sol")`; let us recall that, although u_{Ω^n} is only defined on Ω^n from the mathematical point of view, it is actually stored as a $\mathbb{P}_1$ finite element function on the total mesh $\mathscr{T}^n$, being understood that only the values at the vertices of the interior part $\mathscr{T}^n_{\text{int}}$ are relevant;
- The gradients `curCpgrad` and `curVgrad` of the objective and constraint functionals are saved in the files `path.step(n,"CP.grad.sol")` and `path.step(n,"V.grad.sol")`,
- The descent direction θ^n, called `curgrad`, is saved in the file `path.step(n,"grad.sol")`.

Eventually, the values of the compliance and volume attached to each shape $\Omega^n$are stored in the `path.HISTO` file.

A few shapes produced in the course of the optimization are depicted in Figure 25. The histories describing the evolution of the compliance and volume of the shape can be displayed by calling the **plot.py** module via the following command line:

```
python base/plot.py
```

see Figure 26.

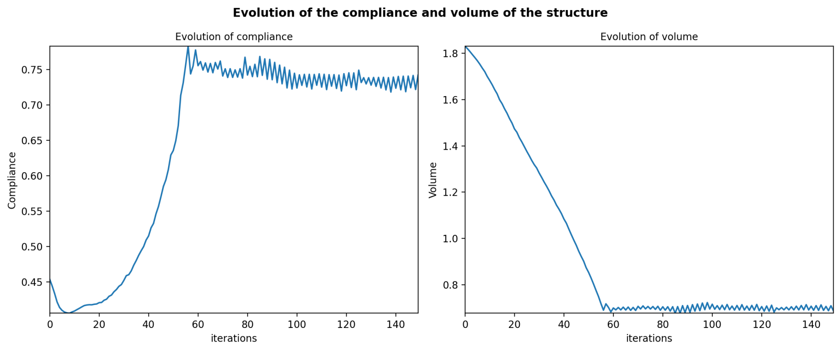

Figure 26. Evolution of the compliance and the volume functionals in the cantilever example of Appendix A.

A.5. *Description of the two files dedicated to the specification of the global variables and paths to executables*

A.5.1. *The file* **path.py**

The file **path.py** contains the global variables used in all the `python` functions.

```
# Global parameters
REFDIR          = 1      # Reference for Dirichlet B.C
REFNEU          = 2    # Reference for Neumann B.C.
REFISO          = 10    # Reference for the boundary edges
                                 # of the shape
REFINT          = 3    # Reference of the triangles
                                # in the interior domain
REFEXT          = 2   # Reference of the triangles
                               # in the exterior domain
```

The parameters regarding the desired size of the elements in the mesh are specified next; their meanings are described in Appendix A.7.

```
# Parameters of the mesh
MESHSIZ         = 0.02
HMIN            = 0.001
HMAX            = 0.02
HAUSD           = 0.001
HGRAD           = 1.3

# Other parameters
EPS             = 1e-10 # Precision parameter
EPSP            = 1e-20 # Precision parameter for packing
MAXIT           = 150   # Maximum number of iterations
                          # in the shape optimization process
MAXITLS         = 3   # Maximum number of iterations
                               # in the line search procedure
[...]
```

The file **path.py** also contains the command lines needed to call the external programs `mshdist`, `advect`, `mmg` and `FreeFem` from the terminal; as hinted at in Appendix A.2, these instructions should be supplied by the user after their installation.

```
# Call for the executables of external codes
FREEFEM  = "FreeFem++ -nw"
MSHDIST  = "mshdist"
ADVECT   = "/Users/dapogny/Advection/build/advect"
MMG2D    = "/Users/dapogny/mmg/build/bin/mmg2d_O3"
```

The names of the various `Freefem` scripts of the archive used in the `python` functions are provided:

```
# Path to FreeFem scripts
FFTEST          = SCRIPT + "testFF.edp"
FFDESCENT       = SCRIPT + "descent.edp"
FFEVALOBJ       = SCRIPT + "evalobj.edp"
...
```

Eventually, a shorthand is defined for the names of various output files at each iteration of the process, see Appendix A.4.3.

```
# Shortcut for various file types
def step(n,typ) :
  return STEP + "." + str(n) + "." + typ
```

A.5.2. *Global variables and macros related to the `Freefem` functions*

The archive contains two files `macros.idp` and `inout.idp` which are loaded in the preamble of all the `FreeFem` scripts, via the following lines.

```
include "./sources/inout.idp"
include "./sources/macros.idp"
```

The file `macros.idp` contains the global variables and the macros shared by the `FreeFem` scripts of the archive. It is organized as follows:

```
/* File for communication of data with python */
string EXCHFILE    = "./res/exch.data";

/* Inner product for extension / regularization */
real alpha         = getrParam(EXCHFILE,"Regularization");
macro psreg(u,v) ( int2d(Th)( alpha^2*(dx(u)*dx(v)+dy(u)*dy(v))
                                               + u*v) ) // EOM

/* Linear elasticity parameters */
real lm   = 0.5769;
real mu   = 0.3846;

/* Load case */
real loadx = 0.0;
real loady = -1.0;
```

In particular, this file contains the definition of the inner product $a(\cdot,\cdot)$ endowing the space of deformations θ with a Hilbertian structure. In the present context, $a(\cdot,\cdot)$ is a bilinear form, associating to two scalar-valued functions $u, v : D \to \mathbb{R}$ the value

$$a(u,v) = \alpha^2 \int_D \nabla u \cdot \nabla v \, dx + \int_D uv \, dx, \tag{39}$$

where α can be interpreted as a regularization length, see Section 4.5.

The file `inout.idp` gathers several functions in charge of reading and printing parameters or files. These functions ensure the communication with the `python` part of the implementation; their syntaxes are meant to be explicit enough so that the user does not need to enter their implementation.

```
/* Get real parameter from file */
func real getrParam(string file,string kwd) {
  [...]
}

[...]

/* Read a .sol file containing a scalar-valued solution */
func int loadsol(string sin, real[int] & u) {
  [...]
}

/* Read a .sol file containing a vector-valued solution in 2d */
func int loadvec2(string sin, real[int] & ux, real[int] & uy) {
  [...]
}

/* Save scalar function u as a .sol file */
func int printsol(string sout, real[int] & u) {
  [...]
}

/* Save vector field [ux,uy] as a .sol file */
func int printvec2(string sout, real[int] & ux,
                                real[int] & uy) {
  [...]
}
```

A.6. *Creation of the initial geometry and specification of the test case*

The function dedicated to the creation of the mesh $\mathcal{T}^0$ of D, in which the initial shape Ω^0 is explicitly discretized as a submesh $\mathcal{T}^0_{\text{int}}$, is contained in the module **inigeom.py**.

```
def iniGeom(mesh) :
```

This function proceeds in several stages:

(1) The exchange file `path.EXCHFILE` is filled with the names of the output mesh and of the intermediate level set function used in the creation of the latter:

```
# Fill in exchange file
inout.setAtt(file=path.EXCHFILE,attname="MeshName",
                                          attval=mesh)
inout.setAtt(file=path.EXCHFILE,attname="PhiName",
                                          attval=path.TMPSOL)
```

(2) A mesh of the computational domain D is generated via the call to the `FreeFem` script `path.FFINIMSH`, whose contents are described in the next listing.

```
/* Get mesh and sol names */
string MESH = getsParam(EXCHFILE,"MeshName");
int    REFDIR = getiParam(EXCHFILE,"Dirichlet");
int    REFNEU = getiParam(EXCHFILE,"Neumann");

/* Create mesh */
/* Mesh definition */
border left(t=0.0,1.0){x=0.0; y=1.0-t; label=REFDIR;};
border bot(t=0.0,2.0){x=t; y=0.0; label=0;};
border right1(t=0.0,0.45){x=2.0; y=t; label=0;};
border right2(t=0.45,0.55){x=2.0; y=t; label=REFNEU;};
border right3(t=0.55,1.0){x=2.0; y=t; label=0;};
border top(t=0.0,2.0){x=2.0-t; y=1.0; label=0;};

mesh Th = buildmesh(left(100)+bot(200)+right1(45)
                    +right2(10)+right3(45)+top(200));

/* Save mesh */
savemesh(Th,MESH);
```

(3) The resulting mesh is modified towards quality improvement, thanks to the external library `mmg`.

```
# Call to mmg2d for remeshing the background mesh
mshtools.mmg2d(mesh,0,None,path.HMIN,path.HMAX,
                              path.HAUSD,path.HGRAD,0,mesh)
```

Again, the syntax of the function **mshtools**.`mmg2d` in charge of making the interface with this external library and our `python` implementation is described in detail in Appendix A.7.

