A NUMERICAL SHAPE OPTIMIZATION FRAMEWORK FOR GENERIC PROBLEMS SOLVED ON UNSTRUCTURED TRIANGULAR MESHES

by

SERGE GOSSELIN
B.Eng. (Mechanical), Université Laval, 2002

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES
(Department of Mechanical Engineering)

We accept this thesis as conforming to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA
August 2004

© Serge Gosselin, 2004
Library Authorization

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Serge Gosselin
Name of Author (please print)

02/09/2004
Date (dd/mm/yyyy)

Title of Thesis: A Numerical Shape Optimization Framework for Generic Problems Solved on Unstructured Triangular Meshes

M. A. Sc.
Degree: Year: 2004

Mechanical Engineering
Department of

The University of British Columbia
Vancouver, BC Canada

Canada
Abstract

Until recently, experiments combined with trial and error have been the preferred methodology to refine designs in fields such as aerodynamics or heat transfer. Numerical shape optimization tools are becoming an important link in the design chain, accelerating this refinement process. Optimal design is a complex task requiring the integration of the many components necessary to perform accurate numerical simulations. In this thesis, a simplified shape optimization framework for generic applications is presented. The partial differential equations describing the physics of the optimization problems are solved on unstructured triangular meshes. The mesh generator guarantees the quality of the triangulation. The finite-volume method is used combined with an implicit Newton-Krylov GMRES solver for generic problems. The numerical and physical aspects of the problem are de-coupled inside the solver. This allows for simplified implementation of new physics package using interior and boundary fluxes. The shape to be optimized is defined with a set of control points. The movements of the boundary vertices are described by cubic spline interpolation. They are then propagated through the internal mesh by an explicit deformation law. Optimization constraints are enforced through a penalty formulation and the resulting unconstrained problem is solved using either the steepest descent method or a quasi-Newton method. Gradients of the objective function with respect to the shape are calculated using finite differences. The correct operation of the optimization framework is verified with validation problems. These allow the assessment of the performance of its different components. To indicate that a minimum has been attained, the gradient norm must be reduced by several orders of magnitude. Overall, the results show that the various components are properly integrated. The framework's range of applications is limited by the implicit solver, currently under development, and by finite-difference gradients. A discussion about necessary requirements to extend this framework to more practical applications is given.
## Contents

1 Introduction .................................................. 1

1.1 Motivation .................................................. 1

1.2 Review of shape optimization algorithms .................. 2

1.2.1 Required elements for shape optimization ................. 2

1.2.2 Optimization algorithms ................................... 6

1.2.3 Methods for gradient computation ......................... 8

1.3 Objectives and scope .......................................... 9

2 Optimization problem formulation ............................ 11

2.1 Generic formulation ........................................... 11

2.2 Mesh generation ............................................... 12

2.2.1 Initial triangulation ....................................... 12

2.2.2 The Delaunay criterion ................................... 13

2.2.3 Edge recovery .............................................. 14

2.2.4 Refinement ................................................ 14

2.2.5 Boundary and control variables definition ............... 16
## CONTENTS

2.3 Shape parameterization ........................................ 17  
   2.3.1 Cubic spline interpolation ................................ 17  
   2.3.2 Deformation parameterization .............................. 19  
2.4 Mesh deformation ................................................ 22  
2.5 Numerical solver .................................................. 23  
   2.5.1 Finite-volume formulation ................................. 25  
   2.5.2 Solution reconstruction .................................... 26  
   2.5.3 Boundary conditions ....................................... 28  
   2.5.4 GMRES algorithm .......................................... 28  
2.6 Objective function and constraints ............................ 30  

3 Optimization algorithm ........................................... 33  
   3.1 Optimality conditions ...................................... 34  
   3.2 Gradient evaluation ......................................... 36  
      3.2.1 Gradient formulation .................................. 37  
      3.2.2 Finite differences ..................................... 37  
      3.2.3 Adjoint methods ....................................... 40  
      3.2.4 Automatic differentiation ............................ 42  
      3.2.5 Incomplete sensitivities .............................. 43  
   3.3 Descent direction ............................................ 44  
      3.3.1 Steepest descent method .............................. 46  
      3.3.2 BFGS method .......................................... 47
## CONTENTS

3.4 Line search procedure ................................................. 48

3.5 Implementation details ............................................. 50

3.5.1 Grid function definition ........................................... 51

3.5.2 Numerical solution of the state .................................... 54

3.5.3 Optimization algorithm ............................................. 55

4 Validation problems ...................................................... 59

4.1 Constrained geometric problem ....................................... 59

4.2 Inverse problem using the Laplace equation ......................... 64

4.3 Inverse problem using the Poisson equation ......................... 67

4.4 Optimization problem in solid mechanics .......................... 71

5 Conclusions and recommendations .................................... 75

5.1 Future work ........................................................... 79
List of Figures

2.1 Geometry to be triangulated .................................................. 13
2.2 Bounding triangles and initial boundary vertices ...................... 13
2.3 Delaunay triangulation ......................................................... 14
2.4 Non-Delaunay triangulation ................................................... 15
2.5 Edge swapping to obtain a Delaunay triangulation ................... 15
2.6 Initial triangulation ............................................................ 15
2.7 Resulting mesh ................................................................. 16
2.8 Cubic splines: $n$ polynomials for $n+1$ points ....................... 18
2.9 Reference curve mapping the Cartesian space .......................... 19
2.10 Illustration of the bisection algorithm .................................... 21
2.11 Motion of the boundary ....................................................... 21
2.12 Mesh deformation examples ................................................ 24
2.13 Averaged and reconstructed solutions ................................... 27
2.14 Representation of the barrier geometric constraints .................. 32
3.1 Typical convergence for the steepest descent algorithm .............. 47
LIST OF FIGURES

3.2 Overview of the optimization process ........................................ 51
3.3 Details of the domain preparation and deformation ........................ 52
3.4 Details of the numerical solver .................................................. 55
3.5 Details of the optimization algorithm ......................................... 56

4.1 Initial and final geometries for the constrained geometric problem .... 61
4.2 Convergence history of the geometric problem .............................. 62
4.3 Initial and final geometries for the inverse Laplace problem ............. 66
4.4 Convergence history of the inverse Laplace problem ...................... 67
4.5 Initial and final geometries for the inverse Poisson problem ............ 69
4.6 Meshed initial domain for the inverse Poisson problem .................. 70
4.7 Convergence history of the inverse Poisson problem ..................... 70
4.8 Meshed domain for the solid mechanics test case .......................... 73
List of Tables

4.1 Locations of the control points in the parametric space for the geometric problem. ................................................................. 63

4.2 Locations of the control points in the parametric space for the Laplace problem. 67

4.3 Locations of the control points in the parametric space for the Poisson problem. 71
Acknowledgements

There are many people I would like to thank and without whom completing this thesis would have been a much more difficult task. First and foremost, I would like to thank my supervisor, Dr. Carl Ollivier-Gooch. Thank you for your availability, your time and most importantly your patience. You always found time to answer my incessant questions about the code and to give me sound advice concerning research. Without your help, I would probably still be lost in the maze of C++ programming. I sincerely appreciated our relationship and I hope I will still be able to learn from you in the sequel to this adventure.

I would also like to thank my lab mates. Charles who made me discover the pleasures of Vancouver. Thank you for your precious help, for sharing your experience with me and for always having been willing to guide me at the beginning of this research project. Chris, thank you for initiating me to the wonders of the open source world. I will never forget the backcountry skiing and the painstaking hikes in the mountains. Amir, with whom I had the most interesting talks about the state of the world at pool side. Veronica, who recently arrived in the lab and who comforted me in regard to my curious accent. Thank you all for giving me your appreciation on this thesis and for coping with my own interpretations of Shakespeare's language. Special thanks to Ashley who patiently read chapter after chapter so that not too many “frenchisms” would remain. Your help was really appreciated.

