POLITECNICO DI MILANO

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA MATEMATICA

Optimization of unsteady PDE systems using a multi-objective descent method

Relatori:
Dr. Régis Duvigneau
Dr. Nicola Parolini

Tesi di Laurea di:
Camilla Fiorini
Matr. 798737

Anno Accademico 2013-2014

Ever tried,
Ever failed.
No matter.
Try again,
Fail again,
Fail better.

— Samuel B. Beckett

Abstract

The aim of this work is to develop an approach to solve minimization problems in which the functional that has to be minimized is time dependent. In the literature, the most common approach when dealing with unsteady problems, is to consider time-average quantities: however, this approach is limited since the dynamical nature of the state is neglected. These considerations lead to the idea, introduced for the first time in this work, of building a set of cost functionals by evaluating a single cost functional at different sampling times: in this way, we reduce the unsteady optimization problem to a multi-objective optimization one, that will be solved using the Multiple Gradient Descent Algorithm. Moreover, we propose new hybrid approach, which will be referred to as *windows* approach: in this case, the set of cost functionals is built by doing a time-average operation over multiple intervals.

The following work has been developed during a five-months internship within the OPALE project team at INRIA Méditerranée (*Institut National de Recherche en Informatique et en Automatique*) - Sophia Antipolis, France.

Sommario

Lo scopo di questo lavoro è quello di sviluppare un approccio per risolvere problemi di minimizzazione in cui il funzionale da minimizzare è tempo-dipendente.
L'approccio più comune in letteratura in questi casi è quello di considerare quantità mediate in tempo: tuttavia, questo approccio risulta limitato in quanto la natura dinamica del fenomeno è trascurata. Queste considerazioni hanno portato all'idea, proposta per la prima volta in questo lavoro, di costruire un insieme di funzionali costo valutando in alcuni istanti temporali il funzionale costo tempo-dipendente: in questo modo il problema di ottimizzazione non stazionaria è stato ridotto a un problema di ottimizzazione multi-obiettivo, per risolvere il quale sarà utilizzato MGDA (Multiple Gradient Descent Algorithm), sviluppato dal team. Inoltre, viene qui proposto anche un nuovo approccio ibrido, che chiameremo approccio *windows*: esso consiste nel costruire l'insieme di funzionali costo facendo un'operazione di media su diversi intervalli temporali.

Il seguente lavoro è stato sviluppato durante uno stage di cinque mesi nel team OPALE presso INRIA Méditerranée (*Institut National de Recherche en Informatique et en Automatique*) - Sophia Antipolis, Francia.

Contents

Introduction

Over the years, optimization has gained an increasing interest in the scientific world, due to the fact that it can be highly valuable in a wide range of applications.

The simplest optimization problem consists in individuating the best point, according to a certain criterion, in a set of admissible points: this can easily be reconduced to finding the minimum (or the maximum) of a given function. However, this model is often too simple to be used in real applications, because of the presence of many different quantities of different nature that interact among themselves, sometimes in a competitive way, sometimes in a synergistic way: this is the framework of *multi-objective optimization*.

Multi-objective optimization problems arise from many different fields: economics is the one in which, for the first time, has been introduced the concept of *optimum* in a multi-objective framework as we intend it in this work; the idea was introduced by Pareto in [Par96]. Other fields in which multi-objective optimization can be applied are, for instance: game theory problems (for the relationship with the Pareto-concepts, see [Rao86]); operations research, that often deals with problems in which there are many competitive quantities to be minimized; finally, an important field is engineering design optimization (see, for instance, [EMDD08]).

In this work, we propose an algorithm to identify a region of Pareto-optimal points. The starting point is the Multiple Gradient Descent Algorithm, that is a generalization of the classical Steepest-Descent Method. It has been introduced in its first version, MGDA I, in [Dés09]; some first improvements to the original version gave birth to MGDA II (see [Dés12a]). Finally, a third version, MGDA III, has been proposed in [Dés12b] from a theoretical point of view. In this work, we implemented MGDA III, by developing a C++ code, and we tested it on some simple cases: the testing process highlighted some limitations of the algorithm, therefore we proposed further modifications, generating in this way MGDA III b.

The main topic of this work is to use MGDA to optimize systems governed by unsteady partial differential equations. More precisely, we considered a model problem defined by a one-dimensional nonlinear advection-diffusion equation with an oscillatory boundary condition, for which a periodic source term is added locally to reduce the variations of the solution: this could mimic a fluid system including an active flow control device, for instance, and was inspired by [DV06], in which, however, single-objective optimization techniques were used. The objective of this work is to apply MGDA to optimize the actuation parameters (frequency and amplitude of the source term), in order to minimize a cost functional evaluated at a set of times.

In the literature, the most common approach when dealing with unsteady problems, is to consider time-average quantities (for instance, in [DV06] and [BTY$^+$99] the cost functional considered is the time-averaged drag force): however, this approach is limited since the dynamical nature of the state is neglected. For this reason, in this work we propose a new approach: the main idea is to built a set of cost functionals by evaluating a single cost functional at different sampling times, and, consequently, applying multi-objective optimization techniques to this set. On one hand, this method is computationally more expensive than a single-objective optimization procedure; on the other hand, it guarantees that the cost functional decrease at every considered instant. Finally, an hybrid approach is proposed, which will be referred to as *windows* approach: in this case, the set of cost functionals is built by doing a time-average operation over multiple intervals.

The work is organized as follows:

Chapter 1: in the first part of the chapter, starting from the definition of a single-objective optimization problem, we define rigorous formulation of a multi-objective optimization one, we recall the Pareto-concepts such as Pareto-stationarity and Pareto-optimality and we analyse the relation between them. Finally we briefly illustrate the classical Steepest-Descent Method in the single-objective optimization framework. In the second part of the chapter, we recall some basic concepts of differential calculus in Banach spaces and we illustrate a general optimization problem governed by PDEs. Finally, we explain how all these concepts will be used in this work.

Chapter 2: initially, we briefly explain the first two versions of MGDA; then follows an accurate analysis of MGDA III: we explain the algorithm in details with the help of some flow charts, we introduce and prove some properties of it, and we illustrate the simple test cases and the problems that they put in light. Finally, we propose some modification and we illustrate the new resulting algorithm.

Chapter 3: first, we illustrate the general solving procedure that has been applied in order to obtain the numerical results: in particular the *Continuous Sensitivity Equation* method (CSE) is introduced, which has been used instead of the more classical *adjoint* approach; then, the specific PDEs considered are introduced, both linear and nonlinear, along with the numerical schemes adopted to solve them: in particular, two schemes have been implemented, a first order and a second order one. Finally, the first numerical results are shown: the order of convergence of the schemes is verified in some simple cases whose exact solution is known, and the CSE method is validated. Moreover, in this occasion, the source term and the boundary condition used in most of the test cases are introduced.

Chapter 4: here, the main numerical results are presented, i.e. the results obtained by applying MGDA to the PDEs systems described in chapter 3. First MGDA III b is applied to the linear case: these results show how this problem is too simple and, therefore, not interesting. For this reason, we focused on the nonlinear PDEs: to them, we applied MGDA III and III b. The results obtained underlined how the modifications introduced allow a better identification of the Pareto-optimal zone. Then, we applied

MGDA III b in a space of higher dimension (i.e. we increased the number of parameters). Finally, we tested the *windows* approach.

Appendix A: in this appendix the C++ code that has been developed is presented; it was used to obtain all the numerical results presented in this work. The main parts of the code are reported and illustrated in details.

1 Multi-objective optimization for PDEs constrained problems

In this chapter, first we recall some basic concepts, like Pareto-stationarity and Pareto-optimality, in order to formulate rigorously a multi-objective optimization problem; then we recall some basic concepts of differential calculus in Banach spaces to illustrate a general optimization problem governed by PDEs.

1.1 From optimization to multi-objective optimization

A generic single-objective optimization problem consists in finding the parameters that minimize (or maximize) a given criterion, in a set of admissible parameters. Formally, one can write:

$$\min_{\mathbf{p}\in\mathcal{P}} J(\mathbf{p}) \tag{1.1}$$

where J is the criterion, also called cost functional, $\mathbf{p}$ is the vector of parameters and $\mathcal{P}$ is the set of admissible parameters. Since every maximization problem can easily be reconduced to a minimization one, we will focus of the latter. Under certain conditions (for instance, $\mathcal{P}$ convex closed and J convex) one can guarantee existence and uniqueness of the solution of (1.1), that we will denote as $\mathbf{p}^{\text{opt}}$. For more details about the well posedness of this kind of problems, see [NW06], [Ber82]. This format can describe simple problems, in which only one quantity has to be minimized. However, it is often difficult to reduce a real situation to a problem of the kind (1.1), because of the presence of many quantities of different natures. This can lead to a model in which there is not only one, but a set of criteria $\{J_i\}_{i=0}^{n}$ to minimize simultaneously, and this is the framework of *multi-objective optimization*. In order to solve this kind of problems, we need to formalize them, since (1.1) does not make sense for a set of cost functionals. In fact, in this context it is not even clear what means for a configuration $\mathbf{p}^0$ to be "better" than another one $\mathbf{p}^1$. First, we give some important definitions that should clarify these concepts.

1.1.1 Basic concepts of multi-objective optimization

The first basic concept regards the relation between two points.

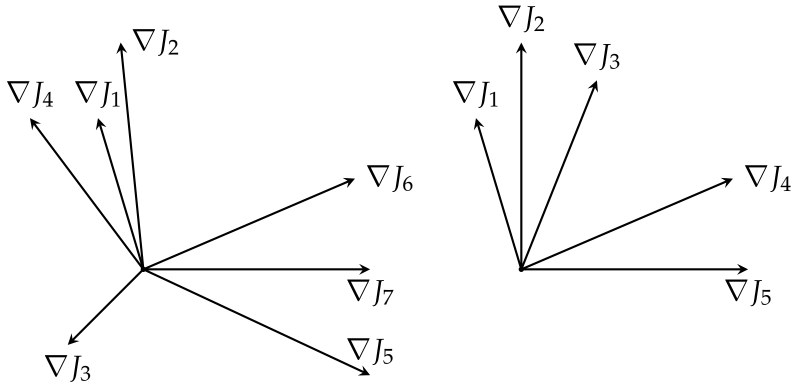


Figure 1.1: On the left a Pareto-stationary point, on the right a non Pareto-stationary point.

Definition 1 (*Dominance*)**.** The design point $\mathbf{p}^0$ is said to dominate $\mathbf{p}^1$ if:

$$J_i(\mathbf{p}^0) \leq J_i(\mathbf{p}^1) \ \ \forall i \quad \text{and} \quad \exists\, k \ : \ J_k(\mathbf{p}^0) < J_k(\mathbf{p}^1).$$

From now on, we consider regular cost functionals such as $J_i \in C^1(\mathcal{P})\ \forall i$ so that it makes sense to write $\nabla J_i(\mathbf{p})$, where we denoted with ∇ the gradient with respect to the parameters $\mathbf{p}$. Now, we want to introduce the concepts of stationarity and optimality in the multi-objective optimization framework.

Definition 2 (*Pareto-stationarity*)**.** The design point $\mathbf{p}^0$ is Pareto-stationary if there exist a convex combination of the gradients $\nabla J_i(\mathbf{p}^0)$ that is zero, i.e.:

$$\exists\, \alpha = \{\alpha_i\}_{i=0}^{n},\ \ \alpha_i \geq 0\ \forall i,\ \ \sum_{i=0}^{n} \alpha_i = 1 \ : \ \sum_{i=0}^{n} \alpha_i \nabla J_i(\mathbf{p}) = 0.$$

In two or three dimension it can be useful to visualize the gradients of the cost functionals as vectors, all applied in the same point. In fact this is a quick and easy way to understand whether or not a point is Pareto-stationary (see Figure 1.1).

Definition 3 (*Pareto-optimality*)**.** The design point $\overline{\mathbf{p}}$ is Pareto-optimal if it is impossible to reduce the value of any cost functional without increasing at least one of the others.

This means that $\overline{\mathbf{p}}$ is Pareto-optimal if and only if $\nexists \mathbf{p} \in \mathcal{P}$ that dominates it. The relation between Pareto-optimality and stationarity is explained by the following theorem:

Theorem 1. $\mathbf{p}^0$ *Pareto-optimal* $\Rightarrow$ $\mathbf{p}^0$ *Pareto-stationary.*

Proof. Let $\mathbf{g}_i = \nabla J_i(\mathbf{p}^0)$, let r be the rank of the set $\{\mathbf{g}_i\}_{i=1}^{n}$ and let N be the dimension of the space ($\mathbf{g}_i \in \mathbb{R}^N$). This means that $r \leq \min\{N, n\}$, $n \geq 2$ since this is multi-objective optimization and $N \geq 1$.

If $r = 0$ the result is trivial.

If $r = 1$, there exist a unit vector $\mathbf{u}$ and coefficients β_i such that $\mathbf{g}_i = \beta_i \mathbf{u}$. Let us consider a small increment $\varepsilon > 0$ in the direction $\mathbf{u}$, the Taylor expansion will be:

$$J_i(\mathbf{p}^0 + \varepsilon \mathbf{u}) = J_i(\mathbf{p}^0) + \varepsilon \left(\nabla J_i(\mathbf{p}^0), \mathbf{u} \right) + \mathcal{O}(\varepsilon^2) = J_i(\mathbf{p}^0) + \varepsilon \beta_i + \mathcal{O}(\varepsilon^2)$$

$$\Rightarrow J_i(\mathbf{p}^0 + \varepsilon \mathbf{u}) - J_i(\mathbf{p}^0) = \varepsilon \beta_i + \mathcal{O}(\varepsilon^2)$$

If $\beta_i \leq 0 \ \forall\ i$, $J_i(\mathbf{p}^0 + \varepsilon \mathbf{u}) \leq J_i(\mathbf{p}^0) \ \forall\ i$ but this is impossible since $\mathbf{p}^0$ is Pareto-optimal. For the same reason it can not be $\beta_i \geq 0 \ \forall i$. This means that $\exists\ j, k$ such that $\beta_j \beta_k < 0$. Let us assume that $\beta_k < 0$ and $\beta_j > 0$. The following coefficients satisfy the Definition 2:

$$\alpha_i = \begin{cases} \frac{-\beta_k}{\beta_j - \beta_k} & i = k \\ \frac{\beta_j}{\beta_j - \beta_k} & i = j \\ 0 & \text{otherwise} \end{cases}$$

then $\mathbf{p}^0$ is Pareto-stationary.

If $r \geq 2$ and the gradients are linearly dependent (i.e $r < n$, with $r \leq N$), up to a permutation of the indexes, $\exists\ \beta_i$ such that:

$$\mathbf{g}_{r+1} = \sum_{i=1}^{r} \beta_i \mathbf{g}_i \quad \Rightarrow \quad \mathbf{g}_{r+1} + \sum_{i=1}^{r} \mu_i \mathbf{g}_i = \mathbf{0} \tag{1.2}$$

where $\mu_i = -\beta_i$. We want to prove that $\mu_i \geq 0 \ \forall\ i$. Let us assume, instead, that $\mu_r < 0$ and define the following space:

$$V := span\{\mathbf{g}_i\}_{i=1}^{r-1}.$$

Let us now consider a generic vector $\mathbf{v} \in V^{\perp}$, $\mathbf{v} \neq \mathbf{0}$. It is possible, since $V^{\perp} \neq \{\mathbf{0}\}$, in fact:

$$dimV \leq r - 1 \leq N - 1 \Rightarrow dimV^{\perp} = N - dimV \geq 1.$$

By doing the scalar product of $\mathbf{v}$ with (1.2) and using the orthogonality, one obtains:

$$0 = \left(\mathbf{v}, \mathbf{g}_{r+1} + \sum_{i=1}^{r} \mu_i \mathbf{g}_i \right) = (\mathbf{v}, \mathbf{g}_{r+1}) + \mu_r (\mathbf{v}, \mathbf{g}_r),$$

then follow that:

$$\left(\mathbf{v}, \nabla J_{r+1}(\mathbf{p}^0) \right) = -\mu_r \left(\mathbf{v}, \nabla J_r(\mathbf{p}^0) \right),$$

and since $\mu_r < 0$, $\left(\mathbf{v}, \nabla J_{r+1}(\mathbf{p}^0) \right)$ and $\left(\mathbf{v}, \nabla J_r(\mathbf{p}^0) \right)$ have the same sign. They cannot be both zeros, in fact if they were, they would belong to V, and the whole family $\{\mathbf{g}_i\}_{i=1}^{n}$ would belong to V, but this is not possible, since:

$$dimV \leq r - 1 < r = rank\{\mathbf{g}_i\}_{i=1}^{n}.$$

Therefore, $\exists\ \mathbf{v} \in V^{\perp}$, such that $\left(\mathbf{v}, \nabla J_{r+1}(\mathbf{p}^0)\right) = -\mu_r \left(\mathbf{v}, \nabla J_r(\mathbf{p}^0)\right)$ is not trivial. Let us say that $\left(\mathbf{v}, \nabla J_{r+1}(\mathbf{p}^0)\right) > 0$, repeating the argument used for $r = 1$ one can show that $-\mathbf{v}$ is a descent direction for both J_r and J_{r+1}, whereas leaves the other functionals unchanged due to the orthogonality, but this is a contradiction with the hypothesis of Pareto-optimality. Therefore $\mu_k \geq 0\ \forall\ k$ and, up to a normalization constant, they are the coefficients α_k of the definition of Pareto-stationarity.

Finally, this leave us with the case of linearly independent gradients, i.e. $r = n \leq N$. However, this is incompatible with the hypothesis of Pareto optimality. In fact, one can rewrite the multi-objective optimization problem as follows:

$$\begin{aligned} &\min_{\mathbf{p}\in\mathcal{P}} J_i(\mathbf{p}) \quad \text{subject to} \\ &J_k(\mathbf{p}) \leq J_k(\mathbf{p}^0) \quad \forall\, k \neq i. \end{aligned} \tag{1.3}$$

The Lagrangian formulation of the problem (1.3) is:

$$L_i(\mathbf{p}, \boldsymbol{\lambda}) = J_i(\mathbf{p}) + \sum_{k\neq i} \lambda_k \left(J_k(\mathbf{p}) - J_k(\mathbf{p}^0)\right),$$

where $\lambda_k \geq 0\ \forall k \neq i$. Since $\mathbf{p}^0$ is Pareto optimal, it stands:

$$\mathbf{0} = \nabla_{\mathbf{p}} L_i|_{(\mathbf{p}^0, \boldsymbol{\lambda})} = \nabla_{\mathbf{p}} J_i(\mathbf{p}^0) + \sum_{k\neq i} \lambda_k \nabla_{\mathbf{p}} J_k(\mathbf{p}^0),$$

but this is impossible since the gradients are linearly independent. In conclusion, it cannot be $r = n$, and we always fall in one of the cases previously examinated, which imply the Pareto-stationarity. □

Unless the problem is trivial and there exist a point $\overline{\mathbf{p}}$ that minimizes simultaneously all the cost functionals, the Pareto-optimal design point is not unique. Therefore, we need to introduce the concept of *Pareto-front*.

Definition 4 (*Pareto-front*). A Pareto-front is a subset of design points $\mathcal{F} \subset \mathcal{P}$ such that $\forall \mathbf{p}, \mathbf{q} \in \mathcal{F}$, $\mathbf{p}$ does not dominate $\mathbf{q}$.

A Pareto-front represents a compromise between the criteria. An example of Pareto-front in a case with two cost functionals is given in Figure 1.2: each point corresponds to a different value of $\mathbf{p}$.

Given these definitions, the multi-objective optimization problem will be the following:

$$\begin{aligned} &\text{given } \mathcal{P},\ \mathbf{p}^{\text{start}} \in \mathcal{P} \text{ not Pareto-stationary, and } \{J_i\}_{i=0}^n \\ &\text{find } \mathbf{p} \in \mathcal{P}\ :\ \mathbf{p} \text{ is Pareto-optimal.} \end{aligned} \tag{1.4}$$

To solve the problem (1.4), different strategies are possible, for instance one can build a new cost functional:

$$J = \sum_{i=0}^{n} \alpha_i J_i \quad \alpha_i > 0\ \forall i,$$

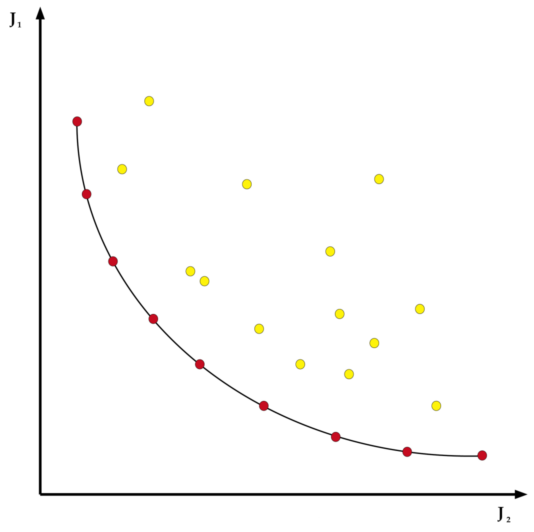


Figure 1.2: Example of Pareto-Front

and apply single-objective optimization techniques to this new J. However, this approach presents heavy limitations due to the arbitrariness of the weights. See [Gia13] for more details. In this work we will consider an algorithm that is a generalization of the steepest descent method: MGDA, that stands for Multiple Gradient Descent Algorithm, introduced in its first version in [Dés09] and then developed in [Dés12a] and [Dés12b]. In particular we will focus on the third version of MGDA, of which we will propose some improvements, generating in this way a new version of the algorithm: MGDA-III b.

1.1.2 Steepest Descent Method

In this section, we recall briefly the steepest descent method, also known as gradient method. We will not focus on the well posedness of the problem, neither on the convergence of the algorithm, since this is not the subject of this work. See [QSS00] for more details.

Let $F(\mathbf{x}) : \Omega \to \mathbb{R}$ be a function that is differentiable in Ω and let us consider the following problem:

$$\min_{\mathbf{x}\in\Omega} F(\mathbf{x}).$$

One can observe that:

$$\exists \rho > 0 : \; F(\mathbf{a}) \geq F(\mathbf{a} - \rho\nabla F(\mathbf{a})) \;\; \forall \mathbf{a} \in \Omega.$$

Therefore, given an initial guess $\mathbf{x}_0 \in \Omega$, it is possible to build a sequence $\{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots\}$ in the following way:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \rho\nabla J(\mathbf{x}_n)$$

such that $F(\mathbf{x}_0) \geq F(\mathbf{x}_1) \geq F(\mathbf{x}_2) \geq \dots$. Under certain conditions on F, Ω and $\mathbf{x}_0$ (for instance, Ω convex closed, $F \in C(\overline{\Omega})$ and $\mathbf{x}_0$ "close enough" to the minimum), it will be:

$$\lim_{n\to+\infty} \mathbf{x}_n = \overline{\mathbf{x}} = \operatorname*{argmin}_{\mathbf{x}\in\Omega} F(\mathbf{x}).$$

1.2 Optimization governed by PDEs

1.2.1 Basic concepts

We start this section by introducing briefly the notion of a control problem governed by PDEs, of which the optimization is a particular case. For a more detailed analysis, see [Lio71], [Trö10].

In general, a control problem consists in modifying a system $(\mathcal{S})$, called *state*, in our case governed by PDEs, in order to obtain a suitable behaviour of the solution of $(\mathcal{S})$. Therefore, the control problem can be expressed as follows: finding the control variable η such that the solution u of the state problem is the one desired. This can be formalized as a minimization problem of a *cost functional* $J = J(\eta)$ subject to some constraints, described by the system $(\mathcal{S})$.

If the system $(\mathcal{S})$ is described by a PDE, or by a system of PDEs, the control variable η normally influences the state through the initial or boundary condition or through the source term.

We will focus on problems of parametric optimization governed by PDEs: in this case, the control variable η is a vector of parameters $\mathbf{p} = (p_1, \dots, p_N) \in \mathbb{R}^N$. Moreover, in this work we will consider only unsteady equations and the optimization parameters will be only in the source term. Therefore, the state problem will be:

$$\begin{aligned} &\frac{\partial u}{\partial t} + \mathcal{L}(u) = s(\mathbf{x}, t; \mathbf{p}) \quad \mathbf{x} \in \Omega,\ 0 < t \leq T \\ &+ \text{b.c. and i.c.,} \end{aligned} \tag{1.5}$$

where $\mathcal{L}$ is an advection-diffusion differential operator, and the optimization problem will be:

$$\min_{\mathbf{p}\in\mathcal{P}} J(u(\mathbf{p})) \quad \text{subject to (1.5)}, \tag{1.6}$$

where $\mathcal{P} \subseteq \mathbb{R}^N$ is the space of admissible parameters. Note that J depends on $\mathbf{p}$ only through u.

1.2.2 Theoretical results

In this subsection, we will present some theoretical results about control problems. For a more detailed analysis, see [Sal]. First, we recall some basic concepts of differential calculus in Banach spaces. Let X and Y be two Banach spaces, and let $\mathfrak{L}(X, Y)$ be the space of linear functionals from X to Y.