(4) In this example, Ω^0 is the total computational domain D, deprived from a collection of holes, see Figure 25 (a). In order to create the associated mesh $\mathcal{T}^0$ of D, one level set function ϕ^0 for this shape is generated via the `FreeFem` script `path.FFINILS`.

```
/* Get mesh and sol names */
string MESH = getsParam(EXCHFILE,"MeshName");
string PHI  = getsParam(EXCHFILE,"PhiName");

/* Read mesh */
mesh Th = readmesh(MESH);

/* Finite element space and functions */
```

```
fespace Vh(Th,P1);
Vh phi;

/* Definition of the initial level set function */
func real iniLS(real xx,real yy) {

 [...]

  return (dd);
}

phi = -iniLS(x,y);

/* Save LS function */
printsol(PHI,phi[]);
```

(5) This level set function ϕ^0 is explicitly discretized in the mesh by another call to mmg, resulting in the mesh displayed in Figure 25 (a).

A.7. *External calls to the libraries mshdist, advection and mmg*

This section is devoted to the python functions through which the external libraries mshdist, advection and mmg are executed in our implementation. Since they are not meant to be modified by the user, we solely present their syntax. We recall that, after the compilation of these three codes, the command lines for their executions have to be specified in the file **path.py**, as described in Appendices A.2 and A.5.1.

Let D be a domain equipped with a mesh $\mathscr{T}$ in which the shape Ω of interest is discretized, as the submesh $\mathscr{T}_{\text{int}}$ of triangles with labels path.REFINT. The calculation of the signed distance function $\phi = d_\Omega$ to Ω is carried out by the function

```
def mshdist(mesh,phi) :
```

from the module **lstools.py**. Here,

- mesh is a string of characters, bearing the name of the .mesh file for $\mathscr{T}$.
- phi is a string of characters, containing the name of the .sol file where ϕ should be saved.

When D is a domain equipped with a mesh $\mathscr{T}$, the resolution of the level set advection equation (15), starting from ϕ^0, with velocity field $V : D \to \mathbb{R}^d$, over a time period $(0, T)$, is realized via the function

```
def advect(mesh,phi,vel,step,newphi) :
```

from the module **lstools.py**. Here,

- mesh is a string of characters, containing the name of the .mesh file for $\mathscr{T}$.
- phi is a string of characters referring to the .sol file for the initial level set function ϕ^0.
- vel is a string of characters, containing the .sol file for the velocity field V.
- step is a real value, standing for the length T of the advection time period.
- newphi is a string of characters, containing the name of the .sol file where the resulting level set function should be written.

Last but not least, the remeshing operations of an input mesh $\mathscr{T}$ of a domain D involved in our framework are realized by the `mmg` library. All the calls to this library are encapsulated in the proposed `python` implementation via the function

```
def mmg2d(mesh,ls,phi,hmin,hmax,hausd,hgrad,nr,out) :
```

defined in the module **mshtools.py**. The parameters of this function are the following:

- `mesh` is a string of characters bearing the name of the input mesh $\mathscr{T}$.
- `ls` is an integer, taking the values 0 or 1. When `ls`= 0, the mesh $\mathscr{T}$ is remeshed towards improving the quality of its elements; when `ls`= 1, the 0 level set of the function `phi` is explicitly discretized in $\mathscr{T}$, which is then remeshed towards improving the quality of its elements.
- `phi` is a string of characters, bearing the name of the level set function in the case where the `ls` parameter is set to 1.
- `hmin` is a real value, corresponding to the minimum authorized length for an edge in the mesh.
- `hmax` is a real value, corresponding to the maximum authorized length for an edge in the mesh.
- `hausd` is a real value, encoding the maximum gap authorized between an edge on the surface part $\mathscr{S}_{\mathscr{T}}$ of $\mathscr{T}$ and the underlying curve drawn on the ideal surface, see Section 4.4.2 and Figure 9.
- `hgrad` is a gradation parameter controlling the shocks of lengths between adjacent edges in the mesh. It is a real value, specifying the maximum ratio allowed between the lengths of two adjacent edges (typical values are 1.3 or 1.4).
- `nr` is an integer taking two values 0, or 1. It equals 1 when sharp angles are to be detected by `mmg`, then treated as corners during remeshing, 0 otherwise.
- `out` is a string of characters containing the name of the output mesh.

The function `mmg2d` returns 1 if the remeshing process is successful and 0 if an error occurred.

A.8. *Mechanical calculations*

Let D be a computational domain, equipped with a mesh $\mathscr{T}$ in which the shape Ω of interest is explicitly discretized, as the submesh $\mathscr{T}_{\text{int}}$ composed of the triangles with label `path.REFINT`.

The resolution of the linear elasticity system for the displacement u_Ω is achieved by the `elasticity` function, pertaining to the module **mechtools.py**.

```
def elasticity(mesh,u) :
```

In this command,

- `mesh` is a string of characters, bearing the name of the mesh $\mathscr{T}$.
- `u` is a string of characters for the name of the output vector field u_Ω. The latter is treated as a $\mathbb{P}_1$ finite element vector field, stored as a `.sol` file containing values at all the vertices of $\mathscr{T}$, being understood that only those values at the vertices of $\mathscr{T}_{\text{int}}$ will be considered.

The function proceeds by first entering this information in the exchange file `path.EXCHFILE`, then calling `Freefem` with the script `path.FFELAS`, that we now touch on.

After a few lines, where notably the mesh `Th` of D is loaded, this mesh is truncated thanks to the `trunc` function.

```
/* Mesh of the interior part and corresponding FE space */
mesh Thi = trunc(Th,(reg(x,y) == REFINT),label=REFINT);
```

```
fespace Vhi(Thi,P1);
Vhi uix,uiy,vix,viy;
```

Hence, `Thi` is the desired mesh $\mathscr{T}_{\text{int}}$ of the shape Ω and `Vhi` is the corresponding finite element space of $\mathbb{P}_1$ Lagrange finite element functions. Then, the linear elasticity system is solved on this interior mesh.

```
/* Variational formulation of the problem */
problem elas([uix,uiy],[vix,viy]) =
  int2d(Thi)(mu*(2.0*dx(uix)*dx(vix)
                    + (dx(uiy)+dy(uix))*(dx(viy)+dy(vix))
                    + 2.0*dy(uiy)*dy(viy))
                    + lm*(dx(uix)+dy(uiy))*(dx(vix)+dy(viy)))
 - int1d(Thi,REFNEU)(loadx*vix+loady*viy)
 + on(REFDIR,uix=0.0,uiy=0.0);

/* Solve problem */
elas;
```

Finally, the result is transferred onto the total mesh of D.

```
/* Transfer the problem on the full mesh */
ux = uix;
uy = uiy;
```

The calculation of the compliance of Ω is easily realized from the datum of the mesh `mesh` and of the resulting elastic displacement `u`. In our implementation, this task relies on the function

```
def compliance(mesh,u) :
```

from the module **mechtools.py**. This function essentially calls the `FreeFem` script `path.FFCPLY`, which is organized as follows.

```
/* Get mesh and sol names, and global parameters */
string MESH      = getsParam(EXCHFILE,"MeshName");
string SOL       = getsParam(EXCHFILE,"DispName");
int REFINT       = getiParam(EXCHFILE,"Refint");
int REFNEU       = getiParam(EXCHFILE,"Neumann");
int REFISO       = getiParam(EXCHFILE,"ReferenceBnd");

/* Loading mesh */
mesh Th = readmesh(MESH);

[...]
/* Read solution */
loadvec2(SOL,ux[],uy[]);

/* Calculate compliance */
cply = int2d(Th,REFINT)(mu*(2.0*dx(ux)*dx(ux)
                             + (dx(uy)+dy(ux))*(dx(uy)+dy(ux))
                             + 2.0*dy(uy)*dy(uy))
                        + lm*(dx(ux)+dy(uy))*(dx(ux)+dy(uy)));
```

```
/* Save result */
setrParam(EXCHFILE,"Compliance",cply);
```

The calculation of the volume of Ω is likewise conducted via the function

```
def volume(mesh) :
```

from the module **mechtools.py**, whose syntax is analogous.

A.9. *Calculation of a descent direction*

In this section, we detail the calculation of shape gradients for the compliance and volume shape functionals, and that of a descent direction θ for the optimization problem (36). The shape Ω of interest is supplied by a mesh $\mathscr{T}$ of the computational domain D in which it is explicitly discretized, as the submesh $\mathscr{T}_{\text{int}}$ of triangles with reference `path.REFINT`. Let us recall from Sections 4.5 and 4.6 that these calculations hinge on the choice of an inner product acting on the vector space of deformation fields, according to the so-called Hilbertian extension-regularization procedure. The macro for this inner product is supplied in the `macros.idp` file, as described in Appendix A.5.2.