Je me permets une petite encartade francophone pour remercier ma famille et mes amis pour leurs encouragements soutenus. Un remerciement tout spécial à Marie-Josée, qui a du supporter les hauts et les bas par lesquels je suis passé au cours des deux dernières années. Sans ton soutien, je ne sais pas si j’y serais parvenu.

Serge
Chapter 1

Introduction

1.1 Motivation

With computational power more easily available, a growing number of scientists and engineers are becoming interested in relying on numerical simulations to study phenomena governed by partial differential equations. Even if these specialists have a complete understanding of the physical aspects underlying the problems being studied, they might lack the knowledge required to create valid numerical models of these phenomena. Making numerical simulations more easily accessible was the main driving force that led to the development of the Advanced Numerical Simulation Library (ANSLib) [51, 9, 5, 7, 8, 49], a simplified toolkit to solve almost any partial differential equation numerically.

A numerical solver, when coupled with proper optimization techniques, can be a powerful design tool. Starting from the same premise that led to the creation of ANSLib, it becomes interesting to study the feasibility of a generic framework allowing one to perform shape optimization with minimal interventions from the user. This framework could then be used with a wide range of practical applications including, but definitely not limited to, acoustics [28], aerodynamics [33] and heat transfer [43].

The goal of optimal design is to accurately determine the contours or surfaces of a body such that, based on a series of criteria, it attains optimal performance. Although the field of aerodynamic shape optimization has received a lot of attention in recent years, optimal design
problems can occur in many engineering applications. Designers could benefit from a simple generic framework for shape optimization. Instead of relying solely on costly experimental methods, they would have the possibility of incorporating numerical optimization into the design loop. The design returned by the computer simulations can then be validated experimentally, in wind tunnels or tow tanks for instance. The reduced use of testing facilities would result in improved design time, possibly an increase in performance and ultimately, a reduction in cost.

However, many challenges must be addressed in order to come up with a functional shape optimization algorithm. Solutions to partial differential equations can be complex and an accurate computational representation might be difficult to obtain. For example, when dealing with aerodynamics calculation in the transonic regime, it is important to be able to precisely locate possible shock waves since they are responsible for a noticeable portion of the total drag. This correct representation must apply throughout the problem's range of application.

The task of transposing an applied optimization problem into a well-posed mathematical framework can also be difficult. To achieve this, accurate spatial discretization, geometry representation and flow simulation are required. Having these elements working in conjunction with a gradient-based optimization algorithm constitutes the main focus of this thesis. However, many efficiency issues must still be dealt with at a later stage of the research.

An overview of various techniques used to perform shape optimization is presented in the following sections. First, section 1.2.1 gives an outline of the elements that must be present in the algorithm, then section 1.2.2 reviews different approaches used to perform optimization and finally, section 1.2.3 describes various methods for computing gradients. Section 1.3 states the objectives of this research project and outlines the contents of the remainder of the thesis.

1.2 Review of shape optimization algorithms

1.2.1 Required elements for shape optimization

Within an optimal design framework, some key components must be present:
• The definition of an objective function and constraints.

• A discretization algorithm reducing the continuous physical problem to a finite dimensional discrete problem governed by a small set of geometric variables. This must include a parameterization of the geometry through control points allowing smooth deformations to any admissible configuration. This deformation must then be carried through the computational mesh.

• A solver for the partial differential equations governing the optimization problem. The solution to these state equations is controlled by the location of the aforementioned control points.

• An optimization algorithm.

• An accurate computation of the objective function’s gradients with respect to the geometry.

The central challenge in shape optimization involves reducing a complex physical phenomenon into a mathematical framework leading to the formulation of a relatively simple objective function. Such a task is difficult and specific to every problem. The range of operating conditions must be determined and an assessment of the validity of the physics model must be made. Geometrically, many levels of complexity can be considered; for example, when modeling a wing, should the slats and flaps, the engine mounts or the winglets be taken into account? It is important to make sure that the geometric modeler and the mesh generator are able to represent the chosen details. A certain level of experience with computer simulation is therefore required to correctly assess the elements to be present in the optimization problem. Laporte and LeTallec [37], Giles [25] and Jou et al. [35] discuss some of the issues to be dealt with before obtaining a correct formulation.

With the objective function defined, constraints might need to be added. Two main families are usually considered: state constraints and geometric constraints. Adding quadratic penalty terms to the objective function is, by far, the easiest way to enforce the state constraints [22, 43, 41, 58]. The main advantage of this method resides in the fact that the problem can still be cast in an unconstrained formulation and therefore, simpler optimization algorithms can still be directly used. There are, however, drawbacks to this method, most notably efficiency issues, created by ill-conditioning, which can appear when approaching the desired minimum [26]. For geometric constraints, limits, inside which the control
points must be located, can be imposed. They therefore prevent the geometry from moving outside the defined design space. Penalty methods can also be used for geometric constraints \cite{46} such as the thickness of an airfoil. Finally, augmented Lagrangian methods \cite{48} are another option to implement constraints. They do not suffer from the problems plaguing penalty methods, but they require the use of more complicated optimization algorithms and are therefore more complicated to implement.

When it comes to domain discretization, both block-structured \cite{47, 46} and unstructured \cite{4, 43} meshes have been used in shape optimization algorithms. Even if structured solvers generally allow for faster solutions of the state equations, there are definite advantages to using unstructured grids. First, the algorithms used to propagate boundary deformations throughout the mesh are generally simpler to implement. It also allows for an easier use of solution-adapted meshes \cite{13}, allowing one to work with mesh independent solutions at a reduced computational cost. In the current implementation, unstructured grids are used. They are created with GRUMMP\textsuperscript{1} \cite{6, 5}, a mesh generator developed by the ANSLab\textsuperscript{2} that can produce guaranteed-quality unstructured grids on domains with curved boundaries.

To parameterize the shape, it is important to choose a technique offering sufficient flexibility to cover the whole design space. This will allow truly optimal design. On the other hand, it is also desirable to use as few control points as possible. This creates a compact design space leading to better convergence rate during the optimization procedure. It is also advised to use a parameterization allowing deformation of the shape without creating discontinuities. A survey of parameterization techniques is made by Samareh in \cite{54}.

Choosing the grid points located on the geometry as the control variables is certainly the simplest possible parameterization \cite{33, 41, 43}. This method, known as the discrete approach\textsuperscript{3}, is easy to implement and the geometry changes are only limited by the number of discretization points located on the boundary. The use of the discrete approach in conjunction with automatic differentiation (see section 1.2.2) is appropriate because, in that case, the mesh becomes the only geometric entity known to the problem. This implies that no geometric modeler has to be differentiated. However, the number of control variables becomes enormous when three-dimensional geometries are studied. Furthermore, a local

\textsuperscript{1}Generation and Refinement of Unstructured Multi-element Meshes in Parallel
\textsuperscript{2}Advanced Numerical Simulation Laboratory. http://tetra.mech.ubc.ca/ANSLab
\textsuperscript{3}Some authors also refer to it as CAD-free parameterization
smoothing algorithm has to be used to prevent discontinuous deformations that might lead to an infeasible geometry.

Using polynomial and spline representations can reduce the total number of control variables. Instead of directly representing the shape, a piecewise polynomial function can parameterize the deformation, based on a reference curve. This approach, used by Laporte and LeTallec [37], is quite versatile because of the countless possibilities available when choosing the reference curve. However, the extension to three dimensions might prove to be difficult. Other spline formulations include basis splines (B-splines) [11] and nonuniform rational B-splines (NURBS) [21].

Linked to the control points' movement is a grid perturbation algorithm, which deforms the computational grid based on the position of the deformable boundaries. Because unstructured meshes are used, the process of creating a new computational grid at every iteration is easily automated for simple two-dimensional geometries. This possibility is attractive, especially when large domain deformations occur. However, the cost of creating new grids at every iteration might become too high, especially when working in three dimensions and deformation strategies need to be implemented. Explicit deformation, where the internal nodes are moved proportionally to their distance from the deformed boundary, is a robust approach and can be applied when the number of grid points is small. This method has been widely used for the propagation of mesh deformations and is the one chosen for this implementation. Examples can be found in [38, 40]. Another approach models every mesh edge as a linear spring with an elasticity inversely proportional to its length [4, 20]. This spring system, which is subjected to constraints at the boundaries, must remain in force equilibrium after a perturbation occurs. A number of other methods are proposed by Mohammadi and Pironneau [43].