Definition 5. $F : U \subseteq X \to Y$, with U open, is Fréchet differentiable in $x_0 \in U$, if there exists an application $L \in \mathfrak{L}(X, Y)$, called Fréchet differential, such that, if $x_0 + h \in U$,

$$F(x_0 + h) - F(x_0) = Lh + o(h),$$

that is equivalent to:

$$\lim_{h \to 0} \frac{\|F(x_0 + h) - F(x_0) - Lh\|_Y}{\|h\|_X} = 0.$$

The Fréchet differential is unique and we denote it with: $dF(x_0)$. If F is differentiable in every point of U, it is possible to define an application $dF : U \to \mathfrak{L}(X, Y)$, the Fréchet derivative:

$$x \mapsto dF(x).$$

We say that $F \in C^1(U)$ if dF is continuous in U, that is:

$$\|dF(x) - dF(x_0)\|_{\mathfrak{L}(X,Y)} \to 0 \quad \text{if} \quad \|x - x_0\|_X \to 0.$$

We now enunciate a useful result, known as *chain rule*.

Theorem 2. *Let X, Y and Z be Banach spaces. $F : U \subseteq X \to V \subseteq Y$ and $G : V \subseteq Y \to Z$, where U and V are open sets. If F is Fréchet differentiable in x_0 and G is Fréchet differentiable in y_0, where $y_0 = F(x_0)$, then $G \circ F$ is Fréchet differentiable in x_0 and:*

$$d(G \circ F)(x_0) = dG(y_0) \circ dF(x_0).$$

We now enunciate a theorem that guarantees existence and uniqueness of the solution of the control problem, when the control variable is a function η living in a Banach space H (more precisely $\eta \in H_{ad} \subseteq H$, H_{ad} being the space of admissible controls).

Theorem 3. *Let H be a reflexive Banach space, $H_{ad} \subseteq H$ convex closed and $J : H \to \mathbb{R}$. Under the following hypotheses:*

1. *if H_{ad} is unbounded, then $J(\eta) \to +\infty$ when $\|\eta\|_H \to +\infty$,*
2. *inferior semicontinuity of J with respect to the weak convergence, i.e.*

$$\eta_j \rightharpoonup \eta \Rightarrow J(\eta) \leq \liminf J(\eta_j) \;\; j \to +\infty.$$

There exists $\hat{\eta} \in H_{ad}$ such as $J(\hat{\eta}) = \min_{\eta \in H_{ad}} J(\eta)$. Moreover, if J is strictly convex than $\hat{\eta}$ is unique.

Let us observe that the Theorem 3 guarantees the well posedness of a control problem. The well posedness of a parametric optimization problem as (1.6) is not a straightforward consequence: it is necessary to add the request of differentiability of u with respect to the vector of parameters $\mathbf{p}$.

1.3 Objective of this work

In this work, as we already said, we will consider only unsteady equation: this will lead to time dependent cost functional $J(t)$. A common approach in the literature is to optimize with respect to a time-average over a period, if the phenomena is periodic, or, if it is not, over an interesting window of time. Therefore, chosen a time interval $(\bar{t}, \bar{t}+T)$, one can build the following cost functional:

$$J = \int_{\bar{t}}^{\bar{t}+T} J(t)dt, \tag{1.7}$$

and apply single-objective optimization techniques to it. However this approach, although straightforward, is limited since the dynamical nature of the state is neglected. Considering only a time-averaged quantity as optimization criterion may yield undesirable effects at some times and does not allow to control unsteady variations, for instance. The objective of this work is thus to study alternative strategies, based on the use of several optimization criteria representing the cost functional evaluated at some sampling times:

$$J_i = J(t_i) \qquad \text{for } i = 1, \ldots, n. \tag{1.8}$$

To this set of cost functionals $\{J_i\}_{i=1}^n$ we will apply the Multiple Gradient Descent Algorithm and Pareto-stationarity concept, already mentioned in Subsection 1.1.1, that will be described in details in the next Chapter. Moreover, an other approach will be investigated in this work that will be referred to as *windows approach*. Somehow, it is an hybrid between considering a time average quantity like (1.7) and a set of instantaneous quantities like (1.8). The set of cost functional which the MGDA will be applied to in this case is:

$$J_i = \int_{t_i}^{t_{i+1}} J(t)dt \qquad \text{for } i = 1, \ldots, n, \tag{1.9}$$

i.e the average operation is split on many subintervals.

Since the idea of coupling MGDA to the same quantity evaluated at different times is new, only scalar one-dimensional equation will be considered, in order to focus more on the multi-objective optimization details, that are the subject of this thesis, and less on the numerical solution of PDEs.

2 Multiple Gradient Descent Algorithm

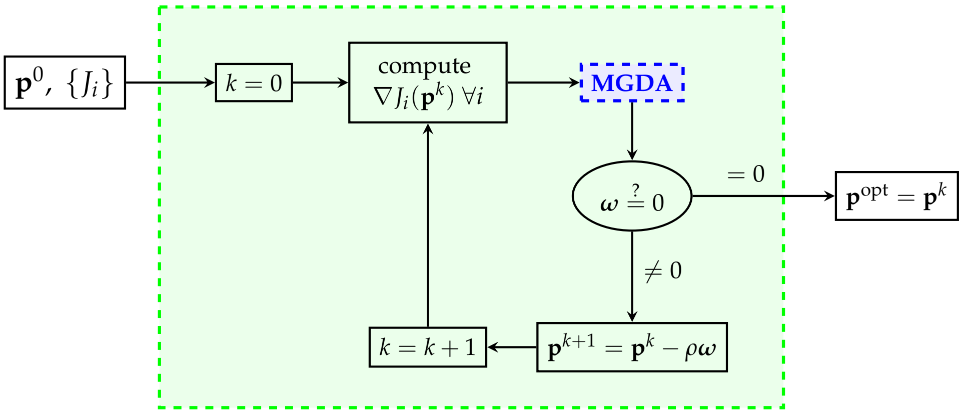
The Multiple Gradient Descent Algorithm (MGDA) is a generalization of the steepest descent method in order to approach problems of the kind (1.4): according to a set of vectors $\{\nabla J(\mathbf{p})\}_{i=0}^{n}$, a search direction $\boldsymbol{\omega}$ is defined such as $-\boldsymbol{\omega}$ is a descent direction $\forall J_i(\mathbf{p})$, until a Pareto-stationary point is reached.

Definition 6 (*descent direction*)**.** A direction $\mathbf{v}$ is said to be a descent one for a cost functional J in a point $\mathbf{p}$ if $\exists\, \rho > 0$ such as:

$$J(\mathbf{p} + \rho\mathbf{v}) < J(\mathbf{p}).$$

If the cost functional is sufficiently regular with respect to the parameters (for instance if $J \in C^2$), $\mathbf{v}$ is a descent direction if $(\mathbf{v}, \nabla J(\mathbf{p})) < 0$.

In this chapter we will focus on how to compute the *research direction* $\boldsymbol{\omega}$. That given, the optimization algorithm will be the one illustrated in Figure 2.1.



$\mathbf{p}^0,\ \{J_i\}$
$k = 0$
compute $\nabla J_i(\mathbf{p}^k)\ \forall i$
MGDA
$\omega \overset{?}{=} 0$
$= 0$
$\mathbf{p}^{\text{opt}} = \mathbf{p}^k$
$\neq 0$
$\mathbf{p}^{k+1} = \mathbf{p}^k - \rho\omega$
$k = k + 1$

Figure 2.1: Flow chart of the optimization algorithm. The optimization loop is highlighted in green.

2.1 MGDA I and II: original method and first improvements

The original version of MGDA defines $\boldsymbol{\omega}$ as the minimum-norm element in the set $\overline{U}$ of the linear convex combinations of the gradients:

$$\overline{U} = \{\mathbf{u} \in \mathbb{R}^N : \mathbf{u} = \sum_{i=1}^{n} \alpha_i \nabla J_i(\mathbf{p}), \text{ with } \alpha_i \geq 0 \ \forall i, \ \sum_{i=1}^{n} \alpha_i = 1\}. \tag{2.1}$$

In order to understand intuitively the idea behind this choice, let us analyse some two and three dimensional examples, shown in Figure 2.2 and Figure 2.3. The gradients can be seen as vectors of $\mathbb{R}^N$ all applied in the origin. Each vector identifies a point $\mathbf{x}_i \in \mathbb{R}^N$, and $\mathbf{d}$ is the point identified by $\boldsymbol{\omega}$. In this framework, $\overline{U}$ is the convex hull of the points $\mathbf{x}_i$, highlighted in green in the Figures. The convex hull lies in an affine subspace of dimension at most $n-1$, denoted with $\mathcal{A}_{n-1}$ and drawn in red. Note that if $n > N$, $\mathcal{A}_{n-1} \equiv \mathbb{R}^N$. We briefly recall the definition of projection on a set.

Definition 7. Let $\mathbf{x}$ be a point in a vector space V and let $C \subset V$ be a convex closed set. The *projection* of $\mathbf{x}$ on C is the point $\mathbf{z} \in C$ such as:

$$\mathbf{z} = \operatorname*{argmin}_{\mathbf{y} \in C} \|\mathbf{x} - \mathbf{y}\|_V.$$

Given this definition, it is clear that finding the minimum-norm element in a set is equivalent to finding the projection of the origin on this set. In the trivial case $O \in \overline{U}$ (condition equivalent to the Pareto stationarity), the algorithm gives $\boldsymbol{\omega} = \mathbf{0}$.

Otherwise, if $O \notin \overline{U}$, we split the projection procedure in two steps: first we project the origin O on the affine subspace $\mathcal{A}_{n-1}$, obtaining $O^\perp$; then, two scenarios are possible: if $O^\perp \in \overline{U}$ then $\mathbf{d} \equiv O^\perp$ (as in the left subfigures) and we have $\boldsymbol{\omega}$, otherwise it is necessary to project $O^\perp$ on $\overline{U}$ finding $\mathbf{d}$ and, consequently, $\boldsymbol{\omega}$ (as shown in the right subfigures).

Let us observe that in the special case in which $O^\perp \in \overline{U}$, we are not "favouring" any gradients, fact explained more rigorously in the following Proposition.

Proposition 4. *If $O^\perp \in \overline{U}$, all the directional derivatives are equal along $\boldsymbol{\omega}$:*

$$(\boldsymbol{\omega}, \nabla J_i(\mathbf{p})) = \|\boldsymbol{\omega}\|^2 \quad \forall i = 1, \dots, n.$$

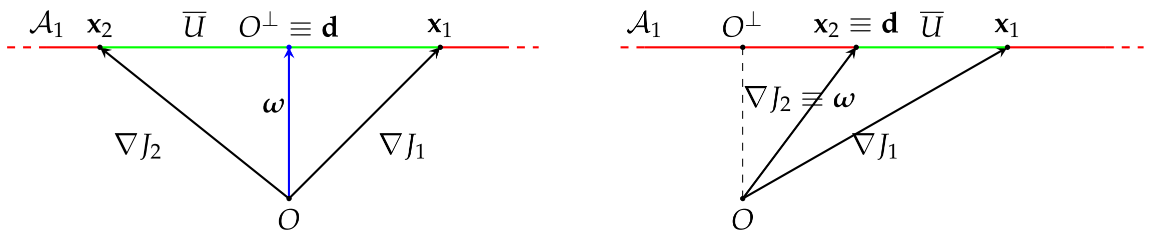


Figure 2.2: 2D examples.

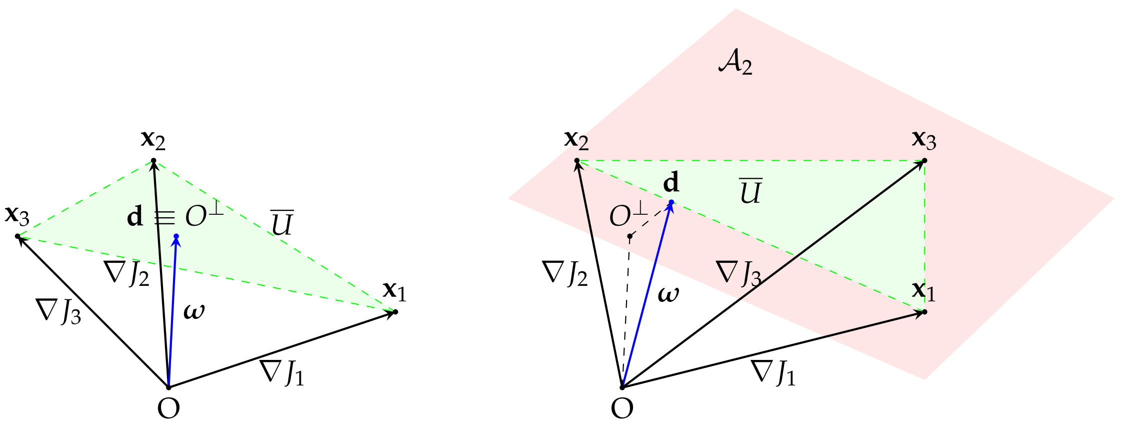


Figure 2.3: 3D examples.

Proof. It is possible to write every gradient as the sum of two orthogonal vectors: $\nabla J_i(\mathbf{p}) = \boldsymbol{\omega} + \mathbf{v}_i$, where $\mathbf{v}_i$ is the vector connecting $O^\perp$ and $\mathbf{x}_i$.

$$\begin{aligned}(\boldsymbol{\omega}, \nabla J_i(\mathbf{p})) &= (\boldsymbol{\omega}, \boldsymbol{\omega} + \mathbf{v}_i) = \\ &= (\boldsymbol{\omega}, \boldsymbol{\omega}) + (\boldsymbol{\omega}, \mathbf{v}_i) = (\boldsymbol{\omega}, \boldsymbol{\omega}) = \|\boldsymbol{\omega}\|^2.\end{aligned}$$

□

The main limitation of this first version of the algorithm is the practical determination of $\boldsymbol{\omega}$ when $n > 2$. For more details on MGDA I, see [Dés09].

In the second version of MGDA, $\boldsymbol{\omega}$ is computed with a direct procedure, based on the Gram-Schmidt orthogonalization process (see [Str06]), applied to rescaled gradients:

$$J_i' = \frac{\nabla J_i(\mathbf{p})}{S_i}.$$

However, this version has two main problems: the choice of the scaling factors S_i is very important but arbitrary, and the gradients must be linearly independent to apply Gram-Schmidt, that is impossible, for instance, in a situation in which the number of gradients is greater than the dimension of the space, i.e. $n > N$. Moreover, let us observe that in general the new $\boldsymbol{\omega}$ is different from the one computed with the first version of MGDA. For more details on this version of MGDA, see [Dés12a]. For these reasons, a third version has been introduced in [Dés12b].

2.2 MGDA III: algorithm for large number of criteria

The main idea of this version of MGDA is that in the cases in which there are many gradients, a common descent direction $\boldsymbol{\omega}$ could be based on only a subset of gradients. Therefore, one can consider only the *most significant* gradients, and perform an incomplete Gram-Schmidt process on them. For instance, in Figure 2.4 there are seven

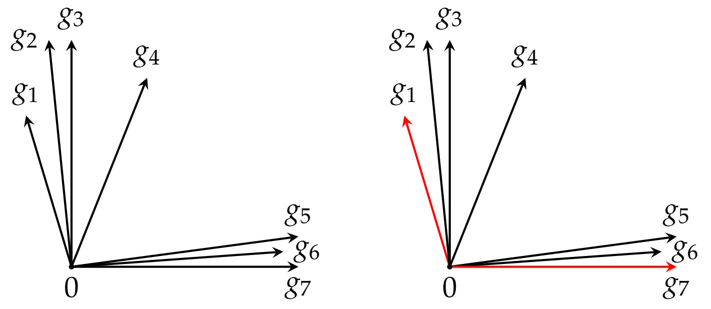


Figure 2.4: 2D example of trends among the gradients. Two good vectors to compute a common descent direction could be the ones in red.

different gradients, but two would be enough to compute a descent direction for all of them. In this contest, the ordering of the gradients is very important: in fact, only the first m out of n will be considered to compute $\boldsymbol{\omega}$, and hopefully it will be $m \ll n$. Let us observe that if the gradients live in a space of dimension N, it will be, for sure, $m \leq N$. Moreover, this eliminates the problem of linear dependency of the gradients.
In this section we will indicate with $\{\mathbf{g}_i\}_{i=1}^n$ the set of gradients evaluated at the current design point $\mathbf{p}$, corresponding respectively to the criteria $\{J_i\}_{i=1}^n$, and with $\{\mathbf{g}_{(i)}\}_{(i)=1}^n$ the reordered gradients, evaluated at $\mathbf{p}$, too. The ordering criterion proposed in [Dés12b] selects the first vector as follows:

$$\mathbf{g}_{(1)} = \mathbf{g}_k \text{ where } k = \underset{i=1,\dots,n}{\operatorname{argmax}} \min_{j=1,\dots,n} \frac{(\mathbf{g}_i, \mathbf{g}_j)}{(\mathbf{g}_i, \mathbf{g}_i)}, \tag{2.2}$$

while the criterion for the following vectors will be shown afterwards. The meaning of this choice will be one of the subject of the next section.

Being the ordering criterion given, the algorithm briefly consists in: building the orthogonal set of vectors $\{\mathbf{u}_i\}_{i=1}^m$ by adding one $\mathbf{u}_i$ at a time and using the Gram-Schmidt process; deciding whether or not to add another vector $\mathbf{u}_{i+1}$ (i.e. stopping criterion); and, finally, computing $\boldsymbol{\omega}$ as the minimum-norm element in the convex hull of $\{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_m\}$:

$$\boldsymbol{\omega} = \sum_{k=1}^m \alpha_k \mathbf{u}_k, \text{ where } \alpha_k = \frac{1}{1 + \sum\limits_{\substack{j=1 \\ j \neq k}}^m \frac{\|\mathbf{u}_k\|^2}{\|\mathbf{u}_j\|^2}}. \tag{2.3}$$

The choice of the coefficients α_k will be justified afterwards.

Now we analyse in detail some of the crucial part of the algorithm, with the support of the flow chart in Figure 2.5. The algorithm is initialized as follows: first $\mathbf{g}_{(1)}$ is selected (according to (2.2)), then one impose $\mathbf{u}_1 = \mathbf{g}_{(1)}$ and the loop starts with $i = 2$. In order to define the stopping criterion of this loop (TEST in Figure 2.5), we introduce an auxiliary structure: the lower-triangular matrix C. Let $c_{k,\ell}$ be the element of the matrix

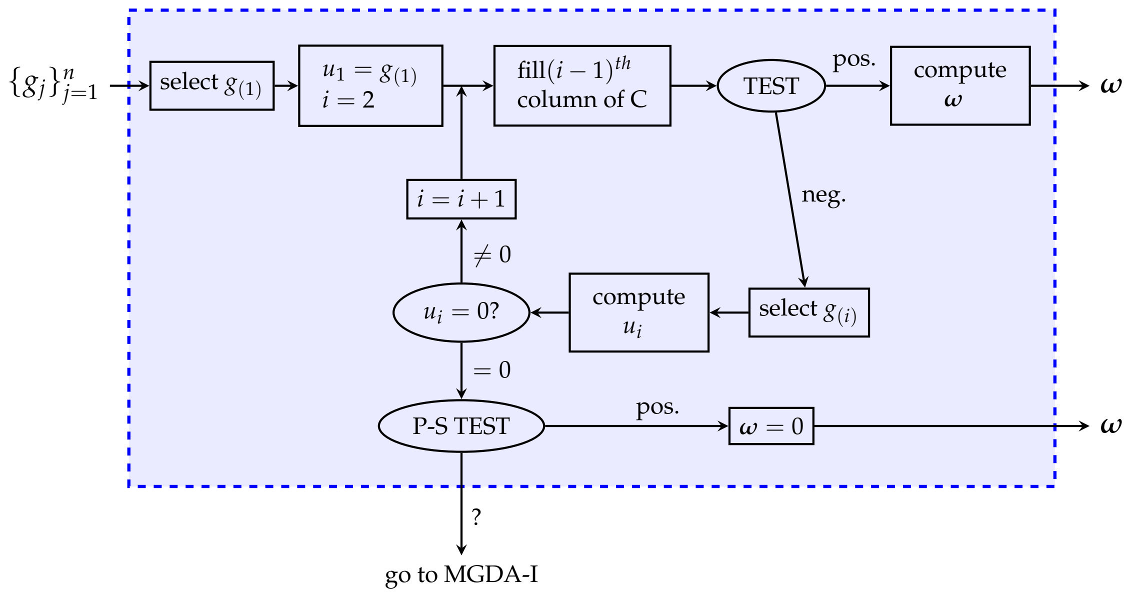


Figure 2.5: Flow chart of third version of MGDA

C in the k^{th} row, ℓ^{th} column.

$$c_{k,\ell} = \begin{cases} \frac{(\mathbf{g}_{(k)},\mathbf{u}_\ell)}{(\mathbf{u}_\ell,\mathbf{u}_\ell)} & \ell < k \\ \sum_{\ell=1}^{k-1} c_{k,\ell} & \ell = k \end{cases} \tag{2.4}$$

This matrix is set to $\mathbf{0}$ at the beginning of the algorithm, then is filled one column at a time (see Figure 2.5). The main diagonal of the matrix C contains the sum of the elements in that row and is updated every time an element in that row changes. Given a threshold value $a \in (0,1)$, chosen by the user and whose role will be studied in section 2.3.1, the test consists in checking if $c_{\ell,\ell} \geq a \ \forall \ell > i$: if it is true for all the coefficient the test is positive, the main loop is interrupted and $\boldsymbol{\omega}$ is computed as in (2.3) with m being the index of the current iteration, otherwise one continues adding a new $\mathbf{u}_i$ computed as follows: first there is the identification of the index ℓ such as:

$$\ell = \operatorname*{argmin}_{j=i,\dots,n} c_{j,j}, \tag{2.5}$$

then it is set $\mathbf{g}_{(i)} = \mathbf{g}_\ell$ (these two steps correspond to "select $\mathbf{g}_{(i)}$" in Figure 2.5), and finally the Gram-Schmidt process is applied:

$$\mathbf{u}_i = \frac{1}{A_i}\left(\mathbf{g}_{(i)} - \sum_{k<i} c_{i,k}\mathbf{u}_k\right), \tag{2.6}$$

where $A_i := 1 - \sum_{k<i} c_{i,k} = 1 - c_{i,i}$.

The meaning of the stop criterion is the following:

$$c_{\ell,\ell} \geq a > 0 \Rightarrow \exists\, i : c_{\ell,i} > 0 \Rightarrow \widehat{\mathbf{g}_{(\ell)}\mathbf{u}_i} \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$$

This means that a descent direction for $J_{(\ell)}$ can be defined from the $\{\mathbf{u}_i\}$ already computed, and adding a new $\mathbf{u}$ computed on the base of $\mathbf{g}_{(\ell)}$ would not add much information.
Once computed a new $\mathbf{u}_i$, before adding it to the set, we should check if it is nonzero, as it is shown in the flow chart. If it is zero, we know from the (2.6) that is:

$$\mathbf{g}_{(i)} = \sum_{k<i} c_{i,k}\mathbf{u}_k \tag{2.7}$$

It is possible to substitute in cascade (2.6) in (2.7), obtaining:

$$\begin{aligned}
\mathbf{g}_{(i)} &= \sum_{k<i} c_{i,k}\left[\frac{1}{A_k}\left(\mathbf{g}_{(k)} - \sum_{\ell<k} c_{k,\ell}\mathbf{u}_\ell\right)\right] = \\
&= \sum_{k<i} c_{i,k}\left[\frac{1}{A_k}\left(\mathbf{g}_{(k)} - \sum_{\ell<k} c_{k,\ell}\left[\frac{1}{A_\ell}\left(\mathbf{g}_{(\ell)} - \sum_{j<\ell} c_{\ell,j}\mathbf{u}_j\right)\right]\right)\right] = \\
&= \dots
\end{aligned} \tag{2.8}$$

Therefore, since $\mathbf{u}_1 = \mathbf{g}_{(1)}$, one can write:

$$\mathbf{g}_{(i)} = \sum_{k<i} c'_{i,k}\mathbf{g}_{(k)} \tag{2.9}$$

Computing the $c'_{i,k}$ is equivalent to solving the following linear system:

$$\underbrace{\begin{bmatrix} \mathbf{g}_{(1)} & \mathbf{g}_{(2)} & \cdots & \mathbf{g}_{(i-1)} \end{bmatrix}}_{=\mathbb{A}} \underbrace{\begin{bmatrix} c'_{1,i} \\ c'_{2,i} \\ \vdots \\ \vdots \\ c'_{i-1,i} \end{bmatrix}}_{=\mathbf{x}} = \underbrace{\begin{bmatrix} \mathbf{g}_{(i)} \end{bmatrix}}_{=\mathbf{b}}$$

Since N is the dimension of the space in which the gradients live, i.e. $\mathbf{g}_i \in \mathbb{R}^N \ \forall i = 1, \dots, n$, the matrix $\mathbb{A}$ is $[N \times (i-1)]$, and for sure is $i-1 \leq N$, because $i-1$ is the number of linearly independent gradients we already found. This means that the system is overdetermined, but we know from (2.8)-(2.9) that the solution exists. In conclusion, to find the $c'_{i,k}$ we can solve the following system:

$$\mathbb{A}^T\mathbb{A}\mathbf{x} = \mathbb{A}^T\mathbf{b} \tag{2.10}$$

where $\mathbb{A}^T\mathbb{A}$ is a square matrix, and it is invertible because the columns of $\mathbb{A}$ are linearly independent (for further details on linear systems, see [Str06]). Once found the $c'_{i,k}$, it is possible to apply the Pareto-stationarity test (P-S TEST in Figure 2.5), based on the following Proposition.

