Geometrical shape optimization in fluid mechanics using FreeFem++

Charles Dapogny, Pascal Frey, Florian Omnès, Yannick Privat, Florian Omnès

GEOMETRICAL SHAPE OPTIMIZATION IN FLUID MECHANICS USING FREEFEM++

C. DAPOGNY[1], P. FREY[2,3], F. OMNÈS[2] AND Y. PRIVAT[2]

[1] *Laboratoire Jean Kuntzmann, CNRS, Université Grenoble-Alpes, BP 53, 38041 Grenoble Cedex 9, France,*
[2] *Sorbonne Universités, UPMC Univ Paris 06, CNRS UMR 7598, Laboratoire Jacques-Louis Lions, F-75005, Paris, France,*
[3] *Sorbonne Universités, UPMC Univ Paris 06, Institut des Sciences du Calcul et des Données (ISCD), F-75005, Paris, France,*

Abstract. In this article, we present simple and robust numerical methods for two-dimensional geometrical shape optimization problems, in the context of viscous flows driven by the stationary Navier-Stokes equations at low Reynolds number. The salient features of our algorithm are exposed with an educational purpose; in particular, the numerical resolution of the nonlinear stationary Navier-Stokes system, the Hadamard boundary variation method for calculating the sensitivity of the minimized function of the domain, and the mesh update strategy are carefully described. Several pedagogical examples are discussed. The corresponding programs are written in the `FreeFem++` environment; they are freely available and can easily be elaborated upon to deal with different, or more complex physical situations.

Contents

1. Introduction

The first industrial developments of shape optimization in contexts involving fluid mechanics arose in the fields of aeronautic and aerospatial engineering. These developments were motivated by the tremendous production and running costs of aircraft: even small improvements on the performance of a design entail very large savings. Perhaps the most famous issue in this field is the design optimization of an airfoil, which dates back to at least 1964 [17]; see also [41, 42], and [60] where optimal profiles for minimum drag problems are calculated thanks to shape sensitivity analyses. We generally refer to [36], Chap. 1 for a historical perspective about the emergence of optimal design techniques in the context of fluid mechanics. Since the aforementioned pioneering works, applications of shape optimization in fluid mechanics have raised a great interest in various areas such as the automotive industry - see [20] about the numerical optimization of a cooling fan - or in computational biology: for instance, in [2, 3], the design optimization of an artery graft for preventing the formation of a stenosis is investigated from a numerical point of view.

Let us briefly outline the main features of the most popular shape optimization strategies in the literature, without looking for exhaustivity. For more in-depth discussions in the context of fluid mechanics, we refer to [36, 50], or to the review article [49]. Any shape optimization method relies on a parametrization of shapes, that is, on the definition of a set of design variables. Depending on the situation, these design variables may be physical parameters of shapes (the length of some pipe, or thickness of a region), control points of a CAD description, or the vertices of a meshed representation. In all cases, the sensitivity (i.e. derivative) of the objective and constraint functionals of the optimization problem with respect to the design variables - which is a key ingredient in most numerical optimization algorithms - can be evaluated either by approximate methods (for instance by finite differences featuring small perturbations of the parameters), or analytically, by relying on adjoint techniques from optimal control theory [43, 47, 61]. In this last class of methods, which is by now quite popular, these sensitivities may be calculated at the discrete level (i.e. the derivative of the finite-dimensional functional resulting from the discretization of the shape and the physical equations is considered), which requires a perfect knowledge of the discretization and numerical methods involved in the resolution of the flow equations (but allows for the use of automatic differentiation methods). The opposite view consists in calculating first the derivative of the optimized criterion at the continuous level, then in discretizing it when it comes to the numerical implementation. This 'continuous' approach relies on advanced mathematical tools, but the stages of the optimization process associated to the calculation of the derivatives and the numerical resolution of the mechanical problem are more independent.

In any event, a great numerical challenge faced by all these methods is that of updating the design of the shape from one iteration of the process to the next, while avoiding that the numerical representation becomes invalid. For instance, if the shape is consistently described by means of a computational mesh, the latter is likely to develop self-intersecting elements in the course of the optimization process, causing it to abort prematurely; see the discussion in Section 3.4. Recently, several strategies have been devised to circumvent this difficulty, and more generally to allow for more freedom in terms of the variety of designs that can be represented, to the point that they make it possible to account for changes in their topology.

In this direction, quite popular density-based methods in structural mechanics - and notably the famous SIMP method (see [10] and references therein) - have been introduced in the context of fluid mechanics in [12]; see also [57] and [1], where a large-scale example is discussed. These relaxation methods rely on an extension of the set of admissible designs: 'black-and-white' shapes Ω contained in a fixed computational domain D, or equivalently their characteristic function $\chi : D \to \{0, 1\}$, taking values 1 inside Ω, and 0 in the 'void' region $D \setminus \overline{\Omega}$, are replaced with density functions $\rho : D \to [0, 1]$, which may assume intermediate, 'grayscale' values in $(0, 1)$. The flow equations have then to be given an appropriate meaning to account for the presence of 'void' and 'grayscale' regions. This is typically achieved by adding a ρ-dependent damping term (or Brinkman's law) to the flow equations [12], a heuristic inspired from the theory of porous media

whereby the void is filled with a fluid with very low permeability, thus mimicking no slip boundary conditions at the interface between the fluid and void domains (see [35] and [30], then [33] for a generalization to the case of Navier-Stokes flows). Let us eventually mention the contribution [46] where topology optimization problems are tackled in the context of the unsteady Navier-Stokes equations, and reveal the limitations of this penalization approach as far as the accuracy of the resolution of the flow equations is concerned.

Another class of shape and topology optimization strategies relies on the level set method, pioneered in [56], then introduced in structural optimization in [62, 5, 66]. Such methods describe a shape Ω via the use of a scalar function ϕ defined on the whole computational domain D: the negative subdomain of ϕ coincides with Ω, while its positive subdomain accounts for void (or, in practice, another fluid with low permeability, according to the aforementioned 'Brinkman' penalization approximation). In the two-dimensional work [28], the level set method is used to deal with Navier-Stokes flows, in a variational framework which alleviates the need for the redistancing stage inherent to many level set based algorithms; this idea is continued in [67] in the three-dimensional setting. See also [14] for another use of the Level Set method in the context of Navier-Stokes flows. Recent contributions have proposed alternative efficient level set methods where the flow equations are solved by the Lattice Boltzmann method [57, 45] or the Extended Finite Element method [44], alleviating the need for the 'Brinkman' penalization method. On a different note, in [18], the Level Set method is used to combine the information supplied by shape and topological derivatives, in the context of Stokes flows, in two and three space dimensions.

Eventually, let us also mention phase-field methods, which share a lot of features with level set methods, except for the fact that they bring into play shapes, or phases, with 'thickened boundaries' [32].

Following the lead of [63] and [6], which take place in the context of structural mechanics, this article is a pedagogical introduction to several basic shape optimization techniques in fluid mechanics. The discussion is didactic: we deliberately keep technicalities to a minimum, and provide adequate references when needed. We present a simple numerical framework, yet robust enough to deal with physically relevant situations, which is dedicated to solving shape optimization problems in the context of fluid mechanics. The proposed examples can be easily reproduced and elaborated upon to deal with more advanced models.

In the setting of the stationary Navier-Stokes equations at low Reynolds number, we optimize shapes in terms e.g. of the dissipated viscous energy, under volume or perimeter constraints. To this end, we rely on an augmented Lagrangian algorithm based on the first order information supplied by shape derivatives, in the sense of the Hadamard boundary variation method. From the numerical point of view, shapes are represented by a computational mesh, on which the flow equations are solved owing to the Finite Element method. The update of the shape between each iteration of the optimization process is achieved by moving the vertices of this mesh according to the calculated descent direction.

The numerical developments proposed in this article rely on the `FreeFem++` [37] software, a free environment allowing to solve a wide variety of Partial Differential Equations (PDE for short) using the Finite Element method within a few command lines.

A particular attention has been paid to the development of a user-friendly source code, which is available online at

`https://github.com/flomnes/optiflow`

with the hope that it serve as a useful basis for further investigations.

The remainder of this article is organized as follows. In Section 2, we introduce the model physical problem at stake, as well as the shape optimization problem considered in this context. In passing, we recall in an elementary way some basic facts about shape derivatives. In Section 3, we describe in more details the main ingredients of the proposed numerical method: after a short motivating outline in Section 3.1, we discuss the salient features of our shape optimization algorithm in Sections 3.2, 3.3, 3.4, 3.5 and 3.6; a sketch of this algorithm is then provided in Section 3.7. Section 4 is then a short guide of our practical implementation; it is expected that, together with the thorough comments left throughout our code, this will allow the user to define and solve his own shape optimization test cases in a user-friendly way. In Section 5, we introduce and comment five test cases which are dealt with by our algorithm. Finally, Section 6 concludes by evoking limitations of our approach as well as perspectives for possible improvements and extensions.

2. Shape optimization for flows governed by the Navier-Stokes equations

In this section, we present the model physical situation and the shape optimization problem at stake, together with the necessary theoretical background. Notice that, while the concrete applications discussed in this article arise in two space dimensions (see Section 5), most of the presented techniques are available in the general, d-dimensional setting. For this reason, the discussion takes place in d dimensions inasmuch as it is possible without giving up simplicity and clarity.

2.1. The Navier-Stokes equations.

In our applications, shapes are smooth bounded domains $\Omega \subset \mathbb{R}^d$ ($d = 2, 3$ in practice), occupied by a homogeneous Newtonian fluid with kinematic viscosity $\nu > 0$. The boundary $\partial\Omega$ is made of three disjoint regions: $\partial\Omega = \Gamma_{\text{in}} \cup \Gamma_{\text{out}} \cup \Gamma$, where

- Γ_{in} is the 'inlet', on which a known velocity profile $\mathbf{u}_{\text{in}}$ is imposed;
- Γ_{out} is the 'outlet', which is free from of surface forces;
- Γ is the 'free' boundary; no slip boundary conditions are imposed on Γ, accounting for the fact that the fluid particles are stuck on it;

see Figure 1 for an illustration. In the applications ahead, Γ is the only region of $\partial\Omega$ which is subject to optimization, i.e. Γ_{in} and Γ_{out} are fixed. From the physical point of view, Ω may represent a mammal's lung, a duct in a ventilation system or a water radiator, etc.

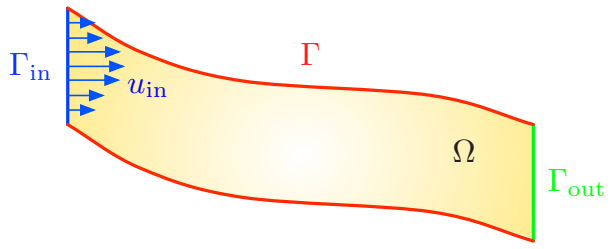


FIGURE 1. *Illustration of the model setting introduced in Section 2.*

The equilibrium behavior of the fluid inside Ω is classically described in terms of its (vector) velocity field $\mathbf{u} = (u_1, ..., u_d) : \Omega \to \mathbb{R}^d$ and (scalar) pressure $p : \Omega \to \mathbb{R}$, which solve the stationary *incompressible Navier-Stokes equations*:

$$\text{(2.1)} \qquad \begin{cases} -\nu\Delta\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p = 0 & \text{in } \Omega, \\ \operatorname{div}(\mathbf{u}) = 0 & \text{in } \Omega, \\ \mathbf{u} = \mathbf{u}_{\text{in}} & \text{on } \Gamma_{\text{in}}, \\ \mathbf{u} = 0 & \text{on } \Gamma, \\ \sigma(\mathbf{u}, p)\mathbf{n} = 0 & \text{on } \Gamma_{\text{out}}. \end{cases}$$

In the above system, the stress tensor $\sigma(\mathbf{u}, p)$ is defined by

$$\sigma(\mathbf{u}, p) = 2\nu e(\mathbf{u}) - p\mathrm{I}, \text{ where } e(\mathbf{u}) = \frac{1}{2}(\nabla\mathbf{u}^T + \nabla\mathbf{u}).$$

From the physical point of view, the first equation in (2.1) is the law of balance of momentum between viscous forces $-\nu\Delta\mathbf{u}$, acceleration forces $(\mathbf{u}\cdot\nabla)\mathbf{u}$ and pressure forces ∇p. The second equation $\operatorname{div}(\mathbf{u}) = 0$ accounts for the incompressibility of the fluid. Because of this incompressibility feature, a simple calculation allows to rewrite the law of balance of momentum under the equivalent form:

$$-\operatorname{div}(\sigma(\mathbf{u}, p)) + (\mathbf{u}\cdot\nabla)\mathbf{u} = 0.$$

For further reference, let us recall that this nonlinear system is often considered from the variational viewpoint, in particular when it comes to its numerical resolution using `FreeFem++`; the pair $(\mathbf{u}, p)$ satisfies:

$$\text{For all } (\mathbf{v}, q) \text{ s.t. } \mathbf{v} = 0 \text{ on } \Gamma_{\text{in}},$$
$$\left\{ \begin{array}{rcl} a(\mathbf{u}, \mathbf{v}) + c(\mathbf{u}, \mathbf{u}, \mathbf{v}) + b(\mathbf{v}, p) & = & 0 \\ b(\mathbf{u}, q) & = & 0 \end{array} \right. \tag{2.2}$$

where we have defined

$$a(\mathbf{u}, \mathbf{v}) = 2\nu \int_\Omega e(\mathbf{u}) : e(\mathbf{v})\, dx,$$
$$b(\mathbf{u}, p) = -\int_\Omega p\, \mathrm{div}(\mathbf{u})\, dx,$$
$$c(\mathbf{u}, \mathbf{v}, \mathbf{w}) = \int_\Omega (\mathbf{u} \cdot \nabla)\mathbf{v} \cdot \mathbf{w}\, dx,$$

and the notation $A : B$ stands for the usual Frobenius inner product of two $d \times d$ matrices A, B, that is $A : B = \sum_{i,j=1}^d A_{ij} B_{ij}$. In the following, we also denote by $||A|| = (A : A)^{1/2}$ the associated Frobenius norm.

In the dimensionless version (2.1), (2.2) of the Navier-Stokes equations, the Reynolds number Re is proportional to $1/\nu$: it is an indicator of the type of regime of the flow [51]. At low Reynolds number, viscous effects are prevailing and the flow is laminar; in particular, its velocity stays relatively low. On the contrary, at moderate to high Reynolds number, convective forces become dominant and the flow is turbulent. The theoretical and numerical study of (2.1) is notoriously much harder in the latter situation, and still leaves room for many open questions. In the present, introductory work, we limit ourselves to the low Reynolds number regime (say, $Re \approx 200$).

Remark 2.1. *Let us say a few words about the functional setting and well-posedness of the stationary Navier-Stokes system (2.1). When the viscosity ν is large enough, i.e. the Reynolds number Re is low, (2.1) is well-posed. It has a unique weak solution $(\mathbf{u}, p) \in H^1(\Omega)^d \times L^2_0(\Omega)$, where $L^2_0(\Omega) := \left\{ p \in L^2(\Omega),\ \int_\Omega p\, dx = 0 \right\}$, in the sense that the variational problem (2.2) is fulfilled; see [65], Chapter II about these matters. In the following, we systematically assume ν to be large enough so that (2.1) is well-posed.*

2.2. **Statement of the shape optimization problem.**

In the context of Section 2.1, the shape optimization problem of interest reads

$$\min_{\Omega \in \mathcal{O}_{\text{ad}}} J(\Omega) \text{ s.t. } G(\Omega) = 0. \tag{2.3}$$

Here, the objective criterion $J(\Omega)$ may stand for

- The energy $E(\Omega)$ dissipated by the fluid owing to the work of viscous forces, i.e.

$$E(\Omega) = \int_\Omega \sigma(\mathbf{u}, p) : e(\mathbf{u})\, dx = 2\nu \int_\Omega \|e(\mathbf{u})\|^2\, dx, \tag{2.4}$$

- A least-square discrepancy

$$D(\Omega) = \frac{1}{2} \int_{\Gamma_{\text{out}}} |\mathbf{u} - \mathbf{u}_{\text{ref}}|^2\, ds \tag{2.5}$$

  between the velocity $\mathbf{u}$ of the fluid, solution to (2.1), and a given reference profile $\mathbf{u}_{\text{ref}}$. Such criteria are often involved in shape optimization-based methods for the detection or the reconstruction of an obstacle immersed in a fluid from the data of boundary measurements [8, 48].

As we have mentioned in Section 2.1, all the considered domains enclose the inlet Γ_{in} and the outlet Γ_{out} as (fixed) subsets of their boundaries, so that the free boundary Γ is the only region of $\partial\Omega$ subject to optimization. Accordingly, the set $\mathcal{O}_{ad}$ of admissible domains featured in (2.3) reads:

$$\mathcal{O}_{ad} = \{\Omega \subset \mathbb{R}^d,\ \text{open, smooth and bounded, such that}$$
$$\Gamma_{\text{in}} \cup \Gamma_{\text{out}} \subset \partial\Omega\} \tag{2.6}$$

Last but not least, as far as the constraint functional $G(\Omega)$ is concerned, we shall restrict ourselves to equality constraints on the volume $\mathrm{Vol}(\Omega) = \int_\Omega dx$ or the perimeter $\mathrm{Per}(\Omega) = \int_{\partial\Omega} ds$ of shapes, namely:

$$G(\Omega) = \mathrm{Vol}(\Omega) - V_T, \text{ or } G(\Omega) = \mathrm{Per}(\Omega) - P_T$$

for some given volume or perimeter target values V_T and P_T.

Remark 2.2. *The existence of global minimizers of problems of the form (2.3) is a long-standing question in shape optimization theory, not only in the context of fluid mechanics, but already in simpler situations, bringing into play the conductivity equation, or the linearized elasticity system. Let us simply mention that, in order to guarantee the existence of optimal shapes, two classical remedies consist in either restricting the set of admissible shapes (for instance by adding constraints on the perimeter, or the regularity of shapes), or on the contrary in enlarging this set, so that it includes 'density functions', and not only 'black and white' shapes. See for instance [15, 38, 64] about these issues, or [39, 40, 11] in the context of fluid mechanics.*

Often, in numerical practice, one is rather interested in searching for local minimizers of (2.3), which are close to an initial guess inspired by physical intuition. These are the 'optimal' shapes which are typically delivered by local optimization methods, such as the steepest-descent algorithms used in the present article.

2.3. Shape sensitivity analysis using Hadamard's boundary variation method.

Most optimization algorithms - such as steepest-descent methods - rely on the knowledge of the derivatives of the objective and constraint functionals. As we have already hinted at in the introduction, two different paradigms exist in the context of PDE constrained optimization problems of the form (2.3). In a nutshell, in 'discretize-then-optimize' approaches, the optimized domain is first discretized into a set of design variables (for instance, the vertices of a mesh); the PDE system (2.1) becomes finite-dimensional (it is e.g. discretized using a Finite Element method), and its coefficients depend on the design variables; accordingly, the objective and constraint functionals $J(\Omega)$ and $G(\Omega)$ are functions of the design variables, and the derivatives of these discrete functionals are calculated. On the contrary, 'optimize-then-discretize' approaches advocate to calculate the derivatives of $J(\Omega)$ and $G(\Omega)$ at the continuous level; the resulting theoretical formulae are then discretized by relying on a discretization of the domain and of the PDE system (2.1).