The numerical solver also plays an important role in the optimization process since many successive flow solutions are required. In order to solve a wide variety of optimal design problems using different physics, the optimization algorithm has to be built around a generic flow solver. The main idea behind such a solver is to separate the numerical and physical aspects of the simulation. This is done by modularizing the solver code, with this modularity often coming from the properties of object-oriented programming languages such as C++. One of the most sophisticated C++ libraries for the numerical solution of partial differential equation is Diffpack [12]. However, it requires much knowledge of the finite-element method from
the users. FEMlab [1] is another finite-element generic solver which uses Matlab to handle the numerical aspects of the simulation. Boivin [5] illustrated that the finite-volume method lends itself better to the decoupling of the numerics and the physics. ANSLib [51, 7, 8, 49], the generic numerical toolkit used in this research, is based on the finite-volume method and presents the advantage of being able to compute high-order solutions [50, 57].

In terms of efficiency, generic solvers cannot be expected to be as fast as a dedicated code, mostly because no assumptions concerning specific aspects of the physics can be made. There are efforts currently being made to reduce the computational time, notably by Nejat [45] who is implementing a preconditioned generalized minimal residual (GMRES) implicit solver into ANSLib. For the moment, the implicit solver has been shown to work for Poisson's equation and for solid mechanics problems. The current study will therefore be limited to these problems, since an efficient solver is imperative to perform shape optimization.

1.2.2 Optimization algorithms

Numerical optimization methods allowing one to find the minimum of an objective function can be subdivided into four categories: the direct search methods, the gradient-based algorithms, the stochastic methods and the full-coupled methods.

To reach the minimum of an objective function, a direct search method relies solely on computing the objective function value for a series of designs. These techniques have the clear advantage of being derivative-free, and they can therefore be applied when the design space is non-smooth or the objective function is non-differentiable. However, they are expected to be very slow, especially if many design variables are considered [56]. Direct search methods can be further separated into three sub-categories: pattern search methods, methods with adaptive sets of search directions and simplex search methods (not to be confused with the simplex method used in linear programming). The latter has recently been investigated on shape optimization problems [19].

One way to accelerate convergence is to locally approximate the objective function by a Taylor series expansion [48, 43, 37]. This is the idea behind gradient-based algorithms, which are potentially the best suited for most shape optimization applications. The optimal

\footnote{Direct search methods are often called zeroth order methods because they do not try to construct an approximation of the objective function through a Taylor series expansion.}
solution can be reached after relatively few function evaluations. There are, however, a few drawbacks to these methods. First, the design space must be smooth and the objective function differentiable. Also, these algorithms will inherently converge to a local minimum, which is not necessarily the absolute minimum inside the design space; the solution can depend on the starting guess. The steepest descent algorithm [41, 33] is the simplest of the unconstrained gradient-based methods. This first order approach directly uses the gradient vector as a search direction in order to determine the next iterate. By using a second-order approximation of the objective function, Newton methods achieve faster convergence rates. This unfortunately requires the Hessian matrix, which is, in the case of shape optimization, usually too costly to compute. Curvature information, coming from the previous gradients, can be utilized to reduce the number of iterations. This approach, used in the quasi-Newton and conjugate gradient methods [16, 48], allows computation of an improved descent direction with no additional cost.

These algorithms are, by design, used to solve unconstrained optimization problems. As was mentioned in section 1.2.1, the constraints can be introduced directly into the objective function expression, via penalty terms, thereby allowing retention of the unconstrained formulation. This approach is simple, but definitely not the most efficient. For example, a more complicated, but potentially more efficient, interior-point algorithm [37] can be used. Throughout the optimization process, these methods guarantee that all the constraints are obeyed. Nonlinearly constrained problems can also be addressed quite effectively using sequential quadratic programming (SQP); an excellent presentation is made by Nocedal and Wright [48] and by Gill et al. [26]. Jameson and Vassberg provide a survey of different gradient-based optimization techniques in [34].

Stochastic optimization methods include two techniques: genetic algorithms [15] and simulated annealing [59]. The former is based on the laws of natural selection while the latter emulates the processes in which a liquid freezes or a metal crystallizes. Stochastic algorithms will almost certainly (but not necessarily) find the global minimum of the objective function. They are also well suited for non-smooth objective functions and most importantly, they can be used with categorical design variables. There are enormous efficiency issues associated with them. An optimization run could require up to 10 000 flow evaluations [46] to reach convergence. This makes these algorithms poorly suited to practical applications.
CHAPTER 1. INTRODUCTION

The last optimization strategy is the fully-coupled method\(^5\). It aims to accelerate the optimization process by solving, all at once, the coupled system of non-linear equations that form the first-order optimality conditions for constrained problems, better known as the Karush-Kuhn-Tucker conditions (see [37, 48] for further details). These optimality conditions are formed using a Lagrangian formulation, assuming the state equations to be constraints of the optimization problem. Solving this system all at once is complex, but it results in important savings since the solution and the gradient do not need to be repeatedly computed. Studies by Laporte and Le Tallec [37] show that this method tends to be faster, but also less robust than gradient-based algorithms. Applications to aerodynamic shape optimization can be found in [36].

### 1.2.3 Methods for gradient computation

When gradient-based algorithms are used in shape optimization, an accurate evaluation of the objective function gradient is required to describe the sensitivity of the design with respect to the control variables. Using a finite difference approximation is certainly the most straightforward way to compute such a gradient [31, 58]. There are well-known difficulties associated with this method: a computational cost proportional to the number of design variables, a risk of introducing round-off errors due to the subtraction of nearly equal terms and a dependence on the choice of the step size. These last two limitations can be avoided by using complex variables [4, 55], but this approach requires at least twice the effort as traditional finite differences, which does not seem appropriate for practical applications.

Alternatively, the gradient may be obtained by solving the linear adjoint state equation. This method was introduced by Pironneau [52] and has the advantage of having a cost virtually independent of the number of design variables. The adjoint method is further subdivided into two families: the continuous approach and the discrete approach. For the continuous approach [33, 4, 32], the adjoint equation is derived from the continuous flow equations and then discretized, while for the discrete approach [46, 47], the adjoint comes directly from the discretized flow equations. The same solution strategy is generally used to solve the adjoint state and the direct problem [37]. In a generic context, the continuous method cannot be used because of its problem-specific nature. However, a common formulation could be implemented to form the linear adjoint equation from various physics discretized equations.

\(^5\)Some authors also use the term one-shot methods
This makes this method particularly attractive for generic optimization. A comparison between the continuous and discrete methods using the Euler equation is made by Nadarajah and Jameson in [44].

Automatic differentiation can also be utilized to compute the objective function's gradient. The idea behind this method is that no matter how complicated the representation of a function is, it can be rewritten using some basic mathematical operations such as addition, multiplication or trigonometric functions. A collection of expressions containing only these operations are then easy to differentiate. If a function can be obtained by a computer program, then each line of this program can be differentiated to calculate its gradient. An excellent review of many automatic differentiation schemes is done by Gilbert et al. [24]. These implementations are classified into two main categories: the direct and the inverse modes. The former is easier to implement but is less efficient, since the computational cost remains dependent on the number of design variables. The inverse mode intrinsically resembles the adjoint method presented above,\(^6\) as every line of the code is considered as a constraint. This then leads to a Lagrangian formulation. However, its implementation is more difficult since memory usage has to be carefully controlled [41, 43]. The major benefit of the reverse mode is that the gradient computation is done at a cost independent of the number of control points. Many differentiators currently exist. Since the computer code used in this research is written in C++, ADOL-C [27] would seem to be the best choice.