Proposition 5. *$c'_{i,k} \leq 0 \; \forall k = 1, \dots, i-1 \Rightarrow$ Pareto-stationarity.*

Proof. Starting from the (2.9), we can write:

$$\mathbf{g}_{(i)} - \sum_{k<i} c'_{i,k} \mathbf{g}_{(k)} = 0 \quad \Rightarrow \quad \sum_{k=1}^{n} \gamma_k \mathbf{g}_{(k)} = 0$$

where:

$$\gamma_k = \begin{cases} -c'_{i,k} & k < i \\ 1 & k = i \\ 0 & k > i \end{cases}$$

$\gamma_k \geq 0 \; \forall k$. If we define $\Gamma = \sum_{k=1}^{n} \gamma_k$, then the coefficients $\{\frac{\gamma_k}{\Gamma}\}_{k=1}^{n}$ satisfy the Definition 2.

□

Note that the condition given in Proposition 5 is sufficient, but not necessary. Therefore, if it is not satisfied we fall in an ambiguous case: the solution suggested in [Dés12b] is, in this case, to compute $\boldsymbol{\omega}$ according to the original version of MGDA, as shown in Figure 2.5.

2.2.1 Properties

We now present and prove some properties of MGDA III. Note that these properties are not valid if we fall in the ambiguous case.

Proposition 6. *$\alpha_i, \;\; i = 1, \dots, n$ defined as in (2.3) are such that:*

$$\sum_{k=1}^{m} \alpha_k = 1 \quad \textit{and} \quad (\boldsymbol{\omega}, \mathbf{u}_k) = \|\boldsymbol{\omega}\|^2 \;\; \forall k = 1, \dots, m.$$

Proof. First, let us observe that:

$$\frac{1}{\alpha_k} = 1 + \sum_{\substack{j=1 \\ j \neq k}}^{m} \frac{\|\mathbf{u}_k\|^2}{\|\mathbf{u}_j\|^2} = \frac{\|\mathbf{u}_k\|^2}{\|\mathbf{u}_k\|^2} + \sum_{\substack{j=1 \\ j \neq k}}^{m} \frac{\|\mathbf{u}_k\|^2}{\|\mathbf{u}_j\|^2} = \sum_{j=1}^{m} \frac{\|\mathbf{u}_k\|^2}{\|\mathbf{u}_j\|^2} = \|\mathbf{u}_k\|^2 \sum_{j=1}^{m} \frac{1}{\|\mathbf{u}_j\|^2}$$

Therefore, it follows that:

$$\sum_{k=1}^{m} \alpha_k = \sum_{k=1}^{m} \frac{1}{\|\mathbf{u}_k\|^2 \sum_{j=1}^{m} \frac{1}{\|\mathbf{u}_j\|^2}} = \sum_{k=1}^{m} \frac{1/\|\mathbf{u}_k\|^2}{\sum_{j=1}^{m} \frac{1}{\|\mathbf{u}_j\|^2}} = \frac{1}{\sum_{j=1}^{m} \frac{1}{\|\mathbf{u}_j\|^2}} \sum_{k=1}^{m} \frac{1}{\|\mathbf{u}_k\|^2} = 1.$$

Let us now prove the second part:

$$(\boldsymbol{\omega}, \mathbf{u}_i) = \left(\sum_{k=1}^{m} \alpha_k \mathbf{u}_k, \mathbf{u}_i \right) = \alpha_i (\mathbf{u}_i, \mathbf{u}_i) = \alpha_i \|\mathbf{u}_i\|^2 = \frac{1}{\sum_{k=1}^{m} \frac{1}{\|\mathbf{u}_k\|^2}} =: D$$

where we used the definition of $\boldsymbol{\omega}$, the linearity of the scalar product, the orthogonality of the $\mathbf{u}_i$ and, finally, the definition of α_i. Let us observe that D does not depend on i. Moreover,

$$\begin{aligned} \|\boldsymbol{\omega}\|^2 = (\boldsymbol{\omega}, \boldsymbol{\omega}) = \left(\sum_{k=1}^{m} \alpha_k \mathbf{u}_k, \sum_{\ell=1}^{m} \alpha_\ell \mathbf{u}_\ell \right) = \sum_{j=1}^{m} \alpha_j^2 (\mathbf{u}_j, \mathbf{u}_j) = \\ = \sum_{j=1}^{m} \alpha_j D = D \sum_{j=1}^{m} \alpha_j = D = (\boldsymbol{\omega}, \mathbf{u}_i) \;\; \forall i = 1, \dots, m. \end{aligned}$$

□

Proposition 7. $\forall i = 1, \dots, m \;\; (\boldsymbol{\omega}, \mathbf{g}_{(i)}) = (\boldsymbol{\omega}, \mathbf{u}_i)$.

Proof. From the definition of $\mathbf{u}_i$ in (2.6), it is possible to write $\mathbf{g}_{(i)}$ in the following way:

$$\begin{aligned} \mathbf{g}_{(i)} = A_i \mathbf{u}_i + \sum_{k<i} c_{i,k} \mathbf{u}_k = \left(1 - \sum_{k<i} c_{i,k} \right) \mathbf{u}_i + \sum_{k<i} c_{i,k} \mathbf{u}_k = \\ = \mathbf{u}_i - \left(\sum_{k<i} c_{i,k} \right) \mathbf{u}_i + \sum_{k<i} c_{i,k} \mathbf{u}_k \end{aligned}$$

and doing the scalar product with $\boldsymbol{\omega}$ one obtains:

$$(\mathbf{g}_{(i)}, \boldsymbol{\omega}) = (\mathbf{u}_i, \boldsymbol{\omega}) \underbrace{- \left(\sum_{k<i} c_{i,k} \right) (\mathbf{u}_i, \boldsymbol{\omega}) + \left(\sum_{k<i} c_{i,k} \mathbf{u}_k, \boldsymbol{\omega} \right)}_{=(\star)}$$

We want to prove that $(\star) = 0$. As we have shown in the proof of Proposition 6:

$$\alpha_k = \frac{1}{\|\mathbf{u}_k\|^2 \sum_{j=1}^{m} \frac{1}{\|\mathbf{u}_j\|^2}} \;\; \Rightarrow \;\; \alpha_i \|\mathbf{u}_i\|^2 = \frac{1}{\sum_{j=1}^{m} \frac{1}{\|\mathbf{u}_j\|^2}}$$

Using the definition of $\boldsymbol{\omega}$, the linearity of the scalar product and the orthogonality of the $\mathbf{u}_i$ one obtains:

$$\begin{aligned}(\star) &= -\left(\sum_{k<i} c_{i,k}\right)\alpha_i\|\mathbf{u}_i\|^2 + \sum_{k<i} c_{i,k}\alpha_k\|\mathbf{u}_k\|^2 = \\ &= -\left(\sum_{k<i} c_{i,k}\right)\frac{1}{\sum_{j=1}^m \frac{1}{\|\mathbf{u}_j\|^2}} + \left(\sum_{k<i} c_{i,k}\right)\frac{1}{\sum_{j=1}^m \frac{1}{\|\mathbf{u}_j\|^2}} = 0.\end{aligned}$$

□

Proposition 8. $\forall i = m+1,\dots,n \ \ (\mathbf{g}_{(i)}, \boldsymbol{\omega}) \geq a\|\boldsymbol{\omega}\|^2$.

Proof. It is possible to write $\mathbf{g}_{(i)}$ as follows:

$$\sum_{k=1}^m c_{i,k}\mathbf{u}_k + \mathbf{v}_i$$

where $\mathbf{v}_i \perp \{\mathbf{u}_1,\dots,\mathbf{u}_m\}$. Since $i > m$, we are in the case in which the test $c_{i,i} \geq a$ is positive. Consequently:

$$(\mathbf{g}_{(i)}, \boldsymbol{\omega}) = \sum_{k=1}^m c_{i,k}(\mathbf{u}_k, \boldsymbol{\omega}) = \sum_{k=1}^m c_{i,k}\|\boldsymbol{\omega}\|^2 = c_{i,i}\|\boldsymbol{\omega}\|^2 \geq a\|\boldsymbol{\omega}\|^2.$$

□

In conclusion, as a straightforward consequence of the Propositions above, we have:

$$(\mathbf{g}_{(i)}, \boldsymbol{\omega}) = \|\boldsymbol{\omega}\|^2 \ \ \forall i = 1,\dots,m \qquad (\mathbf{g}_{(i)}, \boldsymbol{\omega}) \geq a\|\boldsymbol{\omega}\|^2 \ \ \forall i = m+1,\dots,n,$$

which confirms that the method provides a direction that is a descent one even for the gradients not used in the construction of the Gram-Schmidt basis and, consequently, not directly used in the computation of $\boldsymbol{\omega}$.

2.3 MGDA III b: Improvements related to the ambiguous case

In this work we propose some strategies to avoid going back to MGDA-version I, when we fall in the ambiguous case, since it is very unefficient when a large number of objective functional are considered. In order to do this, we should investigate which are the reasons that lead to the ambiguous case. Let us observe that in the algorithm there are two arbitrary choices: the value of the threshold a for the stopping test and the algorithm to order the gradients.

2.3.1 Choice of the threshold

Let us start with analysing how a too demanding choice of a (i.e. of a too large value) can affect the algorithm. We will do this with the support of the example in Figure 2.6, following the algorithm as illustrated in the precedent section. For example, assume

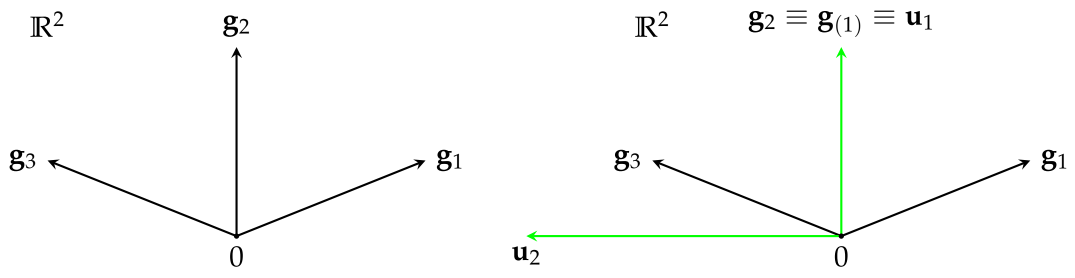


Figure 2.6: Example to illustrate a wrong choice for a.

that the gradients are the following:

$$\mathbf{g}_1 = \begin{pmatrix} 5 \\ 2 \end{pmatrix} \quad \mathbf{g}_2 = \begin{pmatrix} 0 \\ 5 \end{pmatrix} \quad \mathbf{g}_3 = \begin{pmatrix} -5 \\ 2 \end{pmatrix}. \tag{2.11}$$

Using the criterion (2.2) the gradient selected as first is $\mathbf{g}_2$. At the first iteration of the loop the first column of the matrix C is filled in as in (2.4), with the following result:

$$C = \begin{bmatrix} 1 & & \\ 0.4 & 0.4 & \\ 0.4 & 0 & 0.4 \end{bmatrix}$$

If a is set, for instance, at 0.5 (or any value greater than 0.4) the stop test fails. However, looking at the picture, it is intuitively clear that $\boldsymbol{\omega} = \mathbf{u}_1$ is the best descent direction one can get (to be more rigorous, it is the descent direction that gives the largest estimate possible from Proposition 8). If the test fails, the algorithm continues and another vector $\mathbf{u}_i$ is computed. This particular case is symmetric, therefore the choice of $\mathbf{g}_{(2)}$ is not very interesting. Let us say that $\mathbf{g}_{(2)} = \mathbf{g}_3$, than $\mathbf{u}_2$ is the one shown in the right side of Figure 2.11 and, since it is nonzero, the algorithm continues filling another column of C:

$$C = \begin{bmatrix} 1 & & \\ 0.4 & 0.4 & \\ 0.4 & -0.6 & -0.2 \end{bmatrix}$$

Even in this case the stopping test fails (it would fail with any value of a due to the negative term in position (3,3)). Since we reached the dimension of the space, the next $\mathbf{u}_i$ computed is zero: this leads to the Pareto-stationarity test, that cannot give a positive answer, because we are not at a Pareto-stationary point. Therefore, we fall in the ambiguous case.

To understand why this happens, let us consider the relation between a and $\boldsymbol{\omega}$, given by Proposition 8 that we briefly recall:

$$\forall i = m+1, \dots, n \;\; (\mathbf{g}_{(i)}, \boldsymbol{\omega}) \geq a\|\boldsymbol{\omega}\|^2.$$

This means that, for the gradients that haven't been used to compute $\boldsymbol{\omega}$, we have a lower bound for the directional derivative along $\boldsymbol{\omega}$: they are, at least, a fraction a of $\|\boldsymbol{\omega}\|^2$. The problem is that it is impossible to know *a priori* how big a can be. For this reason, the parameter a has been removed. The stopping test remains the same as before, with only one difference: instead of checking $c_{\ell,\ell} \geq a$ one checks if $c_{\ell,\ell} > 0$.

However, removing a is not sufficient to remove completely the ambiguous case, as it is shown in the next subsection.

2.3.2 Ordering criteria

In this section we will analyse the meaning of the ordering criterion used in MGDA III and we will present another one, providing some two-dimensional examples in order to have a concrete idea of different situations.

The ordering problem can be divided in two steps:

(i) choice of $\mathbf{g}_{(1)}$

(ii) choice of $\mathbf{g}_{(i)}$, given $\mathbf{g}_{(k)}\ \ \forall k = 1, \dots i-1$

The naïve idea is to order the gradients in the way that makes the MGDA stop as soon as possible. Moreover, we would like to find an ordering criterion that minimizes the times in which we fall in the ambiguous case (i.e. when we find a $\mathbf{u}_i = 0$ and the P-S test is not satisfied).

Definition 8. Let us say that an ordering is *acceptable* if it does not lead to the ambiguous case, otherwise we say that the ordering is *not acceptable*.

First, we discuss point (ii), since it is simpler. If we are in the case in which we have to add another vector, it is because the stopping test gave a negative result, therefore:

$$\exists \ell,\ i \leq \ell \leq n :\ c_{\ell,\ell} < 0.$$

We chose as $\mathbf{g}_{(i)}$ the one that violates the stopping test the most. This explains the (2.5), that we recall here:

$$\mathbf{g}_{(i)} = \mathbf{g}_k\ :\ k = \operatorname*{argmin}_{\ell} c_{\ell,\ell} \quad i \leq \ell \leq n.$$

The point (i) is more delicate, since the choice of the first vector influences strongly all the following choices. Intuitively, in a situation like the one shown in Figure 2.7, we would pick as first vector $\mathbf{g}_5$ and we would have the stopping test satisfied at the first iteration, in fact $\boldsymbol{\omega} = \mathbf{g}_5$ is a descent direction for all the gradients. Formally, this could be achieved using the criterion 2.2, that we recall:

$$\mathbf{g}_{(1)} = \mathbf{g}_k, \quad \text{where } k = \operatorname*{argmax}_{i} \min_{j} \frac{(\mathbf{g}_j, \mathbf{g}_i)}{(\mathbf{g}_i, \mathbf{g}_i)} \tag{2.12}$$

However, it is clear that this is not the best choice possible in cases like the one shown in Figure 2.8, in which none of the gradients is close to the bisecting line (or, in three dimensions, to the axis of the cone generated by the vectors). In this particular case, it

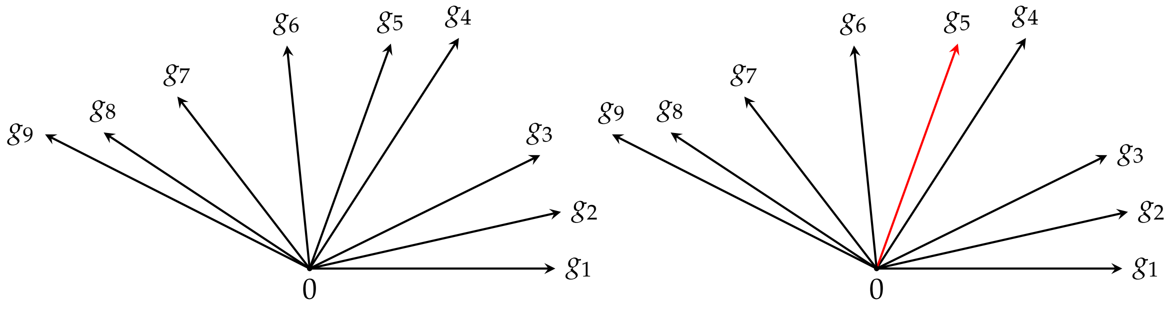


Figure 2.7: Example 1.

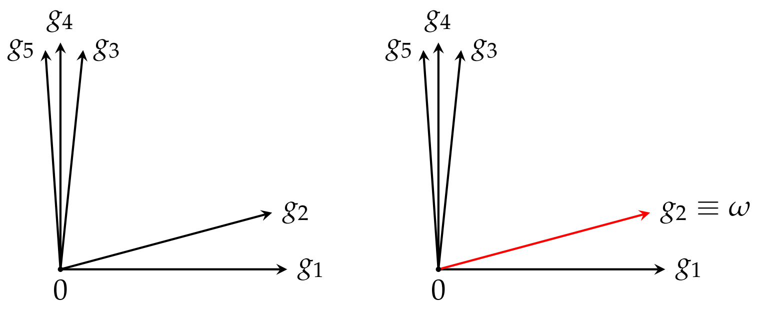


Figure 2.8: Choice of the first gradient with the criterion (2.2)-(2.5).

would be better to pick two external vectors ($\mathbf{g}_1$ and $\mathbf{g}_5$) to compute $\boldsymbol{\omega}$. Anyway, with the criterion (2.2)-(2.5) we still obtain an *acceptable* ordering, and the descent direction is computed with just one iteration. However, there are cases in which the criterion (2.2)-(2.5) leads to a *not acceptable* ordering. Let us consider, for example, a simple situation like the one in Figure 2.9, with just three vectors, and let us follow the algorithm step by step.

Let the gradients be the following:

$$\mathbf{g}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \mathbf{g}_2 = \begin{pmatrix} 0.7 \\ 0.7 \end{pmatrix} \quad \mathbf{g}_3 = \begin{pmatrix} -1 \\ 0.1 \end{pmatrix} \tag{2.13}$$

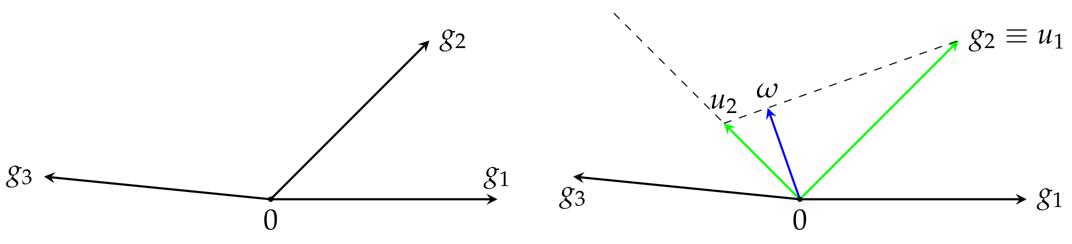


Figure 2.9: Criterion (2.2)-(2.5) leads to a not acceptable ordering.

Given the vectors (2.13), we can build a matrix T such as:

$$[T]_{i,j} = \frac{(\mathbf{g}_j, \mathbf{g}_i)}{(\mathbf{g}_i, \mathbf{g}_i)} \tag{2.14}$$

obtaining:

$$T = \begin{bmatrix} 1 & 0.7 & -1 \\ 0.714 & 1 & -0.643 \\ -0.99 & -0.624 & 1 \end{bmatrix} \begin{array}{l} \rightsquigarrow \min = -1 \\ \rightsquigarrow \min = -0.643 \\ \rightsquigarrow \min = -0.99 \end{array}$$

The internal minimum of (2.2) is the minimum of each row. Among this, we should select the maximum: the index of this identifies the index k such as $\mathbf{g}_{(1)} = \mathbf{g}_k$. In this case, $k = 2$. We can now fill in the first column of the triangular matrix C introduced in (2.4), obtaining:

$$C = \begin{bmatrix} 1 & & \\ 0.714 & 0.714 & \\ -0.642 & 0 & -0.642 \end{bmatrix}$$

Since the diagonal terms are not all positive, we continue adding $\mathbf{g}_{(2)} = \mathbf{g}_3$, according to (2.5). Note that, due to this choice, the second line of the matrix C has to be switched with the third. The next vector of the orthogonal base $\mathbf{u}_2$ is computed as in (2.6), and in Figure 2.9, right subfigure, it is shown the result. Having a new $\mathbf{u}_2 \neq \mathbf{0}$, we can now fill the second column of C as follows:

$$C = \begin{bmatrix} 1 & & \\ -0.642 & -0.642 & \\ 0.714 & -1.4935 & -0.77922 \end{bmatrix}$$

This is a situation analogue to the one examinated in the example of Section 2.3.1, therefore we fall in the ambiguous case.

A possible solution to this problem is the one proposed in the flow chart in Figure 2.10: compute $\boldsymbol{\omega}$ with the $\{\mathbf{u}_i\}$ that we already have and check for which gradients it is not a descent direction (in this case for $\mathbf{g}_1$, as shown in the right side of Figure 2.9). We want this gradient to be "more considered": in order to do that we double their length and we restart the algorithm from the beginning. This leads to a different matrix T:

$$T = \begin{bmatrix} 1 & 0.35 & -0.5 \\ 1.428 & 1 & -0.643 \\ -1.98 & -0.624 & 1 \end{bmatrix} \begin{array}{l} \rightsquigarrow \min = -0.5 \\ \rightsquigarrow \min = -0.643 \\ \rightsquigarrow \min = -1.98 \end{array}$$

With the same process described above, one obtains $\mathbf{g}_{(1)} = \mathbf{u}_1 = \mathbf{g}_1$. The final output of MGDA is shown in Figure 2.11, with $\boldsymbol{\omega} = [0.00222, 0.0666]^T$.

However, rerunning MGDA from the beginning is not an efficient solution, in particular in the cases in which we have to do it many times before reaching a good ordering of the gradients. For this reason, a different ordering criterion is proposed: we chose as a first vector the "most external", or the "most isolated" one. Formally, this

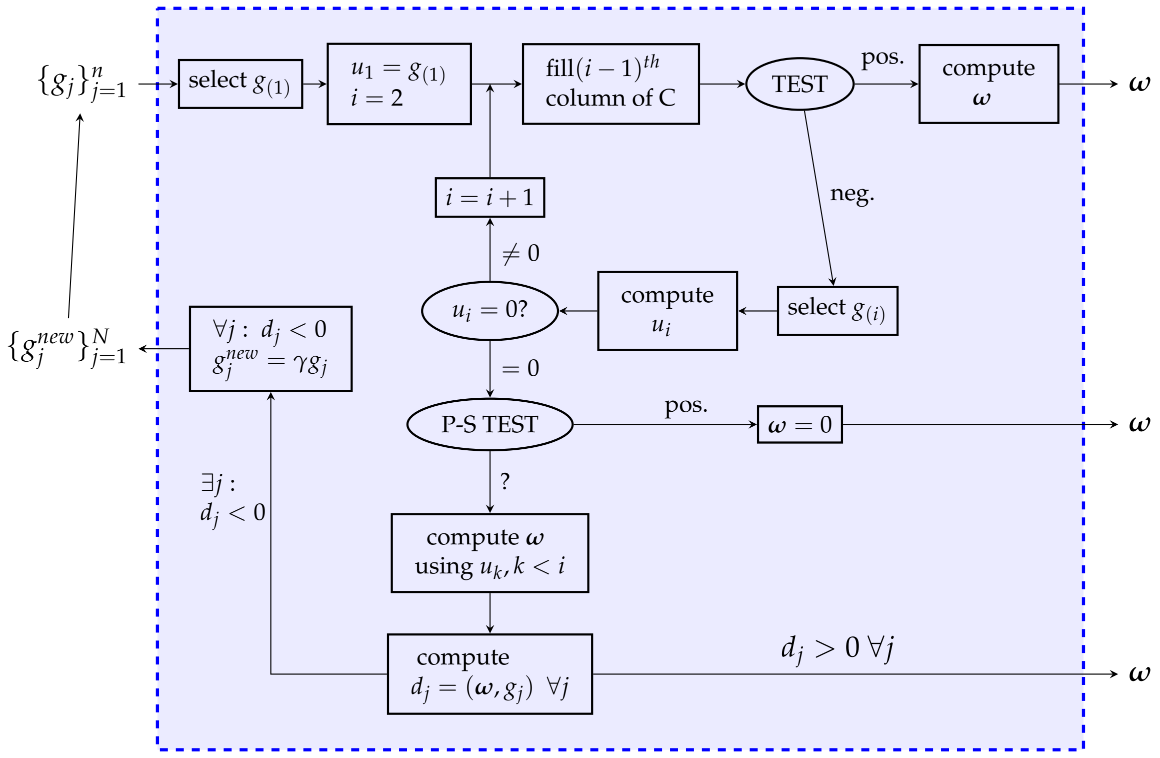


Figure 2.10: Flow chart of MGDA-III b

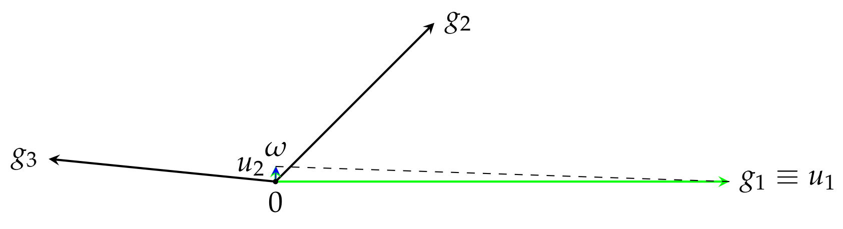


Figure 2.11: Criterion (2.2)-(2.5) after doubling.

means:

$$\mathbf{g}_{(1)} = \mathbf{g}_k, \quad \text{where } k = \operatorname*{argmin}_i \min_j \frac{(\mathbf{g}_j, \mathbf{g}_i)}{(\mathbf{g}_i, \mathbf{g}_i)} \tag{2.15}$$

On one hand with this criterion it is very unlikely to stop at the first iteration, but on the other hand the number of situations in which is necessary to rerun the whole algorithm is reduced. Note that, in this case, instead of doubling the length of the non-considered gradients, we must make the new gradients smaller.