The approach described in this article belongs to the second category, and therefore requires to compute derivatives with respect to the domain. Several ways exist to define a notion of shape derivative, and we rely on Hadamard's boundary variation method, a brief sketch of which is now provided; see for instance to [38], Chap. 5, or [4, 53] for in-depth expositions. See also [55] for an overview of the rival notion of *topological derivative*, and [7] for the calculation of topological derivatives in the context of fluid mechanics.

In the framework of Hadamard's method, the sensitivity of a function of the domain is assessed with respect to small perturbations of its boundary: variations of a given shape Ω are considered in the form

$$\Omega_{\boldsymbol{\theta}} = (\mathrm{Id} + \boldsymbol{\theta})(\Omega), \tag{2.7}$$

where $\boldsymbol{\theta} : \mathbb{R}^d \to \mathbb{R}^d$ is a 'small' vector field, and Id is the identity mapping from $\mathbb{R}^d$ into itself; see Figure 2 for an illustration.

Since admissible shapes $\Omega \in \mathcal{O}_{ad}$ are smooth and only Γ is subject to optimization, it is natural that $\boldsymbol{\theta}$ belong to the set Θ_{ad} of admissible perturbations defined by:

$$\Theta_{ad} = \left\{ \boldsymbol{\theta} : \mathbb{R}^d \to \mathbb{R}^d \text{ smooth}, \ \boldsymbol{\theta} = 0 \text{ on } \Gamma_{\mathrm{in}} \cup \Gamma_{\mathrm{out}} \right\};$$

so that variations (2.7) of admissible shapes stay admissible.

Definition 2.1. *A function of the domain $F(\Omega)$ is shape differentiable at $\Omega \in \mathcal{O}_{ad}$ if the underlying mapping $\boldsymbol{\theta} \mapsto F(\Omega_{\boldsymbol{\theta}})$, from Θ_{ad} into $\mathbb{R}$, is differentiable at $\boldsymbol{\theta} = 0$ (in the sense of Fréchet). The corresponding derivative is denoted by $\boldsymbol{\theta} \mapsto F'(\Omega)(\boldsymbol{\theta})$, and the following Taylor expansion holds:*

$$F(\Omega_{\boldsymbol{\theta}}) = F(\Omega) + F'(\Omega)(\boldsymbol{\theta}) + o(\boldsymbol{\theta}) \tag{2.8}$$

where $o(\boldsymbol{\theta}) \to 0$ as $\boldsymbol{\theta} \to 0$.

When it comes to shape derivatives, the first result of interest deals with the volume and perimeter functionals; see [4, 38] for a proof.

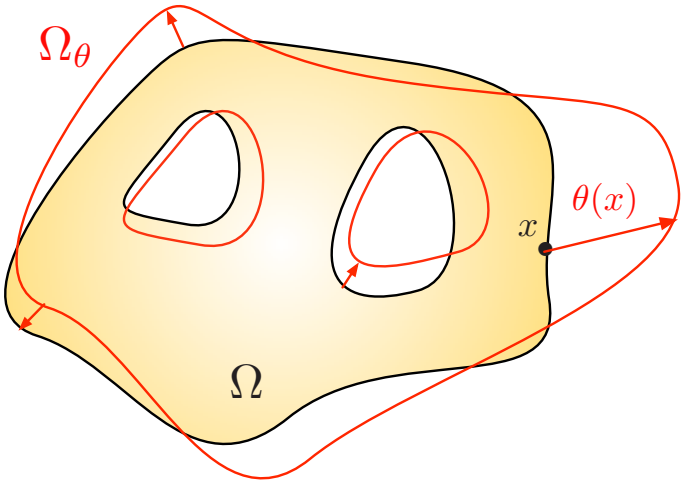


FIGURE 2. *Example of a variation $\Omega_{\boldsymbol{\theta}}$ of a shape Ω.*

Theorem 2.1. *Let Ω be a smooth shape. Then,*

(i) The volume $\mathrm{Vol}(\Omega)$ *is shape differentiable and its derivative reads:*

$$\forall \boldsymbol{\theta} \in \Theta_{ad},\ \mathrm{Vol}'(\Omega)(\boldsymbol{\theta}) = \int_\Gamma \boldsymbol{\theta} \cdot \mathbf{n}\, ds.$$

(ii) The perimeter $\mathrm{Per}(\Omega)$ *is shape differentiable and its derivative reads:*

$$\forall \boldsymbol{\theta} \in \Theta_{ad},\ \mathrm{Per}'(\Omega)(\boldsymbol{\theta}) = \int_\Gamma \kappa\, \boldsymbol{\theta} \cdot \mathbf{n}\, ds,$$

where $\kappa : \partial\Omega \to \mathbb{R}$ is the mean curvature of $\partial\Omega$.

Calculating shape derivatives of functions of the form (2.4) or (2.5) is a little harder, since they bring into play the solution of a partial differential equation posed on Ω (in the present case, the Navier-Stokes system (2.1)). This can however be managed by using quite classical adjoint techniques from optimal control theory. Again, we refer to [4] for a comprehensive introduction to such techniques in the context of shape optimization, and to Appendix A for a sketch of proof.

Theorem 2.2. *Let $\Omega \in \mathcal{O}_{ad}$; then,*

(i) The energy dissipation $E(\Omega)$ given by (2.4) is shape differentiable and its derivative reads:

$$\forall \boldsymbol{\theta} \in \Theta_{ad},\ E'(\Omega)(\boldsymbol{\theta}) = \int_\Gamma (-2\nu e(\mathbf{u}) : e(\mathbf{u}) + 2\nu e(\mathbf{u}) : e(\mathbf{v}_e))\, \boldsymbol{\theta} \cdot \mathbf{n}\, ds, \tag{2.9}$$

where $(\mathbf{v}_e, q_e)$ is an adjoint state, defined as the solution of the linear PDE

$$\begin{cases} -\nu\Delta\mathbf{v}_e + (\nabla\mathbf{u})^T\mathbf{v}_e - (\nabla\mathbf{v}_e)\mathbf{u} + \nabla q_e \\ \qquad = -2\nu\Delta\mathbf{u} & \text{in } \Omega, \\ \mathrm{div}(\mathbf{v}_e) = 0 & \text{in } \Omega, \\ \mathbf{v}_e = 0 & \text{on } \Gamma \cup \Gamma_{\mathrm{in}}, \\ \sigma(\mathbf{v}_e, q_e)\mathbf{n} + (\mathbf{u}\cdot\mathbf{n})\mathbf{v}_e = 4\nu e(\mathbf{u})\mathbf{n} & \text{on } \Gamma_{\mathrm{out}}. \end{cases} \tag{2.10}$$

(ii) The least-square functional $D(\Omega)$ defined by (2.5) is shape differentiable and its derivative reads:

$$\forall \boldsymbol{\theta} \in \Theta_{ad},\ D'(\Omega)(\theta) = \int_\Gamma 2\nu e(\mathbf{u}) : e(\mathbf{v}_d)\, \boldsymbol{\theta} \cdot \mathbf{n}\, ds, \tag{2.11}$$

where the adjoint system for $(\mathbf{v}_d, q_d)$ *reads*

$$\begin{cases} -\nu\Delta\mathbf{v}_d + (\nabla\mathbf{u})^T\mathbf{v}_d - (\nabla\mathbf{v}_d)\mathbf{u} + \nabla q_d = 0 & \text{in } \Omega, \\ \operatorname{div}(\mathbf{v}_d) = 0 & \text{in } \Omega, \\ \mathbf{v}_d = 0 & \text{on } \Gamma \cup \Gamma_{in}, \\ \sigma(\mathbf{v}_d, q_d)\mathbf{n} + (\mathbf{u}\cdot\mathbf{n})\mathbf{v}_d = \mathbf{u} - \mathbf{u}_{ref} & \text{on } \Gamma_{out}. \end{cases} \tag{2.12}$$

Remark 2.3. *(1) As is customary in shape optimization - and in optimal control in general -, the adjoint systems (2.10) and (2.11) are linear, while the original Navier-Stokes system (2.1) is non linear.*

(2) From the mathematical point of view, the adjoint systems (2.10) and (2.12) are well-posed in suitable functional spaces when the parameter ν *is assumed to be large enough (see e.g. [40]).*

Like those of the functions $\mathrm{Vol}(\Omega)$, $\mathrm{Per}(\Omega)$, $E(\Omega)$ and $D(\Omega)$ involved in Theorems 2.1 and 2.2, the shape derivative of a fairly general class of shape functionals $F(\Omega)$ has the generic form:

$$F'(\Omega)(\boldsymbol{\theta}) = \int_\Gamma \phi\, \boldsymbol{\theta}\cdot\mathbf{n}\, ds =: (\phi, \boldsymbol{\theta}\cdot\mathbf{n})_{L^2(\Gamma)}, \tag{2.13}$$

where the scalar function $\phi : \Gamma \to \mathbb{R}$ is the 'shape gradient' of F with respect to the $L^2(\Gamma)$ inner product. This statement is referred to as the *Structure theorem* for shape derivatives; see [38], §5.9. In particular, $F'(\Omega)(\boldsymbol{\theta})$ depends only on the values of the normal component $\boldsymbol{\theta}\cdot\mathbf{n}$ on the free boundary Γ; this reflects the intuitive fact that tangential deformations of Ω leave the values of $F(\Omega)$ unchanged at first order.

For further reference, the structure (2.13) makes it easy to infer descent directions for $F(\Omega)$. Indeed, if $\boldsymbol{\theta}$ coincides with $-\phi\boldsymbol{n}$ on Γ, it readily follows from (2.8) that, for $t > 0$ small enough:

$$F(\Omega_{t\boldsymbol{\theta}}) = F(\Omega) - t\int_\Gamma \phi^2\, ds + o(t) < F(\Omega). \tag{2.14}$$

3. Numerical methods

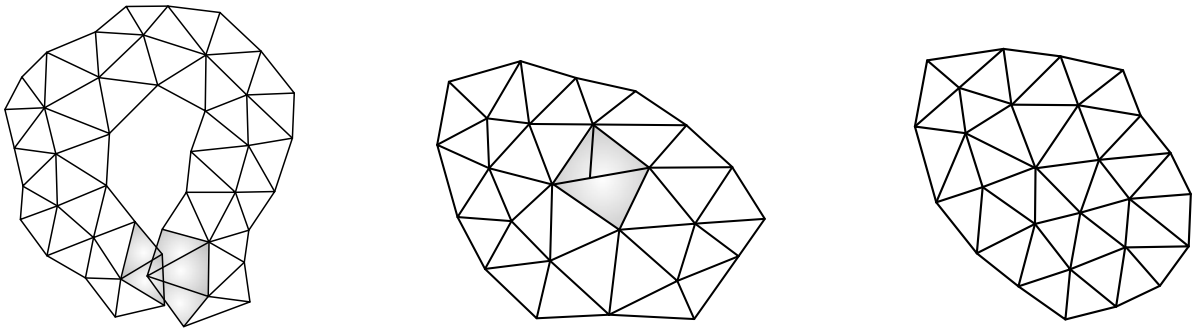
In this section, we describe in more detail the numerical methods involved in the resolution of the shape optimization problem (2.3).

3.1. Description of the numerical setting and outline of the algorithm.

Each shape Ω is represented by means of a simplicial mesh $\mathcal{T}$, composed of K (closed) simplices $T_1, ..., T_K$ (i.e. triangles in $2d$, tetrahedra in $3d$) , and I vertices $\mathbf{x}_1, ..., \mathbf{x}_I$. The mesh $\mathcal{T}$ is computational in the sense of Finite Elements, that is:

- The T_k form a cover of $\overline{\Omega}$, i.e. $\overline{\Omega} = \bigcup_{k=1}^K T_k$,
- The T_k do not overlap, i.e. the intersection between the interiors of T_k and $T_{k'}$ is empty whenever $k \neq k'$,
- The mesh $\mathcal{T}$ is *conforming*; for instance, in two dimensions, the intersection between any two triangles T_k and $T_{k'}$, $k \neq k'$, is either empty, or it is a vertex, or an edge of $\mathcal{T}$.

See Figure 3 for illustrations of these notions.



(a) (b) (c)

FIGURE 3. *Examples of (a) a mesh with overlapping triangles (in grey); (b) a mesh with non overlapping, yet non conforming triangles (in grey); (c) a computational mesh.*

In the following, we shall often consider sequences of shapes Ω^n and meshes $\mathcal{T}^n$, and we denote with a ^{n} superscript all the entities (vertices $\mathbf{x}_i^n$, simplices T_k^n, numbers of vertices I^n and simplices K^n) of $\mathcal{T}^n$.

So as to emphasize the needed numerical methods in the resolution of (2.3), we now give a deliberately hazy sketch of the main stages; a practical version is given in Section 3.7.

- **Initialization:** The initial domain Ω^0 is equipped with a mesh $\mathcal{T}^0$.
- **For $n = 0$, ... until convergence:**
  (1) Compute the solution $(\mathbf{u}, p)$ of the Navier-Stokes equation (2.1), and the adjoint state $(\mathbf{v}, q)$, solution of (2.10) or (2.12) on Ω^n, using the mesh $\mathcal{T}^n$.
  (2) Compute the shape derivatives of $J(\Omega)$ and $G(\Omega)$ (see Theorems 2.1 and 2.2) and infer a descent direction $\boldsymbol{\theta}^n$ for the optimization problem (2.3).
  (3) Choose a sufficiently small time step τ^n and update the shape Ω^n into the new shape $\Omega^{n+1} := (\mathrm{Id} + \tau^n \boldsymbol{\theta}^n)(\Omega^n)$; a mesh $\mathcal{T}^{n+1}$ of Ω^{n+1} is obtained.

This program raises a number of issues:

- The numerical resolution of the systems (2.1), (2.10) and (2.12) is by no means trivial; Section 3.2 below is devoted to this issue.
- The calculation of a descent direction for $J(\Omega)$ which allows to satisfy the constraint $G(\Omega)$ demands the use of an adapted optimization algorithm, which is described in Section 3.3.
- The deformation of the mesh $\mathcal{T}^n$ of Ω^n into a computational mesh $\mathcal{T}^{n+1}$ of Ω^{n+1} is a difficult task. We describe in Section 3.4 the stakes of mesh deformation, and in Section 3.5 a strategy for calculating a nice shape gradient which eases this purpose.

3.2. **Numerical resolution of the Navier-Stokes equations.**

The numerical resolution of the Navier-Stokes system (2.1) with the Finite Element method is faced with two relatively independent difficulties. The first one is related to the treatment of the nonlinear convective term $(\mathbf{u} \cdot \nabla)\mathbf{u}$; the second one is quite common in the resolution of saddle-point problems: it is about the choice of adequate Finite Element spaces for the discretization of the velocity $\mathbf{u}$ and pressure p. Notice that the adjoint systems (2.10) and (2.12) are linear, so their resolution is not concerned by the first issue, but it is by the second one. We only discuss the case of the nonlinear Navier-Stokes system (2.1) in this section, which is in all regards more difficult.

3.2.1. *Dealing with the nonlinear convective term using Newton's method.* We rely on a fairly standard Newton method for nonlinear problems. Writing (2.1) in the abstract form

$$A(\mathbf{u}, p) = 0, \tag{3.1}$$

Newton's method achieves the solution as the limit of the sequence $(\mathbf{u}^k, p^k)$, where each update $(\delta\mathbf{u}^k, \delta p^k)$ between the steps k and $(k+1)$ is calculated as the solution to the linearized version of (3.1) around $(\mathbf{u}^k, p^k)$:

$$d_{(\mathbf{u}^k,p^k)}A(\delta\mathbf{u}^k, \delta p^k) = -A(\mathbf{u}^k, p^k), \tag{3.2}$$

where $d_{(\mathbf{u}^k,p^k)}A$ is the linearization of the mapping $(\mathbf{u}, p) \mapsto A(\mathbf{u}, p)$ at $(\mathbf{u}^k, p^k)$. In the particular case of interest for us, the iterative procedure (3.2) reads as follows:

(1) **Initialization:** The pair $(\mathbf{u}^0, p^0)$ is the solution to the Stokes counterpart of (2.1) (i.e. the version of (2.1) where the non linear term is omitted):

$$\begin{cases} -\nu\Delta\mathbf{u}^0 + \nabla p^0 = 0 & \text{in } \Omega, \\ \operatorname{div}(\mathbf{u}^0) = 0 & \text{in } \Omega, \\ \mathbf{u}^0 = \mathbf{u}_{\text{in}} & \text{on } \Gamma_{\text{in}}, \\ \mathbf{u}^0 = 0 & \text{on } \Gamma, \\ \sigma(\mathbf{u}^0, p^0)\mathbf{n} = 0 & \text{on } \Gamma_{\text{out}}. \end{cases} \tag{3.3}$$

(2) **For** $k = 1, ...,$ $(\mathbf{u}^{k+1}, p^{k+1})$ is obtained by

$$(\mathbf{u}^{k+1}, p^{k+1}) = (\mathbf{u}^k, p^k) + (\delta\mathbf{u}^k, \delta p^k),$$

where $(\delta\mathbf{u}^k, \delta p^k)$ is the solution to the linear system (viz. (3.2)),

$$\begin{cases} -\nu\Delta(\delta\mathbf{u}^k) + (\mathbf{u}^k \cdot \nabla)(\delta\mathbf{u}^k) + ((\delta\mathbf{u}^k) \cdot \nabla)\mathbf{u}^k \\ \quad +\nabla(\delta p^k) = \nu\Delta\mathbf{u}^k - (\mathbf{u}^k \cdot \nabla)\mathbf{u}^k - \nabla p^k & \text{in } \Omega, \\ \operatorname{div}(\delta\mathbf{u}^k) = 0 & \text{in } \Omega, \\ \delta\mathbf{u}^k = 0 & \text{on } \Gamma \cup \Gamma_{\text{in}}, \\ \sigma(\delta\mathbf{u}^k, \delta p^k)\mathbf{n} = 0 & \text{on } \Gamma_{\text{out}}, \end{cases} \tag{3.4}$$

which is sometimes referred to as the Oseen system.

(3) **Ending criterion:** The algorithm ends when

$$e^k < \varepsilon_{\text{stop}}, \text{ with } e^k := \sqrt{\frac{\|\delta\mathbf{u}^k\|^2_{L^2(\Omega)^d} + \|\nabla(\delta\mathbf{u}^k)\|^2_{L^2(\Omega)^{d\times d}}}{\|\mathbf{u}^k\|^2_{L^2(\Omega)^d} + \|\nabla\mathbf{u}^k\|^2_{L^2(\Omega)^{d\times d}}}} \tag{3.5}$$

for a fixed, user-defined tolerance $\varepsilon_{\text{stop}}$.

This ending criterion is inspired from [34, Chapter 6], where the sequence $(\mathbf{u}^k, p^k)$ is proved to converge quadratically to the solution of (2.1), provided the initial pair $(\mathbf{u}^0, p^0)$ is 'close' enough to the latter. In other terms, the error e^k behaves as $e^{k+1} \approx \left(e^k\right)^2$. In practice, only 3 or 4 iterations are required to fulfill (3.5) with $\varepsilon_{\text{stop}} = 10^{-10}$.