After doing some numerical experiments with automatic differentiation, Mohammadi [41] observed that, for small changes in the deformable geometries, the solution to the state equations remains almost constant. This implies that gradients can be obtained using finite-differences without having to recompute the state for every design variable. The result is a considerable reduction in computational costs. This incomplete sensitivity formulation [16, 42, 39, 43] can only be applied to a certain class of problems – namely when the objective function is defined as a boundary integral – and has shown encouraging results.

### 1.3 Objectives and scope

The main objective of this thesis is to investigate the possibility of performing shape optimization using a generic numerical solver. To achieve this, a gradient-based algorithm has

\(^6\)It is called the “adjoint mode” by many authors.
CHAPTER 1. INTRODUCTION

been developed. As optimal shape design is a complicated problem, especially when it comes to practical applications, efficiency issues are not addressed at the current stage of development. The first phase of this work consists of implementing the simplest possible methods allowing performance of generic optimization. Then, by testing these various components on simple problems, it becomes possible to identify areas where improvements in accuracy and/or efficiency have to be made. Finally some unimplemented solutions, corresponding to areas requiring improvements are presented in details to guide future work. This is especially true for gradient calculation.

Because many areas of ANSLib are currently under heavy development, some problems that were originally planned to be studied had to be abandoned. For example, some technical difficulties encountered in the implementation of the GMRES implicit solver have prevented testing aerodynamics applications (either using the Euler or Navier-Stokes equations). Although an explicit solver is available, the cost of computing many flow solutions to obtain a gradient evaluation would be prohibitive. This is part of the reason why efficient gradient calculation could not be explored deeper. As will be seen in chapter 3, an efficient solver is required to solve the adjoint state leading to a relatively cheap gradient evaluation.

The goal at this point is to explore the feasibility of different alternatives that could lead to a robust, accurate and efficient generic shape optimization framework. To achieve this, the mathematical, physical and geometrical formulation of the optimization problem is presented in chapter 2. This is followed by a detailed look at the optimization algorithm in chapter 3. Chapter 4 presents the results of the problems used to validate the framework. Finally chapter 5 gives conclusions and recommendations to guide future work.
Chapter 2

Optimization problem formulation

2.1 Generic formulation

A shape optimization problem can be viewed as the constrained minimization of an objective function $J$ whose evaluation involves the solution of partial differential equations on a geometrical domain $\Omega$. This domain is controlled, in general, by a finite set of parameters $z$, known as the control variables.

The problem's goal consists of obtaining the set $z$ corresponding to $\Omega^*$ such that, among all admissible values,

$$ J(\Omega^*) = \min J(X(z), U(z)) $$

subject to

$$ C_i(X, U) = 0 \quad i \in \mathcal{E} $$

$$ C_j(X, U) \leq 0 \quad j \in \mathcal{I} $$

where $\mathcal{E}$ regroups the equality constraints and $\mathcal{I}$ the inequality constraints. $X$ represents the grid function used to discretize the computational domain while $U$ is the state variable vector obtained by satisfying eq. 2.3 for any admissible set of control variables

$$ \mathcal{F}(X, U) = 0 \quad \forall z \in \Omega_{adm} $$

(2.3)
This implicitly defines the state variables as functions of the control variables. In order to clearly define the problem, the remainder of this chapter details how eqs 2.1 to 2.3 are represented in the optimization framework.

2.2 Mesh generation

The first step to numerically solving a partial differential equation consists of preparing the computational domain by defining its boundaries and then decomposing it into smaller entities known as cells. The resulting discretization is called a mesh. In this research, an unstructured triangular domain decomposition is used.

The mesh quality has a direct influence on a numerical solution's accuracy. Obtaining a high-quality discretization can however be a time consuming process if done manually. Considerable savings can be made by fully automating this process. GRUMMP, the mesh generator used in this work, is able to construct guaranteed-quality meshes from a boundary definition of the domain, provided by the user. The fact that no further manual correction, for example fixing an area with an invalid triangulation, is required once the mesh has been created, makes this tool useful in the context of automatic shape optimization. After many domain deformations, the triangulation may become invalid, therefore requiring an entirely new mesh. Being able to regenerate the mesh without interrupting the optimization process is a considerable advantage.

This section outlines basic the techniques utilized by GRUMMP to build a triangular domain decomposition. It is not meant to be a detailed review of mesh generation algorithms. Further details about the mesh generator can be found in [6, 5].

2.2.1 Initial triangulation

The first part of the meshing process involves creating an initial triangulation based on the user-defined boundaries. This is done by creating bounding triangles inside which all the boundary vertices can fit and then by adding the boundary vertices, updating the triangulation with every addition. To better illustrate this, the domain depicted on figure 2.1 is to be
CHAPTER 2. OPTIMIZATION PROBLEM FORMULATION

2.2.2 The Delaunay criterion

The next step involves obtaining, by swapping edges, a triangulation respecting the Delaunay criterion. A triangulation in which none of the circumcircles (circles passing through every vertices of a triangle) contains another vertex is said to be Delaunay. This is illustrated in figures 2.3 and 2.4. For a given set of vertices, up to degenerate cases\(^1\), there only exists one possible triangulation meeting this criterion. It is also proven that, for these vertices, it

\(^{1}\)A square is an example of a degenerate case where two possible solutions respecting the Delaunay criterion exist.
CHAPTER 2. OPTIMIZATION PROBLEM FORMULATION

2.2.3 Edge recovery

With the Delaunay criterion satisfied, the boundary edges from the initial domain need to be recovered. It is in fact possible that some of the edges might not be present in the Delaunay triangulation. They can always be recovered by swapping. In the example, it is possible to see by looking at figure 2.5 that all the edges are already present. The cells formed by the bounding triangles can then be removed leading to the triangulation shown in figure 2.6, which is the starting point for refinement.

2.2.4 Refinement

Once the initial discretization has been obtained, the mesh needs to be refined to satisfy the sizing required by the user. This process is by far the most complicated and will not be discussed here. However, it is important to note that, starting from a Delaunay triangulation,
Figure 2.4: Non-Delaunay triangulation

Figure 2.5: Edge swapping to obtain a Delaunay triangulation

Figure 2.6: Initial triangulation
none of the cells added in the triangulation will have an angle less than 25.7°. All the details concerning mesh refinement and quality assessments can be found in [5]. The resulting mesh, as generated by GRUMMP, is shown on figure 2.7. This mesh can be made finer by adjusting refinement and grading parameters at run time.

2.2.5 Boundary and control variables definition

In order to define the domain's boundaries, the user needs to provide a list of boundary points. These points are then used to build the geometric entities - lines, circles, arcs, cubic parametric curves or interpolated splines - forming the boundaries. The exact geometric representation allows the insertion of vertices directly on the boundary during the refinement process. This is essential to guarantee the quality of the triangulation [5].

As was mentioned in section 2.1, a set of control variables must also be defined. To minimize the user's interaction, the control variables' definition is made directly from the boundary file GRUMMP reads prior to creating the mesh. The control and boundary points therefore coincide. This allows exploitation of GRUMMP's ability to exactly represent geometric entities to define the boundaries, eliminating the need to include a stand-alone shape parameterization algorithm in the optimization process (see section 2.3).
For the moment, only control variables defining the contours of a boundary are supported. Other parameters such as the angle of attack of a wing or translational variables describing the relative positions of the different parts of a multi-element airfoil could also be used. Taking these additional variables into account will be the subject of future research.

2.3 Shape parameterization

In shape optimization problems, the first difficulty which must be overcome is constructing a grid function \( X(z) \) defining how the mesh behaves with respect to the control variables. This function’s definition is done in two parts. The first, detailed in this section, defines how the boundary vertices are related to the control variables, while the second, presented in section 2.4, shows how boundary deformation are propagated to internal grid points.