These solutions are not the only ones possible, moreover there is no proof that the ambiguous case is completely avoided. For instance, another idea could be the following: keeping the threshold parameter a for the stopping test and, every time the ambiguous case occurs, rerunning the algorithm with a reduced value of a.

A C++ code has been developed in order to implement MGDA III and III b. It is described in details in appendix A.

3 | The PDE constraints

In this chapter, first we will illustrate the general solving procedure that has been applied to solve multi-objective optimization problems with PDE constraints, then we will introduce the specific PDEs used as test cases in this work and finally we will present the first numerical results, related to the convergence of the schemes and the validation of the methods used.

3.1 Solving method

In this section we want to explain the method used in this work to solve problem of the kind (1.6). First, let us recall all the ingredients of the problem:

· state equation:

$$\begin{cases} \frac{\partial u}{\partial t} + \mathcal{L}(u) = s(\mathbf{x}, t; \mathbf{p}) & \mathbf{x} \in \Omega,\ 0 < t \leq T \\ \nabla u \cdot \mathbf{n} = 0 & \Gamma_N \subseteq \partial\Omega,\ 0 < t \leq T \\ u = f(t) & \Gamma_D \subseteq \partial\Omega,\ 0 < t \leq T \\ u = g(\mathbf{x}) & \mathbf{x} \in \Omega,\ t = 0 \end{cases}$$

· cost functional J that we assume quadratic in u:

$$J(u(\mathbf{p})) = \frac{1}{2} q(u, u), \tag{3.1}$$

where $q(\cdot, \cdot)$ is a bilinear, symmetric and continuous form. For instance, a very common choice for J in problems of fluid dynamics is $J = \frac{1}{2}\|\nabla u\|^2_{L^2(\Omega)}$.

Let us observe that every classical optimization method requires the computation of the gradient of the cost functional with respect to the parameters $\mathbf{p}$, that we will indicate with ∇J. In order to compute it, we can apply the chain rule, obtaining:

$$\nabla J(\mathbf{p}^0) = d(J \circ u)(\mathbf{p}^0) = dJ(u_0) \circ du(\mathbf{p}^0),$$

being $u_0 = u(\mathbf{p}^0)$. Therefore, ∇J is composed by two pieces. Let us start analysing the first one.

Proposition 9. *If J is like in (3.1), than its Fréchet differential computed in u_0 and applied to the generic increment h is:*

$$dJ(u_0)h = q(u_0, h)$$

Proof. Using the definition of Fréchet derivative one obtains:

$$\begin{aligned} J(u_0+h) - J(u_0) &= \frac{1}{2}q(u_0+h, u_0+h) - \frac{1}{2}q(u_0, u_0) = \\ &= q(u_0, h) + \frac{1}{2}q(h,h) = q(u_0, h) + O(\|h\|^2), \end{aligned}$$

where the last passage is justified by the continuity of $q(\cdot, \cdot)$. □

The second piece, i.e. $du(\mathbf{p}^0)$, is just the derivative of u with respect to the parameters. This means that the $i-th$ component of the gradient of J is the following:

$$[\nabla J(\mathbf{p}^0)]_i = q\left(u_0, u_{p_i}(\mathbf{p}^0)\right), \tag{3.2}$$

where

$$u_{p_i}(\mathbf{p}^0) = \frac{\partial u}{\partial p_i}(\mathbf{p}^0),$$

and it is called *sensitivity* (with respect to the parameter p_i). We now apply this procedure to the example introduced above, $J = \frac{1}{2}\|\nabla u\|^2_{L^2(\Omega)}$. We observe that the form q is:

$$q(u,v) = (\nabla u, \nabla v)_{L^2(\Omega)},$$

that is bilinear, symmetric and continuous. Therefore we can apply the result of Proposition 9, obtaining:

$$[\nabla J(\mathbf{p}^0)]_i = \left(\nabla u_0, \nabla u_{p_i}(\mathbf{p}^0)\right)_{L^2(\Omega)}.$$

Summing up: in order to solve a minimization problem like (1.6), we need to compute the gradient of J with respect to the parameters. To do that, we see from (3.2) that it is necessary to know $u_0 = u(\mathbf{p}^0)$ (therefore to solve the state equation with $\mathbf{p} = \mathbf{p}^0$) and to know u_{p_i}. The last step we have to do is finding the equations governing the behaviour of the sensitivities u_{p_i}. To do that, we apply the *Continuous Sensitivity Equation* method, shorten in CSE method from now on. See [KED$^+$10]. It consists in formally differentiating the state equation, the boundary and initial condition, with respect to p_i:

$$\begin{cases} \frac{\partial}{\partial p_i}\left(\frac{\partial u}{\partial t} + \mathcal{L}(u)\right) = \frac{\partial}{\partial p_i}s(\mathbf{x}, t; \mathbf{p}) & \mathbf{x} \in \Omega,\ 0 < t \leq T \\ \frac{\partial}{\partial p_i}\left(\nabla u \cdot \mathbf{n}\right) = 0 & \Gamma_N \subseteq \partial\Omega,\ 0 < t \leq T \\ \frac{\partial}{\partial p_i}u = 0 & \Gamma_D \subseteq \partial\Omega,\ 0 < t \leq T \\ \frac{\partial}{\partial p_i}u = 0 & \mathbf{x} \in \Omega,\ t = 0. \end{cases}$$

After a permutation of the derivatives, if $\mathcal{L}$ is linear and Ω does not depend on $\mathbf{p}$, one obtains the *sensitivities equations*:

$$\begin{cases} \frac{\partial u_{p_i}}{\partial t} + \mathcal{L}(u_{p_i}) = s_{p_i}(\mathbf{x}, t; \mathbf{p}) & \mathbf{x} \in \Omega,\ 0 < t \leq T \\ \nabla u_{p_i} \cdot \mathbf{n} = 0 & \Gamma_N \subseteq \partial\Omega,\ 0 < t \leq T \\ u_{p_i} = 0 & \Gamma_D \subseteq \partial\Omega,\ 0 < t \leq T \\ u_{p_i} = 0 & \mathbf{x} \in \Omega,\ t = 0. \end{cases} \tag{3.3}$$

For every $i = 1, \dots, N$, this is a problem of the same kind of the state equation and independent from it, therefore from a computational point of view it is possible to use the same solver and to solve the $N + 1$ problems in parallel (N sensitivity problems plus the state). However, if $\mathcal{L}$ is nonlinear,

$$\mathcal{L}(u_{p_i}) \neq \frac{\partial}{\partial p_i}\mathcal{L}(u),$$

and we need to introduce a new operator $\widetilde{\mathcal{L}} = \widetilde{\mathcal{L}}(u, u_{p_i}) = \frac{\partial}{\partial p_i}\mathcal{L}(u)$. Therefore, the sensitivities equations in this case will be:

$$\begin{cases} \frac{\partial u_{p_i}}{\partial t} + \widetilde{\mathcal{L}}(u, u_{p_i}) = s_{p_i}(\mathbf{x}, t; \mathbf{p}) & \mathbf{x} \in \Omega,\ 0 < t \leq T \\ \nabla u_{p_i} \cdot \mathbf{n} = 0 & \Gamma_N \subseteq \partial\Omega,\ 0 < t \leq T \\ u_{p_i} = 0 & \Gamma_D \subseteq \partial\Omega,\ 0 < t \leq T \\ u_{p_i} = 0 & \mathbf{x} \in \Omega,\ t = 0. \end{cases} \tag{3.4}$$

They are no more independent from the state equation, but they remain independent ones from the others. Let us observe that in this case it is necessary to implement a different (but similar) solver than the one used for the state.

3.2 Test cases and numerical schemes

We now introduce the partial differential equations imposed as constraints of the test cases considered and the numerical schemes adopted to solve them.

3.2.1 Test cases

In this work we will consider only unsteady one-dimensional scalar advection-diffusion PDEs, both linear and nonlinear. Let $\Omega = (x_a, x_b)$ be the domain. The linear equation will be the following:

$$\begin{cases} \partial_t u - b\partial_x^2 u + c\partial_x u = s(x, t; \mathbf{p}) & \text{in } (x_a, x_b),\ 0 < t \leq T \\ u(x_a, t) = f_D(t) & 0 < t \leq T \\ \partial_x u(x_b, t) = f_N(t) & 0 < t \leq T \\ u(x, 0) = g(x) & \text{in } (x_a, x_b) \end{cases} \tag{3.5}$$

where b and c are constant coefficients $b, c > 0$, $c \gg b$ (i.e. dominant advection), $s(x,t)$, $f_D(t)$, $f_N(t)$ and $g(x)$ are given functions and $\mathbf{p}$ is the vector of control parameters. Being $c > 0$ it makes sense to impose Dirichlet boundary condition in x_a, since it is the inlet, and Neumann boundary condition in x_b, since it is the outlet. We observe that the problem (3.5) is well posed, i.e. the solution exists and is unique. For more details on the well posedness of this kind of problems see [Sal10], [Qua09].

Applying the CSE method, introduced in section 3.1, one obtains the following sensitivity equations, for every $i = 1, \dots, N$:

$$\begin{cases} \partial_t u_{p_i} - b\partial_x^2 u_{p_i} + c\partial_x u_{p_i} = s_{p_i}(x,t;\mathbf{p}) & \text{in } (x_a, x_b),\ 0 < t \le T \\ u_{p_i}(x_a, t) = 0 & 0 < t \le T \\ \partial_x u_{p_i}(x_b, t) = 0 & 0 < t \le T \\ u_{p_i}(x, 0) = 0 & \text{in } (x_a, x_b) \end{cases} \tag{3.6}$$

Let us observe that the sensitivity equations (3.6) are independent one from the other and all from the state. Moreover, the same solver can be used for all of them.

On the other hand, for the nonlinear case we consider the following equation:

$$\begin{cases} \partial_t u - b\partial_x^2 u + cu\partial_x u = s(x,t;\mathbf{p}) & \text{in } (x_a, x_b),\ 0 < t \le T \\ u(x_a, t) = f_D(t) & 0 < t \le T \\ \partial_x u(x_b, t) = f_N(t) & 0 < t \le T \\ u(x, 0) = g(x) & \text{in } (x_a, x_b). \end{cases} \tag{3.7}$$

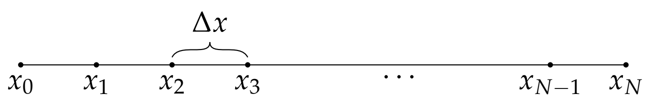
We will impose source term, initial and boundary condition such as $u(x,t) > 0\ \forall x \in (x_a, x_b),\ \forall t \in (0, T)$. In this way the advection field cu is such that x_a is always the inlet and x_b the outlet, and we can keep the same boundary condition and, most important thing, the same upwind scheme for the convective term, introduced in the next Section. However, in this case the sensitivity equations are:

$$\begin{cases} \partial_t u_{p_i} - b\partial_x^2 u_{p_i} + cu\partial_x u_{p_i} + cu_{p_i}\partial_x u = s_{p_i}(x,t;\mathbf{p}) & \text{in } (x_a, x_b),\ 0 < t \le T \\ u_{p_i}(x_a, t) = 0 & 0 < t \le T \\ \partial_x u_{p_i}(x_b, t) = 0 & 0 < t \le T \\ u_{p_i}(x, 0) = 0 & \text{in } (x_a, x_b). \end{cases} \tag{3.8}$$

These are linear PDEs of reaction-advection-diffusion with the advection coefficient $cu(x,t)$ and the reaction coefficient $c\partial_x u(x,t)$ depending on space and time. In this case it is not possible to use the same solver for state and sensitivity, moreover the sensitivity equations depend on the solution of the state.

3.2.2 Numerical schemes

We now introduce the numerical schemes used to solve the problems (3.5)-(3.6)-(3.7)-(3.8). To implement them, a C++ code has been developed, described in appendix A. In every case, the space has been discretized with a uniform grid $\{x_i\}_{i=0}^N$ where $x_i = x_a + i\Delta x$ and Δx is the spatial step, that can be chosen by the user.



Also in time, we considered a uniform discretization: $\{t^n\}_{n=0}^K$ where $t^n = n\Delta t$ and Δt is chosen such as two stability conditions are satisfied:

$$\Delta t < \frac{1}{2}\frac{\Delta x^2}{b} \qquad \wedge \qquad \Delta t < \frac{\Delta x}{c}. \tag{3.9}$$

These conditions (3.9) are necessary, since we use explicit methods in time, as shown in the next subsections.

A first order and a second order scheme have been implemented, by using finite differences in space for both orders, while in time explicit Euler for first order, and a Runge-Kutta scheme for second order. See [Qua09] and [LeV07] for more details on these schemes. The resulting schemes are the following:
all of them are initialized using the initial condition and the Dirichlet boundary condition:

$$u_i^0 = g(x_i) \quad \forall i = 0, \dots, N \qquad u_0^n = f_D(t^n) \quad \forall n = 0, \dots, K.$$

- **First order scheme for the linear case** $\forall\ n = 1, \dots, K, \quad \forall\ i = 1, \dots, N-1$:

$$u_i^{n+1} = u_i^n - \Delta t\ c\ \frac{u_i^n - u_{i-1}^n}{\Delta x} + \Delta t\ b\ \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} + \Delta t\ s(x_i, t^n) \tag{3.10}$$

  Neumann boundary condition $i = N$:

$$u_N^{n+1} = u_{N-1}^{n+1} + \Delta x f_N(t^{n+1})$$

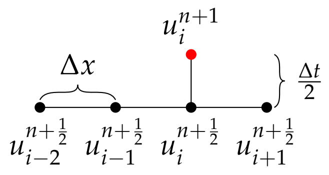
- **Second order scheme for the linear case** $\forall\ n = 1, \dots, K, \quad \forall\ i = 2, \dots, N-1$:

$$\begin{cases} u_i^{n+\frac{1}{2}} = u_i^n & -\frac{\Delta t}{2}\ c\ \frac{3u_i^n - 4u_{i-1}^n + u_{i-2}^n}{2\Delta x} + \frac{\Delta t}{2}\ b\ \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} + \frac{\Delta t}{2} s(x_i, t^n) \\ u_i^{n+1} = u_i^n & -\Delta t\ c\ \frac{3u_i^{n+\frac{1}{2}} - 4u_{i-1}^{n+\frac{1}{2}} + u_{i-2}^{n+\frac{1}{2}}}{2\Delta x} + \\ & +\Delta t\ b\ \frac{u_{i+1}^{n+\frac{1}{2}} - 2u_i^{n+\frac{1}{2}} + u_{i-1}^{n+\frac{1}{2}}}{\Delta x^2} + \Delta t\ s(x_i, t^{n+\frac{1}{2}}) \end{cases} \tag{3.11}$$

  Neumann boundary condition $i = N$:

$$u_N^{n+1} = \frac{1}{3}(4u_{N-1}^{n+1} - u_{N-2}^{n+1} + 2\Delta x\ f_N(t^{n+1}))$$

Since the numerical dependency stencil for the internal nodes ($\forall i = 1, \dots, N-1$) is the following:



u_i^{n+1}
Δx
$\frac{\Delta t}{2}$
$u_{i-2}^{n+\frac{1}{2}}$
$u_{i-1}^{n+\frac{1}{2}}$
$u_i^{n+\frac{1}{2}}$
$u_{i+1}^{n+\frac{1}{2}}$

for the computation of u_1^{n+1} it is necessary to know the value of the solution in a point outside the domain. Different choices for this node lead to different schemes. Unless otherwise indicated, in this work we decided to impose $u_{-1}^n = u_0^n$.

With these two schemes (3.10)-(3.11) it is possible to solve the linear state equation (3.5) and the linear sensitivity equations (3.6).

For the nonlinear state equation (3.7) the scheme is very similar to the one shown above, the only difference is in the advection term.

- **First order scheme for the nonlinear case** $\forall\, n = 1, \ldots, K, \quad \forall\, i = 1, \ldots, N-1$:

$$u_i^{n+1} = u_i^n - \Delta t\, c u_i^n \frac{u_i^n - u_{i-1}^n}{\Delta x} + \Delta t\, b\, \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} + \Delta t\, s(x_i, t^n) \tag{3.12}$$

- **Second order scheme for the nonlinear case** $\forall\, n = 1, \ldots, K, \quad \forall\, i = 2, \ldots, N-1$:

$$\begin{cases} u_i^{n+\frac{1}{2}} = u_i^n & -\frac{\Delta t}{2}\, c u_i^n\, \frac{3u_i^n - 4u_{i-1}^n + u_{i-2}^n}{2\Delta x} + \frac{\Delta t}{2}\, b\, \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} + \frac{\Delta t}{2} s(x_i, t^n) \\ u_i^{n+1} = u_i^n & -\Delta t\, c u_i^{n+\frac{1}{2}}\, \frac{3u_i^{n+\frac{1}{2}} - 4u_{i-1}^{n+\frac{1}{2}} + u_{i-2}^{n+\frac{1}{2}}}{2\Delta x} + \\ & +\Delta t\, b\, \frac{u_{i+1}^{n+\frac{1}{2}} - 2u_i^{n+\frac{1}{2}} + u_{i-1}^{n+\frac{1}{2}}}{\Delta x^2} + \Delta t\, s(x_i, t^{n+\frac{1}{2}}) \end{cases} \tag{3.13}$$

  Note that, since the numerical dependency stencil is the same as in the linear case, we have the same problem for computation of the solution in the first node x_1.

Finally, to solve the equation (3.8), the linear schemes can be used, with two simple modification. The first one is that the coefficient c, constant in (3.10)-(3.11), has to be substituted with c_i^n in (3.10) and in the first equation of (3.11), and with $c_i^{n+\frac{1}{2}}$ in the second equation of (3.13), being $c_i^n \simeq c u(x_i, t^n)$. Note that the equation is linear since we are solving the sensitivity equations in u_{p_k} and $u(x,t)$ is given. The second one is that the reaction term $c(\partial_x u)_i^n (u_{p_k})_i^n$ must be added. Let us observe that in this case, for the second order scheme, it is necessary to know the state u on a temporal grid twice as fine as the one used for the sensitivity equations.

3.3 Convergence of the finite differences schemes

In this section we will present some numerical convergence results in order to verify the actual order of the finite differences schemes in some simple cases. This section is not meant to be exhaustive on the convergence topic, but only a brief check in some simple cases in which it is easy to compute the analytical solution. First let us observe that the discretization of the diffusion term does not change from the first to the second order scheme, therefore we focused on only advection problems.

3.3.1 Advection problem with a steady solution

The first simple case is the following:

$$\begin{cases} \partial_t u + c\,\partial_x u = x & \text{in } (0,1),\ 0 < t \leq T \\ u(0,t) = 0 & 0 < t \leq T \\ u(x,0) = 0 & \text{in } (0,1). \end{cases} \tag{3.14}$$

Since both the source term and the Dirichlet boundary condition do not depend on time, there will be a steady solution, that is $u_{ex} = \frac{x^2}{2c}$. If the final time T is big enough, also the numerical solution will be steady. The error is computed as follows:

$$e(T) = \|u_{ex}(x,T) - u_h(x,T)\|_{L^2(0,1)},$$

where u_{ex} is the exact solution and u_h the numerical solution. In Figure 3.1 is shown the

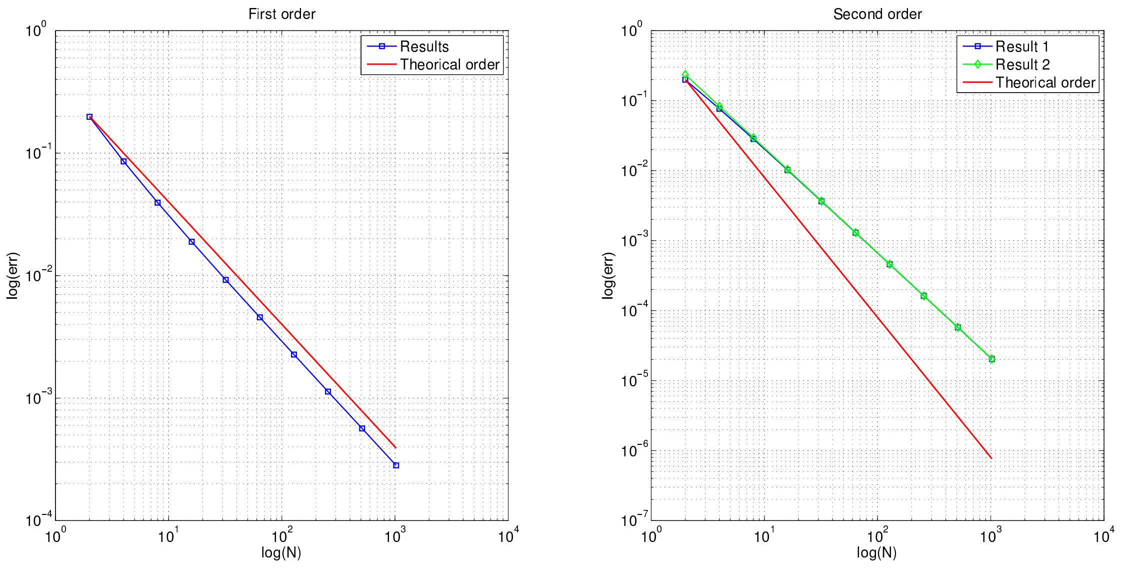


Figure 3.1: Convergence advection steady problem

error with respect to the grid: the theoretical slope is in red. The first order scheme has the expected convergence. For the second order, two different results are plotted: the green one correspond to the choice of $u^n_{-1} = u_{ex}(-\Delta x, t^n)$, the blue one to $u^n_{-1} = f_D(t^n)$. As one can notice, the difference is not significant. However, in both cases the order is not the one expected. This can be explained by the fact that numerically we are imposing two boundary condition to a problem that requires only one. The results obtained removing the Neumann boundary condition and imposing in the last node the same scheme of the internal nodes are shown in Figure 3.2. These results show that the order is the one expected if we use $u^n_{-1} = f_D(t^n)$, otherwise, using the exact solution for the external node leads to a problem that is too simple: the error is comparable to the machine error for all the grids. In fact, we are imposing the exact solution in two nodes (x_{-1} and x_0). Moreover, developing by hand the scheme for the computation of u^n_1 and imposing $u^n_i = u^{n+1}_i$ (because of the stationarity), one finds that the scheme yields $u_1 = \frac{\Delta x^2}{2c}$, that is the exact solution in x_1. Since u_{ex} is a parabola, two nodes inside the domain (plus the external one) are enough to represent the exact solution.