A.9.1. *Calculation of the gradient of the compliance (and volume) functional*

The calculation of the shape gradient for the compliance functional is realized by the function

```
def gradCp(mesh,disp,grad) :
```

from the **mechtools.py** module. This function essentially calls `FreeFem` with the file `path.FFGRADCP`, which is organized as follows.

```
/* Get mesh and sol names */
[...]

/* Loading mesh */
mesh Th = readmesh(MESH);

/* Finite element spaces and functions */
fespace Vh(Th,P1);
fespace Vh0(Th,P0);

Vh ux,uy,g,v;
Vh0 reg = region;

/* Load elastic displacement */
loadvec2(DISP,ux[],uy[]);

/* Mesh of the interior part and corresponding FE spaces */
mesh Thi = trunc(Th,(reg(x,y) == REFINT),label=REFINT);
fespace Vhi(Thi,P1);
fespace Vh0i(Thi,P0);
Vhi uix,uiy;
Vh0i Aeueu;
```

```
/* Integrand of the shape derivative */
uix = ux;
uiy = uy;
Aeueu = mu*(2.0*dx(uix)*dx(uix)
                     + (dx(uiy)+dy(uix))*(dx(uiy)+dy(uix))
                     + 2.0*dy(uiy)*dy(uiy))
              + lm*(dx(uix)+dy(uiy))*(dx(uix)+dy(uiy));

/* Resolution of the extension - regularization problem */
problem velext(g,v) = psreg(g,v)
                        - int1d(Thi,REFISO)(-Aeueu*v)
                        + on(REFNEU,g=0.0);

/* Solve problem and save solution */
velext;
printsol(GRADCP,g[]);
```

In a few words, the variational problem featured in this script solves the identification problem (23) with the inner product $a(\cdot,\cdot)$ – given by (39) and supplied in the file `macros.idp` described in Appendix A.5.2 – and the right-hand side $C'(\Omega)(\cdot n)$ given by (38).

The calculation of the gradient of the volume functional follows the exact same trail, and we do not detail further the contents of the corresponding function

```
def gradV(mesh,grad) :
```

A.9.2. *Derivation of a descent direction*

The null-space optimization algorithm described in Section 4.6 is implemented in the function

```
def descent(mesh,phi,Cp,gCp,vol,gV,g) :
```

from the **mechtools.py** module. The latter essentially calls `FreeFem` with the file `path.FFDESCENT`, which proceeds as follows.

After reading the computational mesh $\mathscr{T}$ of D, finite element spaces and parameters are declared. The finite element functions associated to the level set function ϕ for Ω and the gradients of the compliance and volume functionals are loaded.

```
loadsol(PHI,phi[]);
loadsol(GRADCP,thJ[]);
loadsol(GRADV,thG[]);
```

The contributions ξ_J and ξ_G to total descent direction θ, given by (27), are then calculated.

```
/* Coefficients for the descent direction */
m = psreg(thG,thG);
lambda = 1.0 / m * psreg(thJ,thG);

/* Null space and range contributions to descent direction */
xiJ = thJ - lambda*thG;
xiG = 1.0/m*(vol-vtarg)*thG;
```

The coefficients α_J and α_G are next calculated, and the scalar amplitude of the descent direction is inferred, see again (27) and Section 4.5.

```
maxxiJ = max(-xiJ[].min,xiJ[].max);
mMaxxiJ = max(maxxiJ,maxNormXiJ);
maxxiG = max(-xiG[].min,xiG[].max);
alphaJ = AJ*meshsiz / (eps^2+mMaxxiJ);
alphaG = AG*meshsiz / (eps^2+maxxiG);

/* Scalar descent direction */
g = - alphaJ*xiJ - alphaG*xiG;
```

Then, the extended normal vector field n to the boundary of the shape Ω is calculated on the whole domain D via the formula $n(x) = \frac{\nabla\phi(x)}{|\nabla\phi(x)|}$, first as a $\mathbb{P}_0$ vector field on $\mathscr{T}$, then as a $\mathbb{P}_1$ vector field.

```
/* Extended normal vector as a P0 function over the mesh */
norm0 = sqrt(eps+dx(phi)*dx(phi)+dy(phi)*dy(phi));
nx0 = dx(phi) / norm0;
ny0 = dy(phi) / norm0;

/* Extended normal vector as a P1 function over the mesh */
problem extnx(nx,v) = int2d(Th)(nx*v)
                      - int2d(Th)(nx0*v);

problem extny(ny,v) = int2d(Th)(ny*v)
                       - int2d(Th)(ny0*v);


extnx;
extny;

norm = sqrt(eps^2+nx*nx+ny*ny);
nx = nx / norm;
ny = ny / norm;
```

Eventually, a descent direction for the optimization problem (36) is obtained as a vector field $D \to \mathbb{R}^2$ and the data are saved in the appropriate files.

```
/* Save solution */
printvec2(GRAD,gx[],gy[]);

/* Save coefficients for the merit function */
setrParam(EXCHFILE,"Lagrange",lambda);
setrParam(EXCHFILE,"Penalty",m);
setrParam(EXCHFILE,"alphaJ",alphaJ);
setrParam(EXCHFILE,"alphaG",alphaG);
setrParam(EXCHFILE,"NormXiJ",maxxiJ);
```

A.9.3. *Evaluation of the merit function*

The evaluation of the merit $M(\Omega)$ of a shape Ω, defined in (28), is simply realized via the function `merit` of the **mechtools** module:

```
def merit(Cp,vol) :
```

This simple function takes as arguments the compliance `Cp` and the volume `vol` of the shape, which have been calculated via the functions defined in Appendix A.8. It also relies on the coefficients λ, S, α_J, α_G produced by the optimization algorithm of Appendix A.9, stored in the exchange file `path.EXCHFILE`.

```
  # Read parameters in the exchange file
  [alphaJ] = inout.getrAtt(file=path.EXCHFILE,attname="alphaJ")
  [alphaG] = inout.getrAtt(file=path.EXCHFILE,attname="alphaG")
  [ell] = inout.getrAtt(file=path.EXCHFILE,attname="Lagrange")
  [m] = inout.getrAtt(file=path.EXCHFILE,attname="Penalty")
  [vtarg] = inout.getrAtt(file=path.EXCHFILE,
                                  attname="VolumeTarget")

  merit = alphaJ*(Cp - ell*(vol-vtarg))
                        + 0.5*alphaG/m*(vol-vtarg)**2

  return merit
```

A.10. *Elaborations upon this code*

The code described in this appendix has been prepared with the concern that it should be reasonably simple to adapt to the treatment of a wide variety of shape optimization problems. In this section, we briefly describe four such settings, illustrating as many features that can advantageously enrich the base code. The corresponding source files are provided in the `github` repository of the project.

A.10.1. *A combination of shape and topological derivatives: minimization of the compliance of a 2d bridge*

This section illustrates the resolution of a shape and topology optimization problem by means of a combination of the boundary variation Algorithm 3 with an occasional use of topological derivatives, as originally proposed in [9].

As we have mentioned in Remark 2, the topological derivative $\mathrm{d}J_T(\Omega)(x)$ of a function $J(\Omega)$ at some point $x \in \Omega$ is the first non trivial term in the asymptotic expansion

$$J(\Omega \setminus \overline{B(x,r)}) = J(\Omega) + r^d \mathrm{d}J_T(\Omega)(x) + \mathrm{o}(r^d);$$

in particular, the value of $J(\Omega)$ is decreased if a tiny hole is nucleated around a point $x \in \Omega$ where $\mathrm{d}J_T(\Omega)(x)$ is negative. Let us recall the following expressions of the topological derivatives of the compliance and volume functionals $C(\Omega)$ and $\mathrm{Vol}(\Omega)$:

$$\mathrm{d}C_T(\Omega)(x) = \frac{\pi(\lambda+2\mu)}{2\mu(\lambda+\mu)}\Big(4\mu Ae(u_\Omega) : e(u_\Omega) + (\lambda-\mu)\,\mathrm{tr}(Ae(u_\Omega))\,\mathrm{tr}(e(u_\Omega))\Big)(x),$$
$$\text{and} \quad \mathrm{dVol}_T(\Omega)(x) = -\pi, \qquad (40)$$

see e.g. [61, 84].