Let us mention that many other methods are available for the numerical resolution of (2.1), such as the Oseen iteration method, the Least-Square gradient method, the Peaceman-Racheford method (an increment of the Least-Square gradient method), with different assets and drawbacks which we do not discuss here; see [34].

On a different note, Newton-like algorithms are well-known to experience difficulties as far as convergence is concerned, especially when the initial state is 'far' from the sought solution; in our context of the numerical resolution of the Navier-Stokes system (2.1), this is likely to happen in the case of moderate-to-high Reynolds numbers, where the solution $(\mathbf{u}^0, p^0)$ to the Stokes equation (3.3) is 'too far' from that $(\mathbf{u}, p)$ to (2.1). In such a case, one may resort to mixed strategies (e.g. starting with the Oseen iteration method for some iterations, then branching with the Newton method), or continuation methods (which advocate to increase steadily the Reynolds number) to improve and make the convergence process more robust. As we have already mentioned, the model examples considered in this article (see Section 5) arise in the regime of low Reynolds number, and we did not run into the need for such elaborated strategies.

Remark 3.1. *In practice, we do not solve exactly (3.3), but the slightly modified version*

$$\left\{\begin{array}{cl} -\nu\Delta\mathbf{u}^0 + \nabla p^0 = 0 & \text{in } \Omega, \\ \mathrm{div}(\mathbf{u}^0) + \varepsilon p^0 = 0 & \text{in } \Omega, \\ \mathbf{u}^0 = \mathbf{u}_{in} & \text{on } \Gamma_{in}, \\ \mathbf{u}^0 = 0 & \text{on } \Gamma, \\ \sigma(\mathbf{u}^0, p^0)\mathbf{n} = 0 & \text{on } \Gamma_{out}. \end{array}\right. \tag{3.6}$$

where ε is a very small parameter (typically $\varepsilon = 10^{-6}$). The reason is that only the gradient of p^0 is involved in the system (3.3), which is not well-posed as a result: p^0 is only defined up to a constant; see Remark 3.3. In contrast, (3.6) is well-posed; the matrix associated to its resolution by the Finite Element method is positive definite, which allows to use efficient numerical linear algebra solvers; see e.g. [34, Chapter 4] *about this approach. The same trick applies to (3.4).*

3.2.2. *Choice of the Finite Element discretization.* When it comes to the numerical resolution of linear saddle point problems of the form (3.3) or (3.4), one should pay attention to the choice of the Finite Element spaces used for the discretization of the unknown velocity $\mathbf{u}$ and pressure p. In our case, (3.3) and (3.4) are solved with the Finite Element method in mixed velocity-pressure formulation, using $\mathbb{P}_2$ Lagrange elements for the velocity $\mathbf{u}$ and $\mathbb{P}_1$ Lagrange elements for the pressure p. This choice as regards Finite Element spaces is one among those ensuring that the so-called *Brezzi inequality* holds, and thereby that the discrete linear systems corresponding to (3.3) and (3.4) are invertible. Details about numerical methods for the resolution of saddle point problems can be found in [27] or [29].

3.3. The augmented Lagrangian algorithm for equality-constrained problems.

In order to drive the numerical resolution of (2.3), we rely on the augmented Lagrangian method, a basic sketch of which is provided; we refer to [54] §17.4 for detailed explanations.

The augmented Lagrangian algorithm transforms the constrained optimization problem (2.3) into the series of unconstrained problems (hereafter indexed by the superscript ^{n}):

$$\inf_{\Omega\in\mathcal{O}_{ad}} \mathcal{L}(\Omega, \ell^n, b^n), \tag{3.7}$$

where

$$\mathcal{L}(\Omega, \ell, b) = J(\Omega) - \ell G(\Omega) + \frac{b}{2} G(\Omega)^2 \tag{3.8}$$

In the definition of the augmented Lagrangian $\mathcal{L}$, the parameter b is a (positive) penalty factor for the violation of the constraint $G(\Omega) = 0$, and ℓ is an estimate of the Lagrange multiplier associated with this constraint in (2.3).

The augmented Lagrangian algorithm intertwines the search for the minimizer Ω^n of $\Omega \mapsto \mathcal{L}(\Omega, \ell^n, b^n)$ for fixed values of ℓ^n and b^n, and the update of these coefficients according to the rule:

$$\ell^{n+1} = \ell^n - b^n G(\Omega^n), \text{ and } b^{n+1} = \left\{\begin{array}{ll} \alpha b^n & \text{if } b < b_{\text{target}}, \\ b^n & \text{otherwise;} \end{array}\right. \tag{3.9}$$

in other terms, starting from a 'small' value b^0, the penalty b is increased by a user-defined factor $\alpha > 1$ during the first iterations of the optimization process, until the maximum, 'large' value b_{target} is reached: this smooth increase of b urges the optimized domain to fulfill the constraint in an increasingly stringent way in the course of the optimization process; see Section 5 for the actual values used in our implementation.

We again refer to [54] for an insight about this procedure; let us simply mention that ℓ^n is an increasingly accurate approximation of the Lagrange multiplier for the constraint $G(\Omega) = 0$ featured in (2.3). Notice also that the penalty coefficient b^n is multiplied by a user-defined constant $\alpha > 1$ during the first iterations of the algorithm, and that it is kept fixed afterwards. In particular, the augmented Lagrangian strategy does not require b^n to tend to infinity so to enforce the constraint $G(\Omega) = 0$; this guarantees a better conditioning of (3.7) with respect to the naive quadratic penalty method (featuring only the first and last terms in the definition of $\mathcal{L}$ in (3.7)).

In our context, where the computational burden of minimizing $\Omega \mapsto \mathcal{L}(\Omega, \ell, b)$ is significant, we rely on the following practical implementation of these ideas which limits the number of iterations of the optimization method.

- **Initialization:** Start from an initial shape Ω^0 and coefficients ℓ^0 and b^0.
- **For** $n = 0, \ldots$ **until convergence**
  - choose a descent direction $\boldsymbol{\theta}^n$ for $\Omega \mapsto \mathcal{L}(\Omega, \ell^n, b^n)$,
  - take τ^n small enough so that $\mathcal{L}((\mathrm{Id} + \tau^n \boldsymbol{\theta}^n)(\Omega^n), \ell^n, b^n) < \mathcal{L}(\Omega^n, \ell^n, b^n)$, and set $\Omega^{n+1} = (\mathrm{Id} + \tau^n \boldsymbol{\theta}^n)(\Omega^n)$.
  - update the coefficients ℓ^n and b^n of the augmented Lagrangian $\mathcal{L}$ according to (3.9).

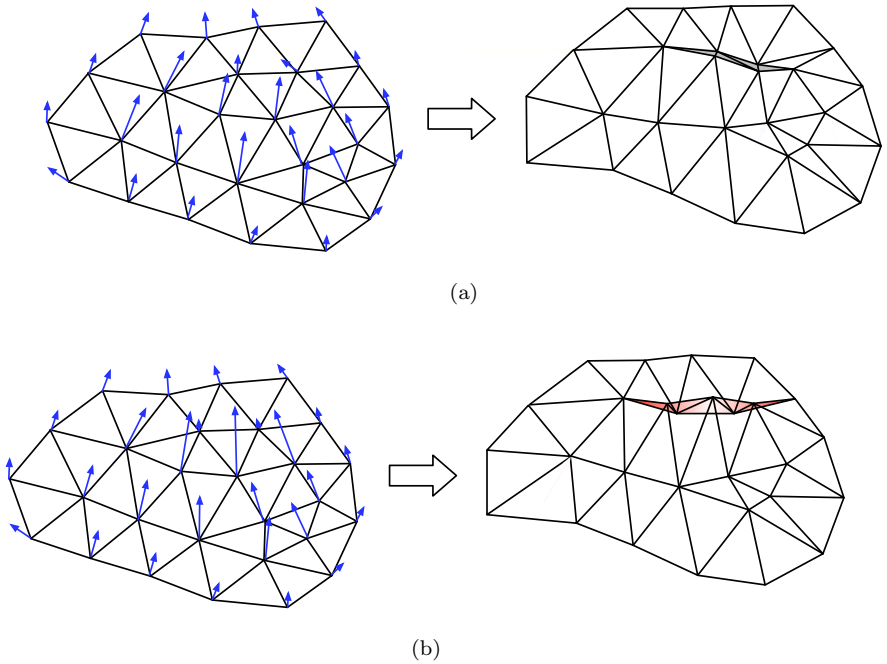
3.4. **Mesh-related issues.**

Assume for one moment that a descent direction $\boldsymbol{\theta}^n$ for (2.3) and a descent step τ^n have been found at the n^{th} iteration of the procedure described in Section 3.1; we are faced with the realization of the operation $\Omega^n \mapsto \Omega^{n+1} = (\mathrm{Id} + \tau^n \boldsymbol{\theta}^n)(\Omega^n)$. If $\mathcal{T}^n$ is the mesh of Ω^n, the natural way to carry it out reads:

$$\mathbf{x}_i^n \mapsto \mathbf{x}_i^{n+1} := \mathbf{x}_i^n + \tau^n \boldsymbol{\theta}^n(\mathbf{x}_i^n), \quad i = 1, \ldots, I^n, \tag{3.10}$$

while the connectivities of the mesh are unchanged, i.e. the considered mesh $\mathcal{T}^{n+1}$ of Ω^{n+1} is made of the same simplices as $\mathcal{T}^n$, but their vertices are relocated according to $\boldsymbol{\theta}^n$.

Unfortunately, this simple procedure is likely to give rise to very stretched (i.e. almost flat) elements within a few iterations. This is problematic since the accuracy of the resolution of PDE with the Finite Element method greatly depends on the *quality* of the elements in the mesh, i.e. on their being close to equilateral [21]. It may also happen that the mesh becomes overlapping in the course of the deformation; see Figure 4 for an illustration of such configurations.



(a)

(b)

FIGURE 4. *Examples of (a) a mesh getting very stretched (grey elements); (b) a mesh developing overlaps (red elements) in the course of its deformation.*

Hence, the numerical resolution of (2.1) may become very inaccurate (not to say impossible) as the computational mesh is successively deformed, causing the whole optimization process to stop prematurely. To circumvent this drawback, we rely on two ingredients:

- The emergence of stretched elements in $\mathcal{T}^n$ may be postponed: in the transformation of Ω^n into Ω^{n+1} (practically, that of $\mathcal{T}^n$ into $\mathcal{T}^{n+1}$ via (3.10)), only the values of $\boldsymbol{\theta}^n$ on the boundary Γ^n determine the new domain Ω^{n+1}; in the numerical framework, the values of $\boldsymbol{\theta}^n$ inside Ω^n are only used to relocate the internal vertices of $\mathcal{T}^n$. In particular, these internal values of $\boldsymbol{\theta}^n$ may be chosen freely, in a way that makes $\mathcal{T}^{n+1}$ of good quality insofar as possible, as we describe in the next Section 3.5.
- When the quality of the mesh becomes poor, i.e. in our context when the volume of one of its elements becomes very small, i.e.

$$\min_{k=1,\dots,K^n} |T_k^n| < \varepsilon_{\text{mesh}},$$

where $\varepsilon_{\text{mesh}}$ is a user-defined parameter (see [31] Chap. 18 for more details, in particular about other possible quality measures of a mesh, which could be easily implemented in FreeFem++.), a *remeshing* of $\mathcal{T}^n$ is carried out: in a nutshell,
  - 'Too long' edges are split,
  - The endpoints of 'too short' edges are merged,
  - The connectivities of ill-shaped triangles (e.g. nearly flat triangles) are swapped,
  - Vertices are moved,

as long as the overall quality of the mesh is improved. See Figure 5 for an illustration of these operations.

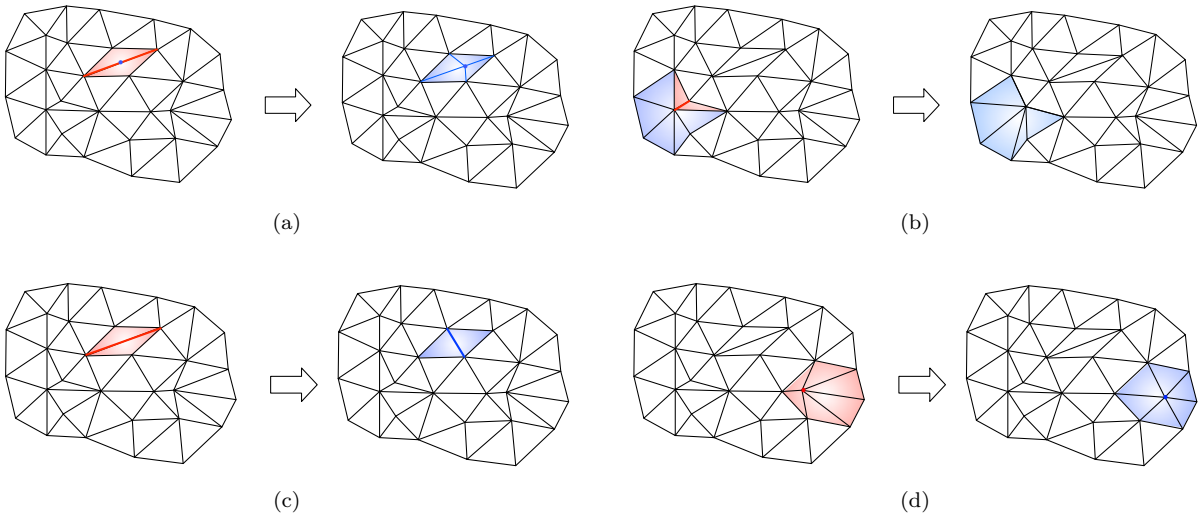


FIGURE 5. *Illustrations of the remeshing operations described in Section 3.4: (a) splitting of a 'long' edge; (b) collapse of the two endpoints of a 'short' edge; (c) swap of the connectivities of a configuration of two ill-shapes triangles; (d) relocation of one vertex.*

From the practical implementation viewpoint, this complex series of operations is conveniently carried out owing to the `movemesh` command in `FreeFem++`.

3.5. **Extension-regularization of the shape gradient.**

As we have seen, our optimization procedure amounts to a series of minimizations of functionals of the form (3.7), which we generically denote by $F(\Omega)$ in the present section. We have seen in Section 2.3 that a natural candidate for a descent direction is

$$\boldsymbol{\theta} = -\phi\,\mathbf{n}, \tag{3.11}$$

where the scalar function $\phi : \Gamma \to \mathbb{R}$ is the $L^2(\Gamma)$-shape gradient of $F(\Omega)$, which is identified from the shape derivative of this functional via (2.13).

Unfortunately, this choice is generally ill-suited for at least two reasons:

(i) Strictly speaking, (3.11) only makes sense on the boundary Γ of the actual shape Ω, while the numerical setting requires the velocity field $\boldsymbol{\theta}$ to be defined on Ω as a whole, see (3.10).

(ii) The $L^2(\Gamma)$ shape gradient ϕ of $F(\Omega)$ may be very irregular, especially in the areas surrounding Γ_{out} because of the change in boundary conditions occurring there. This may cause numerical artefacts when it comes to the mesh procedure (3.10); see for instance [50] §6.2.4. It is therefore often desirable to smooth the velocity field $\boldsymbol{\theta}$ on Γ before performing (3.10).

The popular *extension-regularization* procedure provides alternative ways to calculate a descent direction $\boldsymbol{\theta}$ for $F(\Omega)$ from the knowledge of the shape derivative $F'(\Omega)(\boldsymbol{\theta})$ while overcoming both difficulties; see e.g. [16, 23, 26]. The basic idea consists in identifying a shape gradient for $F(\Omega)$ from its shape derivative $F'(\Omega)(\boldsymbol{\theta})$ (see (2.13)) by means of a different inner product $(\cdot,\cdot)_V$ than $(\cdot,\cdot)_{L^2(\Gamma)}$, acting on a (Hilbert) space V of more regular vector fields, defined on Ω as a whole. More precisely, one searches for $\boldsymbol{\theta} \in V$ such that for all test function $\boldsymbol{\psi} \in V$,

$$(\boldsymbol{\theta}, \boldsymbol{\psi})_V = J'(\Omega)(\boldsymbol{\psi}) = \int_\Gamma \phi\, \boldsymbol{\psi} \cdot \mathbf{n}\, ds. \tag{3.12}$$

Doing so ensures that:

$$J'(\Omega)(-\boldsymbol{\theta}) = -(\boldsymbol{\theta}, \boldsymbol{\theta})_V < 0,$$

which together with (2.14) guarantees that $\boldsymbol{\theta}$ is also a descent direction for $F(\Omega)$.

To be quite precise, in our context, we rely on the space

$$V = \{\mathbf{v} \in H^1(\Omega)^d, \mathbf{v}|_{\Gamma_{\text{in}} \cup \Gamma_{\text{out}}} = 0,\ \nabla_\Gamma \mathbf{v} \in L^2(\Gamma)^d\},$$

where $\nabla_\Gamma f := \nabla f - (\nabla f \cdot \mathbf{n})\mathbf{n}$ is the tangential gradient of a (smooth) function f; V is equipped with the inner product

$$\forall \boldsymbol{\theta}, \boldsymbol{\psi} \in V,\ \ (\boldsymbol{\theta}, \boldsymbol{\psi})_V = \gamma \int_\Omega Ae(\boldsymbol{\theta}) : e(\boldsymbol{\psi})\, dx + (1-\gamma) \int_\Gamma \nabla_\Gamma \boldsymbol{\theta} \cdot \nabla_\Gamma \boldsymbol{\psi}\, ds. \tag{3.13}$$

This definition features two contributions, balanced by the parameter $\gamma \in [0, 1]$:

- The first term in (3.13) is inspired by the linearized elasticity equations. Here, A is the Hooke's law, acting on symmetric matrices e with size $d \times d$,

  $$Ae = 2\mu e + \lambda \text{tr}(e),$$

  where λ and μ are the Lamé coefficients of the fictitious elastic material. This choice - which is widespread in meshing [9, 25] to help in keeping a mesh with fine quality - is motivated by the intuition that elastic displacements tend to induce little compression (i.e. local change in the volume).
- The second term in (3.13) corresponds to the Laplace-Beltrami operator on Γ; its role is to enforce the smoothness of the descent direction $\boldsymbol{\theta}$ on Γ.

With these definitions at hand, the desired 'regularized' shape gradient $\boldsymbol{\theta}$ is calculated by solving (3.12) with a standard Finite Element method on a mesh of Ω.