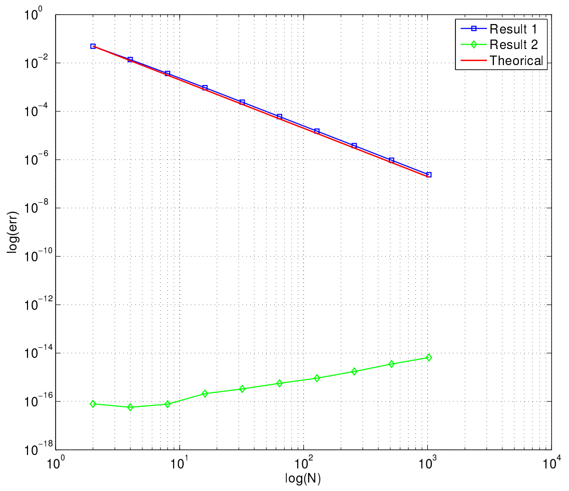


Figure 3.2: Convergence advection steady problem, without Neumann boundary condition

3.3.2 Advection unsteady problem

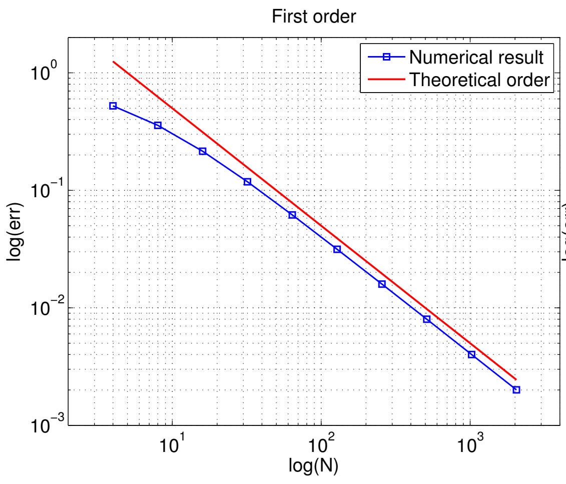
The second simple case considered is the following:

$$\begin{cases} \partial_t u + c\,\partial_x u = 0 & \text{in } (0,1),\ 0 < t \le T \\ u(0,t) = 1 + \alpha \sin(ft) & 0 < t \le T \\ u(x,0) = 0 & \text{in } (0,1). \end{cases} \tag{3.15}$$

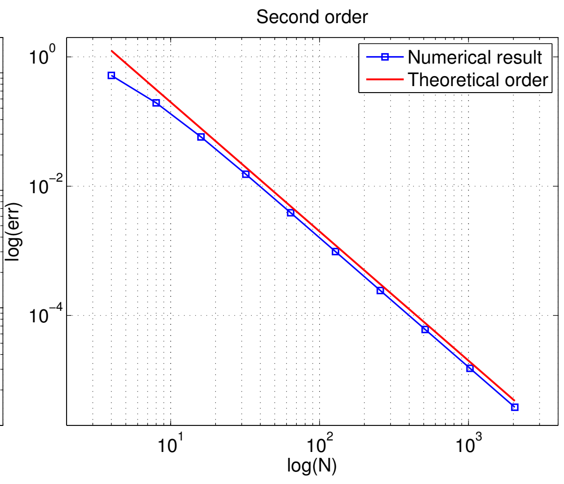
Let us observe that the function $u(x,t) = 1 + \alpha \sin(f(ct - x))$ satisfies the equation and the boundary condition. Moreover, if the time t is big enough (let us say $t > \bar{t}$), the influence of the initial condition is negligible. The results shown in this section are obtained with $c = 1$, $f = \frac{3}{2}\pi$, $\bar{t} = 3$ and $T = 6$. The error is computed as follows:

$$e(t) = \|u_{ex}(x,t) - u_h(x,t)\|_{L^2(0,1)} \qquad \text{err} = \|e(t)\|_{L^\infty(\bar{t},T)},$$

and it is shown in Figure 3.3 with respect to the grid. On the left there is the first order scheme, on the right the second order. The theoretical slope is in red: when the grid becomes fine enough, both schemes have the order expected, for every time considered.



(a) First order scheme



(b) Second order scheme

Figure 3.3: Convergence advection unsteady problem.

3.4 Validation of the CSE method

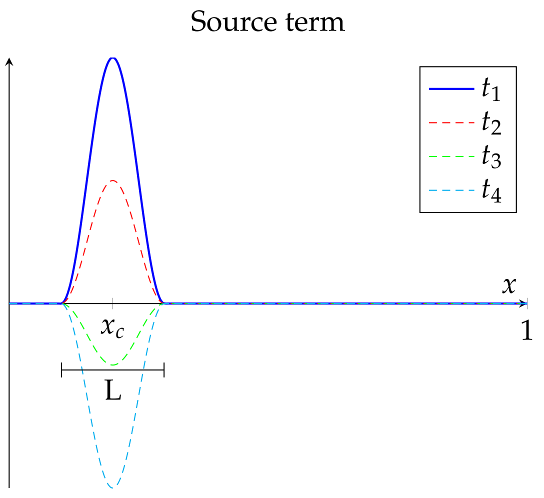
The purpose of this section is to validate the CSE method introduced in Section 3.1. To do that, we will consider the problems (3.5) and (3.7) with specific boundary and initial conditions:

$$f_D(t) = k + \alpha \sin(2\pi f t) \qquad f_N(t) \equiv 0 \qquad g(x) \equiv k \tag{3.16}$$

where α and f are given parameters, k is big enough to guarantee $u(x,t) \geq 0$. We consider the following source term:

$$s(x,t) = \begin{cases} \sqrt{A}\sin(2\pi f t + \varphi)\sin^2(\frac{x-x_c}{L}\pi - \frac{\pi}{2}), & \text{if } x \in (x_c - \frac{L}{2}, x_c + \frac{L}{2}) \\ 0, & \text{otherwise} \end{cases} \tag{3.17}$$

where L is the width of the spatial support, x_c is the centre of the support and A and φ are the design parameters, therefore we have $\mathbf{p} = (A, \varphi)$. The source term at some sample times is shown below.



These being the conditions of the state equation, we now introduce the source terms of the sensitivity equations (3.6) and (3.8):

$$s_A(x,t) = \begin{cases} \frac{1}{2\sqrt{A}} \sin(2\pi f t + \varphi)\sin^2(\frac{x-x_c}{L}\pi - \frac{\pi}{2}), & \text{if } x \in (x_c - \frac{L}{2}, x_c + \frac{L}{2}) \\ 0 & \text{otherwise} \end{cases} \tag{3.18}$$

and

$$s_\varphi(x,t) = \begin{cases} \sqrt{A} \cos(2\pi f t + \varphi)\sin^2(\frac{x-x_c}{L}\pi - \frac{\pi}{2}), & \text{if } x \in (x_c - \frac{L}{2}, x_c + \frac{L}{2}) \\ 0 & \text{otherwise.} \end{cases} \tag{3.19}$$

Note that one must solve each problem ((3.6) and (3.8)) twice: once with (3.18) as source term, and once with (3.19). To validate the method, we observe that a first order Taylor expansion of the solution with respect to the parameters gives:

$$u(A + dA, \varphi) = u(A, \varphi) + dA\, u_A(A, \varphi) + O(dA^2), \tag{3.20a}$$
$$u(A, \varphi + d\varphi) = u(A, \varphi) + d\varphi\, u_\varphi(A, \varphi) + O(d\varphi^2). \tag{3.20b}$$

We now define u_{FD} as the solution computed with the finite differences scheme, and u_{CSE} as an approximation of the solution obtained as follows:

$$u_{CSE}(x, t; p_i + dp_i) = u(x, t; p_i) + dp_i u_{p_i}(x, t; p_i),$$

where p_i can be either A or φ, and $u(x, t; p_i)$ is called *reference solution*. We want to measure the following quantity:

$$\text{diff}(T) := \|u_{FD}(x, T) - u_{CSE}(x, T)\|_{L^2(0,1)},$$

and we expect, due to the Taylor expansion (3.20), that it will be diff$(T) \simeq O(dp_i^2)$. To do that, we varied the design parameters one at a time, following the procedure illustrated in Algorithm 1. Figure 3.4 shows in a logarithmic scale diff$_A$ (on the left) and diff$_\varphi$ (on the right), computed as in line 9-10, Algorithm 1, for the linear case. The

Algorithm 1 Validation of the CSE method

```
1: $A \leftarrow A^{\text{refer}}$, $\varphi \leftarrow \varphi^{\text{refer}}$
2: $r \leftarrow r^{\text{start}}$                              $\triangleright\ r = \text{ratio} = \frac{dA}{A} = \frac{d\varphi}{\varphi}$
3: solve state equation with $(A, \varphi) \longrightarrow$ store solution in $u^{\text{refer}}$
4: solve sensitivity equations with $(A, \varphi) \longrightarrow$ store solutions in $u_A$, $u_\varphi$
5: for $k = 0 \dots N$ do
6:     $dA \leftarrow rA, d\varphi \leftarrow r\varphi$
7:     solve state equation with $(A + dA, \varphi) \longrightarrow$ store solution in $u^{A+dA}$
8:     solve state equation with $(A, \varphi + d\varphi) \longrightarrow$ store solution in $u^{\varphi+d\varphi}$
9:     $\text{diff}_A[k] \leftarrow \|u^{A+dA}(x,T) - u^{\text{refer}}(x,T) - dAu_A(x,T)\|_{L^2(0,1)}$
10:    $\text{diff}_\varphi[k] \leftarrow \|u^{\varphi+d\varphi}(x,T) - u^{\text{refer}}(x,T) - d\varphi u_\varphi(x,T)\|_{L^2(0,1)}$
11:    $r* = 2$
12: end for
```

red line is the theoretical slope. Figure 3.5 shows the same results for the nonlinear case. One can notice that, in the nonlinear case, the slope is the one expected only starting from a certain threshold. This is due to the approach chosen in this work, "differentiate, then discretize", opposed to the other possible approach "discretize, then differentiate": in fact Taylor expansion (3.20) is carried out on the exact solution. By discretizing the equations, one adds discretization error, so:

$$\text{diff}(T) = O(dp_i^2) + O(h^r),$$

where r is the order of the scheme used. The change of slope of the numerical results in Figure 3.5 corresponds to the point in which $O(h^r)$ gets comparable with $O(dp_i^2)$. Note that in this case a spatial step $\Delta x = 0.05$ has been used.

On the contrary, in the "discretize, then differentiate" approach, the Taylor expansion is consistent with the numerical solution computed, whatever grid is used. In the linear case, one can notice that both approaches yield to the same solution, which explains the results.

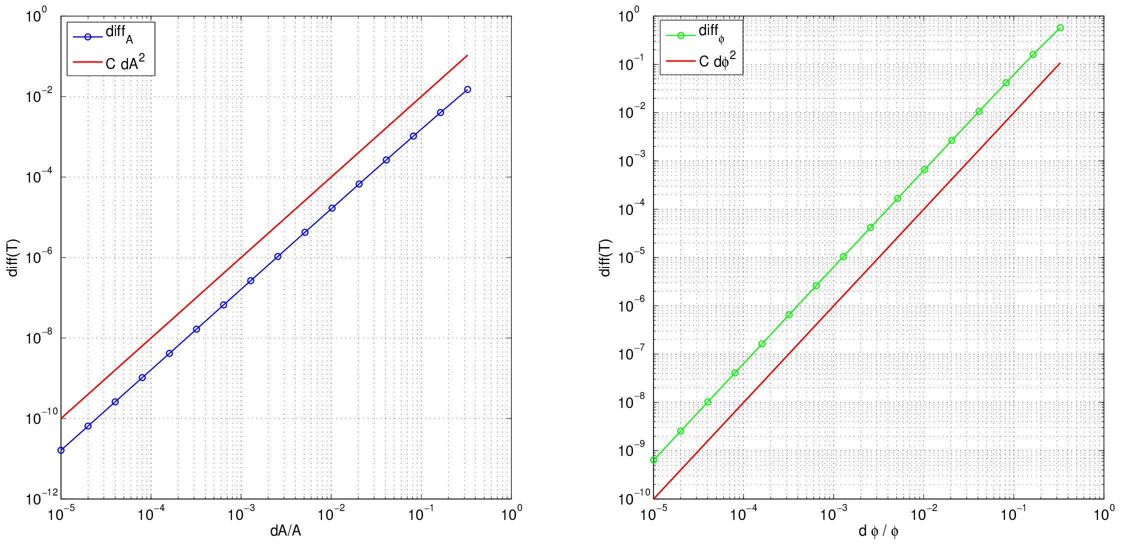


Figure 3.4: Validation of the CSE method, linear case. On the left diff_A, on the right diff_φ. In red the theoretical slope.

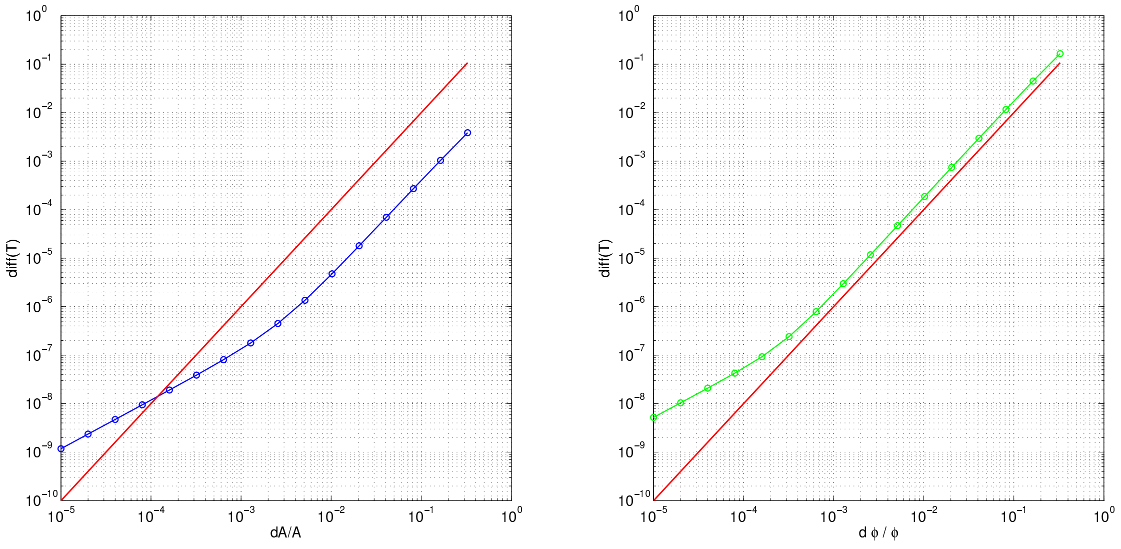


Figure 3.5: Validation of the CSE method, nonlinear case. On the left diff_A, on the right diff_φ. In red the theoretical slope.

4 Numerical results

In this chapter we present the numerical results obtained by applying MGDA III b, to problems governed by PDEs like (3.5) and (3.7), by following the procedure described in Section 3.1.

First of all, we need to introduce the time dependent cost functional J. The same J has been used in all the test cases, and it is the one introduced, as an example, in section 3.1, i.e.:

$$J(\mathbf{p}) = \frac{1}{2}\|\nabla u(\mathbf{p})\|^2_{L^2(\Omega)} = \frac{1}{2}\left(\nabla u(\mathbf{p}), \nabla u(\mathbf{p})\right)_{L^2(\Omega)}. \tag{4.1}$$

We also recall that in this case the gradient of J with respect to the parameters is:

$$\nabla J(\mathbf{p}) = \left(\nabla u(\mathbf{p}), \nabla u_{p_i}(\mathbf{p})\right)_{L^2(\Omega)}, \tag{4.2}$$

whose computation requires to solve the state and the sensitivity equations.

First we will present some results obtained with the *instantaneous approach*, that means, we define the set of objective functionals as:

$$J_k(\mathbf{p}) = J(u(t_k;\mathbf{p})) \quad \forall k = 1, \dots, n,$$

later, we will present a result obtained with the *windows approach*, where:

$$J_k(\mathbf{p}) = \int_{t_k}^{t_{k+1}} J(u(t;\mathbf{p}))dt \quad \forall k = 1, \dots, n.$$

The solving procedure is schematically recalled in the Algorithm 2, in the case of instantaneous approach. Note that the algorithm is general, the only significant difference for the windows approach is in the evaluation of the cost functionals J_k (line 8).

In section 3.4 we introduced some specific source terms, boundary and initial conditions: unless otherwise indicated, all the results of this chapter are obtained with the same data, with fixed values of the non-optimized parameters, reported in Table 4.1.

These choices for the values of the parameters are related to the choice of the time and grid steps: in order to have a spatial grid fine enough to represent accurately the source term we chose

$$\Delta x = \frac{L}{20},$$

that is, being $L = 0.2$, $\Delta x = 0.01$. For the temporal step, we recall that it has to satisfy

Algorithm 2 Solving procedure

```
1:  $\mathbf{p} \leftarrow \mathbf{p}^{start}$
2:  while convergence do
3:      solve state equation with $\mathbf{p} \longrightarrow$ store solution in $u$
4:      if Nonlinear then
5:          set $u$ as advection-reaction field in sensitivity equations
6:      end if
7:      solve sensitivity equations with $\mathbf{p} \longrightarrow$ store solution in $u_{p_i}$
8:      $J_k \leftarrow J(u(t_k))$
9:      compute $\nabla u(t_k)$ and $\nabla u(t_k)_{p_i}$
10:     compute $\nabla_{\mathbf{p}} J_k$ as in (4.2)
11:     $\nabla_{\mathbf{p}} J_k \longrightarrow$ MGDA $\longrightarrow \boldsymbol{\omega}$
12:     $\rho \leftarrow \rho^{start}$
13:     $\mathbf{p}^{temp} \leftarrow \mathbf{p} - \rho\boldsymbol{\omega}$
14:     if $\mathbf{p} \notin \mathcal{P}$ then correction
15:     end if
16:     solve state equation with $\mathbf{p}^{temp} \longrightarrow$ store solution in $u^{temp}$
17:     $J_k^{temp} \leftarrow J(u^{temp}(t_k))$
18:     if $J_k^{temp} < J_k \ \forall k$ then
19:         update: $\mathbf{p} \leftarrow \mathbf{p}^{temp}$
20:     else
21:         $\rho \leftarrow \alpha\rho$                          ▷ $\alpha \in (0,1)$, fixed parameter
22:         go to line 13
23:     end if
24: end while
```

Table 4.1: Values of non-optimized parameters

Parameter	Symbol	Value
frequency	f	4
source centre	x_c	0.2
source width	L	0.2
b.c. average	k	5
b.c amplitude	α	1

the stability condition (3.9).

As we said in the first chapter, unless all the cost functionals have their minimum in the same point, there is not a unique optimal solution: this means that running the multi-objective optimization algorithm from different starting points $\mathbf{p}^{\text{start}}$ we will obtain different optimal points. For this reason, first we find the optimal points of the single cost functionals $\mathbf{p}_k^{\text{opt}}$ (using, for instance, a steepest descent method), and in a second time we run MGDA from different starting points around the zone identified by the $\mathbf{p}_k^{\text{opt}}$.

Let us observe that, being the vector of the optimization parameters composed by an amplitude and a phase, i.e. $\mathbf{p} = (A, \varphi)$, the set of admissible parameters will be:

$$\mathcal{P} = (0, +\infty) \times [0, 2\pi], \tag{4.3}$$

that is an open set, therefore there is no theoretical proof of existence and uniqueness of the minimum of the single cost functional J. However, we verified numerically case by case at least the existence. Moreover, every time the parameters are updated it is controlled whether or not they are in the admissible set (see Algorithm 2, line 14): if the phase falls outside $[0, 2\pi]$ it is set to a value on the boundary (for instance, if $\varphi < 0 \Rightarrow \varphi^{new} = 2\pi$), while if the amplitude is negative it is set to a threshold, chosen by the user.

Finally, while running the single objective optimization algorithm, we noticed an anisotropy problem: the functionals are much flatter in A then in φ, i.e. to have a significant difference in the value of the cost functionals, big variations of A are necessary (big if compared with the variation necessary for φ to obtain the same difference for the cost functionals), and this can affect significantly the numerical optimization method. To solve this problem quickly, we multiplied by a factor $\mu > 1$ the source term (3.17) and, consequently, the source terms of the sensitivities. This is equivalent to a changement of variable, therefore we are now optimizing with respect to a new parameter A^{new}, related to the old one as follows:

$$\mu\sqrt{A^{new}} = \sqrt{A^{old}}. \tag{4.4}$$

From (4.4), it is clear that, if μ is big enough, big variations of A^{old} correspond to small variations of A^{new}.

4.1 Instantaneous approach

We start by considering the instantaneous approach: first, it requires to chose suitable (and interesting) times for the evaluation of the cost functionals. Let us observe that, since in our case the Dirichlet boundary condition and the source term are both periodic, with the same frequency $f = 4$, we expect also the solution u, and therefore the cost functional J, to be periodic with frequency $f = 4$. For this reason it makes sense to fix all the instant t_k considered inside the same period. Moreover, we would like to skip the transient part of the phenomena. This given, the choice made for the instant is the one in Table 4.2. Let us observe that it would be better if the instant considered t_k

Table 4.2: Instant considered

t_1	t_2	t_3	t_4	t_5	t_6	t_7
3.00	3.04	3.08	3.12	3.16	3.20	3.24

were actual points of the temporal grid, i.e. if $\exists m \in \mathbb{N} : t_k = m\Delta t \ \forall k$.

4.1.1 Linear case

For the linear case (3.5)-(3.6) we chose the following values for the advection and diffusion coefficients, and for μ:

$$c = 1 \qquad b = 0.01 \qquad \mu = 10.$$

The values of the parameters corresponding to the minimum of the single objective functionals are shown in Table 4.3. As one can notice, all the values are very close to each other, therefore it is not an interesting case from a multi-objective optimization point of view. Anyway, in Figure 4.1 it is shown the final result: the red points correspond to the optimal value of the parameters for each functional; each blue point, instead, correspond to one iteration of the **while** loop of Algorithm 2, this means that for each blue point at least three equations have been solved (in fact in the cases in which the control of line 18 does not succeed it is necessary to solve the state equation more than once). From Figure 4.1 we can observe that there is a region strictly contained in the convex hull of the "red points" in which all the points are Pareto-stationary. Moreover, for smaller amplitudes (0.875 $\sim$ 0.885), it is possible to identify quite clearly the border of the set of Pareto-stationary points. However, being the region of the Pareto-stationary parameters so small, it is not possible to appreciate a significant decrement in the cost functional. In Figure 4.2 it is shown the cost functional J with respect to time, over a period, for three different choices of parameters: one outside the Pareto-stationarity zone, one inside and one on the border. As one can notice, it is almost impossible to distinguish one from the other: to do that it is necessary to zoom in, as done for Figure 4.3. However, it is clear that this is not a particularly interesting case, therefore from now on we will consider nonlinear equation.

Table 4.3: Single-objective optimization results - linear case

time $t_k[s]$	3.00	3.04	3.08	3.12	3.16	3.20	3.24
A	0.907108	0.882811	0.876467	0.906197	0.887208	0.873657	0.904139
φ	5.12258	5.11216	5.12904	5.1204	5.1116	5.12704	5.12732

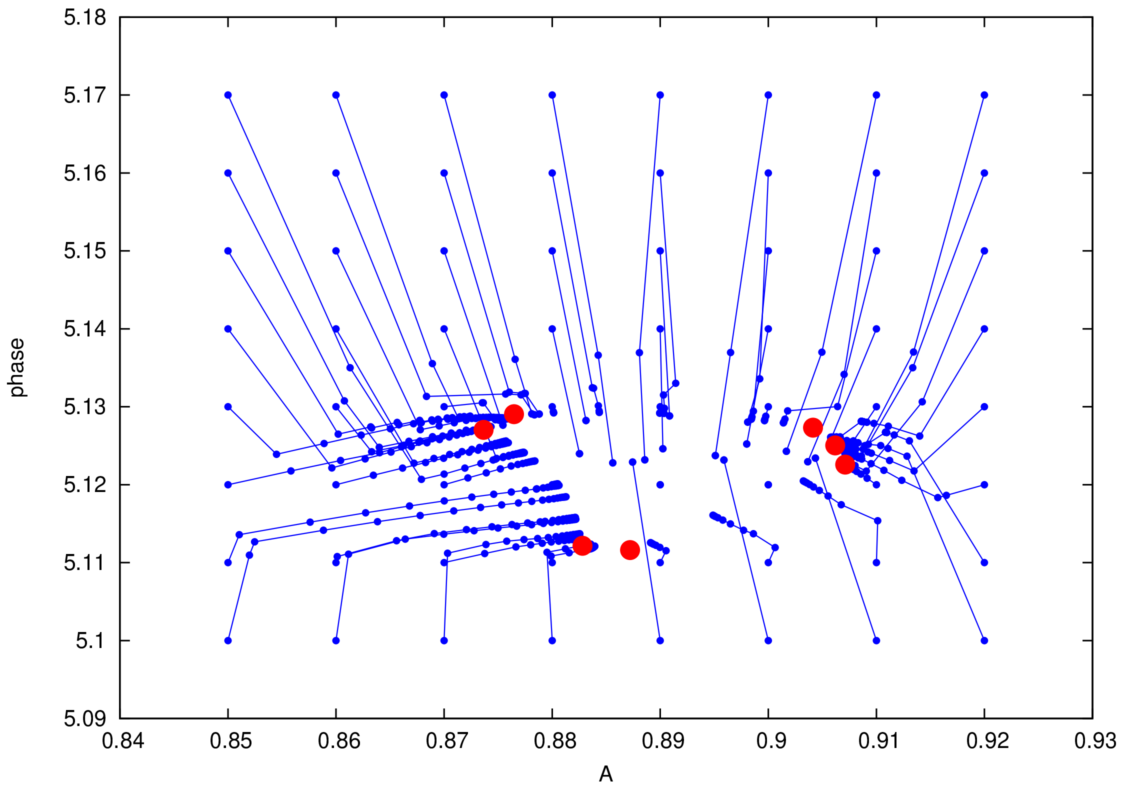


Figure 4.1: Result of MGDA applied to the linear case from different starting points.