2.3.1 Cubic spline interpolation

The geometric representation of regular two dimensional curves can be done using cubic spline interpolation [11]. Given \( n + 1 \) interpolation points, where the function’s value is known, and assuming continuous first and second derivatives at these points, it is possible to build, at low cost, a piecewise cubic function respecting these interpolation values. This is shown on figure 2.8.

Using the same notation as on the figure, the spline equation inside the interval \([t_{i-1}, t_i]\) is

\[
p_i(t) = -f''_{i-1} \frac{(t - t_i)^3}{6h_i} + f''_i \frac{(t - t_{i-1})^3}{6h_i}
- \left( \frac{f(t_{i-1})}{h_i} - \frac{h_if''_{i-1}}{6} \right) (t - t_i)
+ \left( \frac{f(t_i)}{h_i} - \frac{h_if''_i}{6} \right) (t - t_{i-1})
\]

(2.4)

where

\[ h_i = t_i - t_{i-1} \quad \text{for} \quad i = 1, 2, \ldots, n \]
The values of $f''_i$ in eq. 2.4 still need to be computed. By imposing the first derivative continuity, obtained by equating the derivatives of eq. 2.4 at points $i$ and $i + 1$, the linear system of eq. 2.5 is obtained for $i = 1, 2, \ldots, n - 1$.

$$\frac{h_i}{h_i + h_{i+1}} f''_{i-1} + 2f''_i + \frac{h_{i+1}}{h_i + h_{i+1}} f''_{i+1} = \frac{6}{t_{i+1} - t_{i-1}} \left[ \frac{f(t_{i+1}) - f(t_i)}{t_{i+1} - t_i} - \frac{f(t_i) - f(t_{i-1})}{t_i - t_{i-1}} \right]$$

There are, in this system, only $n - 1$ equations for $n+1$ unknowns. Two arbitrary conditions must therefore be added. In the present case, the second derivatives at the beginning and the end of the spline are set to be equal to 0:

$$f''_0 = f''_n = 0.$$

The resulting linear system is tridiagonal and can therefore be solved at low cost using, for instance, the Thomas algorithm [2].

---

**Figure 2.8:** Cubic splines: $n$ polynomials for $n + 1$ points
2.3.2 Deformation parameterization

As mentioned in the first chapter, there are many options available to describe shapes as functions of a set of control points. Since GRUMMP readily supports geometries with curved boundaries and since boundary points coincide with control points (see section 2.2.5), the domain's shape parameterization is done automatically during mesh generation. However, shape deformations are made outside the mesh generator. This implies that an independent algorithm must ensure that boundaries are smoothly deformed after the control points are displaced. To achieve this, the original admissible Cartesian space can be remapped based on a parametric reference curve. As shown in figure 2.9, parameter $t$ represents the position along a curvilinear abscissa $s$ while parameter $z$ describes the positions of the points in the normal direction, at a given abscissa. On this figure, $z_0$ would have a negative value and $z_1$ a positive one.

This reference curve can be defined in various ways. One simple solution is to have it coincide with the starting shape. All the required parametric information can be directly obtained from GRUMMP. For instance, the normal vectors are readily available because an exact geometric representation of the boundaries is used when generating the meshes. The values for the $z$ parameter would of course be set to 0 for all the points since they lie directly on the reference curve. This feature is not currently implemented, but would be desirable for practical applications. At the moment, a slightly more complicated method is, in which the reference curve is defined analytically in a form similar to the one of eq. 2.6. This definition suits inverse problems well because the reference curve can be set as the target solution,
allowing an easier assessment of the results.

\[
\begin{pmatrix}
  x \\
  y
\end{pmatrix} = \begin{pmatrix}
  f(t) \\
  g(t)
\end{pmatrix} \quad t \in [t_{\text{min}}, t_{\text{max}}]
\]  

(2.6)

The increased difficulty comes from the fact that, for a point \( P \) not located on the reference curve, it is impossible to obtain the pair \((t, z)\) directly. The only information available is the Cartesian coordinate \((x, y)\) of \( P \) and the normal vectors, \( \hat{n}(t) \), at any \( t \) on the reference curve \( \hat{n}(t) \). To determine \( t \), a point \( P_i \), located at \( t = t_i \) on the reference curve, must be found such that \( P \) and \( P_i \) are collinear along the normal \( \hat{n}(t_i) \). Since the normal is a function of the yet undetermined \( t_i \), a search algorithm is required to obtain the parameter’s value. Knowing that the cross product of \( \vec{a} \) and \( \vec{b} \) obeys

\[
\vec{a} \times \vec{b} > 0 \quad \text{if} \quad 0 < \theta < \pi \\
\quad = 0 \quad \text{if} \quad \theta = 0, \pi \\
\quad < 0 \quad \text{if} \quad -\pi < \theta < 0
\]

(2.7)

where \( \theta \) is the angle from \( \vec{a} \) to \( \vec{b} \), then from two starting guesses \( t_a \) and \( t_b \) as shown on figure 2.10, a bisection method [23] looking for the zero of the cross product between \( \hat{n} \) and \( \vec{u} \) (the vector from \( P' \) to \( P \)) will eventually converge to the value of \( t_i \).

Once all boundary vertices and control points are mapped in the parametric space, the domain’s boundaries can be deformed with the cubic spline interpolation approach of section 2.3.1. As presented in figure 2.11, based on the parametric coordinates \((t, z)\) of the control points, it is now possible to locate, by interpolation, the location of the boundary vertices given their curvilinear abscissae. The new parametric locations are described by the vector

\[
\vec{\bar{w}}(t) = p_i(t) \hat{n}(t)
\]

(2.8)

where \( p_i(t) \) is the interpolation polynomial’s value, for \( t \in [t_a, t_b] \), obtained with 2.4. Since the connectivity information remains unchanged, the only remaining operation is projecting the boundary vertices from the parametric to the Cartesian space and updating the mesh information and moving the boundaries accordingly.

There is one important pitfall associated with this method. Since the deformation parameterization is implemented without any dependence on the mesh generator, the high-order
CHAPTER 2. OPTIMIZATION PROBLEM FORMULATION

Figure 2.10: Illustration of the bisection algorithm

Figure 2.11: Motion of the boundary
CHAPTER 2. OPTIMIZATION PROBLEM FORMULATION

capabilities of the numerical solver (see section 2.5) cannot be exploited for geometries with curved boundaries. To obtain a high-order accurate solution, the curvature of the boundaries must be taken into account when performing operations necessary to the solution of the state. This information does not exist independently inside the solver, but is instead provided by GRUMMP. The routines used for deformation parameterization would need to calculate the curvature of the deformed boundaries for high-order schemes to work properly. This has not been implemented in this work and therefore, numerical solutions have to be limited to second-order accuracy.

With the boundary movements calculated as a function of the control points, the construction of the grid function $X(z)$ can be completed by defining how these deformations propagate to the internal grid points.

2.4 Mesh deformation

In order to handle domain deformations, an algorithm must be implemented to spread geometry modifications throughout the mesh. A robust, yet expensive, explicit deformation scheme accomplishes this task. This technique has been widely used, notably in [37, 43, 38].

Knowing the displacement of the control variables located on the boundary and having computed the displacement vectors of boundary vertices, $\delta r$, the goal is to make the movement of each internal vertex $\delta r_I$ proportional to its distance from the boundary vertices. This is described by eq. 2.9.

$$\delta r_I = \frac{1}{\alpha_I} \sum_{m \in \partial \Omega} w_m \alpha_m \delta r_m$$

(2.9)

where $\partial \Omega$ represents the deformed boundary. In eq. 2.9, $w_m$ is a scalar weight which is, in two dimensions, equal to the sum of half the length of edges sharing the boundary vertex $m$,

$$\alpha_m = \frac{1}{|r_m - r_I|^{\beta}}$$

represents the local value of a discrete Green's function with $\beta \geq 2$, an arbitrary parameter. The lower this parameter is set, the more localized the deformations will remain. $\alpha_I$ is a normalization factor given by

$$\alpha_I = \sum_{m \in \partial \Omega} w_m \alpha_m$$
CHAPTER 2. OPTIMIZATION PROBLEM FORMULATION

The deformation law of eq. 2.9 has a theoretical interpretation which will not be discussed here; details can be found in [38]. The cost of mesh deformation with this algorithm is proportional to the product of the number of boundary vertices by the number of internal vertices. The number of operations can become very high for three dimensional problems. Other approaches might need to be considered for bigger meshes, but for the two dimensional problems studied in this work, this technique is appropriate. Deformed meshes are presented in figure 2.12.