Remark 3.2. *In our implementation, the Lamé parameters λ, μ of the elastic material used for the extension-regularization procedure are homogeneous over Ω. Notice that the above strategy could be easily improved by considering inhomogeneous elasticity coefficients λ, μ, for instance coefficients characterized by a larger Young's modulus (which measures the resistance to traction and compression efforts) in regions where the mesh of Ω has stretched elements, so to penalize the relative compression rate they undergo.*

Remark 3.3. *A perhaps more natural idea consists in choosing*

$$V = \left\{\mathbf{v} \in H^1(\Omega)^d,\ \ \mathbf{v}|_{\Gamma_{in} \cup \Gamma_{out}} = 0\right\},$$

with associated inner product:

$$(\boldsymbol{\theta}, \boldsymbol{\psi})_V = \gamma \int_\Omega \nabla \boldsymbol{\theta} : \nabla \boldsymbol{\psi}\, dx + \int_\Omega \boldsymbol{\theta} \cdot \boldsymbol{\psi}\, dx,$$

where $\gamma > 0$ is a 'small' parameter. In this context, (3.12) amounts to solving the regularizing, elliptic system:

$$\left\{\begin{array}{ll} -\gamma\Delta\boldsymbol{\theta} + \boldsymbol{\theta} = 0 & \text{in } \Omega, \\ \boldsymbol{\theta} = 0 & \text{on } \Gamma_{in} \cup \Gamma_{out}, \\ \gamma\frac{\partial\boldsymbol{\theta}}{\partial\mathbf{n}} = -\phi\mathbf{n} & \text{on } \Gamma. \end{array}\right. \tag{3.14}$$

However easy to implement, this choice is less efficient than (3.13) insofar as it does not show the same efficiency in preventing the emergence of stretched elements; see the example in Section 5.3 about this point.

3.6. **Calculation of the curvature.**

Most of the numerical methods involved in the resolution of the shape optimization problem (2.3) imply the calculations of the normal vector $\mathbf{n}$ and the curvature κ of the boundary $\partial\Omega$ of a shape Ω (see for instance Theorem 2.1). In practice, these quantities are evaluated from the discrete geometry of a mesh $\mathcal{T}$ of Ω, which is not a completely straightforward task. In this section, following [31], we describe a simple, yet robust method to achieve this goal in the case of two space dimensions: $d = 2$. Similar approximations hold in the general case, which involve more tedious notations.

Let $\mathbf{x}_i$ be a vertex of $\mathcal{T}$ lying on $\partial\Omega$, and let $\mathbf{x}_{i-1}$ (resp. $\mathbf{x}_{i+1}$) be the vertex on $\partial\Omega$ located immediately before (resp. after) $\mathbf{x}_i$ when $\partial\Omega$ is oriented counterclockwise; see Figure 6.

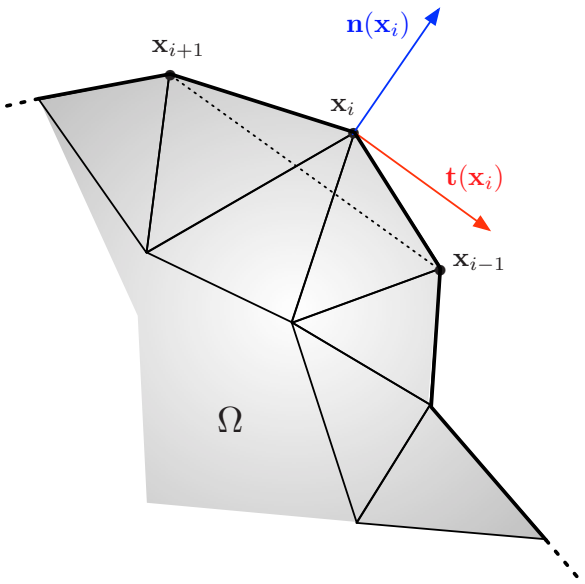


FIGURE 6. *Calculation of the tangent and normal vectors to $\partial\Omega$ from the data of a triangular mesh.*

In this situation, the tangent vector $\mathbf{t}(\mathbf{x}_i)$ to $\partial\Omega$ at $\mathbf{x}_i$ is calculated as:

$$\mathbf{t}(\mathbf{x}_i) = \frac{\overrightarrow{\mathbf{x}_{i+1}\mathbf{x}_{i-1}}}{|\overrightarrow{\mathbf{x}_{i+1}\mathbf{x}_{i-1}}|},$$

and the unit normal vector $\mathbf{n}(\mathbf{x}_i)$ to $\partial\Omega$ at $\mathbf{x}_i$, pointing outward Ω is estimated as the rotate of $\mathbf{t}(\mathbf{x}_i)$:

$$\mathbf{n}(\mathbf{x}_i) = \begin{pmatrix} -t_2(\mathbf{x}_i) \\ t_1(\mathbf{x}_i) \end{pmatrix}.$$

Thence, the curvature radius $r(\mathbf{x}_i)$ at $\mathbf{x}_i$ is approximated as:

$$r(\mathbf{x}_i) = \frac{1}{4}\left(\frac{\overrightarrow{\mathbf{x}_i\mathbf{x}_{i-1}} \cdot \overrightarrow{\mathbf{x}_i\mathbf{x}_{i-1}}}{-\mathbf{n}(\mathbf{x}_i) \cdot \overrightarrow{\mathbf{x}_i\mathbf{x}_{i-1}}} + \frac{\overrightarrow{\mathbf{x}_i\mathbf{x}_{i+1}} \cdot \overrightarrow{\mathbf{x}_i\mathbf{x}_{i+1}}}{-\mathbf{n}(\mathbf{x}_i) \cdot \overrightarrow{\mathbf{x}_i\mathbf{x}_{i+1}}}\right), \tag{3.15}$$

and the curvature $\kappa(\mathrm{x}_i)$ at $\mathbf{x}_i$ is simply $\kappa(\mathrm{x}_i) = \frac{1}{r(\mathbf{x}_i)}$ if none of the denominators featured in (3.15) equals 0 (it is set to 0 otherwise).

3.7. **Algorithmic description of the implemented method.**

We are now ready to provide a precise sketch of the shape optimization algorithm arising from the previous considerations. The brief account below follows exactly the steps of the file `main.edp` of the (commented) supplied code.

(1) **Initialization.**
- The initial shape $\Omega^0 \in \mathcal{O}_{ad}$ is equipped with a triangular mesh $\mathcal{T}^0$.
- Select initial values for the coefficients ℓ^0, $b^0 > 0$ of the augmented Lagrangian algorithm.

(2) **Main loop: for** $n = 0, ...$
- (i) Calculate the solution $(\mathbf{u}^n, p^n)$ to the Navier-Stokes system (2.1) on the mesh $\mathcal{T}^n$ of Ω^n by using the material in Section 3.2.
- (ii) Calculate the solution $(\mathbf{v}^n, q^n)$ to the adjoint system (2.10) or (2.12) on Ω^n.
- (iii) Calculate the $L^2(\Gamma^n)$ shape gradient ϕ^n of $\Omega \mapsto \mathcal{L}(\Omega, \ell^n, b^n)$ by using Theorem 2.2.
- (iv) Infer a descent direction $\boldsymbol{\theta}^n$ for $\Omega \mapsto \mathcal{L}(\Omega, \ell^n, b^n)$ by solving (3.12) (3.13) on the mesh $\mathcal{T}^n$.
- (v) Find a descent step τ^n such that

$$\mathcal{L}((\mathrm{Id} + \tau^n \boldsymbol{\theta}^n)(\Omega^n), \ell^n, b^n) < \mathcal{L}(\Omega^n, \ell^n, b^n) \tag{3.16}$$

(possibly up to a small tolerance)
- (vi) Move the vertices of $\mathcal{T}^n$ according to τ^n and $\boldsymbol{\theta}^n$:

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \tau^n \boldsymbol{\theta}^n(\mathbf{x}_i^n) \tag{3.17}$$

  - If the resulting mesh is invalid, go back to step (v), and use a smaller value for τ^n,
  - Else, the positions (3.17) define the vertices of the new mesh $\mathcal{T}^{n+1}$.
- (vii) If the quality of $\mathcal{T}^{n+1}$ is too low, use a local remeshing as described in Section 3.4.
- (viii) Update the augmented Lagrangian parameters according to (3.9).

(3) **Ending criterion.** Stop if

$$\|\boldsymbol{\theta}^n\|_{L^2(\Gamma^n)} < \varepsilon_{\mathrm{stop}}$$

Return Ω^n.

4. Practical implementation of the shape optimization algorithm

In this section, we describe the practical code used in the numerical experiments of Section 5. In addition to the detailed comments accompanying the sources, we focus our discussion on the parts that should be modified for the user to implement a different geometric or physical situation (i.e. to change the initial shape, the objective function, the shape derivative, etc.).

The user is supposed to have installed the free software `FreeFem++`. This environment allows to solve partial differential equations by the Finite Element method from the input of their variational formulation via an adapted pseudo-language. We recommend using the latest release, although our programs work with any version above 3.42. `FreeFem++` is available at

http://www.freefem.org/ff++/,

`--config`	Number of the considered test-cases; `config` ranges from 1 to 7
`--navsto`	The Stokes (resp. Navier-Stokes) system models the flow if `navsto` is 0 (resp. 1)
`--tau`	Value of τ, initial step in the gradient descent, see (3.10)
`--errc`	Value of the stopping criterion $\varepsilon_{\text{stop}}$
`--gamma`	Value of the regularization parameter γ; see (3.13)
`--beta`	The constraint function $G(\Omega)$ is $\text{Vol}(\Omega)$ if `beta` is 1, and $\text{Per}(\Omega)$ if `beta` is 0.
`--delta`	The objective $J(\Omega)$ is the dissipated energy (2.4) if `delta` is 1, and the least-square discrepancy (2.5) if `delta` is 0.
`--binit`	Initial value for the penalty parameter `b` in (3.8)
`--btarget`	Limiting value for `b`
`--cv`	Desired constraint (volume or perimeter) over initial value for the constraint $G(\Omega)$
`--optraff`	0 for no remeshing, **1** for remeshing when necessary
`--raffinit`	Value of the `raff` parameter used in the routines for mesh adaptation (see Section 4.8)

TABLE 1. Main parameters passed on the command line.

and it comes along with a comprehensive documentation [37].

4.1. **Organization of the repository and of the program.**

Our code may be downloaded from the address:

`https://github.com/flomnes/optiflow`.

The main repository is organized as follows:

- The folder `./meshes` contains the mesh files associated to the initial shapes of the test cases of Section 5: `mesh1.mesh`, `mesh2.mesh`, etc.
- The `FreeFem++` source code used to generate these meshes is in the file `geometry.edp`.
- As the name suggests, the file `main.edp` contains the main routines of the optimization process.
- The file `macros.edp` contains several useful macros; see Section 4.3.
- The file `curvature.edp` gathers the routines involved in the calculation of the mean curvature κ of shapes; see Section 4.5.
- The files `run_case.sh` and `run_all.sh` are `shell` scripts containing the sample command lines needed to launch any, or all of the proposed test cases in Section 5.

4.2. **Main parameters.**

The main program, written in the file `main.edp`, is executed by using the command line

```
FreeFem++ --param1 value1 ...  main.edp,
```

where `--param1`, ... are the computational parameters of the considered test case; see Table 1.

Seven geometric settings (associated to different meshes of the initial shape and applied boundary conditions) are implemented in our code, corresponding to values of the `config` parameter ranging from 1 to 7. The precise command lines used to launch these examples are supplied in the file `run_case.sh`. The first five configurations correspond to the numerical results of Section 5.

4.3. **Main macros.**

Our program relies on macros insofar as possible: it is a convenient way in `FreeFem++` to ensure that the various operations carried out resemble their mathematical counterparts. The shortcuts that are consistently used throughout the implementation are stored in the file `macros.edp`; see Listing 1 for a sample.

```
/* Strain tensor */
macro EPS(u, v) [dx(u), 1./2*(dx(v)+dy(u)), 1./2*(dx(v)+dy(u)), dy(v)] // EOM

/* Jacobian matrix */
macro GRAD(u, v) [dx(u), dy(u), dx(v), dy(v)] // EOM

/* (u \cdot \nabla) V */
macro UgradV(u1,u2,v1,v2) [ [u1,u2]'*[dx(v1),dy(v1)] , [u1,u2]'*[dx(v2),dy(v2)] ]// EOM
```

Listing 1. Several macros (from `macros.edp`)

4.4. **Definition of the geometry and of the Finite Element setting.**

The meshes associated to the proposed test cases are supplied in the folder `./meshes`. The mesh `Th` corresponding to the considered situation (i.e. associated to the actual value of the `config` parameter) is read at the beginning of the `main.edp` file; see Listing 4.

```
/* Load initial mesh */
string meshname = "meshes/mesh"+config+".mesh";
cout << "Loading mesh " << meshname << "...";
Th = readmesh(meshname);
cout << "done." << endl;
cout.flush;
```

Listing 2. Reading the initial shape (from `main.edp`)

The Finite Element spaces on the mesh `Th` are then defined as in Listing 3.

```
fespace Qh(Th,P1);
fespace Vh(Th,P2);

Vh    ux, uy, vx, vy, wx, wy, dux, duy, uxx, uyy, clx, cly;
Qh    p,q, mx, dpx, dpy, dp, qq, phix, phiy, kappa, phi, psi;
```

Listing 3. Definition of the Finite Element spaces and functions (from `main.edp`)

These meshes may be generated using the code in the file `geometry.edp`, which can easily be modified and adapted to describe a different physical setting.

For instance, the code in Listing 4 allows to create the mesh of the initial shape in the bend test case of Section 5.1; see Figure 10 (top).

```
/* Bend with orthogonal inlet and outlet */
if (config==1) {
  border in(t=0,1){x=param(0,1./3,t);
                   y=0;
                   label=2;};
  border sig1(t=0,1){
           x=circlearcx(1,2./3,pi,pi/2,t);
           y=circlearcy(0,2./3,pi,pi/2,t);
           label=3;};
  border out(t=0,1){x=1;
```

```
                        y=param(2./3,1,t);
                        label=1;};
  border sig2(t=0,1){
            x=circlearcx(1,1,pi/2,pi,t);
            y=circlearcy(0,1,pi/2,pi,t);
            label=3;};
  Th=buildmesh(in(pp/2)+sig1(pp)
                  +out(pp/2)+sig2(pp));
  Th=adaptmesh(Th,IsMetric=1,1./30);
}
```

LISTING 4. Creation of the initial mesh in the bend example of Section 5.1 (from `geometry.edp`)

4.5. **Practical calculation of the mean curvature.**

The routines dedicated to the calculation of the mean curvature `kappa` of the boundary of the optimized shape are a little involved. They are gathered in the file
`curvature.edp` and in principle, they do not need to be modified.

The calculation of `kappa` in `main.edp` is then carried out along the lines of Listing 5.

```
kappa=0;
calculconnect(Th,ordre);
courbure(Th,ordre,kappa[]);
kappa=kc*kappa;
```

LISTING 5. Calculation of the mean curvature in `main.edp`

4.6. **Resolution of the flow equations.**

As outlined in Section 3.2, the Navier-Stokes equations are solved iteratively thanks to the Newton method.

To achieve this, the Stokes equation is first defined as a variational problem; see Listing 6 and Remark 3.1:

```
problem stokes([ux, uy, p], [vx, vy, q]) =
  int2d(Th)(2*mu*tr(EPS(ux,uy))*EPS(vx,vy) - p * div(vx,vy))
  +int2d(Th)(div(ux,uy)*q)
  -int2d(Th)(p*q*epsilon)
  +on(3,ux=0,uy=0)
  +on(1,ux=clx, uy=cly);
```

LISTING 6. Variational problem for the Stokes system (from `main.edp`)

The Navier-Stokes system is solved for the velocity and pressure `[ux,uy,p]` by using the macro `ns` reprinted in Listing 7. In a nutshell, the Stokes system is solved as an initial guess; then, if the parameter `navsto` is set to 1, a loop is performed during which the Oseen equation is solved for the increment `[dux,duy,dp]`, from which `[ux,uy,p]` is updated.

```
  macro ns () {
/* Only solve when necessary if ns has
never been executed or if the mesh
has changed since the last resolution */
  if (solvefluid) {
/* Initialize Newton loop with the solution of Stokes */
    stokes;
/* If we want to solve Navier-Stokes,
iterate successive Oseen problems */
    if(navsto) {
      int n;
      real err=0;
      cout << "Navier-Stokes";
      /* Newton Loop  */
```

```
      for(n=0; n< 15; n++) {
  solve Oseen([dux,duy,dp],[vx,vy,qq]) =
    int2d(Th)(2*nu*tr(EPS(dux,duy))*EPS(vx,vy)
        + tr(UgradV(dux,duy, ux, uy))*[vx,vy]
        + tr(UgradV(ux,uy,dux,duy))*[vx,vy]
        - div(dux,duy)*qq - div(vx,vy)*dp
        - epsilon*dp*qq)
    +int2d(Th)(2*nu*tr(EPS(ux,uy))*EPS(vx,vy)
         + tr(UgradV(ux,uy, ux, uy))*[vx,vy]
         - div(ux,uy)*qq - div(vx,vy)*p
         - epsilon*p*qq)
    +on(1,3,dux=0,duy=0);

  ux[] += dux[];
  uy[] += duy[];
  p[]  += dp[];
  err = sqrt(int2d(Th)(tr(GRAD(dux,duy))*GRAD(dux,duy) + tr([dux,duy])*[dux,duy]) / int2d(Th
      )(tr(GRAD(ux,uy))*GRAD(ux,uy) + tr([ux,uy])*[ux,uy]));
  cout << ".";
  cout.flush;
  if(err < arrns) break;
      }
      /* Newton loop has not converged */
      if(err > arrns) {
  cout << "NS Warning : non convergence : err = " << err << " / eps = " << epsilon << endl;
      }
    }
    cout << endl;
/* It is not necessary to solve ns until the mesh is moved or adapted */
    solvefluid = 0;
    nflsolved++;
  }
}//EOF
```

LISTING 7. Resolution of the flow equations (from `main.edp`)

4.7. **Calculation of the objective function and of the shape derivative.**

The considered objective function $J(\Omega)$ in (2.3) is the energy dissipation (2.4) if the parameter `delta` is set to 1, and a least-square difference (2.5) between the fluid velocity and a target velocity if `delta` is 0. These are calculated from the macro in Listing 8.

```
/* Objective function = weighted sum of energy dissipation and least-square difference with
    a
 prescribed flow */
macro J() (2*delta*mu*int2d(Th)(tr(EPS(ux,uy))*EPS(ux,uy)) + ((1.-delta)/2)*int1d(Th,2)((ux-
    uxx)^2+(uy-uyy)^2)) //EOM
```

LISTING 8. Macro for the objective function (from `macros.edp`)

Likewise, the constraint function $G(\Omega)$ is $\mathrm{Vol}(\Omega)$ if `beta` is 1, and $\mathrm{Per}(\Omega)$ if `beta` is 0; these are calculated from the macro in Listing 9.

```
/* Constraint function = weighted sum of volume and perimeter */
macro contr(Th) (beta*int2d(Th)(1.)
                +(1.-beta)*int1d(Th)(1.)) //EOM
```

LISTING 9. Macro for the constraint function (from `macros.edp`)

Thence, the value of the augmented Lagrangian functional is calculated by means of the macro `EL`, reprinted in Listing 10.

```
/* Augmented Lagrangian */
macro EL() (J/J0 + l*(contr(Th) - ctarget)/c0 + b/2 * ((contr(Th) - ctarget)^2)/(c0^2)) //
    EOM
```

Listing 10. Macro for the augmented Lagrangian (from `macros.edp`)