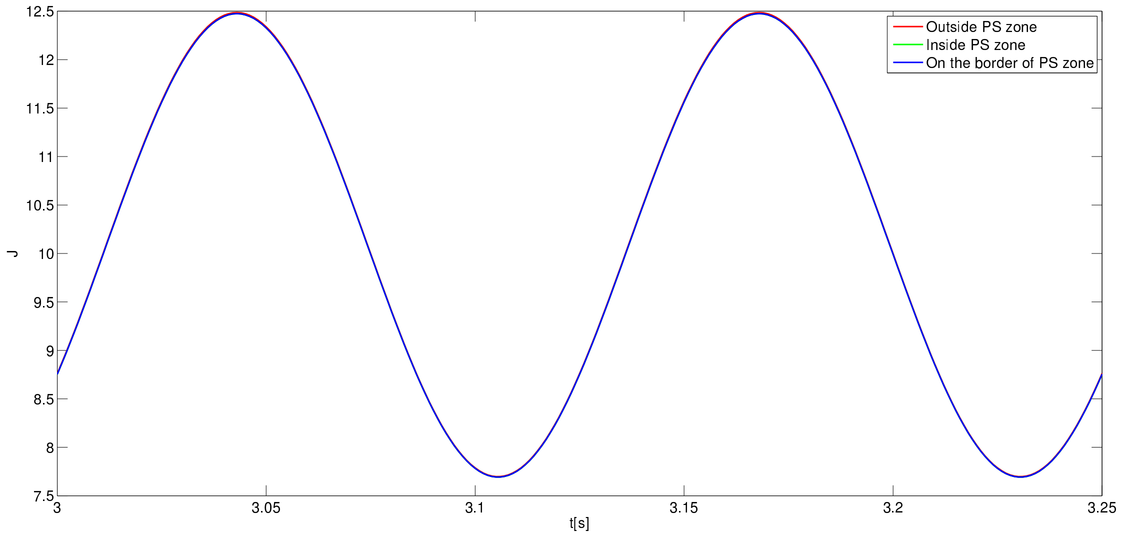


Figure 4.2: Cost functional for different parameters with respect to time, over a period.

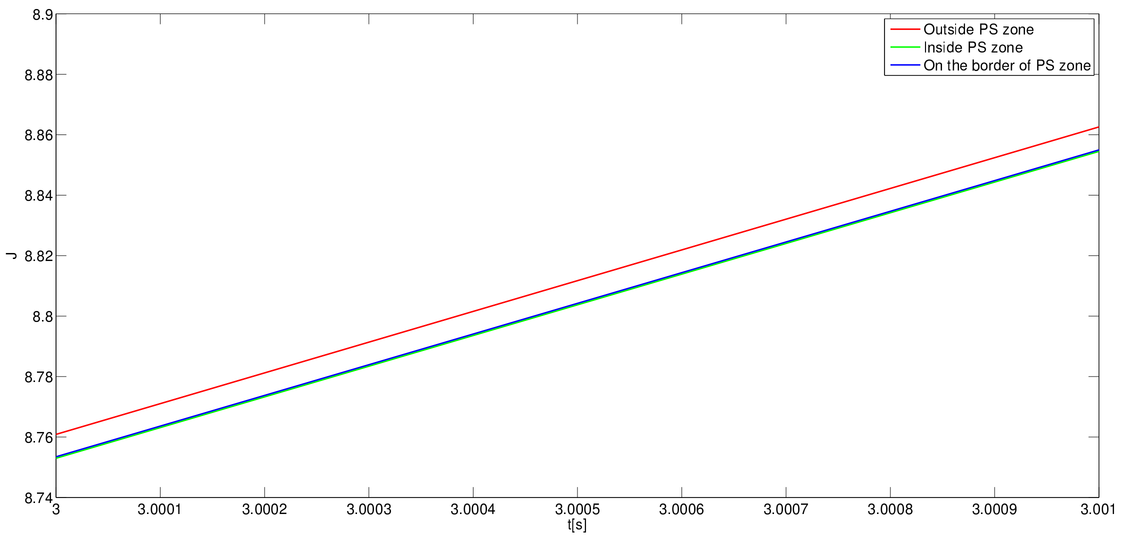


Figure 4.3: Cost functional for different parameters with respect to time - zoom.

4.1.2 Nonlinear case

For the nonlinear case (3.7)-(3.8) we chose the following values for advection and diffusion coefficients, and for μ:

$$c = 1 \qquad b = 0.1 \qquad \mu = 50.$$

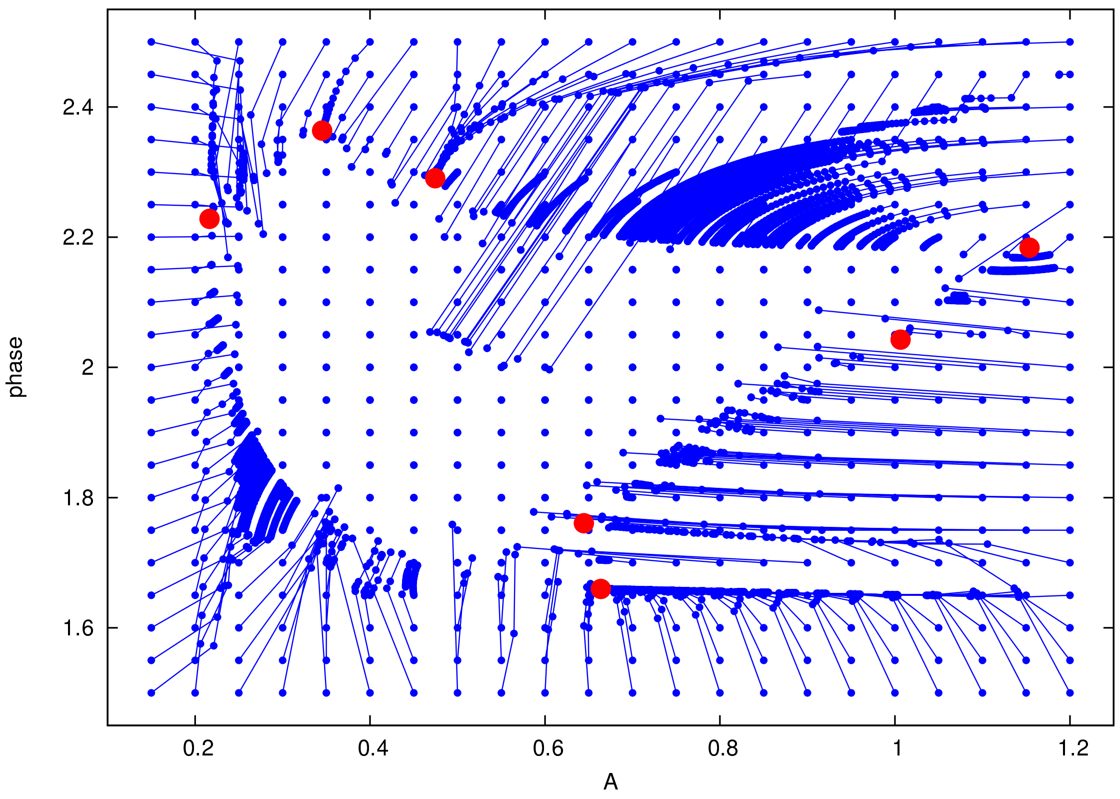
We followed the same procedure as in the linear case, therefore at first we computed the optimal points of the single cost functionals, reported in Table 4.4 and drawn in red in Figure 4.4. In this case, the optimal points found are much more varied, compared to the linear case, therefore from a multi-objective optimization point of view it is a more interesting case. For this reason we decided to run MGDA with both the ordering criteria proposed in Section 2.3.2 to see the differences. The results are shown in Figure 4.4. In order to analyse the differences, we briefly recall the ordering criteria:

· Criterion (2.2) selects the first gradient as:

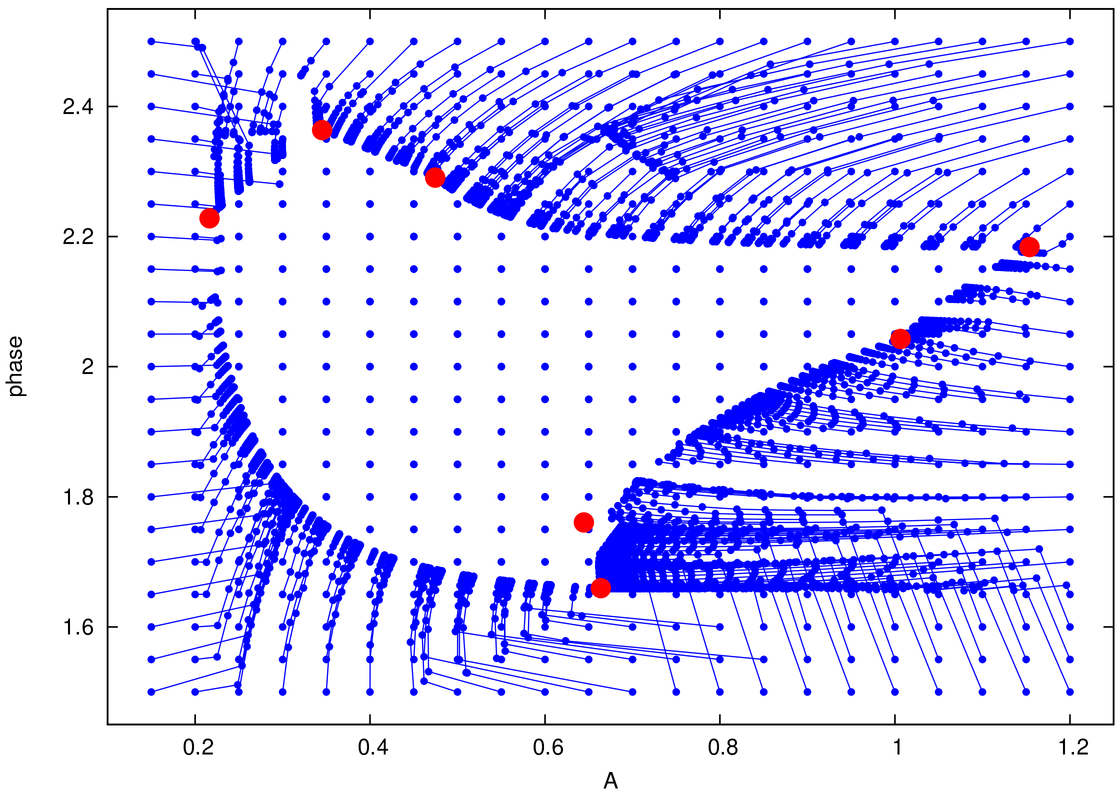
$$\mathbf{g}_{(1)} = \mathbf{g}_k \text{ where } k = \underset{i=1,\dots,n}{\operatorname{argmax}} \min_{j=1,\dots,n} \frac{(\mathbf{g}_i, \mathbf{g}_j)}{(\mathbf{g}_i, \mathbf{g}_i)}, \tag{4.5}$$

Table 4.4: Single-objective optimization results - nonlinear case

time $t_k[s]$	3.00	3.04	3.08	3.12	3.16	3.20	3.24
A	0.34512	1.00652	0.644476	0.47365	1.15389	0.663846	0.216462
φ	2.36373	2.04267	1.76051	2.29071	2.1838	1.65968	2.22826



(a) criterion (4.5).



(b) criterion (4.6).

Figure 4.4: Result of MGDA applied to the nonlinear case from different starting points.

while criterion (2.15) select the first as:

$$\mathbf{g}_{(1)} = \mathbf{g}_k \text{ where } k = \underset{i=1,\dots,n}{\operatorname{argmin}} \min_{j=1,\dots,n} \frac{(\mathbf{g}_i, \mathbf{g}_j)}{(\mathbf{g}_i, \mathbf{g}_i)}, \tag{4.6}$$

· The next gradients $\mathbf{g}_{(i)}$ are chosen in the same way from both the criteria, that is:

$$\mathbf{g}_{(i)} = \mathbf{g}_\ell \text{ where } \ell = \underset{j=i,\dots,n}{\operatorname{argmin}} c_{j,j},$$

First, let us observe that the shape of the Pareto-stationary set is qualitatively the same in the two cases, even if in case (4.4b) is much more defined. This means that, globally, the choice of the first gradient does not affect too much the final result. However, it is clear from the Figures that starting from the same point the two criteria leads to different final points. Moreover, in the result shown in Figure 4.4a there seems to be a tendency to converge at the closest single-objective optimal, if there is one close: this can be explained by the fact that the criterion (4.5) tends to select the "middle vector" and therefore is very likely to stop the MGDA loop at the first iteration and to compute $\boldsymbol{\omega}$ on the base only of one gradient; on the other hand, the criterion (4.6) prefers as first vector an external one and therefore is very unlikely to stop MGDA at the first iteration, this implies that in most of the cases the direction $\boldsymbol{\omega}$ is computed on the base of two gradients (obviously, in this case the maximum number of gradients considered from MGDA is two, since it is the dimension of the space).

In Figure 4.5 it is shown the cost functional J with respect to time, for different choices of parameters: the red ones are obtained evaluating the cost functional in some parameters outside the Pareto-stationarity zone, while the green ones correspond to parameters inside the region (see the exact value in Table 4.5). From this Figure it is clear that the functional is periodic and that, sampling the time from only one period, we minimize the cost functional also for all the other times, as expected. In Figure 4.6 there is the same image, zoomed over a period. The vertical lines denote the sampling times: at a first view can seem strange that, even at some sampling times, the green functionals are greater than the red ones. However, this can happen because the notion of Pareto-stationarity is somehow local: in fact it is based on the evaluation of the gradients in that point. What we can assure is the monotonicity of all the cost functionals during the MGDA loop: for instance, considering Figure 4.4, we have monotonicity "along every blue line", but we cannot say anything if we skip from one to the other. In Figures 4.7-4.8-4.9-4.10 it is shown the monotonicity "along the blue lines" in some particular cases: in each subfigure, the red line represents the cost functional evaluated in the starting parameters, while the purple and the green lines are the cost functional evaluated in the convergence points of MGDA obtained using, respectively, criterion (4.5) and criterion (4.6).

Table 4.5: Values of the parameters chosen for Figure 4.5-4.6

A	0.15	0.2	1.0	1.1	0.45	0.4	0.6
φ	2.35	1.6	2.3	1.6	1.85	2.15	2.0
collocation	outside	outside	outside	outside	inside	inside	inside

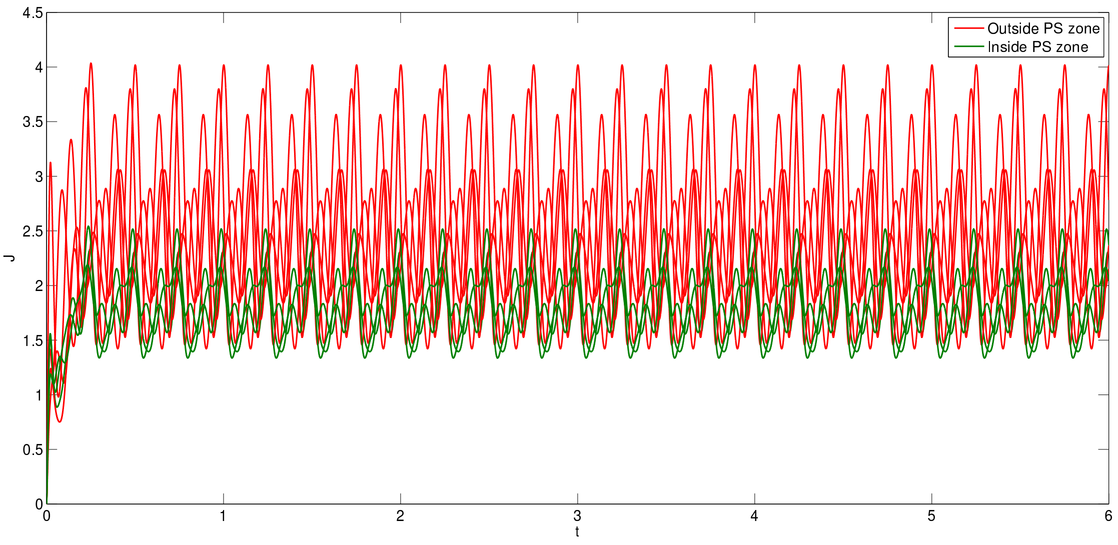


Figure 4.5: Cost functional with respect to time for different parameters values - non-linear case.

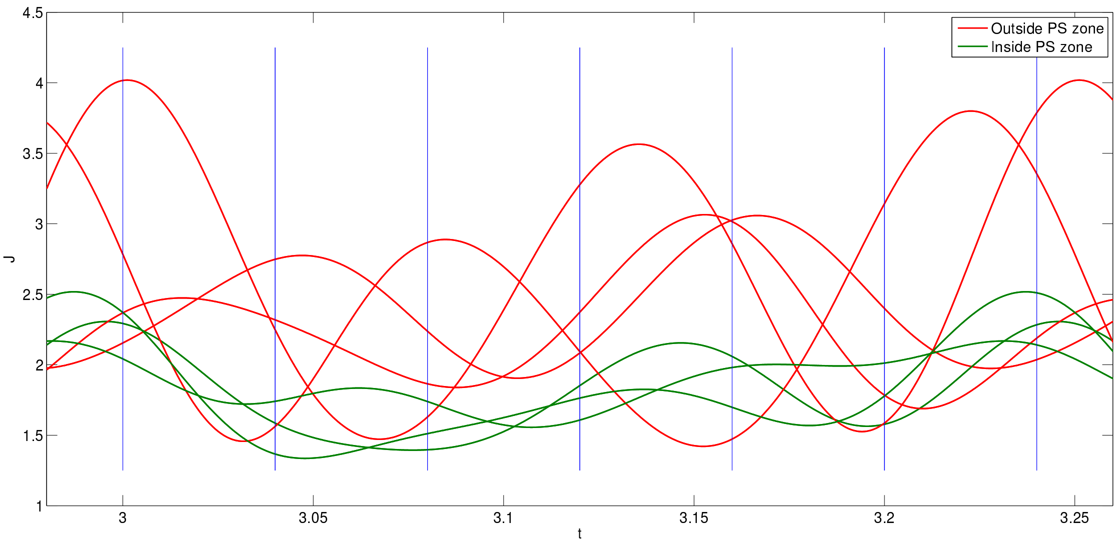


Figure 4.6: Cost functional with respect to time for different parameters values over a period - nonlinear case.

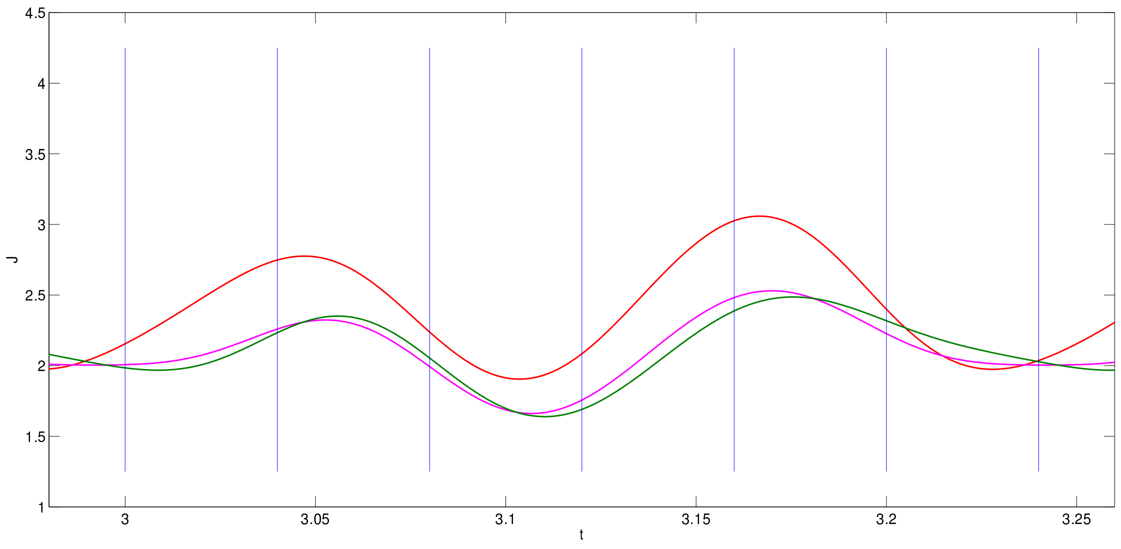


Figure 4.7: Monotonicity of MGDA - starting parameters: $A = 0.15$, $\varphi = 2.35$.

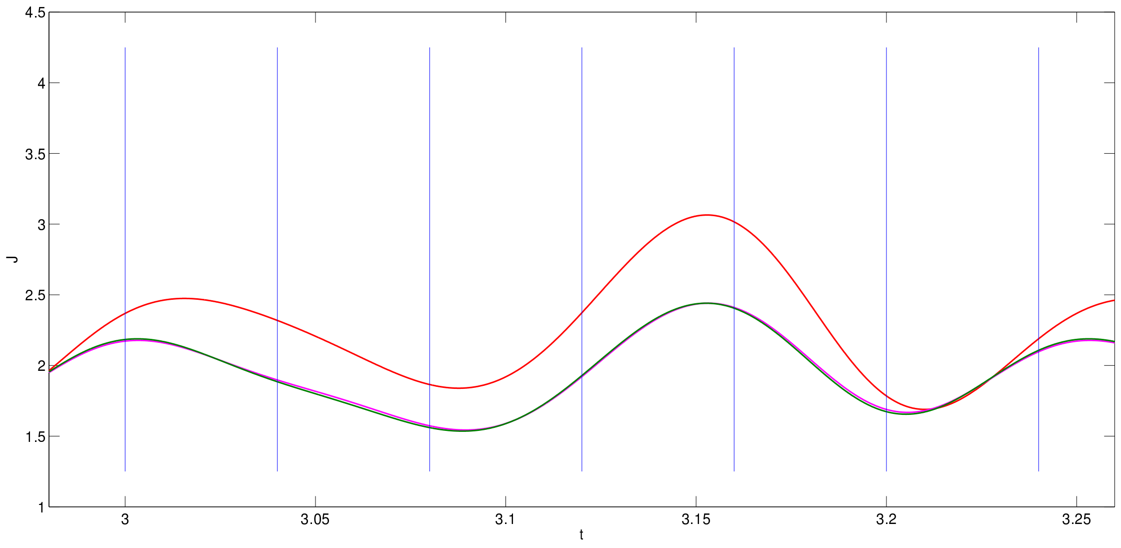


Figure 4.8: Monotonicity of MGDA - starting parameters: $A = 0.2$, $\varphi = 1.6$.

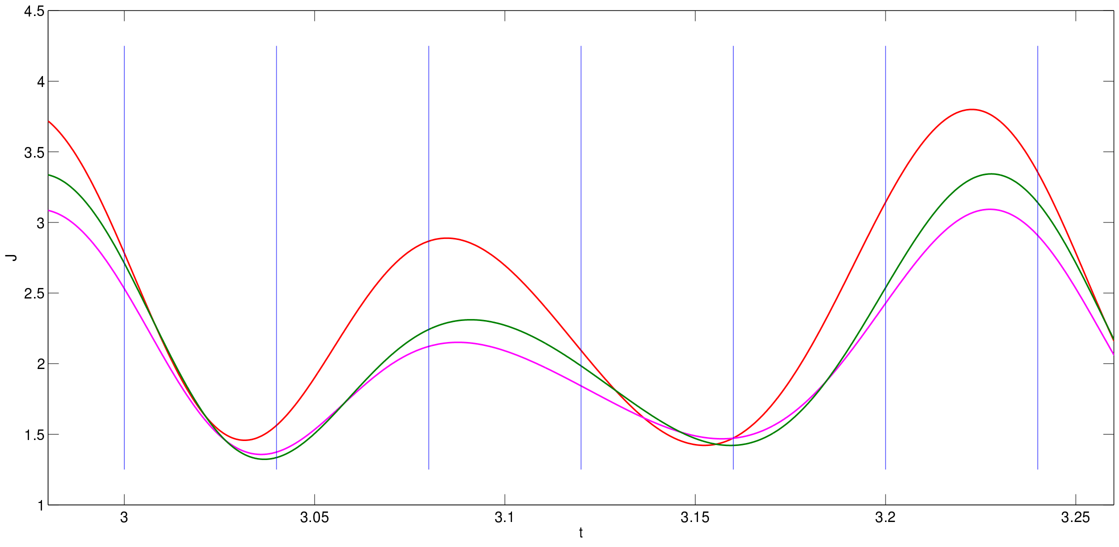


Figure 4.9: Monotonicity of MGDA - starting parameters: $A = 1,\ \varphi = 2.3$.