With successive large shape variations, the quality of the mesh will eventually degrade. The only option at this point is to create a new computational grid, effectively causing the optimization problem to reset. As mentioned in section 2.2.5, the re-meshing process is made simpler by using boundary points as control variables.

With the grid function now defined, the method used to compute the state variables $U$ can be presented.

2.5 Numerical solver

In the context of shape optimization, the numerical solver is used to compute the solution to the partial differential equations describing the physics of the problem at hand. This solution, obtained by satisfying 2.3, allows one to obtain the state variables $U(z)$ on the domain $\Omega$.

A central part of this research consists of investigating the feasibility of generic shape optimization. A toolkit capable of numerically simulating a wide array of partial differential equations is therefore required. It was outlined in chapter 1 that the finite-element method has been the method of choice in previous research on generic solution of PDEs. However, as illustrated in [5, 51], the finite-volume method might provide a simpler alternative to de-coupling the physical and numerical aspects of the problem. The physics is taken into account by the calculation of fluxes, which are free from numerical intricacy.

This section focuses on the general operation of ANSLib, the numerical solver used in the present work. With the domain decomposed into a number of smaller control volumes and the grid function $X(z)$ defined, the PDEs of the problem can be discretized using a formulation presented in section 2.5.1. The resulting system of equations is then solved for
CHAPTER 2. OPTIMIZATION PROBLEM FORMULATION

Figure 2.12: Mesh deformation examples

(a) Original mesh

(b) Deformed mesh, $\beta = 4$

(c) Deformed mesh, $\beta = 32$
the control-volume averaged values of the unknowns. In order to get a smooth solution for accuracy purposes, the numerical solver performs a reconstruction of this solution over the computational domain. This process is detailed in section 2.5.2. The handling of the boundary conditions by ANSLib is then discussed in section 2.5.3. Finally, a brief introduction to the Newton-Krylov implicit solver is made in 2.5.4.

2.5.1 Finite-volume formulation

To derive the finite-volume formulation for a partial differential equation, the first step is to write this equation in a general conservation form. For a two dimensional problem, this leads to

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = S$$

(2.10)

where $U$ is the solution vector, corresponding to the state variables, while $F$ and $G$ are respectively the $x$- and $y$-component of the flux vector. The source term is represented by $S$. Integrating eq. 2.10 over the control volume area$^2$, the following is obtained:

$$\int_A \frac{\partial U}{\partial t} dA + \int_A \frac{\partial F}{\partial x} dA + \int_A \frac{\partial G}{\partial y} dA = \int_A S dA$$

$$\int_A \frac{\partial U}{\partial t} dA + \int_A \left( \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} \right) dA = \int_A S dA$$

(2.11)

Defining the vector $\vec{F} = F \hat{i} + G \hat{j}$, then eq. 2.11 can be rewritten as

$$\int_A \frac{\partial U}{\partial t} dA + \int_A \nabla \cdot \vec{F} dA = \int_A S dA$$

(2.12)

Applying Gauss’s theorem to the second term of the left-hand side, eq. 2.12 becomes

$$\int_A \frac{\partial U}{\partial t} dA + \oint_{\partial A} \vec{F} \cdot \hat{n} ds = \int_A S dA$$

(2.13)

If it is assumed that the size and shape of the control volume is fixed, eq. 2.13 can be simplified further.

$$\frac{d}{dt} \int_A U dA + \oint_{\partial A} \vec{F} \cdot \hat{n} ds = \int_A S dA$$

(2.14)

$^2$In three dimensions, the integration would be done over the volume.
The contour integral in eq. 2.14 is evaluated along the boundaries of the control volumes, with \( \bar{F} \) representing the flux vector across these boundaries.

Equation 2.14 is mathematically equivalent to eq. 2.10. It has only been rewritten in such a way that the fluxes are evaluated through a contour integral. The next step introduces the finite-volume approximation into eq. 2.14. It supposes that the value of \( U \), within a control volume, is constant and equal the control-volume average. For a variable \( Y \) of the control volume \( j \), the control-volume average is defined as

\[
\bar{Y}_j = \frac{1}{A_j} \int_{A_j} Y \, dA_j
\]  

By substituting the state variables vector \( U \) in eq. 2.13 by the approximation of eq. 2.15 and by rearranging terms, the finite-volume approximation of eq. 2.10 is finally obtained.

\[
\frac{d\bar{U}_j}{dt} = -\frac{1}{A_j} \oint_{\partial A_j} \bar{F} \cdot \hat{n} \, ds_j + \bar{S}_j
\]  

The right-hand side of this equation is known as the flux integral, even though it includes the source term integral. Eq. 2.16 is solved for every control volume. The integration is carried out with the Gauss quadrature [17] whose accuracy must match the solution accuracy (see section 2.5.2). This method seeks to optimize the integration scheme by carefully selecting points at which the function is evaluated. It allows exact integration of a polynomial function of degree \( 2m - 1 \) with \( m \) points. Third and fourth order solutions, using respectively quadratic and cubic reconstruction, use two integration points while second order (linear reconstruction) only requires one point.

By preserving the average over every control volume, the finite-volume method allows the conservation of the state variables. This makes the method particularly attractive for fluid flows where the conservation of mass, momentum and energy is critical.

### 2.5.2 Solution reconstruction

In the previous section, it was mentioned that by using the finite-volume method, only the control-volume averaged values are computed. The difference between the average solution in adjacent control volumes is first-order in the mesh spacing. The solution of eq. 2.16 would
therefore only be first-order accurate if the fluxes at the control volume boundaries were calculated using the averaged values.

The solution needs to be reconstructed over the domain to achieve a higher order of accuracy [57, 50]. By this process, the averaged solution is replaced, in every cell, by a low-degree polynomial, creating a smooth approximation over the entire control volume. Depending on the desired order of accuracy, the polynomial function can be linear (second order), quadratic (third order) or cubic (fourth order). By minimizing, in a least-squares sense, the error in predicting the values in the neighboring control volumes, while preserving the mean in the current cell\(^3\), the coefficients of the polynomial function can be determined.

These polynomial functions are then used to compute, with an improved accuracy, the fluxes at the cells interfaces. Figure 2.13 \(^4\), on which the height of a point represents its value, illustrates the difference between a control averaged solution and a second-order accurate reconstructed solution. By comparing both representations, it is evident that the fluxes computed using the reconstructed values will be more accurate.

It is important to note that the polynomials are only valid over their given control volumes. At a common interior boundary face, the values extrapolated from two neighboring cells will

\(^3\)The computed control volume average should match the average of the polynomial produced by reconstruction. Conservation of the mean acts as a constraint on the coefficients.

\(^4\)Taken from [5]
be different. Both of these values are available for flux computation. It is up to the user to decide whether one or the other, or perhaps a combination or both, should be used.

### 2.5.3 Boundary conditions

The fluxes at control volume boundaries coinciding with the domain's boundaries need to agree with the problem's boundary conditions. To impose these conditions, constraints can be directly added to the least-squares reconstruction problem. Dirichlet, Neumann or mixed conditions are enforced by imposing the desired solution or its derivative at Gauss points directly located on the boundary. However, this means that the conditions are only strictly enforced at these points. Elsewhere, they are satisfied to within truncation error (the error coming from the use of reconstruction polynomials). The flux $F$, calculated by reconstruction at these boundary faces, can then be used directly to solve eq. 2.16.