At each iteration of the optimization loop (see Section 4.8 below), the adjoint states `[vx,vy,q]` are calculated as the solution to (2.10) if `delta` is 1 or (2.12) if `delta` is 0. This is achieved by calling the macro `adjoint` reprinted in Listing 11. Notice the presence of the `navsto` variable in the variational problem for the adjoint states, corresponding to the term induced by the non linearity of the flow equation (2.1) if `navsto` equals 1.

```
macro adjoint() {
  solve probadjoint([vx, vy, q], [wx, wy, qq]) =
  int2d(Th) (2*nu*tr(EPS(vx, vy))*EPS(wx, wy)
          -q*div(wx, wy)
          -qq*div(vx, vy)
          +navsto*(tr(UgradV(wx, wy, ux, uy))*[vx, vy]+tr(UgradV(ux,uy,wx,wy))*[vx,vy]))
 +int2d(Th)(-4*nu*delta*tr(EPS(ux,uy))*EPS(wx,wy))
 +int1d(Th,2)(-(1-delta)*((ux-uxx)*wx+(uy-uyy)*wy))
 +on(1,3,5, vx=0, vy=0);
}//EOM
```

Listing 11. Macro for the resolution of the adjoint system (from `main.edp`)

The shape derivatives of the considered objective and constraint functions $J(\Omega)$ and $G(\Omega)$, and that of the augmented Lagrangian $\mathcal{L}(\Omega, \ell, b)$ are then computed, again, thanks to a set of macros defined in the file `macros.edp`; see Listing 12.

```
/* Strain tensor */
macro EPS(u, v)
  [dx(u), 1./2*(dx(v)+dy(u)),
   1./2*(dx(v)+dy(u)), dy(v)] // EOM
/* Shape derivative of the objective function */
macro IJ()
  (-2*delta*nu*tr(EPS(ux,uy))*EPS(ux,uy)
  +2*nu*tr(EPS(ux,uy))*EPS(vx,vy)) //EOM
/* Shape gradient of the constraint function */
macro gradC() (beta*1+(1.-beta)*kappa) //EOM
/* Shape-gradient of the Lagrangian */
macro gradDF()
  (IJ/J0 + l*gradC/c0
  +b*gradC*(contr(Th)-ctarget)/(c0^2))//EOM
```

Listing 12. Macros for shape derivatives (from `macros.edp`)

The shape gradient of the augmented Lagrangian on Γ is then extended to the whole computational mesh using the `regulbord` macro; the result of which is stored in the variable `[dpx, dpy]`; see Listing 13.

```
macro regulbord() {
solve regb([dpx, dpy],[phix, phiy]) =
  int2d(Th)(gamma*tr(SIG(dpx, dpy))*EPS(phix, phiy))
  +int1d(Th,3)(gamma1*tr(gradT(dpx))*gradT(phix))
  +int1d(Th,3)(gamma1*tr(gradT(dpy))*gradT(phiy))
  +int1d(Th,3)(gradDF*dotN(phix, phiy))
  +int1d(Th,4)(1./epspen*dotN(dpx,dpy)*dotN(phix,phiy))
  +on(1, 2, dpx=0)
  +on(1, 2, dpy=0);
}//EOM
```

Listing 13. Macro for the extension-regularization procedure of the shape gradient (from `main.edp`)

4.8. **Main optimization loop : gradient descent with line search.**

Last but not least, we now discuss our implementation of the algorithm of Section 3.7 for the resolution of the shape optimization problem (2.3), properly speaking.

This is achieved by means of two nested loops; see Listings 14 and 15. The main, outermost loop, reprinted in Listing 14, drives the update of the shape. At the beginning of each iteration, the actual shape is stored in the mesh `Th2`; then the direct and adjoint problems are solved thanks to the macros `ns` and `adjoint` respectively (see Line 4); a descent direction `[dpx,dpy]` from the actual shape `Th2` is inferred by using the macro `regulbord` (Line 8). Meanwhile, the performance `L0` - i.e. the value of the augmented Lagrangian - of `Th2` is calculated (Line 9).

Then, starting from the input parameter `tau` an appropriate value of the time step `tau1` is found by the inner loop of Listing 15, which is described below.

This inner loop results in a mesh `Th` of the new shape; the coefficients ℓ^n and b^n of the augmented Lagrangian are eventually updated (Line 29). This main loop stops if either the maximum number of iterations `jjmax` is reached or if the ending criterion

$$\texttt{sv} \leq \texttt{errc}$$

is fulfilled; see Line 23 in Listing 14 and Section 3.7.

```
1  /* The algorithm stops when jj reaches jjmax or the ending criterion is low enough */
2  for (jj = 0; (sv > errc) && (jj < jjmax); jj++) {
3    Th2 = Th; // Keep a copy of the mesh
4    ns; // Solve the NS equation if needed
5    adjoint;  // Solve the adjoint system
6    /* Solve the velocity extension/regularization problem to get the descent direction;
7       the descent direction is [dpx,dpy] */
8    regulbord;
9    L0 = EL;  // Value of the augmented Lagrangian
10   tau1 = tau;
11
12   /* Inner loop for line search */
13   ****************************************
14   ** Linear seach loop: see Listing 15 below **
15   ****************************************
16
17   /* Maximum number of iterations has been reached, and no decrease in the value of the
         augmented Lagrangian is observed */
18   if (kk == kkmax) {
19     cout << "Warning : L_{n+1}>L_{n} (L0 = " << L0 << ", l = " << l << ")" << endl;
20   }
21
22   /* L^2 norm of the shape gradient, used as the ending criterion */
23   sv = sqrt(int1d(Th,3)(dpx^2+dpy^2));
24
25   /* Print output */
26   r << J << " " <<  EL << " " << contr(Th) << " " << l << " " << sv << " " << b << " " <<
         minarea << endl;
27
28   /* Update of the values of the coefficients of the augmented Lagrangian */
29   l = l + b * (contr(Th) - ctarget);
30
31   /* Increase b if it is less than btarget */
32   if (b < btarget) {
33     b *= alpha;
34   }
35   cout << "jj = " << jj << endl;
```

```
}
```

LISTING 14. Main loop of the optimization algorithm (from `main.edp`)

Let us now describe the inner loop, which is nothing but a basic line search procedure for finding a suitable value of the time step `tau1`; see Listing 15. This procedure is initialized while `tau1=tau`, `tau` being a user-defined value. At each iteration of the inner loop, the current shape `Th2` is deformed along the descent direction `[dpx,dpy]` for a time `tau1`; this yields a new, 'attempt' mesh `Th` (Lines 6-30).

This shape `Th` is then evaluated: the flow equations are solved on `Th` (see Line 37), and the value `L1` of the augmented Lagrangian associated to `Th` is calculated (Line 40). If `L1` is smaller than the value `L0` of the augmented Lagrangian of the current shape `Th2`, the loop ends, and the 'attempt' mesh `Th` is accepted as the updated shape. Otherwise, the procedure is repeated from the beginning once the value of `tau1` has been divided by 2.

Note that, if after `kkmax=10` iterations of the line search procedure, none of the produced 'attempt' meshes `Th` has produced a value `L1` of the augmented Lagrangian smaller than `L0`, the last iteration `kk = kkmax` is accepted nevertheless; the step used in this case being $\tau/2^{10}$, `Th` is then very close to `Th2`.

```
1  for (kk = 0;kk < kkmax; kk++) {
2    cout << "movemesh tau = "<< tau1 << endl;
3    minarea = checkmovemesh(Th2, [x + tau1*dpx, y + tau1*dpy]);
4
5    /* Try to adapt the mesh in case one of the triangles becomes degenerate */
6    if(optraff) {
7      /* No adaptation if the minimal area is larger than parameter minarea0 or if we already
           tried remeshing 3 times in this loop */
8      if (minarea > minarea0 || adaptcount>=3) {
9        Th = movemesh(Th2, [x + tau1*dpx, y + tau1*dpy]);
10       solvefluid = 1;
11     }
12     else {
13       cout << "*** ADAPTMESH *** minarea = " << minarea << " minarea0 = " << minarea0 <<
             endl;
14       Th = adaptmesh(Th, hmax=raff, hmin=raff/sqrt(2), ratio=1.5);
15       cout << " new minarea = " << minarea << endl;
16       minarea = checkmovemesh(Th2, [x + tau1*dpx, y + tau1*dpy]);
17       solvefluid = 1;
18       kappa = 0;
19       calculconnect(Th, ordre);
20       adaptcount++;
21     }
22
23     if(adaptcount>=3) {
24       cout << "Too many consecutive mesh adaptations. Giving up mesh adaptation" << endl;
25     }
26   }
27   else {
28     Th = movemesh(Th2, [x + tau1*dpx, y + tau1*dpy]);
29     solvefluid = 1;
30   }
31
32   /* Calculate the mean curvature of the new shape */
33   courbure(Th, ordre, kappa[]);
34   kappa = kc * kappa;
35
36   /* Solve the Navier-Stokes and adjoint equations on the new shape */
37   ns;
38
39   /* New value of the objective function */
```

Case number	Duration
1	3m24s
2	10m8s
3	4m28s
4	12m7s
5	6m43s

TABLE 2. CPU time for the numerical examples of Section 5.

	$\frac{V_T}{V_0}$	$\frac{P_T}{P_0}$	γ	τ^0	$\varepsilon_{\text{stop}}$	ℓ^0	b_0	b_{target}
Case 1	1		10^{-2}	10^{-2}	10^{-2}	0	1	10^1
Case 2	1		10^{-2}	10^{-2}	5×10^{-3}	0	10^{-1}	10^1
Case 3			1	10^{-2}		0	0	0
Case 4	1		1	3×10^{-3}	2×10^{-2}	15	10^1	10^2
Case 5		0.97	1	10^{-2}	10^{-2}	0	10^2	10^2

TABLE 3. *Parameters used for the numerical examples of Section 5. From left to right: desired volume over initial volume, desired perimeter over initial perimeter, regularization parameter, initial gradient step, stopping criterion, initial Lagrange multiplier, initial value of b, target value of b.*

```
40   L1 = EL;
41   tau1/= 2;
42   cout << "L = " << L1 << " / L0 = " << L0 << " (variation = " << 100*(L1-L0)/L0 << "%)" <<
         endl;
43
44   /* Accept iteration as soon as the value of the augmented Lagrangian is decreased */
45   if (L1 < L0) {
46     break;
47   }
48 }
```

LISTING 15. Line search in the optimization algorithm (from `main.edp`)

One final word about remeshing is in order. If the `optraff` input parameter equals 1, then whenever the mesh of the shape is deformed, the minimum area of an element in the tentative mesh `Th` is compared to the parameter `minarea0` (see Lines 6-30 in Listing 15). If smaller, the mesh is adapted thanks to the `adaptmesh` command of `FreeFem++` (Line 14 in Listing 15). The resulting mesh has edges with length comprised betwen `raff` and `raff/sqrt(2)`, where `raff` stems from the `raffinit` parameter from the command line.

Note that, if mesh adaptation occurs, the connectivity of the boundary has to be calculated anew by calling `calculconnect` (viz. line 33), so that the routines described in Section 4.5 may be used to calculate the curvature of the shape.

5. NUMERICAL ILLUSTRATIONS

In this section, we present five two-dimensional applications of the numerical algorithm presented in Section 3.7. The geometric configurations associated to these examples are represented in Figure 7, and the parameters used in the different test cases (initial parameters of the augmented Lagrangian algorithm, target volume, etc.) are reported in Table 3. The approximate CPU time when running each example on a workstation with an Intel Core i5-7600T @ 2.80GHz CPU is indicated in Table 2.

5.1. **Minimization of the dissipated energy in a bend.**

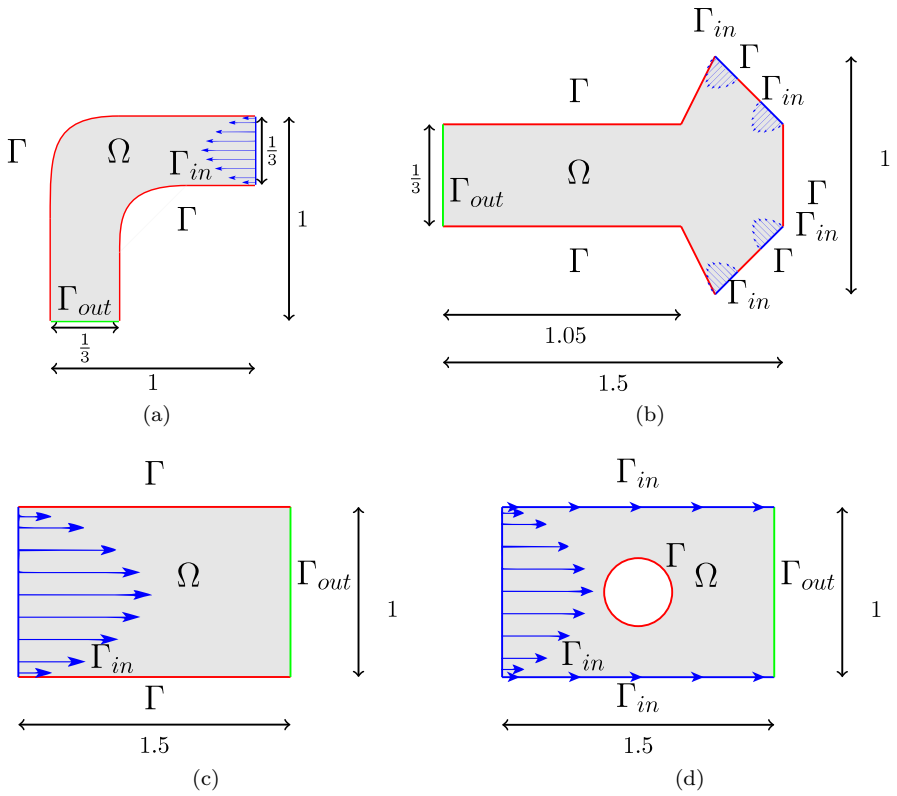


FIGURE 7. *Settings of the five test cases discussed in Section 5; (a) the bend, discussed in Section 5.1, (b) the ramified structure of Sections 5.2 and 5.3, (c) the straight pipe with one inlet, one outlet where a least-square criterion is considered, as studied in Section 5.4, (d) the dissipated energy minimization example of Section 5.5.*

Our first benchmark example is concerned with the optimization of the shape of a pipe with orthogonal inlet and outlet, as depicted in Figure 7 (a); see for instance [22, 12]. In a nutshell, this test case answers the question:

> "How to build a pipe with fixed volume that spends the least amount of energy to convey a fluid from Γ_{in} to Γ_{out}?"

The inlet flow is given by the parabolic profile

$$\mathbf{u}_{\text{in}}(x_1, x_2) = ((1 - x_2)(\frac{2}{3} - x_2), 0).$$

Starting from the initial shape Ω^0 represented in Figure 10 (top), we minimize the work of viscous forces, i.e. $J(\Omega) = E(\Omega)$, as defined by (2.4), under the volume constraint $\text{Vol}(\Omega) = V_T$, where $V_T = \text{Vol}(\Omega^0)$, i.e. the target volume is that of Ω^0.

The results are displayed on Figure 10, and the associated convergence histories are included in Figure 9. The dissipated viscous energy decreases by roughly 25% during the process, and as expected, the optimized design looks like a straight pipe. It is worth mentioning that theoretical arguments in [40] support this observation for a very close model.

Eventually, let us mention that this test case is fairly insensitive to the computation parameters ℓ^0 and τ, which makes it the easiest of all five to run.

5.2. **Minimization of the dissipated energy in a ramified structure with volume constraint.**

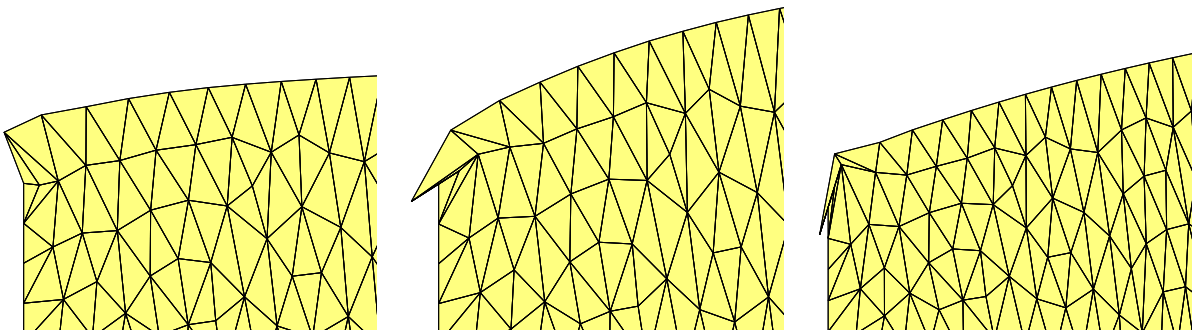


FIGURE 8. *Left upper corner of intermediate shapes Ω^n obtained in the ramified structure example of Section 5.3 at iterations (from left to right) $n = 15, 60$ and 130.*

Our second example is a simple model for the ramified structure of a human lung. It can be considered as an extension of the study in [24].

The situation is that of Figure 7 (b), where incoming parabolic profiles are imposed on each component of Γ_{in}. More precisely, Γ_{in} is the reunion of four disjoint line segments; for any of these segments, let us denote by (x_1^A, x_2^A) and (x_1^B, x_2^B) the two ending points, which are assumed to be distributed counterclockwise on Γ_{in}. The imposed inlet flow on the considered segment is then defined by:

$$\mathbf{u}_{\text{in}}(x_1, x_2) = s(1-s)\begin{pmatrix} -(x_2^B - x_2^A) \\ x_1^B - x_1^A \end{pmatrix}$$

where $s = \frac{x_1^A - x_1}{x_1^A - x_1^B} = \frac{x_2^A - x_2}{x_2^A - x_2^B}$ so that in particular $\mathbf{u}_{\text{in}}$ is oriented toward inside Ω: $\mathbf{u}_{\text{in}} \cdot \mathbf{n} \leq 0$ on Γ_{in}.

In this context, we again aim at optimizing the energy dissipated owing to viscous effects, i.e. $J(\Omega) = E(\Omega)$, under the volume constraint $\text{Vol}(\Omega) = V_T$, where $V_T = \text{Vol}(\Omega_0)$.

The results are presented in Figure 11, and the associated convergence histories are those in Figure 12. Interestingly enough, ramifications appear in the course of the iterations and the optimized shape is much smoother than the initial one. These results are also in accordance with those obtained in [24].

This example shows large mesh deformations, which justifies the importance of using a good extension-regularization procedure, such as that introduced in section 2.3.

5.3. **Minimization of the dissipated energy in a ramified structure with perimeter constraint.**

This third example arises in the exact same physical context as that of Section 5.2 (again, see Figure 12. The only difference with the latter is that we now impose a constraint on the perimeter of shapes: $\text{Per}(\Omega) = P_T$, with $P_T = 0.97\ \text{Per}(\Omega^0)$. The convergence histories of the computation are reported on Figure 13, and the shape at several intermediate stages is represented on Figure 14.

Let us emphasize the role of the regularizing parameter γ featured in the definition of the extension-regularization inner product (3.13). In this example (as in the previous ones), the $L^2(\Gamma)$ shape gradient of $E(\Omega)$ is not smooth in the vicinity of the transitions between parts of the boundary bearing different types of boundary conditions (that is, the transitions between Γ_{in}, Γ_{out} or Γ). Therefore, if no regularization of this gradient is applied, mesh intersections appear within a few iterations in this region.This phenomenon is illustrated in Figure 8: all parameters retain the same values except for γ, which is changed to 1. The mesh irregularities are caused by an irregular shape gradient, and are not observed for $0 < \gamma < 1$.