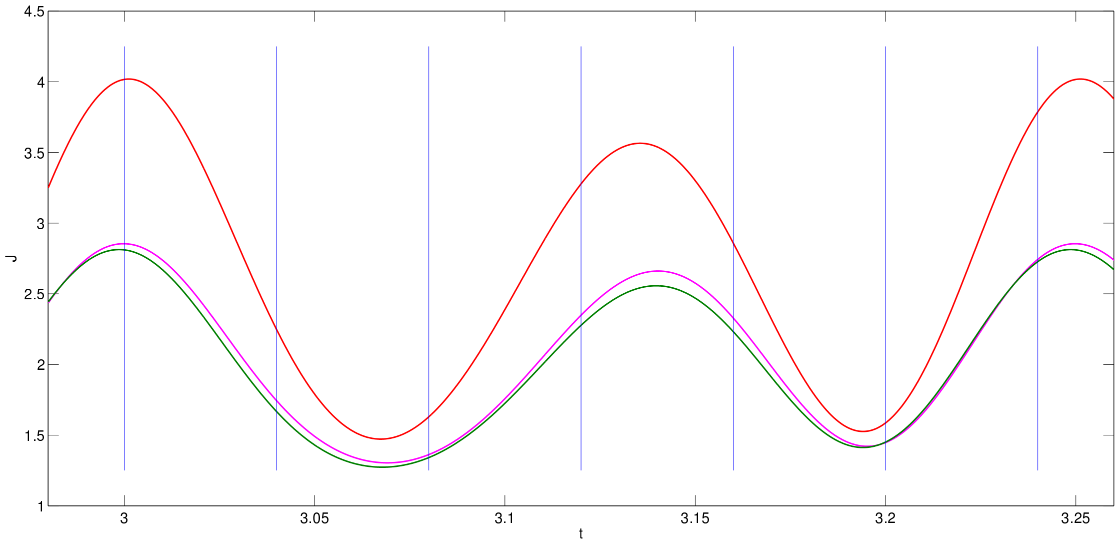
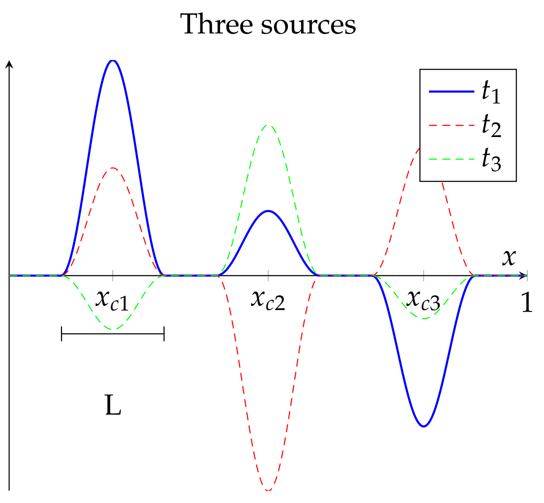


Figure 4.10: Monotonicity of MGDA - starting parameters: $A = 1.1,\ \varphi = 1.6$.

Three sources

In order to test MGDA in dimension higher than 2, we increased the number of parameters adding two source terms, of the same type of (3.17), with different centres x_c. A possible result of the complete source term is shown below.



As one can notice, the width L of the three sources is the same; moreover, also the frequency f is the same (and for this simulation we kept the same values than in the previous sections). In this way there are 6 parameters to optimize (three amplitudes and three phases), with 7 different cost functionals. It is clear that it makes no sense to find the single-objective optimal points and then run MGDA systematically in a grid around these values, because there is no way to visualize the region. Therefore, in this case we run MGDA starting from some random points.

For the first starting point we set all the amplitudes at 0.5 and all the phases at π. Figure 4.11 shows the different cost functionals with respect to the iterations in a logarithmic scale: one can observe that at each iteration none of the functionals increases and at least one of them decreases. In Figure 4.12 are compared the cost functional *pre* and *post* optimization (respectively in red and blue) over a period. In Table 4.6 we report the values of the parameters during the main MGDA loop: one can notice that the second source's amplitude tends to zero.

We here present the same results described above for a different starting point. One can notice that, even in this case, the second source's amplitude tends to zero.

4.2 Windows approach

The final result presented in this work is obtained with the *windows* approach applied to the nonlinear case. In Table 4.8 are reported the intervals (t_k, t_{k+1}) considered in this case. All the non-optimized parameters have the same values used for the nonlinear case, with the simple source, and the criterion (4.6) has been used. In Figure 4.15 are shown in red the optimal points of the single objective optimization at in blue the MGDA iterations, as in the previous cases. The shape of the Pareto-stationary zone is

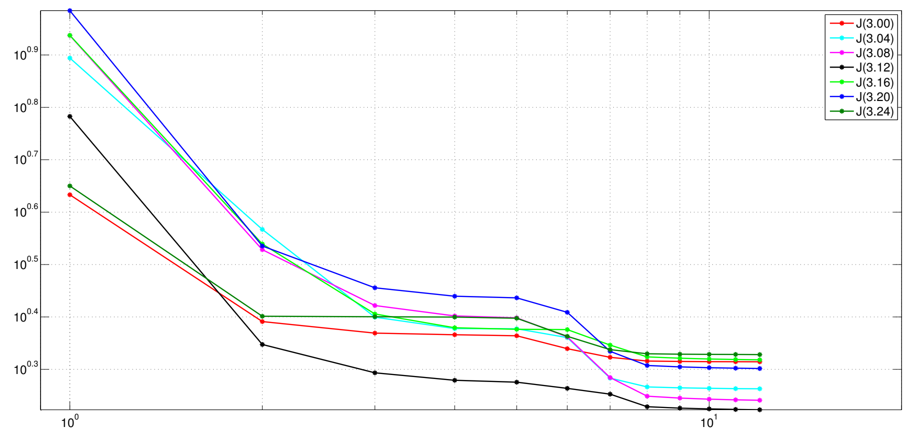


Figure 4.11: Cost functionals with respect to iteration - three sources - symmetric starting point.

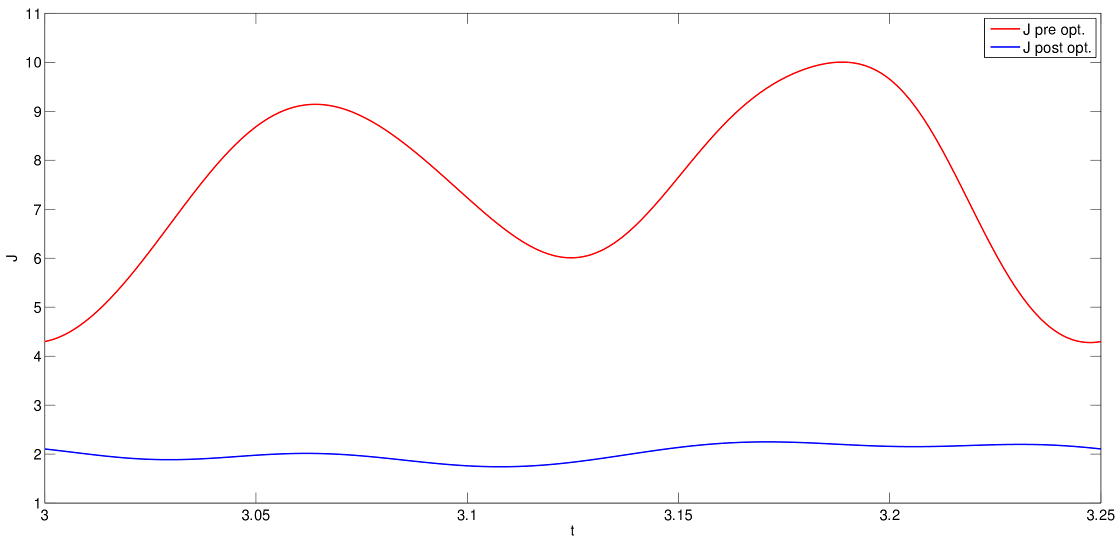


Figure 4.12: Cost functionals with respect to time - three sources - symmetric starting point.

Table 4.6: Parameters during MGDA main loop - three sources - symmetric starting point.

A_1	A_2	A_3	φ_1	φ_2	φ_3
0.5	0.5	0.5	3.14159	3.14159	3.14159
0.326183	0.01	0.497761	2.54601	3.031	3.55003
0.422592	0.0155267	0.276467	2.285	3.02927	3.51984
0.43974	0.0033347	0.248952	2.26425	3.02819	3.51827
0.434359	0.00126382	0.242303	2.26661	3.02778	3.52063
0.366366	0.01	0.126234	2.29823	3.02349	3.55691
0.348423	0.01	0.01	2.13807	3.01764	3.59255
0.349058	0.00062438	0.0102434	2.137	3.01759	3.59259
0.349064	0.000237505	0.0102312	2.13699	3.01759	3.59259
0.349065	9.08649e-05	0.0102276	2.13698	3.01759	3.59259
0.349065	3.4122e-05	0.0102267	2.13698	3.01759	3.59259
0.349065	1.16204e-05	0.0102264	2.13698	3.01759	3.59259

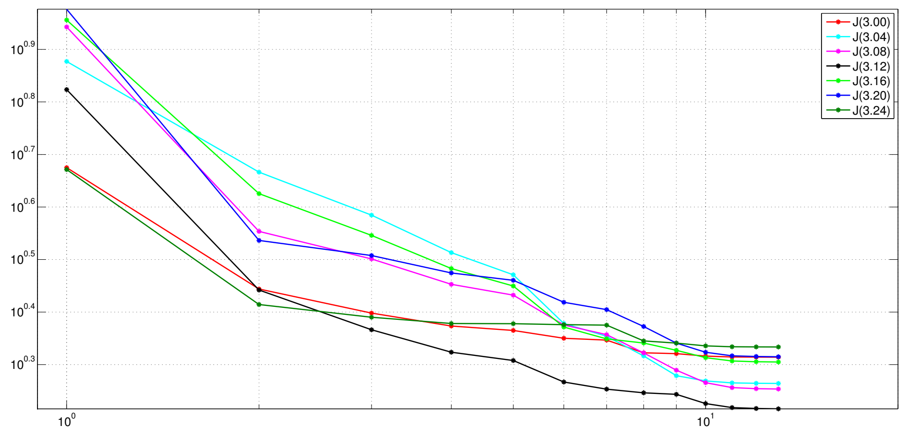


Figure 4.13: Cost functionals with respect to iteration - three sources - asymmetric starting point.

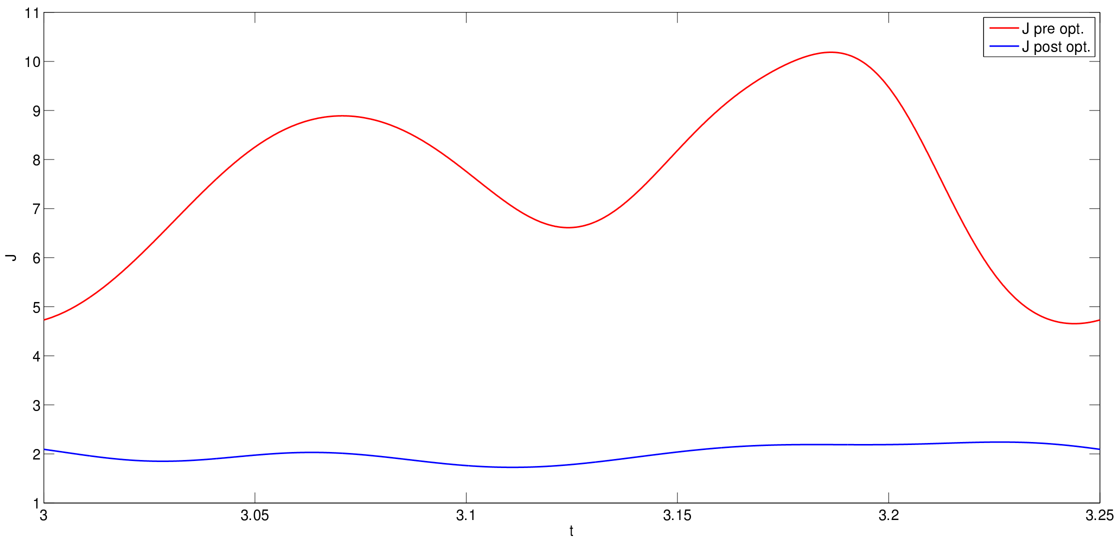


Figure 4.14: Cost functionals with respect to time - three sources - asymmetric starting point.

Table 4.7: Parameters during MGDA main loop - three sources - asymmetric starting point.

A_1	A_2	A_3	φ_1	φ_2	φ_3
0.25	0.5	0.75	3.14159	3.14159	3.14159
0.209686	0.01	0.671903	2.70767	2.9716	3.70705
0.28742	0.01	0.516188	2.56182	2.96944	3.7173
0.32335	0.01	0.385196	2.46421	2.96649	3.7297
0.352622	0.0151327	0.325739	2.40728	2.96619	3.72979
0.416572	0.01	0.209501	2.29981	2.96446	3.73147
0.432156	0.00440427	0.179948	2.2774	2.96396	3.73208
0.367466	0.01	0.0574286	2.25788	2.9599	3.74702
0.369833	0.01	0.0247545	2.1731	2.95696	3.75488
0.37018	0.00183642	0.0246056	2.17192	2.9569	3.75495
0.370197	0.000162565	0.024526	2.17177	2.9569	3.75495
0.370198	2.57116e-05	0.0245225	2.17177	2.9569	3.75495
0.370198	5.76837e-06	0.0245223	2.17177	2.9569	3.75495

Table 4.8: Intervals considered - window approach.

I_1	I_2	I_3	I_4	I_5	I_6
$(3.00, 3.04)$	$(3.04, 3.08)$	$(3.08, 3.12)$	$(3.12, 3.16)$	$(3.16, 3.20)$	$(3.20, 3.25)$

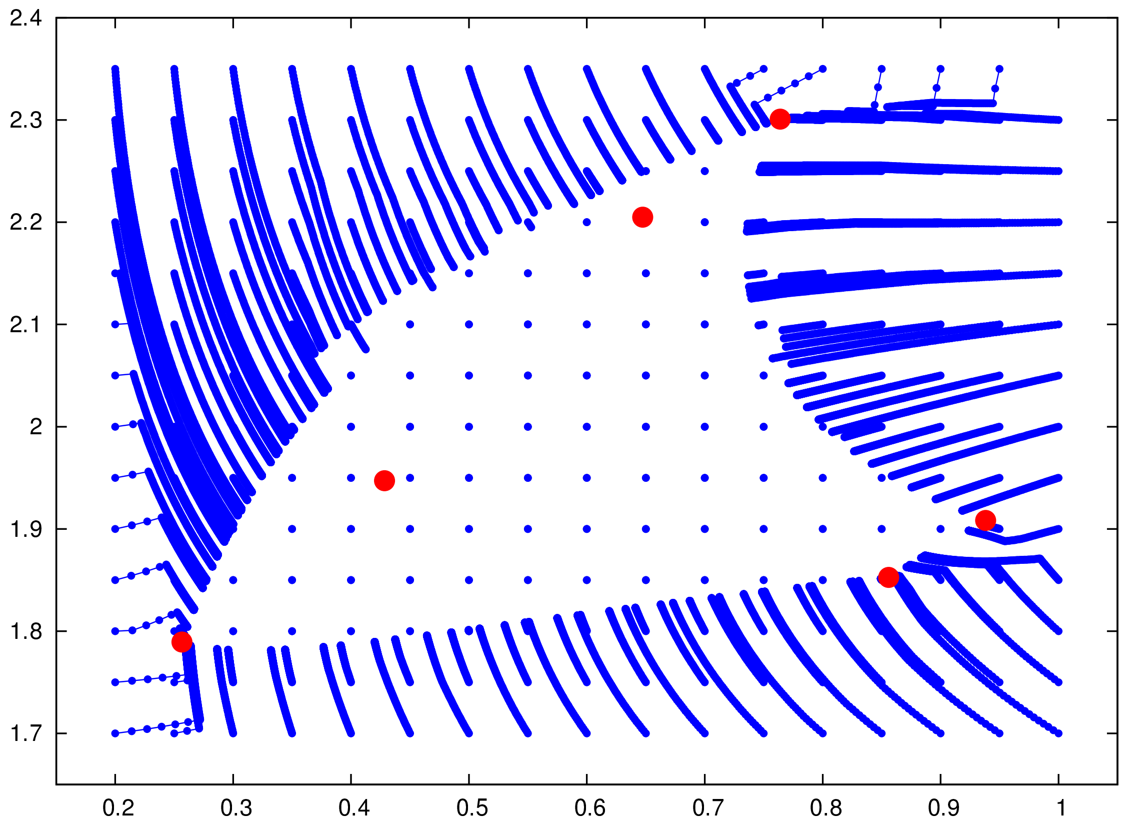


Figure 4.15: MGDA iterations for windows approach.

significantly different from the one obtained in the same case with the instantaneous approach, however, one can say that approximatively in both cases the zone is centred around the same value $(A_c, \varphi_c) \sim (0.6, 2.0)$.

To see how the window approach and the instantaneous one work, and if there is any difference between them, we selected two starting points (in particular $(A_1, \varphi_1) = (1, 1.8)$ and $(A_2, \varphi_2) = (0.6, 2.3)$) that are outside the Pareto-stationary zone in both cases, and we plotted $J(t)$ with respect to time, computed in the optimal parameters obtained with the two approaches. The results are shown in Figure 4.16: what appears from this images is that none of the approaches is better that the other one, it depends on the final result one is interested in. Moreover, it is not fair to compare two methods whose aim is to identify a region of points, only on the base of two or three points in the identified region. The results shown in Figure 4.16 are meant as an example of how MGDA, coupled with different approaches, works. In Tables 4.9-4.10-4.11-4.12 the precise values are reported: J_{pre} correspond to the cost functional *pre* optimization, J_{win} correspond to the cost functional optimized with the window approach and J_{inst} is the one optimized with the instantaneous approach. The values shown in the Tables are a further confirmation of the monotonicity of MGDA. Moreover, we recall that the approaches (windows and instantaneous) are not alternative: in fact, it is possible to consider a set of cost functionals of different nature, some of them generated by instantaneous evaluation of the cost functional, and some generated by an average operation

Table 4.9: Comparing the windows and the instantaneous approach - starting parameters $(A, \varphi) = (1, 1.8)$ - instantaneous values.

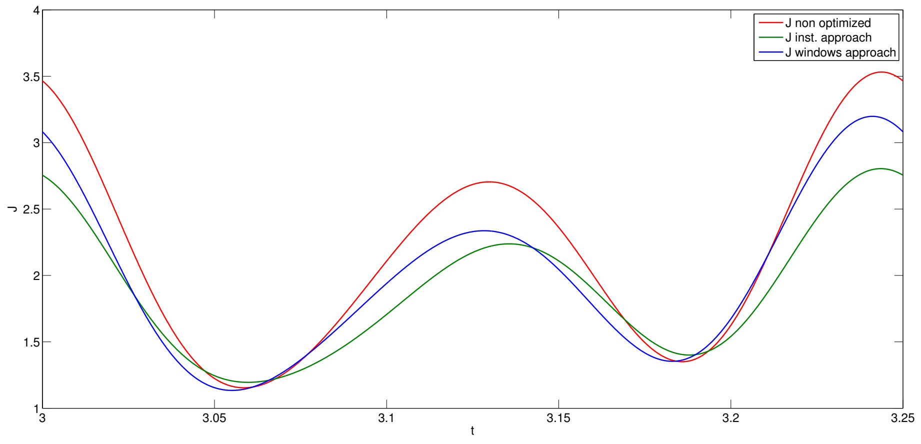
	J_{pre}	J_{win}	J_{inst}
$t_1 = 3.00$	3.4651	3.0819	2.7550
$t_2 = 3.04$	1.4981	1.3320	1.4196
$t_3 = 3.08$	1.4826	1.4694	1.3487
$t_4 = 3.12$	2.6260	2.2950	2.1037
$t_5 = 3.16$	2.0030	1.7714	1.8955
$t_6 = 3.20$	1.6249	1.6738	1.5410
$t_7 = 3.24$	3.5074	3.1961	2.7891

Table 4.10: Comparing the windows and the instantaneous approach - starting parameters $(A, \varphi) = (1, 1.8)$ - averaged values.

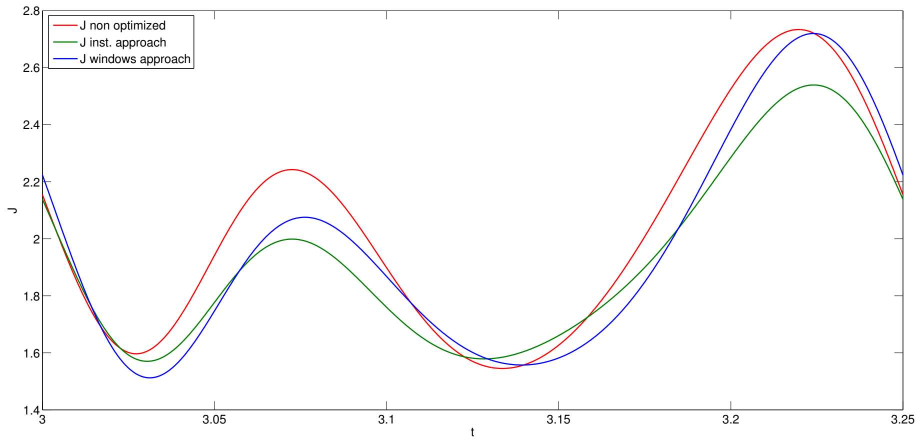
	J_{pre}	J_{win}	J_{inst}
$I_1 = (3.00, 3.04)$	0.1015	0.0883	0.0847
$I_2 = (3.04, 3.08)$	0.0509	0.0495	0.0505
$I_3 = (3.08, 3.12)$	0.0838	0.0770	0.0687
$I_4 = (3.12, 3.16)$	0.1006	0.0871	0.0861
$I_5 = (3.16, 3.20)$	0.0618	0.0597	0.0620
$I_6 = (3.20, 3.25)$	0.1415	0.1339	0.1167

over a window of time.

Finally, we report that, in this case (i.e. with only one source term and a non-linear state equation), the optimal parameters obtained by minimizing with a single objective technique the time-average over the whole period of the cost functional are $(\hat{A}, \hat{\varphi}) = (0.528949, 1.96501)$. We observe that these values are in the middle of both the Pareto-stationarity zones, the one obtained with the instantaneous approach and the one with the windows approach. Moreover, we observe that this single objective approach is equivalent to minimizing the sum of the cost functionals used for the windows approach. In Table 4.13 we report the values (instantaneous and time-averaged over the intervals) of the cost functional computed in the optimal parameters $(\hat{A}, \hat{\varphi})$: as expected, with this approach there are some instant at which the cost functional increases, and also some intervals over which the average increases; this is balanced by other instant, or intervals, in which the reduction of the cost functional value is bigger than the one obtained with the multi-objective approach.



(a) Starting parameters $(A, \varphi) = (1, 1.8)$.



(b) Starting parameters $(A, \varphi) = (0.6, 2.3)$,

Figure 4.16: Comparing the windows and the instantaneous approach.

Table 4.11: Comparing the windows and the instantaneous approach - starting parameters $(A, \varphi) = (0.6, 2.3)$ - instantaneous values.

	J_{pre}	J_{win}	J_{inst}
$t_1 = 3.00$	2.1546	2.2237	2.1386
$t_2 = 3.04$	1.7273	1.5738	1.6303
$t_3 = 3.08$	2.2081	2.0684	1.9736
$t_4 = 3.12$	1.6072	1.6375	1.5941
$t_5 = 3.16$	1.7478	1.6514	1.7390
$t_6 = 3.20$	2.5253	2.3834	2.2855
$t_7 = 3.24$	2.4536	2.5198	2.3795

Table 4.12: Comparing the windows and the instantaneous approach - starting parameters $(A, \varphi) = (0.6, 2.3)$ - averaged values.

	J_{pre}	J_{win}	J_{inst}
$I_1 = (3.00, 3.04)$	0.0702	0.0691	0.0696
$I_2 = (3.04, 3.08)$	0.0833	0.0759	0.0753
$I_3 = (3.08, 3.12)$	0.0763	0.0747	0.0709
$I_4 = (3.12, 3.16)$	0.0641	0.0636	0.0652
$I_5 = (3.16, 3.20)$	0.0850	0.0789	0.0795
$I_6 = (3.20, 3.25)$	0.1293	0.1288	0.1213

Table 4.13: Results of the single-objective optimization.

	$J_{\text{S-O}}$
$t_1 = 3.00$	2.2961
$t_2 = 3.04$	1.4410
$t_3 = 3.08$	1.4575
$t_4 = 3.12$	1.7497
$t_5 = 3.16$	1.8121
$t_6 = 3.20$	1.7042
$t_7 = 3.24$	2.3827

	$J_{\text{S-O}}$
$I_1 = (3.00, 3.04)$	0.0736
$I_2 = (3.04, 3.08)$	0.0568
$I_3 = (3.08, 3.12)$	0.0631
$I_4 = (3.12, 3.16)$	0.0744
$I_5 = (3.16, 3.20)$	0.0669
$I_6 = (3.20, 3.25)$	0.1076

5 Conclusions and future developments

In this work we have presented a procedure that identifies a zone of parameters that are Pareto-stationary, i.e. a region of interesting values for the parameters of a complex problem. This identification has a very high computational cost that, however, can be reduced on three levels:

(i) it is possible to parallelize the whole process of identification of the Pareto-stationary zone at the highest level, since the application of MGDA from one starting point is independent from its application to another starting point;

(ii) it is also possible to parallelize the solution of the sensitivities, since they are independent one from the other, and, in the linear case, also the solution of the state; this can be useful as the number of parameters increases;

(iii) finally, it is possible to use reduced methods to solve the PDEs more efficiently.

A new ordering criterion of the gradients has been proposed in this work, to perform an incomplete Gram-Schmidt process: the numerical results have shown how this criterion leads to a better identification of the Pareto-stationary region, which is more defined.

Moreover, we would like to underline the generality of this multi-objective optimization method: in fact, even if this work was focused on systems governed by unsteady PDEs, the method can be applied to any kind of problem, it is sufficient to provide a set of cost functionals.