For more complex boundary conditions (imposed stress in solid mechanics, compressible flows conditions), a weak formulation must be used. Instead of using a constraint, a boundary flux $F_b$, defined by the user, is imposed. When solving the system of eq. 2.16, the solver will use this user-defined flux instead of $F$.

ANSLib recognizes two different types of fluxes: the interior fluxes, at faces inside the computational domain; and the boundary fluxes, at faces corresponding to the domain's boundaries. The interior fluxes are always computed using $F$, while at the boundaries, $F$ or $F_b$ can be used depending on the type of boundary conditions. The solver will choose what type of flux must be used depending on the implementation of the physics of the problem. More elaborate details can be found in [51].

With the numerical aspects of the problem defined, the system of eq. 2.16 can now be solved to steady state by a Newton-Krylov GMRES technique presented in the next subsection.

### 2.5.4 GMRES algorithm

Because of the repeated solution evaluations required in optimization problems, convergence to steady state should ideally be attained in a relatively short amount of time. An efficient solver is an essential part of a shape optimization algorithm. Even if explicit time-advance
solvers are accurate, their convergence rates are far too slow for optimization applications. An implicit solver using Newton’s method with a preconditioned Krylov GMRES (generalized minimal residual) is currently being implemented in ANSLib [45]. It allows substantial savings in computational time compared to an explicit solver and is therefore well suited for this research work. However, since the implementation work is still under way, application had to be limited to problems in conduction and solid mechanics.

At steady state, eq. 2.16 can be rewritten as

$$\frac{1}{A} \left[ \int_{\partial A} \vec{F} \cdot \hat{n} \, ds - \int_A S \, dA \right] \equiv R(X, U) = 0$$  \hspace{1cm} (2.17)

where $R$ is the nonlinear system of equations formed from the spatial discretization\(^5\). Newton’s method can be utilized to solve eq. 2.17:

$$J_n \Delta U_n = -R_n$$ \hspace{1cm} (2.18)

with $J_n$ being the flow Jacobian matrix evaluated at iteration $n$.

$$J = \frac{\partial R}{\partial U}.$$  

Equation 2.18 is solved for $\Delta U$ and the update in solution can is made according to eq. 2.19.

$$U_{n+1} = U_n + \Delta U_n$$ \hspace{1cm} (2.19)

The Jacobian is a large sparse matrix. It is usually non-symmetric and often ill-conditioned. An effective Newton’s method implementation requires the efficient solution of the linear system of eq. 2.18. A Krylov GMRES techniques in matrix-free form [53] is used to achieve this. To keep memory requirements at a reasonable level, the restarted version of the GMRES algorithm, denoted GMRES($k$), is utilized. It aims to minimize the norm of the residual vector over a Krylov subspace of size $k$. Increasing the number of search directions (increasing $k$) leads to a more accurate solution update for each GMRES iteration. The drawback is that memory usage increases linearly and computational time quadratically with $k$.

The matrix-vector products on the right-hand side of eq. 2.18 are approximated, at every

\(^5\)Until steady state is reached, the values taken by $R$ are the residuals.
iteration, with a first-order forward difference:

\[ J_v \approx \frac{R(X,U + \epsilon v) - R(X,U)}{\epsilon} \]  

(2.20)

where \( v \) is a normalized vector and \( \epsilon \) a small scalar value typically chosen to be equal to the square root of machine precision. Consequently, the Jacobian matrix does not need to be formed explicitly. This is the reason why this implementation is dubbed matrix-free GMRES.

To improve the efficiency, preconditioning of the GMRES approximation of the Jacobian matrix, clustering the eigenvalue spectrum around unity, can be applied to the right-hand side of eq. 2.18.

\[ J^\mathcal{P}^{-1} \mathcal{P} \Delta U = -R \]

where \( \mathcal{P} \) represents the preconditioner matrix. Since preconditioning is still under development, it is neither discussed, nor used, in this research.

### 2.6 Objective function and constraints

With the grid function \( X(z) \) and the state variables \( U(z) \) characterized, the objective function \( J(X(z),U(z)) \) can now be defined. This objective function is obviously specific to every optimization problem. Object-oriented programming simplifies the implementation of the objective function. Only certain problem specific routines need to be written\(^6\). The optimization framework remains exactly the same.

The objective function can take many forms. For example, for aerodynamics problems, a contour integral, evaluating the drag or the lift, is often used. On the other hand, domain integrals can be used in diffusion problems. It is important to note that the Gauss integration schemes, presented in section 2.5.2, are used to calculate the value of the objective function requiring the evaluation of an integral. For gradient-based algorithms, this function must be continuously differentiable over the design space.

Most practical optimization problems have to be constrained to be physically feasible. For example, a drag minimization problem requires the lift to be constrained. Mathematically,\(^6\)

\(^6\)Implementation details for the complete optimization framework are presented in section 3.5.
the shape that would provide the absolute minimum drag, in such cases, is an infinitely small point. Obviously, this has no practical interest.

Shape optimization problems are usually subject to two types of constraints: state and geometric constraints. The first family includes everything that is related to the physics of the problem (every quantity that depends directly on \( U(z) \)). The second family is used to constrain the shape of the domain and its boundaries.

The simplest way of implementing state constraints is by adding quadratic penalty terms directly into the objective function definition [22, 58]. By doing so, the problem can still be casted in an unconstrained framework. Consequently, the relatively simple unconstrained optimization algorithm can still be used even if constraints are present in the problem. As presented in eq. 2.2, equality and inequality constraints can be encountered. For equality constraints of the form \( \mathcal{A} - \mathcal{A}^0 = 0 \), the penalty term \( C_\mathcal{E} \) is written as

\[
C_\mathcal{E} = \sum_{i \in \mathcal{E}} \eta_i \left( \frac{A_i - A_i^0}{A_i^0} \right)^2
\]

while for inequality constraints \( \mathcal{B} - \mathcal{B}^0 \leq 0 \), the penalty \( C_\mathcal{I} \) is expressed as

\[
C_\mathcal{I} = \sum_{j \in \mathcal{I}} \gamma_j \left[ \max \left( 0, \frac{B_j - B_j^0}{B_j^0} \right) \right]^2
\]

where \( \mathcal{A} \) and \( \mathcal{B} \) are parameters computed based on the actual values of the functions \( X \) and \( U \), \( \mathcal{A}^0 \) and \( \mathcal{B}^0 \) are the target values of these parameters and finally, \( \eta \) and \( \gamma \) are constants dictating the magnitude of each penalty term. Setting these constants can be challenging: if they are set too high, the convergence of the optimization process will be extremely slow, while if they are set too low, the constraints will not be properly enforced. One common approach starts with low values to approach the desired solution quickly. After convergence is reached, \( \eta \) and \( \gamma \) are gradually increased until, after two successive restarts, no more change is observed in the solution. However, this technique requires a more involved implementation on the part of the user.

With the penalty terms included, the objective function can now be written as

\[
\mathcal{J} = \Phi + C_\mathcal{E} + C_\mathcal{I}
\]
where $\Phi = \Phi(X,U)$ is the expression that is to be minimized with respect to the set of constraints.

In this work, two different methods are used to enforce the geometric constraints. Firstly, penalty methods can be utilized to constrain geometric parameters such as the thickness or the area contained inside a closed boundary. On the other hand, some problems, such as the design of turbine blades, require the geometry to be contained inside a bounding box. This prevents two neighboring blades to be too close from each other. Control points are therefore forced to remain inside a certain part of the design space. As illustrated in figure 2.14, maximum and minimum values of the parameter $z$ are assigned to every control point at the beginning of the optimization run. They are then limited to move within $z_{\text{min}} \leq z \leq z_{\text{max}}$. If the absolute minimum is located outside the constrained design space, some of the points will be located on this boundary of the bounding box when convergence is reached. It is important to note that, to respect the first order optimality condition (see section 3.1), the gradient components corresponding to the points located on the limits of the design space should be set to zero.