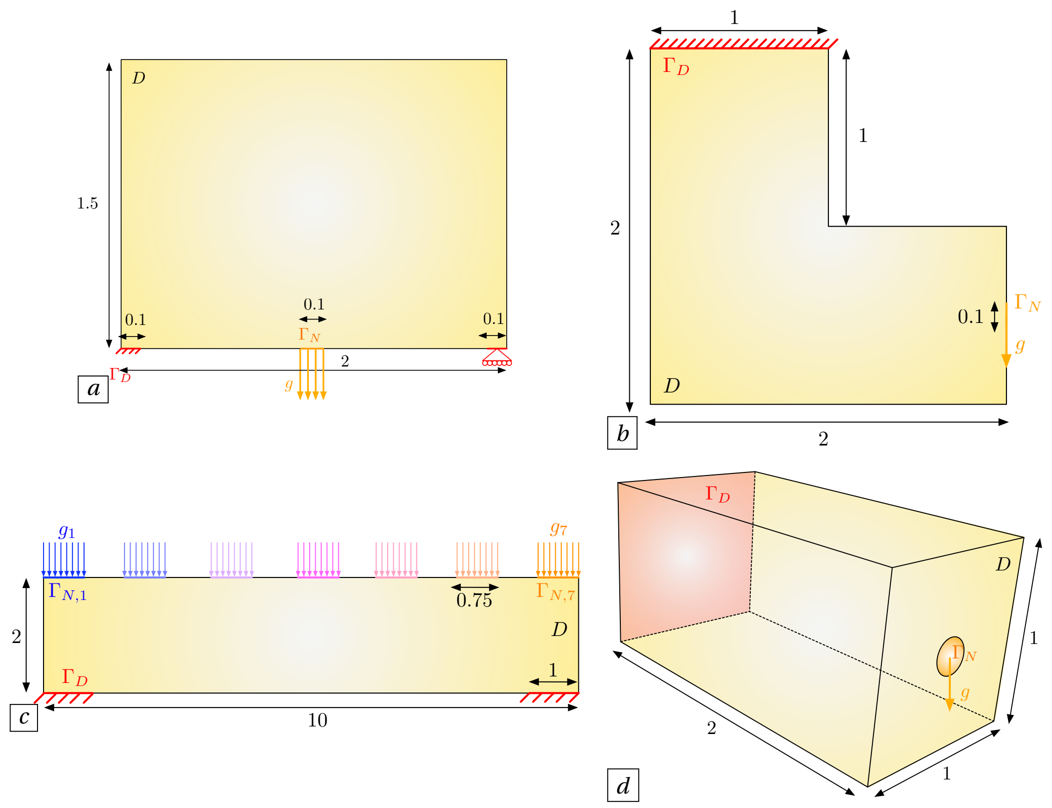

Figure 27. (a) Setting of the bridge test case considered in Appendix A.10.1; (b) setting of the L-shaped beam example considered in Appendix A.10.2; (c) setting of the multi-load bridge considered in Appendix A.10.3; (d) setting of the 3d cantilever test case considered in Appendix A.10.4.

The considered example deals with the optimization of a 2d bridge, as depicted on Figure 27 (a). The shapes Ω of interest are contained in a box D with size 2×1.5; they are clamped on their lower-left corner, their vertical displacement is prevented on their lower-right corner, and a unit vertical load $g = (0,-1)$ is applied on a small region Γ_N around the middle of their bottom side. In this context, we aim to minimize the compliance $C(\Omega)$ of Ω under a constraint on its volume $\text{Vol}(\Omega)$: we solve problem (36) with a target volume $V_T = 0.7$.

To achieve this, we rely on the boundary variation Algorithm 4 with an extra ingredient: at each iteration n of the main loop, a boolean value `dotopder` is calculated:

```
 # Equals 1 if topological derivative operation, 0 else
dotopder = 1 if ( it % 5 == 0 and it <= 50 ) else 0
```

The iterations n where this parameter equals 0 unfold exactly as those of the base code: the shape gradients of $C(\Omega)$ and $\text{Vol}(\Omega)$ are calculated, a descent direction for the problem (36) is inferred from the null space optimization algorithm described in Section 4.6 and Appendix A.9, then the shape Ω^n is updated into the new iterate Ω^{n+1} (a level set function ϕ^n for Ω^n is calculated on the mesh $\mathcal{T}^n$ of D, the level set advection equation is solved, etc.).

The iterations n where `dotopder` equals 1 are associated to a stage of topological sensitivity analysis, which proceeds along the following lines.

```
# Modification of the shape using topological gradients
if ( dotopder ) :
  # Calculation of the topological derivatives of
  # the compliance and volume
  print("  Computation of topological derivatives")
  mechtools.topgradCp(curmesh,curu,curtopCpgrad)
  mechtools.topgradV(curmesh,curtopVgrad)

  # Calculation of the topological derivative of merit
  mechtools.descentTG(curmesh,curphi,curCp,curtopCpgrad,curvol,
                                  curtopVgrad,curtopgrad)

  # Update of the level set function of the shape
  lstools.creaHoles(curmesh,curphi,curtopgrad,
                          path.VFRACTG,newphi)

  # Creation of a mesh associated to the new shape
  print("  Local remeshing")
  retmmg = mshtools.mmg2d(curmesh,1,newphi,path.HMIN,path.HMAX,
                              path.HAUSD,path.HGRAD,1,newmesh)

  [...]

  # Decision
  [...]
```

Briefly, the topological derivatives of $C(\Omega)$ and $\text{Vol}(\Omega)$ are calculated from the formulas (40), thanks to the functions `topgradCp` and `topgradV` of the **mechtools.py** module. These essentially call the `FreeFem` scripts `path.FFTOPGRADCP` and `path.FFTOPGRADV`, whose syntaxes are quite straightforward. Then, the function `descentTG` of the **mechtools.py** module infers a descent direction for (36) as a scalar field on the computational domain, owing to a simple adaptation of the null space algorithm of Section 4.6. Finally, the function

```
def creaHoles(mesh,phi,grad,volfrac,newphi) :
```

of the **lstools.py** module is called. The latter expects the following arguments:

- A string of characters `mesh` for the name of the current mesh of D.
- The name `phi` of the file containing a level set function for the shape Ω^n.
- The name `grad` of the file containing the descent direction for (36) resulting from the topological sensitivity analysis.
- A real parameter `volfrac` (taking values between 0 and 1).
- The name `newphi` of the output file containing the level set function of the new shape Ω^{n+1}.

This function essentially relies on the `FreeFem` script `path.FFUPTG`. It first identifies by dichotomy the `volfrac` percentage of elements of the input shape `phi` where the values of `grad` are the most negative, and it then returns a new level set function `newphi` featuring holes in place of these elements.

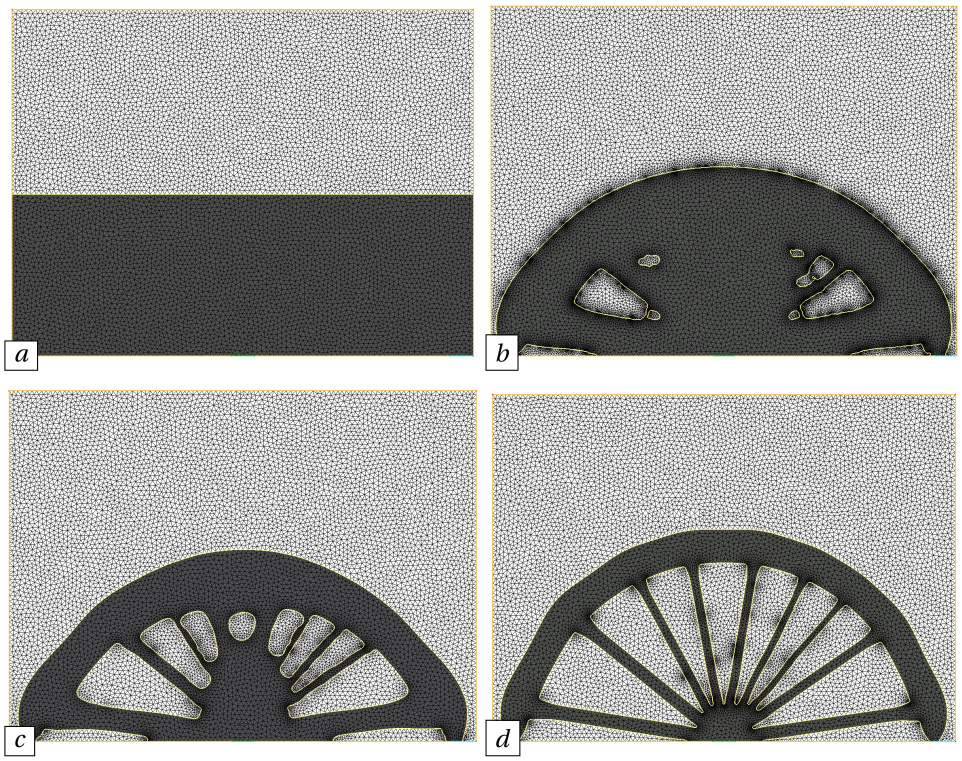

Figure 28. Iterations (a) 0, (b) 26, (c) 40 and (d) 150 of the optimization of the shape of a 2d bridge with the combined use of shape and topological derivatives proposed in Appendix A.10.1.

A few iterations of the resolution of (36) with this strategy in the 2d bridge example are depicted on Figure 28. The corresponding source code is contained in the following folder:

```
https://github.com/dapogny/sotuto/topder
```

A.10.2. *Handling shape functionals whose derivative involve an adjoint state: minimization of the stress within an L-shaped beam*

The present example illustrates how the base code described in this appendix can be adapted to handle other objective functionals than the compliance, whose shape derivatives involve an adjoint state.

Let us consider the physical situation depicted on Figure 29. Shapes Ω are beams contained in an L-shaped domain D with size 1×1; they are clamped on their upper side Γ_D, and a unit vertical load $g = (0, -1)$ is applied on a small region Γ_N around the middle of their right-hand side. In this context, we consider the shape optimization problem

$$\min_{\Omega} S(\Omega) \text{ s.t. } \mathrm{Vol}(\Omega) = V_T, \tag{41}$$

where $S(\Omega)$ is the integral measure of the stress inside Ω defined in (4) with a weight function $k(x)$ equal to 1 everywhere except in two small regions around Γ_D and Γ_N, where it equals 0. The volume target is set to $V_T = 0.7$. The shape derivative $S'(\Omega)(\theta)$ of $S(\Omega)$ is provided in (9); its expression involves the adjoint state $p_\Omega \in H^1_{\Gamma_D}(\Omega)^d$, which is defined as the solution to the system (10).