5.4. **Minimization of the discrepancy with a reference velocity profile.**

Our third example considers pipes Ω in the situation depicted on Figure 7 (c), where the parabolic profile

$$\mathbf{u}_{\text{in}}(x_1, x_2) = (x_2(1-x_2), 0)$$

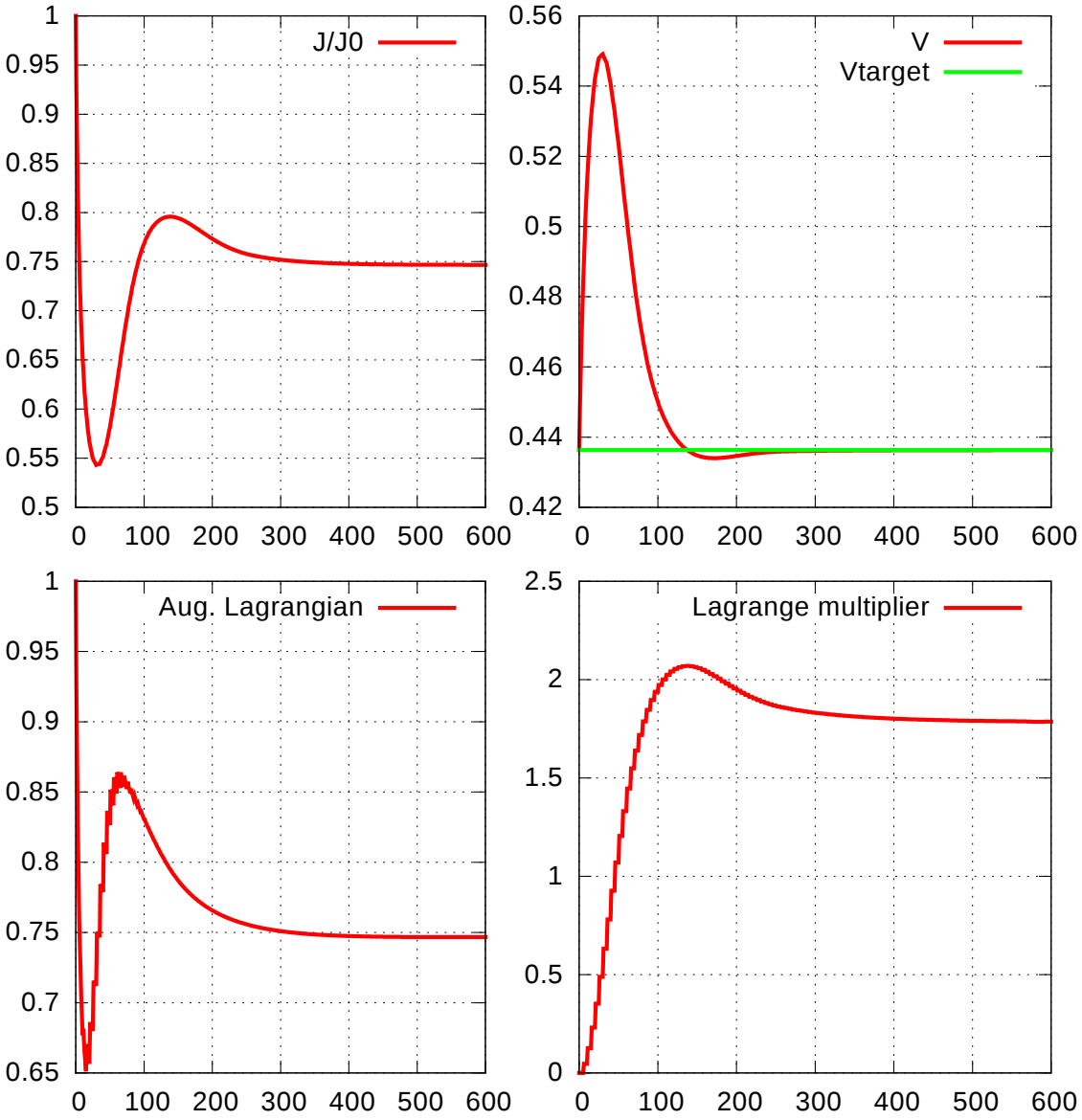


FIGURE 9. *Convergence histories of (from left to right, top to bottom)* $J(\Omega)$, $\mathrm{Vol}(\Omega)$, $\mathcal{L}(\Omega, \ell, b)$ *and* ℓ^n *in the bend optimization example of Section 5.1*

is imposed on the inlet Γ_{in}. Our aim is to optimize the shape of Ω with respect to the least-square criterion $J(\Omega) = D(\Omega)$ given by:

$$D(\Omega) = \int_{\Gamma_{\mathrm{out}}} |\mathbf{u} - \mathbf{u}_{\mathrm{ref}}|^2 \, ds, \tag{5.1}$$

where the reference profile $\mathbf{u}_{\mathrm{ref}}$ is slightly off-centered (see Figure 16 (top)), namely:

$$\mathbf{u}_{\mathrm{ref}}(x_1, x_2) = (2x_2^2(1 - x_2), 0). \tag{5.2}$$

This example is motivated by the following consideration: the initial shape Ω^0 is a straight cylinder, and it is well-known that the laminar flow satisfying the Navier-Stokes equations (2.1) is of Poiseuille type ([52], chapter 6):

$$\mathbf{u}(x_1, x_2) = (x_2(1 - x_2), 0). \tag{5.3}$$

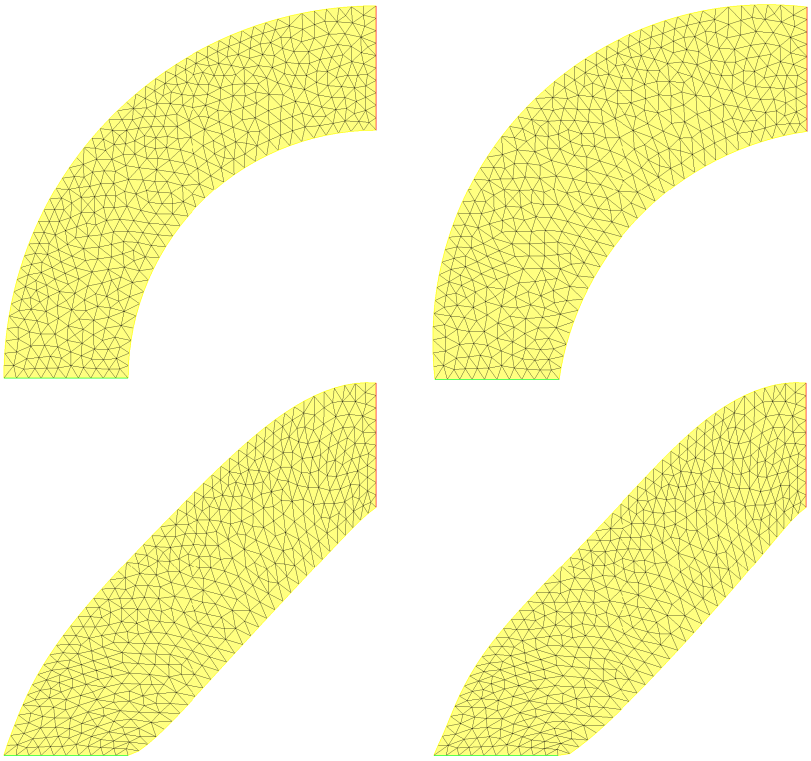


FIGURE 10. *Intermediate shapes Ω^n obtained in the bend optimization example of Section 5.1 at iterations (from left to right, top to bottom) $n = 0, 5, 100$ and 500.*

Hence, imposing that the velocity $\mathbf{u}$ in the pipe resemble (5.2) on Γ_{out} should shift the maximum of the horizontal component u_1 towards the top of the pipe.

No constraint is applied, so that the algorithm of Section 3.7 reduces to a simple gradient method in this case, i.e. $b = \ell_0 = 0$.

Several intermediate shapes are presented on Figure 17, and the corresponding one-dimensional profiles of u_1 and u_2 on Γ_{out} are reported on Figure 16. The least-square criterion $D(\Omega)$ decreases by roughly 63% in the course of the optimization. Note that, as the final value of $D(\Omega)$ is not exactly 0, the velocity $\mathbf{u}$ does not match exactly the reference profile $\mathbf{u}_{\text{ref}}$ on Γ_{out}.

To achieve this decrease of the least-square criterion, the shape Ω develops a 'bump' on its bottom side, so that the flow is deviated towards the top. After iteration 430, a cusp appears at the bottom-right corner of Ω, which interrupts the algorithm. However, at this stage, $J(\Omega^n)$ is only a few percents lower than $J(\Omega^{430})$, so we decided to stop the optimization process at iteration 430.

5.5. **Energy dissipation around an obstacle.**

In our last example, depicted on Figure 7 (d), a solid obstacle is immersed in a cavity filled with a fluid, and the shapes Ω stand for the fluid domain, which is the complement of the obstacle in the cavity. Our aim is to minimize the dissipated energy in the cavity with respect to the shape of the obstacle, i.e. $J(\Omega) = E(\Omega)$ with the volume constraint $\text{Vol}(\Omega) = V_T$, $V_T = \text{Vol}(\Omega^0)$.

A very similar version of this problem is considered in [59] and [13] in the context of Stokes flows. The same problem was later investigated using more modern topology optimization techniques in [12].

In the model situation discussed here, we impose a horizontal flow on Γ_{in}, namely

$$\mathbf{u} = \mathbf{u}_{\text{in}}(x_1, x_2) = (1, 0) \quad \text{on } \Gamma_{\text{in}},$$

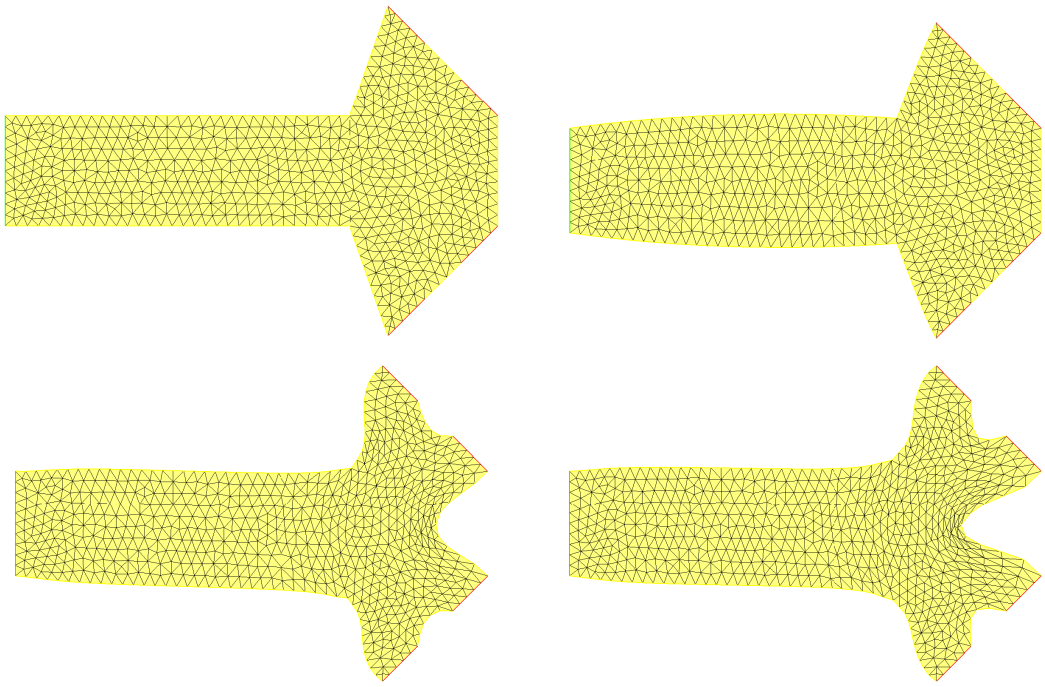


FIGURE 11. *From left to right, top to bottom, successive shapes* Ω^n *at iterations* $n = 0, 5, 240, 1000$ *in the dissipated energy minimization example in a ramified structure with volume constraint of Section 5.2.*

and no-slip boundary conditions are prescribed on the boundary of the obstacle. The convergence histories are presented on Figure 18. The resulting shape (see Figure 19, bottom) is roughly similar to those obtained in references [13, 12], having the visual aspect of a sharp rugby ball.

Finally, let us mention that, from the numerical point of view, this test-case is the hardest to run, since ℓ_0 and τ have to be chosen carefully in order to avoid the collapse of the obstacle, which would result in an invalid mesh. From a practical point of view, this choice relies on a few trials on very coarse meshes.

6. Conclusion and perspectives

In this article, we have presented a numerical framework for shape optimization in the context of fluid mechanics, consisting of well-established techniques which we have strived to present in an elementary and pedagogical way. The resulting strategy has been successfully applied to several benchmark test cases in the literature; admittedly, the techniques involved suffer from limitations, and there is a lot of room for improvements, notably:

- As we have explained in Section 3.4, the deformation of the computational mesh according to the shape gradient throughout the iterations of the optimization process is a delicate operation. Even though the heuristics described in Section 3.5 allow to overcome this difficulty in many cases, it may still happen that at some point the computational mesh becomes invalid; this is especially likely to happen when the evolving shape changes topology (for instance, two holes merge). This stake is a burning issue in the literature, and it calls for other means to represent shapes numerically than by a computational mesh, e.g. via the level set method [58, 19], or the SIMP method [46]. To keep a valid mesh, the gradient step must also be limited to small enough values, which can make convergence slow.
- The augmented Lagrangian algorithm described in Section 3.3 is well-tailored to impose one or two equality constraints on shapes. However, many natural constraints are inequality constraints, and it may be desirable to impose several of them. In such a case, it would be necessary to rely on more

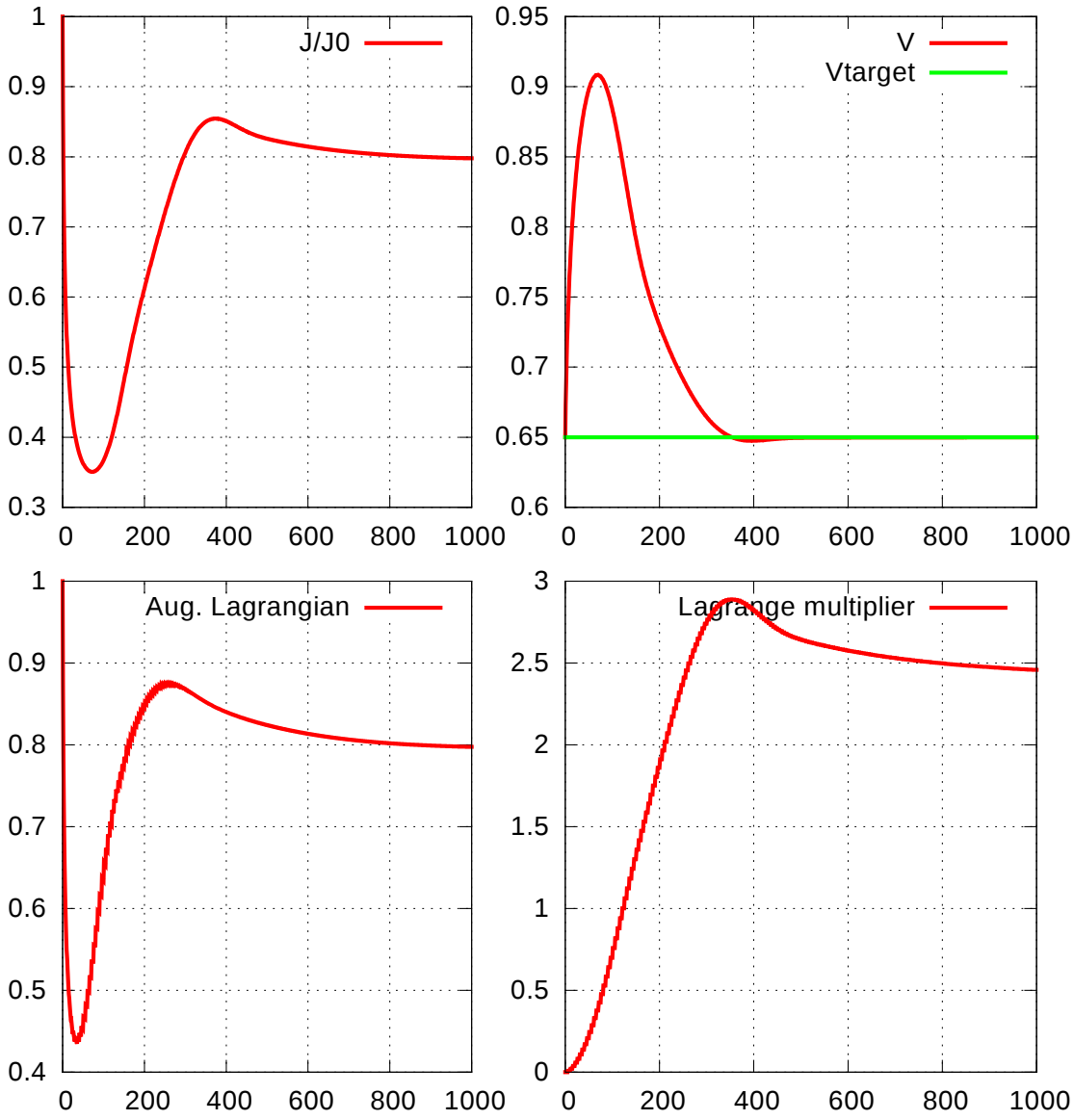


FIGURE 12. *Convergence histories of (from left to right, top to bottom)* $J(\Omega)$, $\mathrm{Vol}(\Omega)$, $\mathcal{L}(\Omega, \ell, b)$ *and* ℓ^n *in the dissipated energy minimization example in a ramified structure of Section 5.2.*

elaborated constrained optimization algorithms, such a Sequential Linear Programming (SLP); see for instance [54] about this point.

The discussed numerical examples have been implemented in the `FreeFem++` environment; the source code and a short manual are available online at

`https://github.com/flomnes/optiflow`

and it can be handled easily. We hope that it proves useful to students, researchers and industrials, as a basis for further developments and applications, such as the study of different geometric configurations, involving for instance other optimization criteria and constraints.