One can apply this method also to problems where there are different temporal phases and one wants to minimize different cost functionals for each phase, as shown in Figure 5.1. In fact the quantities that have to be minimized during a transient phase are usually different from the ones to minimize when the steady state has been reached. For instance, one can think to the optimization of an air foil of a plane: the interesting quantities during the take-off are different from the one interesting during the steady state or during the landing. The classical approach in these cases is to minimize a convex combination of the different cost functionals, however the multi-objective optimization approach guarantees that all the cost functionals considered decrease (or,

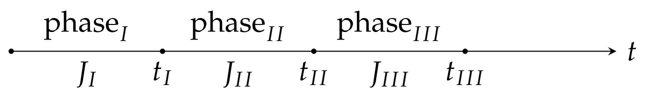


Figure 5.1: Example of different cost functionals for different phases.

in the worst case, remain at the same value) at every iteration, which instead is not guaranteed using the classical approach.

Finally, this method will be applied by the OPALE team to optimize problems in which the governing PDEs are three-dimensional incompressible unsteady Reynolds-averaged Navier-Stokes equations: it will be used to optimize the parameters of a flow control device, in particular a synthetic jet actuator as the one described in [DV06]. Another application proposed for the future is to use this technique to minimize the drag coefficient for cars, since this is one of the main source of pollution, without changing their shape: the idea is to apply some flow control device, the position of whose is one of the parameters to optimize.

Appendices

A Implementation

In this appendix we describe the C++ code (for a detailed manual, see [LLM05] and [Str08]), developed for the implementation of the last version of MGDA introduced in this work and for the resolution of some one-dimensional scalar PDEs used as test case. Therefore, the code is built around two principal *classes*: the class mgda, which implements the multi-objective optimization algorithm, and the class simulator, which implements a solver for a specific kind of PDEs.

Since this code has been developed in the framework of an internship at INRIA, it is an INRIA property, this is why we don't report the whole code, but only some parts. Moreover, this code is meant to be used from the researchers of OPALE team, in order to solve multi-objective optimization problems.

A.1 Class mgda

We start describing the class mgda. Note that this class can be used independently from the kind of problem that generated the multi-objective optimization one: the user only needs to provide the set of gradients and the main method of the class computes the descent direction ω. The following code shows only the most important structures and method contained in the class:

```
class mgda
{
    private:
        vector<double> omega;
        vector<vector<double> > grads;
        vector<vector<double> > Cmatr;
        vector<int> idx;
        vector<vector<double> > basis;
        bool initialization();
        ...
    public:
        void compute_direction();
        ...
};
```

Let us now analyse every structure:

- **omega** is the vector in which is stored the research direction.
- **grads** is provided by the user such as the vector grads[i] corresponds to $\mathbf{g}_i$, i.e. it is the matrix of non-ordinate gradients.
- **idx** is the vector containing the indexes of the reordered gradients, in fact the matrix **grads** given by the user remains unchanged during the algorithm. The vector idx is necessary to access to it in the proper order, as follows: $\mathbf{g}_{(i)}$ = grads[idx [i]].
- **basis** is a matrix built by the algorithm and contains the vectors $\mathbf{u}_i$ that compose the orthogonal basis, computed as in (2.6).
- **Cmatr** contains the lower triangular matrix C introduced in (2.4). In this case, when at the $i-th$ iteration the index ℓ is identified and $\mathbf{g}_{(i)} = \mathbf{g}_\ell$, the $i-th$ and the $\ell-th$ lines of C are actually exchanged. To be more precise, only the elements in the columns that have been filled (i.e. from the first to the $(i-1)-th$), plus the diagonal term, are exchanged, as shown in (A.1)-(A.2).

$$C = \begin{bmatrix} \bullet &&&&&& \\ \bullet & \bullet &&&&& \\ \bullet & \bullet & \bullet &&&& \\ \star & \diamond & \triangleright & \circ &&& \\ \bullet & \bullet & \bullet && \bullet && \\ \star & \diamond & \triangleright &&& \circ & \\ \bullet & \bullet & \bullet &&&& \bullet \end{bmatrix} \begin{array}{l} \\ \textit{before permutation} \\ \\ \longleftarrow \text{curr. line } i \\ \\ \longleftarrow \text{line } \ell \\ \\ \end{array} \tag{A.1}$$

$$C = \begin{bmatrix} \bullet &&&&&& \\ \bullet & \bullet &&&&& \\ \bullet & \bullet & \bullet &&&& \\ \star & \diamond & \triangleright & \circ &&& \\ \bullet & \bullet & \bullet && \bullet && \\ \star & \diamond & \triangleright &&& \circ & \\ \bullet & \bullet & \bullet &&&& \bullet \end{bmatrix} \quad \textit{after permutation} \tag{A.2}$$

- some other members not reported in the code are: the integers **NP** and **NCF** to store respectively the number of parameters and the number of cost functionals, the integer **m** to store the number of gradients actually used to compute ω, two integers **recCount** and **recToll** to control recoursive loops, a boolean **verbosity** to decide whether or not to print the auxiliary structures as they change along the algorithm, and some methods to set the parameters and to get the auxiliary structures. In particular, there is the inline method **void** get_direction(vector< **double**>&) and a second version of the method that computes the direction: **void** compute_direction(vector<**double**>&).

Before describing the main method **void** compute_direction(), we introduce the private method **bool** initialization (). The main part of it consists in building the matrix T

introduced in (2.14) in order to chose the first gradient, and to initialize all the auxiliary structures. We now report and comment some of the fundamental parts.

```
1  bool mgda::initialization ()
2  {
3     if(grads.empty() )
4     {
5        cerr << "It␣is␣necessary␣to␣set␣the␣gradients␣first.\n";
6        return false;
7     }
8     for(int i = 0; i < NCF; ++i)
9     {
10       if(sqrt(dot(grads[i], grads[i]))<toll)
11       {
12          omega.assign(NP, 0);
13          return false;
14       }
15    }
16    ...
```

At the beginning of initialization it is checked whether or not it makes sense to run the algorithm to compute the research direction. In particular, the second control (line 10) verifies that none of the gradients is zero. The function **double** dot(**const** vector<**double** >&, **const** vector<**double**>&) is implemented in the file `utilities.hpp` and returns the scalar product between two vectors. If one of the gradients is zero, the problem is trivial: $\omega = \mathbf{0}$ and there's no need to run the whole algorithm.

Done these controls, the actual initialization can start: the auxiliary structures are cleared, resized and initialized as shown in the following code.

```
   ...
   idx.clear();
   idx.resize(NCF);
   for (int i = 0; i < NCF; ++i)
      idx[i] = i;
   basis.clear();
   basis.reserve(NCF);
   m = NCF;
   Cmatr.clear();
   Cmatr.resize(NCF);
   ...
```

Note that for **basis** the method reserve(NCF) has been used, instead of resize(NCF), since it is the only structure whose size is unknown at the beginning of the algorithm.

We now need to build the matrix T introduced in (2.14): since this matrix is necessary only to select the first gradient and than can be deleted, it makes sense to use **Cmatr** to do that. In the following code is shown the process to chose the first gradient with the ordering criterion (2.15)-(2.5). The functions **double** min(**const** vector<**double** >&) and **int** posmin(**const** vector<**double**>&) are implemented in `utilities.cpp` and they simply call the function min_element contained in the library `algorithm`. They have been introduced only to make the code easier to read.

```
   ...
   for(int i = 0; i < NCF; ++i)
   {
      Cmatr[i].resize(NCF);
      for(int j = 0; j < NCF; ++j)
         Cmatr[i][j] = dot(grads[i], grads[j]);
   }
   vector<double> a(NCF);
   for(int i = 0; i < NCF; ++i)
      a[i] = min(Cmatr[i])/Cmatr[i][i];
   if (posmin(a) != 0)
      swap(idx[0], idx[posmin(a)]);
   basis.push_back(grads[idx[0]]);
   ...
```

After this, **Cmatr** is resized in order to contain a lower triangular matrix and is set to zero, except for the element in position (0,0), set to one. Finally, the method returns **true**.

```
   ...
   for(int i = 0; i < NCF; ++i)
      Cmatr[i].assign(i+1,0);
   Cmatr[0][0]=1;
   return true;
}
```

We now report some of the fundamental parts of the body of the method compute_direction( ), analysing them step by step. At the beginning, the vector **omega** is cleared and the private method **bool** initialization () described above is called: only if the boolean returned from it is true, **omega** is set to zero and the algorithm can start.

```
vector<double> mgda::compute_direction()
{
   omega.clear();
   bool flag = this->initialization();
   if(!flag) return;
   omega.assign(NP, 0);
   double pmin (0);
    bool amb (false);
   ...
```

We do not report the main loop for the Gram-Schmidt process, since it is just the coding translation of the algorithm explained in the previous chapter. However, we would like to focus on a particular part: the Pareto-stationarity test. Right after the computation of a new $\mathbf{u}_i$, if this is zero, the test illustrated in Proposition 5 must be done, for which it is necessary to solve the linear system (2.10). To do this, the lapack library has been used as follows.

```
 1  ...
 2  if(sqrt(dot(unew,unew)) < toll)
 3  {
 4      ...
 5      // define A'A
 6      vector<double> AtA(i*i);
 7      for(int row = 0; row < i; ++row)
 8          for(int col = 0; col < i; ++col)
 9              AtA[row+i*col] = dot(grads[idx[row]],grads[idx[col]]);
10      // define A'b
11      vector<double> Cprime(i);
12      for(int k = 0; k < i; ++k)
13          Cprime[k] = dot(grads[idx[i]], grads[idx[k]]);
14      // compute the solution
15      ...
16      dgetrs_(&TRANS,&N,&NRHS,&AtA[0],&LDA,IPIV,&Cprime[0],&LDB,&
            INFO);
17      ...
18     m = i;
19      if(max(Cprime)<=0)
20      {
21          if(verbosity) cout << "Pareto-stationarity␣detected\n";
22          return;
23      }
24      else
25      {
26          if(verbosity) cout << "Ambiguous␣case\n";
27          amb = true;
28      }
29      break;
30  }
31  ...
```

Note that the vector **Cprime** is initialized as the right handside of the linear system and it is overridden with the solution by the function `dgetrs_`. The **break** at line 29 exits the main loop for the Gram-Schmidt process, whose index is i.

Finally, the ambiguous case is treated recursively, as follows:

```
 1      ...
```

```
    if(amb)
    {
        bool desc = true;
        vector<vector<double> > newgrads = grads;
        for(int i = 0; i < NCF; ++i)
        {
            if(dot(omega, newgrads[i])<0)
            {
                if(verbosity) cout << "omega␣is␣not␣a␣d.d.␣for␣i␣=␣
                    " << i << endl;
                for(int comp = 0; comp < NP; ++comp)
                    newgrads[i][comp] *= gamma;
                desc = false;
            }
        }
        if(!desc)
        {
            if(verbosity) cout << "recoursive\n";
            if(recCount > recToll)
            {
                cerr << "Recoursive␣limit␣reached\n";
                omega.clear();
                return;
            }
            mgda rec(newgrads, verbosity, toll);
            rec.set_recCount(this->recCount+1);
            rec.set_recToll(this->recToll);
            rec.compute_direction(this->omega);
        }
    }
    return;
}
```

A.2 Class simulator

The second main class of the code is the class simulator: the purpose of this class is to solve three specific types of PDEs, introduced in (1.5) and (3.4), and used as test case. We recall that the differential operators considered are of advection-diffusion, with constant coefficients, and the PDEs are one-dimensional, scalar and unsteady.

The structure of the class is reported in the following code, in its main parts. Note that this class stores pointers to polymorphic objects: this cooperation between classes is know as *aggregation*. For more details on this programming technique, see [For].

```
class simulator
```

```
{
    private:
        double c;
        double b;
        double xa;
        double xb;
        double T;
        double dx;  int N;
        double dt;  int K;
        vector<double> sol;
        function * bcD;
        function * bcN;
        function * ic;
        function * source;
        ...
    public:
        simulator ();
        simulator(const simulator&);
        void run (int order = 1);
        ...
};
```

We now explain every member and method listed above.

- **c** and **b** are, respectively, the advection and the diffusion coefficient.
- **xa** and **xb** are the extreme points of the domain and **T** is the final time.
- **dx** and **dt** are the spatial and time discretization step, while **N** and **K** are, respectively, the number of nodes and of time steps.
- **sol** is the vector in which the solution is stored. There is also a boolean member, **memory**, to decide whether to store the whole solution or only the last two temporal steps (in this second case, the solution is printed on a file).
- **bcD**, **bcN**, **ic** and **source** are the analytical functions for boundary and initial condition and for the source term. The abstract class **function** will be explained in detail afterwards.
- simulator() is the default constructor: it initializes all the parameters to default values (see Table A.1). Note that the pointer are initialized to the NULL pointer: it is necessary to set all the analytical functions before running any simulation.
- simulator(**const** simulator&) is the copy constructor. It is necessary, for instance, to store more equations in a vector (useful in case in which there are many sensitivities).

Table A.1: Default values.

Parameter	Symbol	Value
advection coeff.	c	1
diffusion coeff.	b	0.1
extremal point	x_a	0
extremal point	x_b	1
spatial step	Δx	0.02
final time	T	6
temporal step	Δt	0.0004
analytical functions		NULL pointer

· the method **run** is the main method of the class: it solves the equation according to the schemes presented in Chapter 2.

· some other methods and members not listed in the code above are: some booleans to handle the type of the equation (for instance if it is linear or not); a method that, given the exact solution, computes the spatial convergence; a method that, given an output stream, prints the solution in a gnuplot compatible format; many methods to set and/or get the members.

Class function

To store the boundary and initial conditions and the source term in an analytical form, the class function has been implemented. It is an abstract class, meant just to define the public interface. The declaration of the class is reported below.

```
class function
{
   protected:
      double a;
   public:
      function () {};
      function (double A) : a(A) {};
      virtual ~function () {};
      virtual function* clone () const = 0;
      virtual double in(double,double) const;
      virtual double in(double) const;
      ...
};
```

Some observation about the class:

· a parameter **a** is defined because even the most simple functions need it and, even though it is not necessary, it is useful;

· it is necessary to declare the destructor virtual because some classes derived from function may add new parameters;

· a method clone is implemented, since it is necessary for the copy constructor of the class simulator;

· the methods **in** are the ones that evaluate the functions in the given point: there are two methods **in** because there can be functions depending only on space, only on time or depending on both. Note that the decision to implement a method **in** instead of an **operator**() is merely esthetic, due to the fact that we use pointers and not references in the class simulator.

From this class derive many other concrete ones, that implement the operator **in** according to the problem one wants to solve. It is possible to implement every kind of function, but the structure of the new class must be the following:

```
class newFun : public function
{
   protected:
      double newParam;
   public:
      newFun () {};
      newFun (double A, double newP) : function(A), newParam(
         newP) {};
      virtual ~newFun () {};
      virtual function* clone () const {return ((function*)(new
          newFun(*this)));};
      ...
};
```

Therefore, a derived class must implement:

· a virtual destructor, in order to override the one of the base class, if new members are added;

· a method clone().

Naturally, the derived class can add as many parameters as needed: in the above example there is only one new parameter for simplicity. Moreover, the derived class should override at least one of the methods **in**.

On the destructor of the class simulator

Since the class simulator contains some pointers, the synthetic destructor does not work properly: in fact it would destroy the pointer itself without freeing the pointed resource. Before proposing some possible implementations of the destructor, let us observe that in this work all the objects `simulator` have been declared at the beginning of the code always in the main scope, never inside loops, and destroyed only at the end of the program: therefore, even if a proper destructor is not implemented, generally at the end of a program the modern OSs free all the memory that has not been freed by the program itself. However, it is good practice to implement the destructor and it gets necessary if the class is used in more complex programs. The main problem for the implementation of the destructor is that the class simulator does not have the full ownership of the pointed resources, but only a shared one. For this reason, here we propose some possible implementations:

· the straightforward approach is to pass to C++11 (instead of C++) and use smart pointers (in particular the shared ones);

· another approach can be to give to the class the full ownership of the functions: however, this can lead to multiple (and useless) declaration of the same function;

· another approach is to remove the method clone from the class function and, in the copy constructor of the class simulator, simply make the pointers of the new object point to the same data of the copied object. However, in this way it is not possible to build a simulator object with the copy constructor and changing the parameters of the data afterwards, without changing also the ones of the original object.

· finally, one can write his own class of smart pointers.

Bibliography

[Ber82] D.P. Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic Press, New York, 1982.

[BTY$^+$99] D. S. Barrett, M. S. Triantafyllou, D. K. P. Yue, M. A. Grosenbaugh, and M. J. Wolfgang. Drag reduction in fish-like locomotion. *Journal of Fluid Mechanics*, 392:183–212, 1999.

[Dés09] J. A. Désidéri. Multiple-gradient descent algorithm (MGDA). *INRIA Research Report*, RR-6953, 2009.

[Dés12a] J. A. Désidéri. MGDA II: A direct method for calculating a descent direction common to several criteria. *INRIA Research Report*, RR-7922, 2012.

[Dés12b] J. A. Désidéri. Multiple-gradient descent algorithm (MGDA) for multiobjective optimization. *Comptes Rendus de l'Académie des Sciences Paris*, 350:313–318, 2012.

[DV06] Régis Duvigneau and Michel Visonneau. Optimization of a synthetic jet actuator for aerodynamic stall control. *Computers & fluids*, 35(6):624–638, 2006.

[EMDD08] Badr Abou El Majd, Jean-Antoine Désidéri, and Régis Duvigneau. Multilevel strategies for parametric shape optimization in aerodynamics. *European Journal of Computational Mechanics/Revue Européenne de Mécanique Numérique*, 17(1-2):149–168, 2008.

[For] L. Formaggia. Appunti del corso di programmazione avanzata per il calcolo scientifico.

[Gia13] M. Giacomini. Multiple-gradient descent algorithm for isogeometric shape optimization. Master's thesis, Politecnico di Milano – INRIA Méditerranée Sophia Antipolis, 2013.

[KED$^+$10] A. Kozub, P. Ellinghaus, R. Duvigneau, J. C. Lombardo, and N. Dalmasso. Interactive and immersive simulation using the continuous sensitivity equation method with application to thermal transport. *INRIA Research Report*, RR-7439, 2010.

[LeV07] R.J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. Society for Industrial and Applied Mathematics, 2007.

[Lio71] J.L. Lions. *Optimal Control of Systems governed by Partial Differential Equations*. Springer-Verlag, 1971.

[LLM05] S. B. Lippman, J. Lajoie, and B. E. Moo. *C++ Primer (4th Edition)*. Addison-Wesley Professional, 2005.

[NW06] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, New York, 2006.

[Par96] Vilfredo Pareto. *Cours d'Economie Politique*. Droz, Genève, 1896.

[QSS00] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical mathematics*. Springer, 2000.

[Qua09] A. Quarteroni. *Numerical models for differential problems*, volume 2 of *MS&A Series*. Springer, 2009.

[Rao86] S. S. Rao. Game theory approach for multiobjective structural optimization. 1986.

[Sal] S. Salsa. Note su ottimizzazione in spazi di Banach e Hilbert e sul controllo per EDP.

[Sal10] S. Salsa. *Equazioni a derivate parziali: Metodi, modelli e applicazioni*. Springer Italia Srl, 2010.

[Str06] G. Strang. *Linear algebra ant its applications, fourth edition*. Thomson Brooks Cole, 2006.

[Str08] B. Stroustrup. *Programming: Principles and Practice Using C++*. Addison-Wesley Professional, 1st edition, 2008.

[Trö10] F. Tröltzsch. *Optimal Control of Partial Differential Equations: Theory, Methods, and Applications*. Graduate Studies in Mathematics. American Mathematical Society, 2010.

Ringraziamenti

Anche questa avventura è giunta al suo termine: sono stati cinque anni intensi, a tratti felici e a tratti difficili, durante i quali sono cresciuta molto. Vorrei quindi ringraziare tutte le persone che ci sono state e che hanno contribuito a rendere questo periodo della mia vita speciale.

Ringrazio il professor Parolini per aver sostenuto la mia idea di fare la tesi all'estero, ancor prima dell'esistenza di un progetto concreto, e per la passione che trasmette quando insegna.

Donc, je veux remercier Régis pour le projet très intéressant qu'il m'a proposé et parce que travailler avec lui a été une expérience nouvelle et stimulante. Je remercie Jean-Antoine, pour toutes les "nouvelles idées" et Abdou, pour l'aide pour la préparation des examens. Je remercie toute l'équipe OPALE, plus les deux CASTOR, pour m'avoir accuelli à Sophia: Matthias et José, pour l'aide, en particulier au début, Élise, Aekta, Maria Laura et Paola. Un ringraziamento particolare a Enrico, per aver portato un po' di aria di casa e di Poli e per il grande aiuto con il francese.

Ringrazio alcuni dei professori che hanno segnato questi cinque anni: la prof. Paganoni, per il buonumore, la simpatia e l'umanità; il prof. Fuhrman perché nel mio modo di studiare si distingue chiaramente un "ante Fuhrman" e un "post Fuhrman"; il prof. Tomarelli, perché spiega matematica come se parlasse di letteratura; i prof. Grasselli e Formaggia per la grandissima disponibilità. Un grazie particolare va al professor Salsa: per il modo di insegnare, per gli infiniti aneddoti e per l'attenzione che dedica agli studenti. Infine, grazie a Francesca perché se non fosse stato per lei probabilmente non avrei mai iniziato Ingegneria Matematica.

Ringrazio tutti gli amici che hanno condiviso con me questo percorso fin dai primi passi: Franci, ottima compagna di tesi triennale e di grandi risate; Nico, per aver arricchito la cultura popolare di nuovi, interessantissimi proverbi; Eugenio, per essere di poche, ma mai banali, parole; Anna, grande compagna di gossip (perché ormai ringraziarla per le torte è diventato mainstream); Cri, nonostante ci abbia abbandonati; Rik e Fiz, per tutte le feste organizzate negli anni; il Pot, per essere sempre un passo avanti a tutti e non farlo pesare mai; infine ringrazio Andrea, per avermi spronata a dare sempre il massimo e per avermi fatta crescere a livello personale.

Un grande ringraziamento va all'AIM, per tutte le cose che ho imparato facendo parte dell'associazione ma soprattutto per tutte le persone che ho conosciuto. In particolare Marco, per avermi insegnato a ridere di me stessa ma soprattutto perché senza di lui non avrei mai raggiunto l'incredibile traguardo di 120 stelle a supermario64; Nic, per il modo in cui si appassiona a tutto quello che fa; Teo, per essere un'inesauribile fonte di soluzioni a problemi burocratici; Nahuel, per il mese in cui non abbiamo mai visto la luce per preparare ARF; Ale, per avermi indirizzata verso questa esperienza in Francia; Abele, per tutte le risate nel preparare la famosa "domanda per la lode"; Jacopo, per essere l'amico che ho sempre voluto e il miglior organizzatore di eventi che il Poli abbia mai visto. Un grazie a tutto il direttivo 2014 per il grandissimo lavoro di rinnovo.

Un ringraziamento particolare per i luoghi che hanno segnato questi anni: il Poli, la Nave e il Tender, ormai seconde case; Milano, la mia città che tanto mi è mancata in questi ultimi mesi; Rasun e Rimini, e con loro tutti gli amici riminesi e le persone incontrate in questi ventiquattro anni, per essere dei rifugi d'infanzia; Chiavari, per essere diventata una "casa nel bosco"; Antibes, per avermi accolta nella sua stagione migliore. Ringrazio poi U&B, Friends, i Beatles, Guccini, Dalla e De Gregori per la compagnia in tutte le serate passate da sola. Grazie anche alle guance di Stefano, per essere il migliore antistress di sempre.

Un grazie di cuore ai miei amici di sempre: Maui, per le ormai famose "battute alla Maui" e per essere sempre presente quando chiamato; Eli perché, nonostante sia sempre in giro per il mondo, alla fine torna sempre; Gue, per la capacità di ascoltare e di mantenere sempre la calma; Giorgia, per l'intelligenza con cui ascolta chi ha punti di vista diversi dal suo senza voler imporre le sue idee; Sonia, per essere la mia coscienza e l'unica in grado di riportarmi coi piedi per terra quando ne ho bisogno; Martina, per essere la sorella che non ho mai avuto.

Tutto questo non sarebbe mai stato possibile senza il sostegno della mia famiglia: un grazie agli zii, che si sono sempre interessati; ai nonni Luigia e Silvio che pur non essendo ancora riusciti a capire quello che faccio ne sono sempre stati fieri; alla nonna Angela, che ha visto questo percorso iniziare ma purtroppo non finire, e al nonno Luciano, a cui spesso ho pensato nei momenti di indecisione e difficoltà. Grazie a una mamma incredibile, sempre presente e pronta ad ascoltare le mie lamentele, e in grado di risolvere ogni problema pratico esistente; grazie a papà per tutte le passioni che mi ha trasmesso, dallo sport allo studio, e per la sicurezza che mi dà la sua presenza, in ogni occasione.

Infine, il ringraziamento più grande va a Stefano: per avermi aiutata e supportata in ogni cambio di idee, per essersi preso cura di me in ogni momento, per la straordinaria capacità di rendermi quotidianamente felice.