The numerical resolution of this problem is almost identical to that featured in the base implementation, except when it comes to the calculation of a shape gradient for the stress-based functional $S(\Omega)$. At each iteration n of the main loop, this operation proceeds along the following lines:

```
# Calculation of a descent direction
print("  Computation of the adjoint state ")
mechtools.adjoint(curmesh,curu,curp)

# Calculation of the gradients of stress and volume
print("  Computation of a descent direction")
mechtools.gradS(curmesh,curu,curp,curSgrad)
mechtools.gradV(curmesh,curVgrad)
mechtools.descent(curmesh,curphi,curS,curSgrad,curvol,
                                                  curVgrad,curgrad)
```

In short, the adjoint state p_{Ω^n} is calculated by the finite element method, thanks to the function `adjoint` of the **mechtools.py** module; the latter consists in turn in executing the `FreeFem` script `path.FFADJ`, which is very similar to the script `path.FFELAS` described in Appendix A.8. Then, the function `gradS` of the `mechtools.py` module is in charge of calculating a shape gradient for $S(\Omega)$ at Ω^n, in a analogous way to the computation described in the above Appendix A.9.1.

A few iterations of the resolution of (41) in the setting of the L-shaped beam example are presented on Figure 29. The source code associated to this development is contained in the following folder:

`https://github.com/dapogny/sotuto/stress`

A.10.3. *Using multiple equality or inequality constraints: optimization of the shape of a bridge submitted to multiple loads*

In this section, we illustrate how shape optimization problems featuring multiple equality or inequality constraints can be handled with an adaptation of the base code. This task relies on the open-source implementation of the null space optimization algorithm of Section 4.6, which is provided at the address

`https://gitlab.com/florian.feppon/null-space-optimizer`

where a documentation of the library is available.

The physical situation of interest is that depicted on Figure 27(c): the considered shape Ω represents a two-dimensional bridge contained in a box D with size 10×2. It is clamped on its lower-left and lower-right corners, and 7 loads $g_i = (0, -1)$ are applied on as many different regions $\Gamma_{N,i}$ $(i = 1, \dots, 7)$ of its upper side. In this setting, we aim to solve the optimization problem

$$\min_{\Omega} \mathrm{Vol}(\Omega) \text{ s.t } C_i(\Omega) \leq C_T, \tag{42}$$

In this formulation, $C_i(\Omega)$ is the compliance of Ω when the i^{th} load g_i is applied on $\Gamma_{N,i}$, involving the solution $u_{\Omega,i}$ to the linear elasticity system (1) in this situation. The threshold C_T is calculated from the values $C_i(\Omega^0)$ of the compliances of the initial design Ω^0 in each load scenario:

$$C_T = 0.9 \max_{i=1,\dots,7} C_i(\Omega^0).$$

The treatment of this problem with the null space optimization library essentially requires a modification of the main loop of the optimization strategy. Briefly, the main steps of the optimization strategy are re-organized as the methods of a new class `Bridge`.

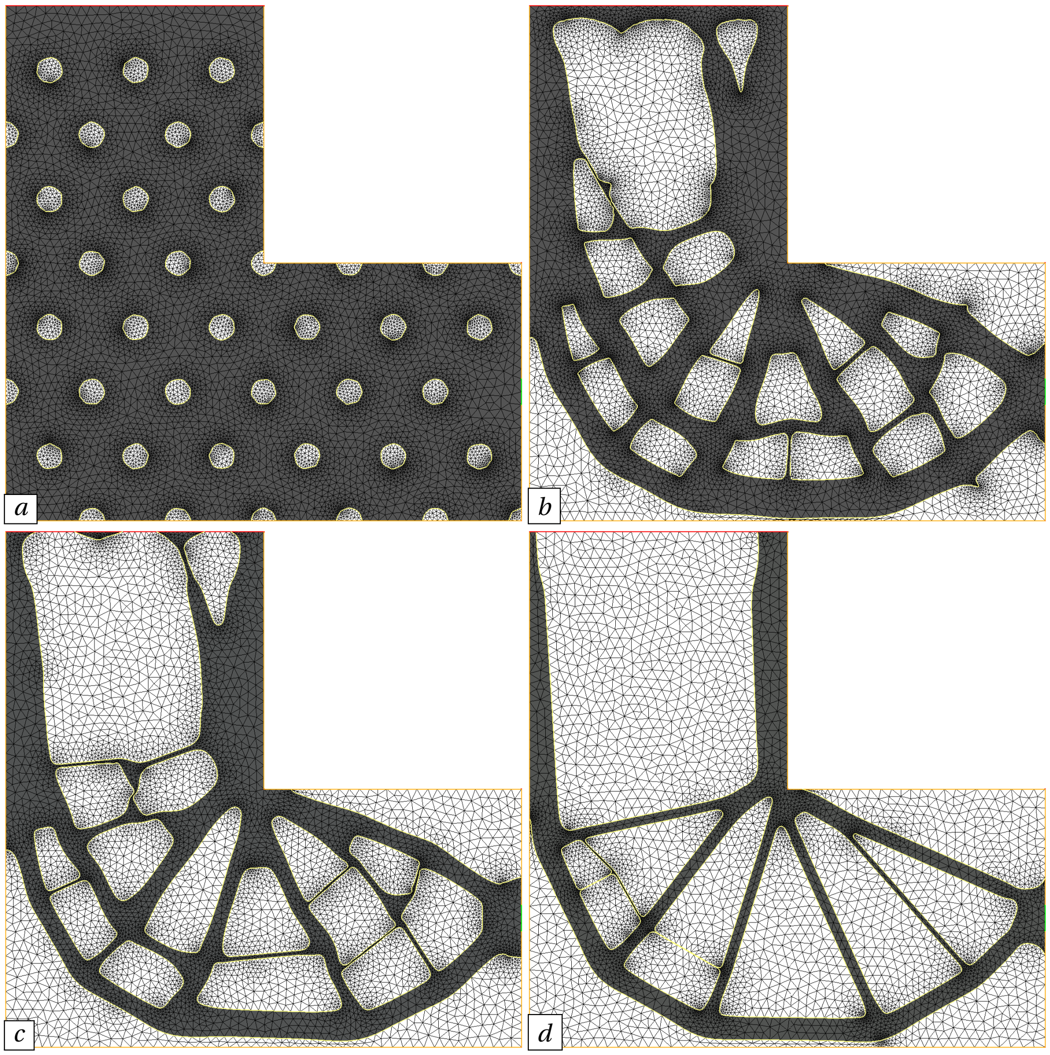

Figure 29. Iterations (a) 0, (b) 26, (c) 35 and (d) 200 (final) of the optimization of the shape of an L-beam with respect to an integral measure of the stress, as presented in Appendix A.10.2.

```
########### Definition of Optimizable class Bridge ############
class Bridge(Optimizable) :

  # Initialization
  def x0(self) :
    return path.step(0,"mesh")

  [...]

  # Calculation of the objective and constraint functions
  def evalObjective(self,x) :
      [...]
      for i in range(0,path.NC) :
        refneu = path.REFNEU + i
        gx     = path.GX[i]
        gy     = path.GY[i]
        ui     = nam + ".u."+str(i)+".sol"
        mechtools.elasticity(x,refneu,gx,gy,ui)

      # Calculate J and H
      J = mechtools.volume(x)
      Cptab = []
      for i in range(0,path.NC) :
        Cp = mechtools.compliance(x,nam+".u."+str(i)+".sol")

  [...]

  # Shape derivatives, sensitivity of objective and constraint
  def evalSensitivities(self,x) :
      [...]
      # Calculate dJ and gradJ
      mechtools.gradV(x,nam+".diffV.sol",nam+".gradV.sol")

      # Calculate dH and gradH
      for i in range(0,path.NC) :
        mechtools.gradCp(x,nam+".u."+str(i)+".sol",
                 nam+".diffCp."+str(i)+".sol",
                              nam+".gradCp."+str(i)+".sol")

      [...]

  # Retraction: shape update
  # dx = array of np values containing values
  #                              of the scalar velocity field
  def retract(self, x, dx) :
    [...]

    # Generation of a level set function for $\Omega^n$ on $D$
    print("  Creation of a level set function")
    lstools.mshdist(curmesh,curphi)

    # Put scalar velocity defined on D in the normal direction
    inout.saveSol(dx,path.TMPSOL)
    lstools.norvec(curmesh,curphi,path.TMPSOL,curgrad)

    # Advection of the level set function
    print("  Level Set advection")
    lstools.advect(curmesh,curphi,curgrad,1.0,newphi)

    # Creation of a mesh associated to the new shape
    print("  Local remeshing")
    retmmg = mshtools.mmg2d(curmesh,1,newphi,path.HMIN,
                  path.HMAX, path.HAUSD,path.HGRAD,1,newmesh)
    return newmesh

  # Accept step
  def accept(self,results) :
    [...]
```

The optimization is then launched by applying the `nlspace_solve` function to an object of this class.

```
# Run optimization solver
optSettings = {"dt" : path.MESHSIZ,
               "alphaJ" : 1.0,
               "alphaC" : 2.0,
               "maxit" : 300,
               "provide_gradient" : True,
               "maxtrials" : 3,
               "itnormalisation" : 3
              }
results=nlspace_solve(Bridge(), optSettings)
```

A few intermediate shapes arising during the resolution of (42) are represented on Figure 30; the source code corresponding to this example is contained in the following folder:

`https://github.com/dapogny/sotuto/constraints`

A.10.4. *A 3d implementation: optimization of a 3d cantilever beam*

In this section, we briefly illustrate how our base implementation can be adapted to handle three-dimensional problems.