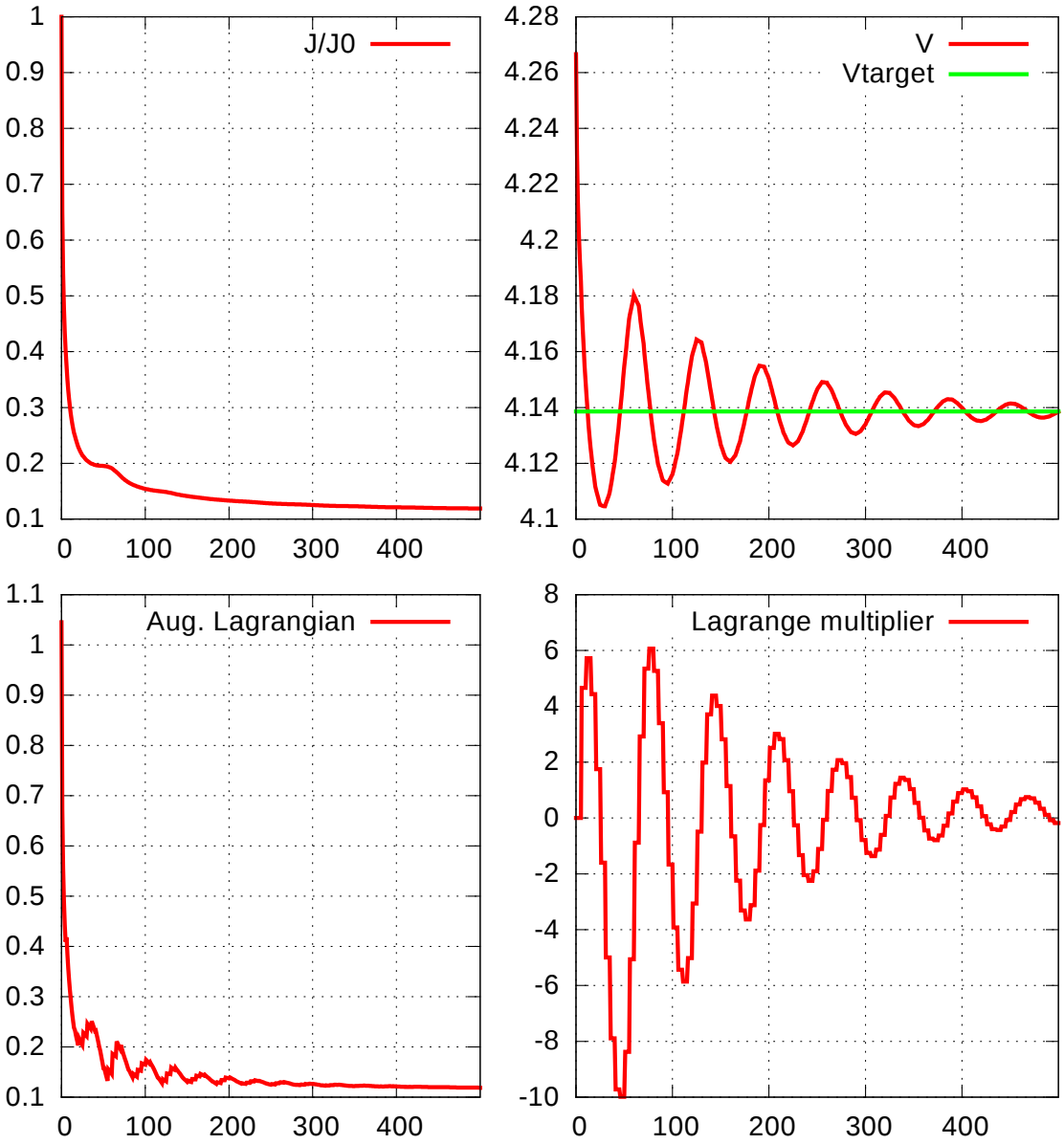


FIGURE 13. *Convergence histories of (from left to right, top to bottom)* $J(\Omega)$, $\mathrm{Per}(\Omega)$, $\mathcal{L}(\Omega, \ell, b)$ *and* ℓ^n *in dissipated energy minimization in a ramified structure example of Section 5.3*

APPENDIX A. SKETCH OF THE PROOF OF THEOREM 2.2

The differentiability of the solution $(\mathbf{u}, p)$ to the Navier-Stokes system (2.1) with respect to the domain is a technical, albeit quite classical matter, and we admit the result, referring for instance to [38] for the rigorous definition of this notion, and to [40] or [24] for this precise calculation. The derivative $(\mathbf{u}', p')$ of $(\mathbf{u}, p)$ with respect to the domain, in the direction $\boldsymbol{\theta} \in \Theta_{ad}$, is solution to the problem:

$$\text{(A.1)} \qquad \begin{cases} -\nu\Delta\mathbf{u}' + (\nabla\mathbf{u})\mathbf{u}' + (\nabla\mathbf{u}')\mathbf{u} + \nabla p' = 0 & \text{in } \Omega, \\ \mathrm{div}(\mathbf{u}') = 0 & \text{in } \Omega, \\ \sigma(\mathbf{u}', p')\mathbf{n} = 0 & \text{on } \Gamma_{\mathrm{out}}, \\ \mathbf{u}' = 0 & \text{on } \Gamma_{\mathrm{in}}, \\ \mathbf{u}' = -\left(\frac{\partial \mathbf{u}}{\partial n}\right)(\boldsymbol{\theta}.n) & \text{on } \Gamma. \end{cases}$$

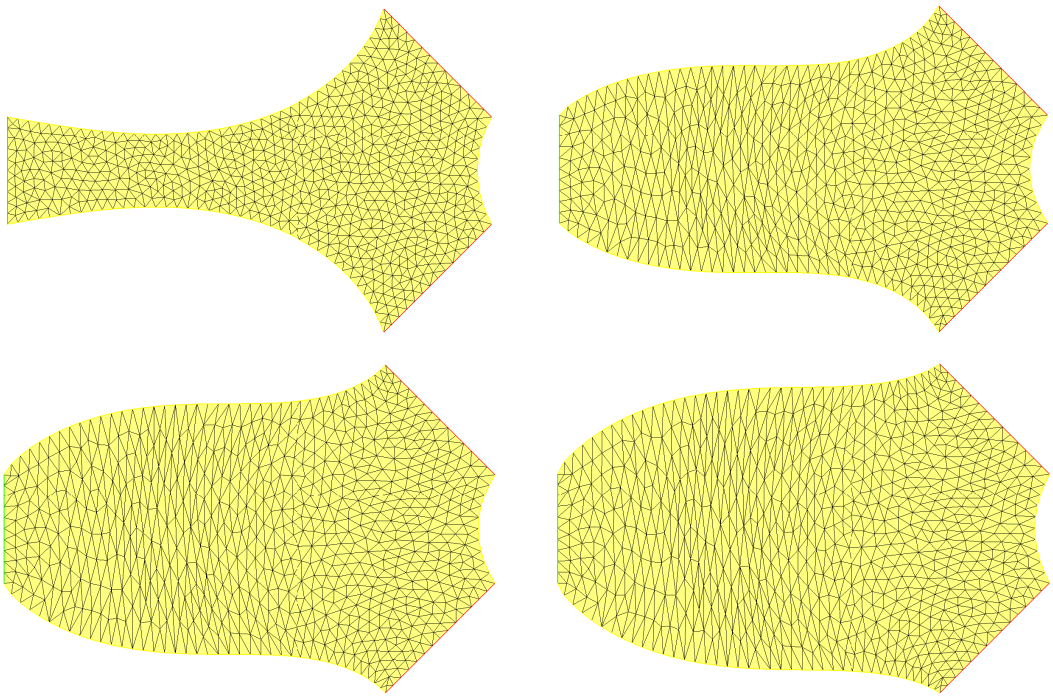


FIGURE 14. *From left to right, top to bottom, successive shapes* Ω^n *at iterations* $n = 0, 100, 250, 500$ *in the energy dissipation example in a ramified structure with perimeter constraint of Section 5.3.*

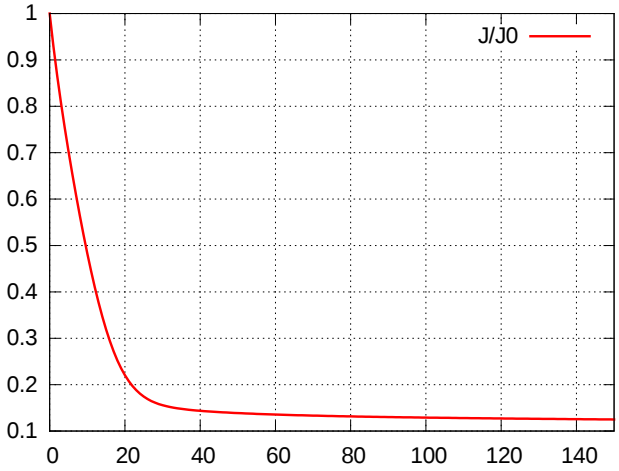


FIGURE 15. *Convergence history of* $J(\Omega)$ *in the least-square criterion minimization example of Section 5.4.*

Also, we only present the calculation of the shape derivative of the functional $D(\Omega)$ given by (2.5), the calculation being on any point easier in the case of $E(\Omega)$; see [24] if need be.

Using the chain rule from the definition (2.5) of $D(\Omega)$ yields:

$$D'(\Omega)(\boldsymbol{\theta}) = \int_{\Gamma_{\text{out}}} \mathbf{u}' \cdot (\mathbf{u} - \mathbf{u}_{\text{ref}})\, ds. \tag{A.2}$$

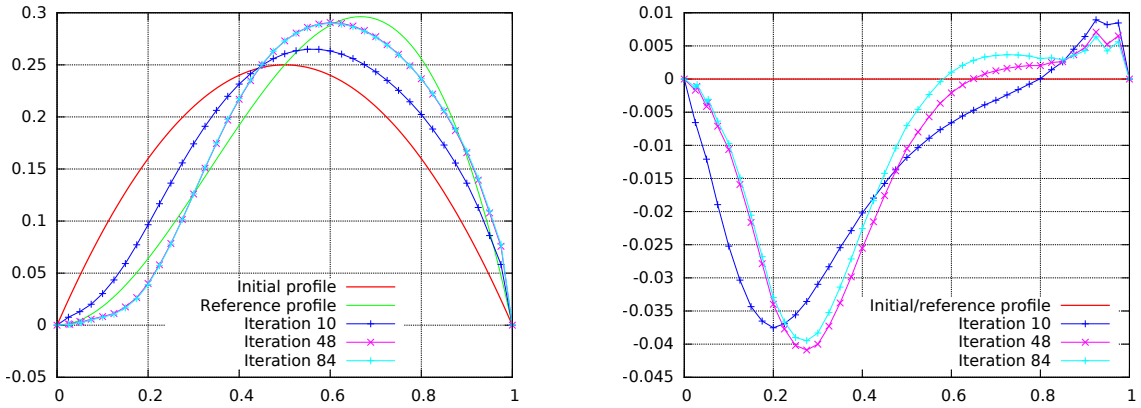


FIGURE 16. *One-dimensional profiles of (left)* u_1 *and (right)* u_2 *on* Γ_{out} *at several stages in the example of Section 5.4.*

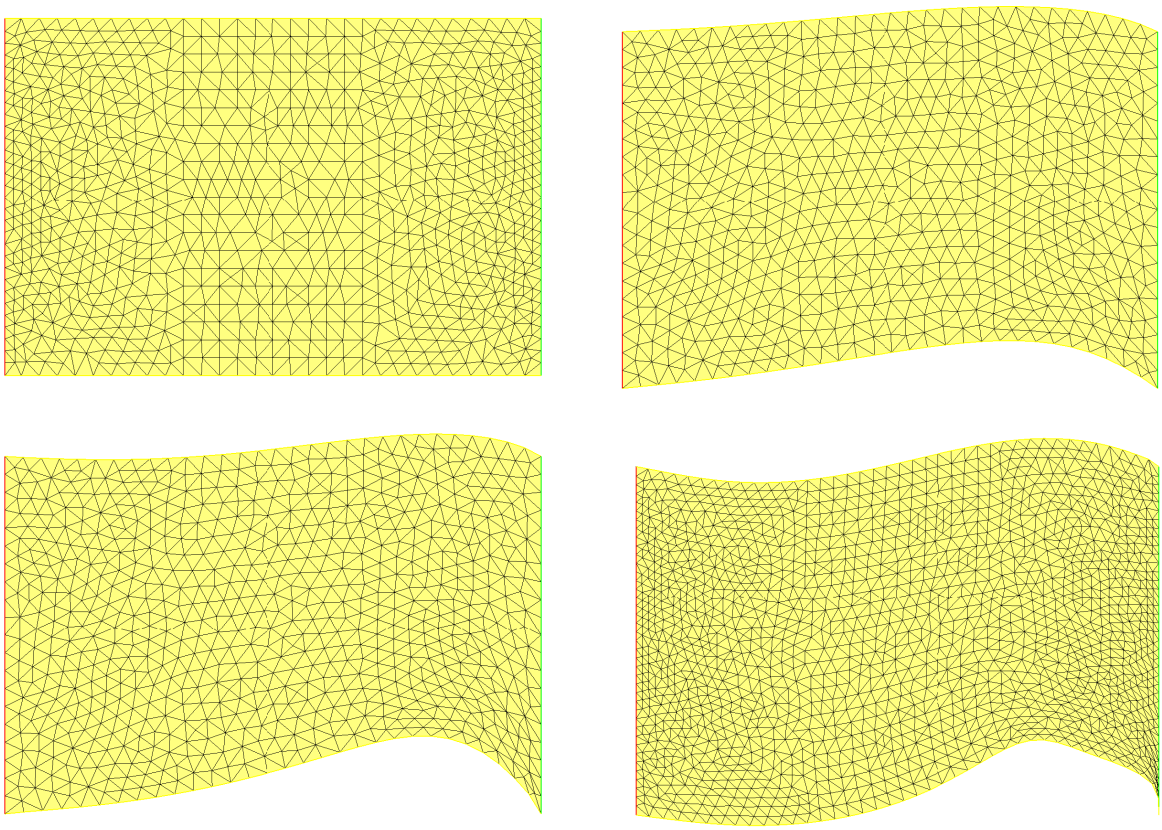


FIGURE 17. *From left to right, top to bottom, successive shapes* Ω^n *at iterations* $n = 0, 10, 48, 430$ *in the least-square criterion minimization example of Section 5.4.*

The main idea of the proof consists in using the adjoint state $(\mathbf{v}_d, q_d)$, solution to (2.12): performing several integrations by parts allows to eliminate the unknown derivatives $(\mathbf{u}', p')$ from the expression (A.2). More

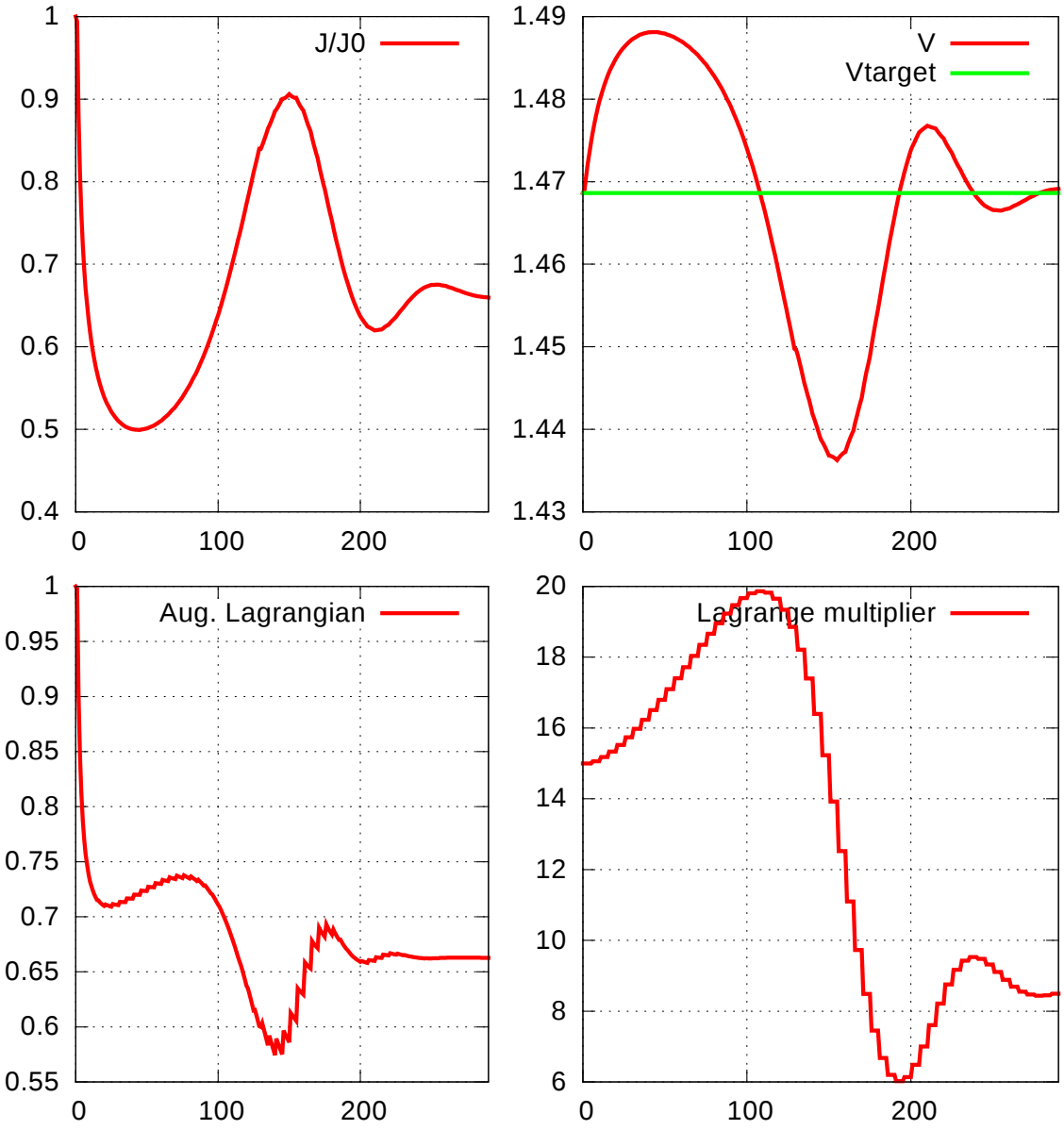


FIGURE 18. *Convergence histories of (from left to right, top to bottom)* $J(\Omega)$, $\mathrm{Vol}(\Omega)$, $\mathcal{L}(\Omega, \ell, b)$ *and* ℓ^n *in the dissipated energy minimization example of Section 5.5*

precisely, multiplying the first equation in (A.1) by $\mathbf{v}_d$ and integrating by parts yields

$$
\begin{aligned}
0 &= \int_\Omega (-\nu\Delta\mathbf{u}' + (\nabla\mathbf{u})\mathbf{u}' + (\nabla\mathbf{u}')\mathbf{u} + \nabla p') \cdot \mathbf{v}_d \, dx \\
&= \int_\Omega (2\nu \, e(\mathbf{u}') : e(\mathbf{v}_d) - \mathrm{div}(\mathbf{v}_d)p' + (\nabla\mathbf{u})\mathbf{u}' \cdot \mathbf{v}_d \\
&\quad + (\nabla\mathbf{u}')\mathbf{u} \cdot \mathbf{v}_d) \, dx - \int_{\partial\Omega} \sigma(\mathbf{u}', p')\mathbf{n} \cdot \mathbf{v}_d \, ds \\
&= \int_\Omega (2\nu \, e(\mathbf{u}') : e(\mathbf{v}_d) + (\nabla\mathbf{u})\mathbf{u}' \cdot \mathbf{v}_d + (\nabla\mathbf{u}')\mathbf{u} \cdot \mathbf{v}_d) \, dx
\end{aligned}
\tag{A.3}
$$