The physical situation of interest is that of a 3d cantilever beam, which is sketched on Figure 27 (d): the shapes Ω are contained in a box D with size $2 \times 1 \times 1$; they are clamped on their face lying in the plane $\{x = (x_1, x_2, x_3) \in \partial D,\ x_1 = 0\}$ and a unit vertical load $g = (0, 0, -1)$ is applied on a small disk Γ_N around the middle of the opposite side. In this context, we aim to minimize the compliance $C(\Omega)$ of Ω under a volume constraint: problem (36) is solved with a volume target $V_T = 0.45$.

The main numerical strategy is very similar to the two-dimensional base implementation. The three-dimensional version of `FreeFem` is used for the finite element calculation involved, which demands minor adaptations of the scripts described in the main part of this tutorial; see the documentation of `FreeFem` for further details. The external libraries `mshdist` and `advect` are called in exactly the same way as in the previous two-dimensional cases, and the three-dimensional version `mmg3d` of the remeshing software `mmg` is used in place of `mmg2d`.

```
# Call for the executables of external codes
FREEFEM  = "FreeFem++ -nw"
MSHDIST  = "mshdist"
ADVECT   = "/Users/dapogny/Advection/build/advect"
MMG3D    = "/Users/dapogny/mmg/build/bin/mmg3d_O3"
```

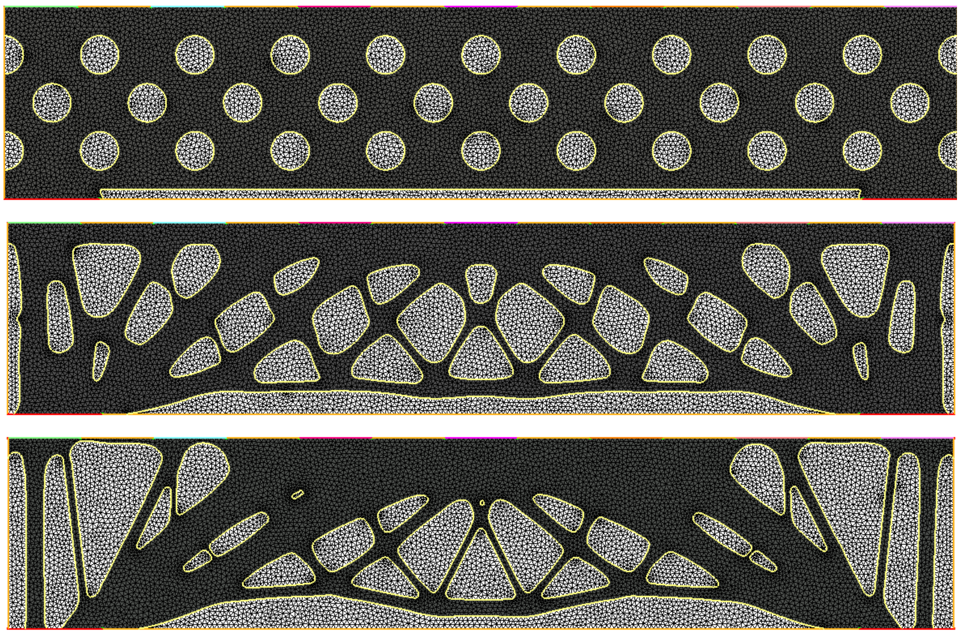

Figure 30. (From top to bottom) Iterations 0, 15 and 200 of the optimization of the shape of a 2d bridge submitted to 7 constraints on its compliance in as many load situations, as described in Appendix A.10.3.

The code mmg3d is called with the help of the python function

```
def mmg3d(mesh,ls,phi,hmin,hmax,hausd,hgrad,nr,out) :
```

of the mshtools.py module, which operates in exactly the same way as the function mmg2d described in Appendix A.7.

A few iterations of the resolution of problem (36) are represented on Figure 31; the associated source code is contained in the following folder:

https://github.com/dapogny/sotuto/base3d

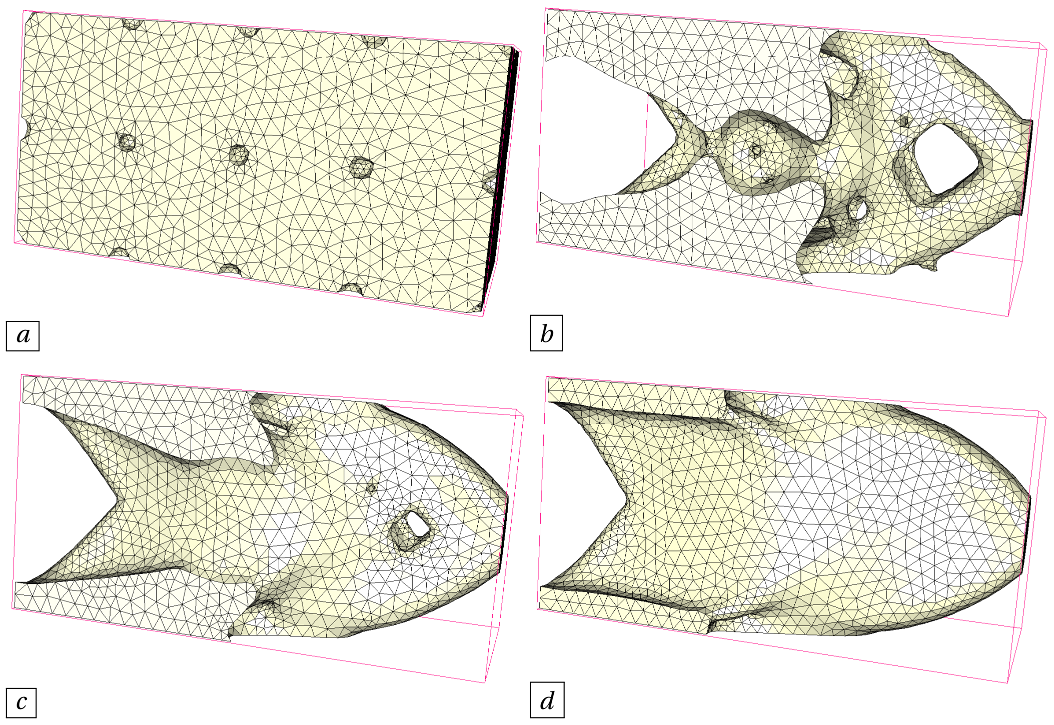

Figure 31. Iterations (a) 0, (b) 50, (c) 120 and (d) 200 (final) of the three-dimensional cantilever example presented in Appendix A.10.4.

References

[1] R. Abgrall, "Numerical discretization of the first-order Hamilton-Jacobi equation on triangular meshes", *Commun. Pure Appl. Math.* **49** (1996), no. 12, p. 1339-1373.

[2] R. A. Adams, J. J. F. Fournier, *Sobolev spaces*, Pure and Applied Mathematics, vol. 140, Academic Press Inc., 2003.

[3] G. Allaire, *Shape optimization by the homogenization method*, vol. 146, Springer, 2002.

[4] G. Allaire, E. Bonnetier, G. Francfort, F. Jouve, "Shape optimization by the homogenization method", *Numer. Math.* **76** (1997), no. 1, p. 27-68.

[5] G. Allaire, C. Dapogny, P. Frey, "Topology and geometry optimization of elastic structures by exact deformation of simplicial mesh", *C. R. Math. Acad. Sci. Paris* **349** (2011), no. 17-18, p. 999-1003.

[6] ———, "A mesh evolution algorithm based on the level set method for geometry and topology optimization", *Struct. Multidiscip. Optim.* **48** (2013), no. 4, p. 711-715.

[7] ———, "Shape optimization with a level set based mesh evolution method", *Comput. Methods Appl. Mech. Eng.* **282** (2014), p. 22-53.

[8] G. Allaire, C. Dapogny, F. Jouve, "Shape and topology optimization", in *Geometric partial differential equations. Part II*, Handbook of Numerical Analysis, vol. 22, Elsevier; North-Holland, 2021, p. 1-132.

[9] G. Allaire, F. De Gournay, F. Jouve, A.-M. Toader, "Structural optimization using topological and shape sensitivity via a level set method", *Control Cybern.* **34** (2005), no. 1, p. 59.

[10] G. Allaire, F. Jouve, G. Michailidis, "Thickness control in structural optimization via a level set method", *Struct. Multidiscip. Optim.* **53** (2016), no. 6, p. 1349-1382.

[11] G. Allaire, F. Jouve, A.-M. Toader, "Structural optimization using sensitivity analysis and a level-set method", *J. Comput. Phys.* **194** (2004), no. 1, p. 363-393.

[12] G. Allaire, O. Pantz, "Structural optimization with `FreeFem++`", *Struct. Multidiscip. Optim.* **32** (2006), no. 3, p. 173-181.

[13] G. Allaire, M. Schoenauer, *Conception optimale de structures*, vol. 58, Springer, 2007.

[14] S. Amstutz, H. Andrä, "A new algorithm for topology optimization using a level-set method", *J. Comput. Phys.* **216** (2006), no. 2, p. 573-588.

[15] H. Azegami, *Shape Optimization Problems*, Springer, 2020.

[16] H. Azegami, Z. C. Wu, "Domain optimization analysis in linear elastic problems: approach using traction method", *JSME Int. J. Ser. A, Mech. Mater. Eng.* **39** (1996), no. 2, p. 272-278.

[17] T. Baker, P. Cavallo, "Dynamic adaptation for deforming tetrahedral meshes", in *14th Computational Fluid Dynamics Conference*, 1999, p. 3253.

[18] G. Balarac, F. Basile, P. Bénard, F. Bordeu, J.-B. Chapelier, L. Cirrottola, G. Caumon, C. Dapogny, P. Frey, A. Froehly, G. Ghigliotti, R. Laraufie, G. Lartigue, C. Legentil, R. Mercier, V. Moureau, C. Nardoni, S. Pertant, M. Zakari, "Tetrahedral remeshing in the context of large-scale numerical simulation and high performance computing", *MathS In Action* **11** (2022), p. 129-164.

[19] C. Barbarosie, S. Lopes, A.-M. Toader, "An algorithm for constrained optimization with applications to the design of mechanical structures", in *International Conference on Engineering Optimization*, Springer, 2018, p. 272-284.

[20] N. Barral, F. Alauzet, "Three-dimensional CFD simulations with large displacement of the geometries using a connectivity-change moving mesh approach", *Eng. Comput.* **35** (2019), no. 2, p. 397-422.

[21] T. J. Barth, J. A. Sethian, "Numerical schemes for the Hamilton–Jacobi and level set equations on triangulated domains", *J. Comput. Phys.* **145** (1998), no. 1, p. 1-40.

[22] M. P. Bendsøe, N. Kikuchi, "Generating optimal topologies in structural design using a homogenization method", *Comput. Methods Appl. Mech. Eng.* **71** (1988), no. 2, p. 197-224.

[23] M. P. Bendsøe, O. Sigmund, *Topology optimization: theory, methods, and applications*, Springer, 2013.

[24] L. Blank, H. Garcke, L. Sarbu, T. Srisupattarawanit, V. Styles, A. Voigt, "Phase-field approaches to structural topology optimization", in *Constrained optimization and optimal control for partial differential equations*, Springer, 2012, p. 245-256.

[25] H. Borouchaki, P.-L. George, *Meshing, Geometric Modeling and Numerical Simulation 1: Form Functions, Triangulations and Geometric Modeling*, John Wiley & Sons, 2017.

[26] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, B. Lévy, *Polygon mesh processing*, CRC Press, 2010.

[27] B. Bourdin, A. Chambolle, "Design-dependent loads in topology optimization", *ESAIM, Control Optim. Calc. Var.* **9** (2003), p. 19-48.

[28] B. Braida, J. Dalphin, C. Dapogny, P. Frey, Y. Privat, "Shape and topology optimization for maximum probability domains in quantum chemistry", *Numer. Math.* (2022), p. 1-48.

[29] D. Bucur, G. Buttazzo, *Variational methods in some shape optimization problems*, Springer, 2002.

[30] C. Bui, C. Dapogny, P. Frey, "An accurate anisotropic adaptation method for solving the level set advection equation", *Int. J. Numer. Methods Fluids* **70** (2012), no. 7, p. 899-922.

[31] M. Burger, "A framework for the construction of level set methods for shape optimization and reconstruction", *Interfaces Free Bound.* **5** (2003), no. 3, p. 301-329.

[32] E. Burman, D. Elfverson, P. Hansbo, M. G. Larson, K. Larsson, "Shape optimization using the cut finite element method", *Comput. Methods Appl. Mech. Eng.* **328** (2018), p. 242-261.

[33] D. L. Chopp, "Computing Minimal Surfaces via Level Set Curvature Flow", *J. Comput. Phys.* **106** (1993), no. 1, p. 77-91.

[34] A. N. Christiansen, J. A. Bærentzen, M. Nobel-Jørgensen, N. Aage, O. Sigmund, "Combined shape and topology optimization of 3D structures", *Computers & Graphics* **46** (2015), p. 25-35.

[35] A. N. Christiansen, M. Nobel-Jørgensen, N. Aage, O. Sigmund, J. A. Bærentzen, "Topology optimization using an explicit interface representation", *Struct. Multidiscip. Optim.* **49** (2014), no. 3, p. 387-399.

[36] P. G. Ciarlet, *The finite element method for elliptic problems*, vol. 40, Society for Industrial and Applied Mathematics, 2002.

[37] M. Dambrine, D. Kateb, "On the ersatz material approximation in level-set methods", *ESAIM, Control Optim. Calc. Var.* **16** (2010), no. 3, p. 618-634.

[38] C. Dapogny, "The topological ligament in shape optimization: an approach based on thin tubular inhomogeneities asymptotics", *SMAI J. Comput. Math.* (2021), p. 185-266.

[39] C. Dapogny, C. Dobrzynski, P. Frey, "Three-dimensional adaptive domain remeshing, implicit domain meshing, and applications to free and moving boundary problems", *J. Comput. Phys.* **262** (2014), p. 358-378.

[40] C. Dapogny, P. Frey, "Computation of the signed distance function to a discrete contour on adapted triangulation", *Calcolo* **49** (2012), no. 3, p. 193-219.

[41] C. Dapogny, P. Frey, F. Omnès, Y. Privat, "Geometrical shape optimization in fluid mechanics using FreeFem++", *Struct. Multidiscip. Optim.* (2017), p. 1-28.

[42] C. Dapogny, N. Lebbe, E. Oudet, "Optimization of the shape of regions supporting boundary conditions", *Numer. Math.* **146** (2020), no. 1, p. 51-104.

[43] F. De Gournay, "Velocity extension for the level-set method and multiple eigenvalues in shape optimization", *SIAM J. Control Optim.* **45** (2006), no. 1, p. 343-367.

[44] L. Dedè, M. J. Borden, T. Hughes, Jr, "Isogeometric analysis for topology optimization with a phase field model", *Arch. Comput. Methods Eng.* **19** (2012), no. 3, p. 427-465.

[45] M. C. Delfour, J.-P. Zolésio, *Shapes and geometries: metrics, analysis, differential calculus, and optimization*, Society for Industrial and Applied Mathematics, 2011.

[46] J. Desai, G. Allaire, F. Jouve, "Topology optimization of structures undergoing brittle fracture", *J. Comput. Phys.* **458** (2022), article no. 111048 (35 pages).

[47] J. Desai, G. Allaire, F. Jouve, C. Mang, "Topology optimization in quasi-static plasticity with hardening using a level-set method", *Struct. Multidiscip. Optim.* **64** (2021), no. 5, p. 3163-3191.

[48] A. Doi, A. Koide, "An efficient method of triangulating equi-valued surfaces by using tetrahedral cells", *IEICE Trans. Inf. Syst.* **74** (1991), no. 1, p. 214-224.

[49] P. D. Dunning, H. A. Kim, "Introducing the sequential linear programming level-set method for topology optimization", *Struct. Multidiscip. Optim.* **51** (2015), no. 3, p. 631-643.

[50] P. Duysinx, L. V. Miegroet, T. Jacobs, C. Fleury, "Generalized shape optimization using X-FEM and level set methods", in *IUTAM symposium on topological design optimization of structures, machines and materials*, Springer, 2006, p. 23-32.

[51] B. Epstein, A. Jameson, S. Peigin, D. Roman, N. Harrison, J. Vassberg, "Comparative study of three-dimensional wing drag minimization by different optimization techniques", *J. Aircraft* **46** (2009), no. 2, p. 526-541.

[52] A. Ern, J.-L. Guermond, "Discontinuous Galerkin methods for Friedrichs' systems. I. General theory", *SIAM J. Numer. Anal.* **44** (2006), no. 2, p. 753-778.