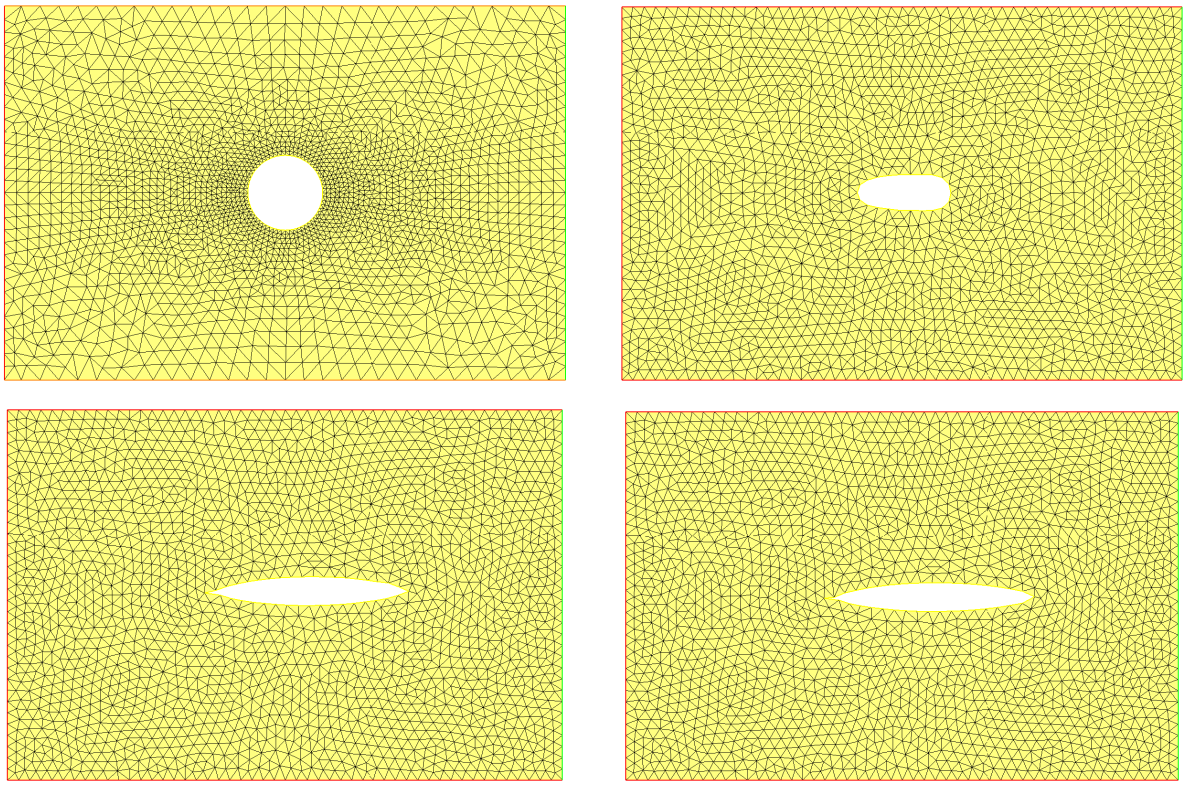


FIGURE 19. *From left to right, top to bottom, successive shapes* Ω^n *at iterations* $n = 5,\ 100,\ 650$ *in the dissipated energy minimization example of Section 5.5.*

where the boundary integral has vanished thanks to the boundary conditions satisfies by $(\mathbf{u}', p')$ and $(\mathbf{v}_d, q_d)$. Likewise, multiplying the first equation in (2.12) by $\mathbf{u}'$ and integrating by parts, we obtain:

$$\begin{aligned} 0 &= \int_\Omega \left(-\nu\Delta\mathbf{v}_d + (\nabla\mathbf{u})^T\mathbf{v} - (\nabla\mathbf{v}_d)\mathbf{u} + \nabla q_d\right)\cdot\mathbf{u}'\,dx \\ &= \int_\Omega \left(2\nu\, e(\mathbf{u}') : e(\mathbf{v}_d) + (\nabla\mathbf{u})\mathbf{u}'\cdot\mathbf{v}_d - (\nabla\mathbf{v}_d)\mathbf{u}\cdot\mathbf{u}'\right)\,dx \\ &\quad - \int_{\partial\Omega} \sigma(\mathbf{v}_d, q_d)\mathbf{n}\cdot\mathbf{u}'\,ds. \end{aligned} \tag{A.4}$$

Combining equations (A.3) and (A.4) leads to:

$$-\int_\Omega \left((\nabla\mathbf{u}')\mathbf{u}\cdot\mathbf{v}_d + (\nabla\mathbf{v}_d)\mathbf{u}\cdot\mathbf{u}'\right)dx = \int_{\partial\Omega}\sigma(\mathbf{v}_d, q_d)\mathbf{n}\,ds. \tag{A.5}$$

Now using the identity

$$\begin{aligned} \int_\Omega (\nabla\mathbf{v}_d)\cdot\mathbf{u}\cdot\mathbf{u}'\,dx &= \int_{\partial\Omega}(\mathbf{v}_d\cdot\mathbf{u}')(\mathbf{u}\cdot\mathbf{n})\,ds \\ &\quad - \int_\Omega (\nabla\mathbf{u}')\mathbf{u}\cdot\mathbf{v}_d\,dx, \end{aligned} \tag{A.6}$$

which again follows from integration by parts, Equation (A.5) rewrites:

$$\int_{\partial\Omega}\left(\sigma(\mathbf{v}_d, q_d)\mathbf{n}\cdot\mathbf{u}' + (\mathbf{u}'\cdot\mathbf{v}_d)(\mathbf{u}\cdot\mathbf{n})\right)ds = 0. \tag{A.7}$$

Eventually, taking into account the boundary conditions in the systems (2.1), (2.10) and (A.1) yields:

$$\begin{aligned} D'(\Omega)(\boldsymbol{\theta}) &= \int_{\Gamma_{\text{out}}} (\mathbf{u} - \mathbf{u}_{\text{ref}}) \cdot \mathbf{u}' \, ds \\ &= \int_{\Gamma_{\text{out}}} (\sigma(\mathbf{v}_d, q_d)\mathbf{n} + (\mathbf{u}\cdot\mathbf{n})\mathbf{v}_d) \cdot \mathbf{u}' \, ds \\ &= -\int_{\Gamma} (\sigma(\mathbf{v}_d, q_d)\mathbf{n} + (\mathbf{u}\cdot\mathbf{n})\mathbf{v}_d) \cdot \mathbf{u}' \, ds \\ &= \int_{\Gamma} (\sigma(\mathbf{v}_d, q_d)\mathbf{n} + (\mathbf{u}\cdot\mathbf{n})\mathbf{v}_d) \cdot \frac{\partial \mathbf{u}}{\partial n} \boldsymbol{\theta} \cdot n \, ds. \end{aligned} \tag{A.8}$$

We now use the boundary conditions $\mathbf{u} = 0$ and $\mathbf{v}_d = 0$ on Γ to simplify this last expression. For any tangential vector field $\boldsymbol{\tau} : \Gamma \to \mathbb{R}^d$ to Γ, they imply that $\frac{\partial \mathbf{u}}{\partial \boldsymbol{\tau}} = 0$, and so, using that $\operatorname{div}(\mathbf{u}) = 0$,

$$\frac{\partial \mathbf{u}}{\partial n} \cdot \mathbf{n} = 0 \tag{A.9}$$

the same relation holds for $\mathbf{v}_d$. Hence (A.8) rewrites:

$$D'(\Omega)(\boldsymbol{\theta}) = \int_{\Gamma} 2\nu \, e(\mathbf{v}_d)\mathbf{n} \cdot \frac{\partial \mathbf{u}}{\partial n} \boldsymbol{\theta} \cdot \mathbf{n} \, ds.$$

After a few algebraic manipulations based again on (A.9), we eventually obtain:

$$D'(\Omega)(\boldsymbol{\theta}) = \int_{\Gamma} 2\nu \, e(\mathbf{u}) : e(\mathbf{v}_d)(\boldsymbol{\theta}\cdot\mathbf{n}) \, ds, \tag{A.10}$$

which is the desired result, and terminates the proof of Theorem 2.2.

Acknowledgements. Y. Privat was partially supported by the Project 'Analysis and simulation of optimal shapes - application to lifesciences' of the Paris City Hall.

References

[1] N. Aage, T. H. Poulsen, A. Gersborg-Hansen, and O. Sigmund, *Topology optimization of large scale stokes flow problems*, Struct. Multidisc. Optim., 35 (2008), pp. 175–180.

[2] F. Abraham, M. Behr, and M. Heinkenschloss, *Shape optimization in unsteady blood flow: A numerical study of non-newtonian effects*, Computer Methods in Biomechanics and Biomedical Engineering, 8:3 (2005).

[3] V. Agoshkov, A. Quarteroni, and G. Rozza, *Shape design in aorto-coronaric bypass anastomoses using perturbation theory*, SIAM Journal on Numerical Analysis, 44 (2006), pp. 367–384.

[4] G. Allaire, *Conception optimale de structures*, vol. 58, Springer, 2007.

[5] G. Allaire, F. Jouve, and A.-M. Toader, *Structural optimization using sensitivity analysis and a level-set method*, Journal of computational physics, 194 (2004), pp. 363–393.

[6] G. Allaire and O. Pantz, *Structural optimization with freefem++*, Structural and Multidisciplinary Optimization, 32 (2006), pp. 173–181.

[7] S. Amstutz, *The topological asymptotic for the navier-stokes equations*, ESAIM: Control, Optimisation and Calculus of Variations, 11 (2005), pp. 401–425.

[8] M. Badra, F. Caubet, and M. Dambrine, *Detecting an obstacle immersed in a fluid by shape optimization methods*, unpublished, (2011).

[9] T. J. Baker, *Mesh movement and metamorphosis*, Eng. Comput., 18 (2002), pp. 188–198.

[10] M. P. Bendsoe and O. Sigmund, *Topology optimization: theory, methods, and applications*, Springer Science & Business Media, 2013.

[11] M. Bergounioux and Y. Privat, *Shape optimization with Stokes constraints over the set of axisymmetric domains*, SIAM J. Control Optim., 51 (2013), pp. 599–628.

[12] T. Borrvall and J. Petersson, *Topology optimization of fluids in stokes flow*, Int. J. Numer. Meth. Fluids, 41 (2003), pp. 77–107.

[13] J.-M. Bourot, *On the numerical computation of the optimum profile in stokes flow*, Journal of Fluid Mechanics, 65 (1974), pp. 513–515.

[14] C.-H. Bruneau, F. Chantalat, A. Iollo, B. Jordi, and I. Mortazavi, *Modelling and shape optimization of an actuator*, Structural and Multidisciplinary Optimization, 48 (2013), pp. 1143–1151.

[15] D. Bucur and G. Buttazzo, *Variational methods in some shape optimization problems*, Appunti dei Corsi Tenuti da Docenti della Scuola. [Notes of Courses Given by Teachers at the School], Scuola Normale Superiore, Pisa, 2002.

[16] M. Burger, *A framework for the construction of level set methods for shape optimization and reconstruction*, Interfaces and Free boundaries, 5 (2003), pp. 301–329.

[17] H. W. Carlson and W. D. Middleton, *A numerical method for the design of camber surfaces of supersonic wingswith arbitrary planforms*, NASA Technical report, (1964).

[18] V. Challis and J. Guest, *Level set topology optimization of fluids in stokes flow*, Int. J. Numer. Meth. Engng, 79 (2009), pp. 1284–1308.

[19] V. J. Challis and J. K. Guest, *Level set topology optimization of fluids in stokes flow*, International journal for numerical methods in engineering, 79 (2009), pp. 1284–1308.

[20] J.-H. Choi, K.-Y. Kim, and D.-S. Chung, *Numerical optimization for design of an automotive cooling fan*, tech. rep., SAE Technical Paper, 1997.

[21] P. Ciarlet, *The Finite Element Method for Elliptic Problems*, Society for Industrial and Applied Mathematics, 2002.

[22] H. Çlabuk and V. Modi, *Optimum plane diffusers in laminar flow*, Journal of Fluid Mechanics, 237 (1992), pp. 373–393.

[23] F. De Gournay, *Velocity extension for the level-set method and multiple eigenvalues in shape optimization*, SIAM journal on control and optimization, 45 (2006), pp. 343–367.

[24] X. D. de La Sablonière, B. Mauroy, and Y. Privat, *Shape minimization of the dissipated energy in dyadic trees*, Discrete Contin. Dyn. Syst. Ser. B, 16 (2011), pp. 767–799.

[25] C. Dobrzynski and P. Frey, *Anisotropic delaunay mesh adaptation for unsteady simulations*, Proc. 17th Int. Meshing Roundtable, (2008).

[26] G. Dogan, P. Morin, R. H. Nochetto, and M. Verani, *Discrete gradient flows for shape optimization and applications*, Computer methods in applied mechanics and engineering, 196 (2007), pp. 3898–3914.

[27] J. Donea and A. Huerta, *Finite element methods for flow problems*, John Wiley & Sons, 2003.

[28] X.-B. Duan, Y.-C. Ma, and R. Zhang, *Shape-topology optimization for navier-stokes problem using variational level set method*, Journal of Computational and Applied Mathematics, 222 (2008), pp. 487–499.

[29] A. Ern and J.-L. Guermond, *Theory and practice of finite elements*, vol. 159, Springer Science & Business Media, 2013.

[30] A. Evgrafov, *Topology optimization of slightly compressible fluids*, ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik, 86 (2006), pp. 46–62.

[31] P. Frey and P.-L. George, *Mesh generation, application to Finite Elements*, Wiley & Sons, 2008.

[32] H. Garcke, C. Hecht, M. Hinze, and C. Kahle, *Numerical approximation of phase field based shape and topology optimization for fluids*, SIAM Journal on Scientific Computing, 37 (2015), pp. A1846–A1871.

[33] A. Gersborg-Hansen, O. Sigmund, and R. B. Haber, *Topology optimization of channel flow problems*, Structural and Multidisciplinary Optimization, 30 (2005), pp. 181–192.

[34] V. Girault and P.-A. Raviart, *Finite Element methods for Navier-Stokes Equations*, Springer Verlag, 1986.

[35] J. Guest and J. Prévost, *Topology optimization of creeping fluid flows using a darcystokes finite element*, Int. J. Numer. Meth. Engng, 66 (2006), pp. 461–484.

[36] M. D. Gunzburger, *Perspectives in flow control and optimization*, vol. 5, Siam, 2003.

[37] F. Hecht, O. Pironneau, A. Le Hyaric, and K. Ohtsuka, *Freefem++ manual*, 2005.

[38] A. Henrot and M. Pierre, *Variation et optimisation de formes*, vol. 48, Springer-Verlag Berlin Heidelberg, 2005.

[39] A. Henrot and Y. Privat, *Une conduite cylindrique n'est pas optimale pour minimiser l'énergie dissipée par un fluide*, C. R. Math. Acad. Sci. Paris, 346 (2008), pp. 1057–1061.

[40] ———, *What is the optimal shape of a pipe?*, Arch. Ration. Mech. Anal., 196 (2010), pp. 281–302.

[41] R. M. Hicks and P. A. Henne, *Wing design by numerical optimization*, Journal of Aircraft, 15 (1978), pp. 407–412.

[42] R. M. Hicks, E. M. Murman, and G. N. Vanderplaats, *An assessment of airfoil design by numerical optimization*, (1974).

[43] A. Jameson, *Aerodynamic design via control theory*, Journal of scientific computing, 3 (1988), pp. 233–260.

[44] S. Kreissl and K. Maute, *Levelset based fluid topology optimization using the extended finite element method*, Structural and Multidisciplinary Optimization, 46 (2012), pp. 311–326.

[45] S. Kreissl, G. Pingen, and K. Maute, *An explicit level set approach for generalized shape optimization of fluids with the lattice boltzmann method*, International Journal for Numerical Methods in Fluids, 65 (2011), pp. 496–519.

[46] ———, *Topology optimization for unsteady flow*, International Journal for Numerical Methods in Engineering, 87 (2011), pp. 1229–1253.

[47] J. L. Lions, *Optimal control of systems governed by partial differential equations*, vol. 170, Springer Verlag, 1971.

[48] A. Litman, D. Lesselier, and F. Santosa, *Reconstruction of a two-dimensional binary obstacle by controlled evolution of a level-set*, Inverse Problems, 14 (1998), p. 685.

[49] B. Mohammadi and O. Pironneau, *Shape optimization in fluid mechanics*, Annu. Rev. Fluid Mech., 36 (2004), pp. 255–279.

[50] B. Mohammadi and O. Pironneau, *Applied shape optimization for fluids*, Oxford University Press, 2010.

[51] B. Munson, A. Rothmayer, T. Okiishi, and W. Huebsch, *Fundamentals of Fluid Mechanics*, John Wiley & Sons, Inc., 7th edition ed., 2013.

[52] B. R. Munson, T. H. Okiishi, W. W. Huebsch, and A. P. Rothmayer, *Fundamentals of fluid mechanics, 7th edition*, Wiley & Sons, 2012.

[53] F. Murat and J. Simon, *Sur le contrôle par un domaine géométrique*, Technical report RR-76005, (1976).

[54] J. Nocedal and S. Wright, *Numerical Optimization*, Springer, 2006.

[55] A. A. Novotny and J. Sokołowski, *Topological derivatives in shape optimization*, Springer Science & Business Media, 2012.

[56] S. Osher and J. A. Sethian, *Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations*, Journal of computational physics, 79 (1988), pp. 12–49.
[57] G. Pingen, A. Evgrafov, and K. Maute, *Topology optimization of flow domains using the lattice boltzmann method*, Structural and Multidisciplinary Optimization, 34 (2007), pp. 507–524.
[58] G. Pingen, M. Waidmann, A. Evgrafov, and K. Maute, *A parametric level-set approach for topology optimization of flow domains*, Structural and Multidisciplinary Optimization, 41 (2010), pp. 117–131.
[59] O. Pironneau, *On optimum profiles in stokes flow*, Journal of Fluid Mechanics, 59 (1973), pp. 117–128.
[60] ———, *On optimum design in fluid mechanics*, Journal of Fluid Mechanics, 64 (1974), pp. 97–110.
[61] ———, *Optimal shape design for elliptic systems*, Springer Science & Business Media, 2012.
[62] J. A. Sethian and A. Wiegmann, *Structural boundary design via level set and immersed interface methods*, Journal of computational physics, 163 (2000), pp. 489–528.
[63] O. Sigmund, *A 99 line topology optimization code written in matlab*, Structural and multidisciplinary optimization, 21 (2001), pp. 120–127.
[64] J. Sokołowski and J.-P. Zolésio, *Introduction to shape optimization*, Springer Series in Computational Mathematics, Springer-Verlag, Berlin, 1992.
[65] R. Temam, *Navier-Stokes Equation: Theory and Numerical Analysis*, North Holland, 1977.
[66] M. Y. Wang, X. Wang, and D. Guo, *A level set method for structural topology optimization*, Computer methods in applied mechanics and engineering, 192 (2003), pp. 227–246.
[67] S. Zhou and Q. Li, *A variational level set method for the topology optimization of steady-state navier-stokes flow*, J. Comput. Phys., 227 (2008), pp. 10178–10195.