[53] H. A. Eschenauer, N. Olhoff, "Topology optimization of continuum structures: a review", *Appl. Mech. Rev.* **54** (2001), no. 4, p. 331-390.

[54] L. C. Evans, R. F. Gariepy, *Measure theory and fine properties of functions*, CRC Press, 2015.

[55] F. Feppon, "Shape and topology optimization of multiphysics systems", PhD Thesis, Université Paris-Saclay (ComUE), 2019.

[56] F. Feppon, G. Allaire, F. Bordeu, J. Cortial, C. Dapogny, "Shape optimization of a coupled thermal fluid–structure problem in a level set mesh evolution framework", *SeMA J.* (2019), p. 1-46.

[57] F. Feppon, G. Allaire, C. Dapogny, "Null space gradient flows for constrained optimization with applications to shape optimization", *ESAIM, Control Optim. Calc. Var.* **26** (2020), article no. 90 (45 pages).

[58] F. Feppon, G. Allaire, C. Dapogny, P. Jolivet, "Topology optimization of thermal fluid–structure systems using body-fitted meshes and parallel computing", *J. Comput. Phys.* (2020), article no. 109574 (29 pages).

[59] ———, "Body-fitted topology optimization of 2D and 3D fluid-to-fluid heat exchangers", *Comput. Methods Appl. Mech. Eng.* **376** (2021), article no. 113638 (36 pages).

[60] P. Frey, P.-L. George, *Mesh generation: application to finite elements*, ISTE, 2007.

[61] S. Garreau, P. Guillaume, M. Masmoudi, "The topological asymptotic for PDE systems: the elasticity case", *SIAM J. Control Optim.* **39** (2001), no. 6, p. 1756-1778.

[62] Y. Giga, *Surface evolution equations*, Springer, 2006.

[63] J. S. Gray, J. T. Hwang, J. R. R. A. Martins, K. T. Moore, B. A. Naylor, "OpenMDAO: an open-source framework for multidisciplinary design, analysis, and optimization", *Struct. Multidiscip. Optim.* **59** (2019), no. 4, p. 1075-1104.

[64] J. Hadamard, *Mémoire sur le problème d'analyse relatif à l'équilibre des plaques élastiques encastrées*, Mém. Sav. étrang., vol. 33, Imprimerie nationale, 1908.

[65] O. Hassan, K.-A. Sørensen, K. Morgan, N. P. Weatherill, "A method for time accurate turbulent compressible fluid flow simulation with moving boundary components employing local remeshing", *Int. J. Numer. Methods Fluids* **53** (2007), no. 8, p. 1243-1266.

[66] F. Hecht, "New development in FreeFem++", *J. Numer. Math.* **20** (2012), no. 3-4, p. 251-266.

[67] A. Henrot, M. Pierre, *Shape Variation and Optimization. A geometrical analysis*, EMS Tracts in Mathematics, vol. 28, European Mathematical Society, 2018.

[68] A. Henrot, Y. Privat, "What is the optimal shape of a pipe?", *Arch. Ration. Mech. Anal.* **196** (2010), no. 1, p. 281-302.

[69] R. Hiptmair, A. Paganini, S. Sargheini, "Comparison of approximate shape gradients", *BIT Numer. Math.* **55** (2015), no. 2, p. 459-485.

[70] A. Jameson, "Aerodynamic design via control theory", in *Recent advances in computational fluid dynamics (Princeton, NJ, 1988)*, Lecture Notes in Engineering, vol. 43, Springer, 1989, p. 377-401.

[71] ———, "Computational algorithms for aerodynamic analysis and design", *Appl. Numer. Math.* **13** (1993), no. 5, p. 383-422.

[72] R. Kimmel, J. A. Sethian, "Computing geodesic paths on manifolds", *Proc. Natl. Acad. Sci. USA* **95** (1998), no. 15, p. 8431-8435.

[73] M. H. Kobayashi, R. A. Canfield, R. M. Kolonay, "On a cellular developmental method for layout optimization via the two-point topological derivative", *Struct. Multidiscip. Optim.* **64** (2021), no. 4, p. 2343-2360.

[74] R. V. Kohn, G. Strang, "Optimal design and relaxation of variational problems, I", *Commun. Pure Appl. Math.* **39** (1986), no. 1, p. 113-137.

[75] A. B. Lambe, J. R. R. A. Martins, "Matrix-free aerostructural optimization of aircraft wings", *Struct. Multidiscip. Optim.* **53** (2016), no. 3, p. 589-603.

[76] A. Laurain, "A level set-based structural optimization code using FEniCS", *Struct. Multidiscip. Optim.* **58** (2018), no. 3, p. 1311-1334.

[77] J. L. Lions, *Optimal control of systems governed by partial differential equations*, Grundlehren der Mathematischen Wissenschaften, vol. 170, Springer, 1971.

[78] W. E. Lorensen, H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm", *ACM SIGGRAPH Comput. Graph.* **21** (1987), no. 4, p. 163-169.

[79] S. Mauch, "A fast algorithm for computing the closest point and distance transform", https://www.researchgate.net/publication/2393786_A_Fast_Algorithm_for_Computing_the_Closest_Point_and_Distance_Transform, 2000.

[80] B. Mohammadi, O. Pironneau, *Applied shape optimization for fluids*, Numerical Mathematics and Scientific Computation, Oxford University Press, 2010.

[81] F. Murat, J. Simon, "Sur le contrôle par un domaine géométrique", pré-publication du Laboratoire d'Analyse Numérique,(76015), 1976.

[82] S. Nazarov, J. Sokołowski, "The topological derivative of the Dirichlet integral due to formation of a thin ligament", *Sib. Math. J.* **45** (2004), no. 2, p. 341-355.

[83] J. Nocedal, S. J. Wright, *Numerical optimization 2nd*, Springer, 2006.

[84] A. A. Novotny, J. Sokołowski, *Topological derivatives in shape optimization*, Springer, 2012.

[85] S. Osher, R. Fedkiw, *Level set methods and dynamic implicit surfaces*, vol. 153, Springer, 2006.

[86] S. Osher, F. Santosa, "Level set methods for optimization problems involving geometry and constraints: I. Frequencies of a two-density inhomogeneous drum", *J. Comput. Phys.* **171** (2001), no. 1, p. 272-288.

[87] S. Osher, J. A. Sethian, "Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations", *J. Comput. Phys.* **79** (1988), no. 1, p. 12-49.

[88] O. Pironneau, "On optimum profiles in Stokes flow", *J. Fluid Mech.* **59** (1973), no. 1, p. 117-128.

[89] ———, *Optimal shape design for elliptic systems*, Springer, 1982.

[90] ———, *Finite element methods for fluids*, Wiley Publishing, 1989.

[91] R.-E. Plessix, "A review of the adjoint-state method for computing the gradient of a functional with geophysical applications", *Geophys. J. Int.* **167** (2006), no. 2, p. 495-503.

[92] J. A. Sethian, "Fast marching methods", *SIAM Rev.* **41** (1999), no. 2, p. 199-235.

[93] ———, *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*, vol. 3, Cambridge University Press, 1999.

[94] J. A. Sethian, A. Wiegmann, "Structural boundary design via level set and immersed interface methods", *J. Comput. Phys.* **163** (2000), no. 2, p. 489-528.

[95] O. Sigmund, "A 99 line topology optimization code written in Matlab", *Struct. Multidiscip. Optim.* **21** (2001), no. 2, p. 120-127.

[96] J. Sokołowski, J.-P. Zolésio, *Introduction to shape optimization*, Springer, 1992.

[97] J. Strain, "Semi-Lagrangian methods for level set equations", *J. Comput. Phys.* **151** (1999), no. 2, p. 498-533.

[98] K. Svanberg, "The method of moving asymptotesÑa new method for structural optimization", *Int. J. Numer. Methods Eng.* **24** (1987), no. 2, p. 359-373.

[99] A. Takezawa, S. Nishiwaki, M. Kitamura, "Shape and topology optimization based on the phase field method and sensitivity analysis", *J. Comput. Phys.* **229** (2010), no. 7, p. 2697-2718.

[100] Y.-H. R. Tsai, "Rapid and accurate computation of the distance function using grids", *J. Comput. Phys.* **178** (2002), no. 1, p. 175-195.

[101] C. Wang, Z. Zhao, M. Zhou, O. Sigmund, X. S. Zhang, "A comprehensive review of educational articles on structural and multidisciplinary optimization", *Struct. Multidiscip. Optim.* **64** (2021), no. 5, p. 2827-2880.

[102] M. Y. Wang, X. Wang, D. Guo, "A level set method for structural topology optimization", *Comput. Methods Appl. Mech. Eng.* **192** (2003), no. 1-2, p. 227-246.

[103] Q. Xia, M. Y. Wang, "Topology optimization of thermoelastic structures using level set method", *Comput. Mech.* **42** (2008), no. 6, p. 837-857.

[104] H. Zhao, "A fast sweeping method for eikonal equations", *Math. Comput.* **74** (2005), no. 250, p. 603-627.