All the elements necessary to express a numerical shape optimization problem into a well-posed mathematical framework have been described. The next step is to define the optimization algorithm that uses these tools to find the minimum of the objective function.
Chapter 3

Optimization algorithm

It has been shown, in the previous chapter, that the constrained generic shape optimization problem presented in section 2.1 can be expressed with an unconstrained formulation. This is accomplished by imposing the constraints with a set of penalty terms added to the objective function. Other constraints, such as the ones imposed by the solution of the state equations (eq. 2.3), are automatically satisfied, at every location in the domain, by the numerical solver. Therefore, they do not need to be explicitly considered.

The simplicity of the algorithms designed for unconstrained problems is, most certainly, their predominant advantage. A gradient-based algorithm, relying on a line search procedure, is used in the present work. Because of the relatively small number of state solutions generally required to find a minimum, it represents an appropriate choice for practical applications. To ensure that a minimum is found by the implemented algorithm, a set of optimality conditions must be fulfilled. These conditions are elaborated on in section 3.1.

As the name indicates, in gradient-based methods, the objective function's gradient evaluation, with respect to the design variables, is a critical operation. The accuracy and the computational cost of the gradient play a central role in determining the performance of the whole optimization framework. Having the shape modifications exclusively ruled by the value of these gradients is enough to justify the need for accuracy. Furthermore, because repeated re-evaluations are required at every iteration, efficiency is also essential. More details on gradients are offered in section 3.2. At this initial state of research, the finite-difference method - a simple yet inefficient alternative - has been adopted. However, due to its importance,
CHAPTER 3. OPTIMIZATION ALGORITHM

an overview of preferable methods for gradient computation is deemed necessary to guide
future research. Adjoint methods and automatic differentiation techniques are respectively
outlined in sections 3.2.3 and 3.2.4.

With knowledge of the gradient, a search direction vector, dictating the motion of the control
points, can be obtained. These displacements should lead to an improved design, which
causes the value of the objective function to decrease. Two approaches yielding a search
vector from the gradient are implemented: the steepest descent method and the BFGS
method. They are explained in more detail in section 3.3. The line search procedure then
determines by what distance the control points should be moved, along the search direction,
to decrease the value of the objective function. The choice of this step length is explained
in section 3.4. Finally, in section 3.5, implementation details are presented. This allows
appreciation of how all the elements presented in chapter 2 interact with the optimizer to
solve a given shape optimization problem.

3.1 Optimality conditions

In unconstrained optimization, the goal is to minimize an objective function $J$ that depends
on a certain number of variables $z$, with no restriction at all on the values of these variables.
Unfortunately, gradient-based algorithms do not guarantee convergence to the global mini-
mizer\footnote{A point $z^*_g$ is a global minimizer if $J(z^*_g) < J(z)$ for all feasible $z.$} $z^*_g$ of $J$ over the domain $Z_{adm}$. They are however able to find a local minimizer $z^*_l$, which is a point that achieves the smallest value of $J$ in its neighborhood. For the remainder
of this work, only local minimizers, denoted $z^*$, are considered.

When the objective function is smooth and twice continuously differentiable, it is possible to
recognize a minimizer $z^*$ by examining the objective function's gradient $\nabla J(z^*)$ and Hessian $\nabla^2 J(z^*)$ [48]. It is possible to write, by Taylor’s theorem, in the neighborhood of $z$:

$$
J(z + aq) = J(z) + a\nabla J(z)^T q + O\left(a^2 \|q\|^2\right)
$$

(3.1)

where $q$ represents feasible search directions over the design space and $a$ is a scalar value
such that $z + aq$ is included inside the design space. Thus to have a minimizer at $z^*$, eq. 3.2
must be respected, in the neighborhood of $z^*$.

$$\mathcal{J}(z^* + aq) \geq \mathcal{J}(z^*)$$

(3.2)

Therefore, from eqs 3.1 and 3.2, a necessary condition for $\mathcal{J}$ to have a local minimizer at $z^*$ is

$$\nabla \mathcal{J}(z^*) q \geq 0$$

(3.3)

for every feasible $q$. For unconstrained optimization, every direction is feasible, which means that for every $q$, the direction $-q$ must also be considered. The condition of eq. 3.3 will only hold if

$$\nabla \mathcal{J}(z^*) = 0.$$

(3.4)

Equation 3.4 is called the first order necessary condition and holds for any stationary point (minima, maxima and saddle points). To find the nature of the stationary point, the next term in the Taylor expansion of eq. 3.1 must also be considered.

$$\mathcal{J}(z + aq) = \mathcal{J}(z) + a\nabla \mathcal{J}(z)^T q + \frac{a^2}{2} q^T \nabla^2 \mathcal{J}(z) q + O(a^3 \|q\|^3)$$

If it is assumed that eq. 3.3 must hold at the stationary point, then, to have a minimum as defined in eq. 3.2, it is necessary that

$$q^T \nabla^2 \mathcal{J}(z^*) q \geq 0$$

(3.5)

holds for every $q \neq 0$. For unconstrained optimization, the situation simplifies to:

- Second order necessary conditions for a minimum are

$$\nabla \mathcal{J}(z^*) = 0$$

$$\nabla^2 \mathcal{J}(z^*)$$ is positive semi-definite.

- Second order sufficient conditions for a minimum are

$$\nabla \mathcal{J}(z^*) = 0$$
\[ \nabla^2 J(z^*) \] is positive definite.

In the current implementation, the second order conditions do not need to be enforced explicitly. Since the optimization is done through a line search method (see sections 3.3 and 3.4), it is required that the value of \( J \) decrease at every iteration. It is therefore obvious that the algorithm cannot converge to a maximum. It is also highly unlikely that it terminates on a saddle point because such a point is numerically unstable.

### 3.2 Gradient evaluation

The computation of the objective function's sensitivities with respect to the control variables plays a central role in gradient-based shape optimization algorithms [43, 37, 48]. An accurate knowledge of the gradient is required to eventually reach the desired minimum. Unfortunately, gradient calculation also constitutes the most challenging part of the process. Even though the steepest descent method (see section 3.3.1) has allowed solution of optimization problems, as well as linear systems of equations, since the nineteenth century [14], efficient techniques for gradient calculation with respect to the shape only appeared in the mid-1980's [52].

In the context of generic shape optimization, the gradient computation problem is further complicated because some methods, such as solving the adjoint equation at the continuous level, are problem specific. Either automatic differentiation or an adjoint method implemented at the discrete level could be used. However, the current implementation only uses finite-differences. As will be shown in section 3.2.2, this choice can lead to some unavoidable problems.

This section first presents how the gradient of the objective function can be formulated based on the different components of the problem. Afterwards, a detailed review of gradient evaluation by finite differences is made. Even though they were not implemented, a brief introduction to adjoint methods and automatic differentiation has been deemed necessary, as this is where future work will need to be concentrated. Finally, a new idea allowing cheap computation of gradients is presented.
### 3.2.1 Gradient formulation

As mentioned earlier, the optimization problem’s goal is to find the minimum of an objective function $J$. In the case of shape optimization, this objective function depends on a set of $N_C$ points controlling the deformable geometry through parameterization. These control points, in turn, define the position of $N_G$ grid locations on which the $N_v$ state variables are calculated by solving the state equations. Formally, the objective function can be written as

$$ J : Z_{adm} 
i z \rightarrow J(z) = J(X(z), U(z)) $$

where $Z_{adm}$ defines the admissible design space for the control points, $z$ contains the set of control points located inside the geometric domain $\Omega$, $X$ is the grid function and $U$ is the vector of state variables. These are the solutions to a non-linear system of equations obtained by spatial discretization

$$ R(X, U) = 0 $$

The objective function expresses dependencies on the grid and on the solution to the state equations, which both are functions of the control points. Using the chain rule, the differentiation of the objective function with respect to the control points gives

$$ \frac{dJ}{dz} = \frac{\partial J}{\partial X} \frac{\partial X}{\partial z} + \frac{\partial J}{\partial U} \frac{\partial U}{\partial z} $$

(3.6)

where $\frac{\partial J}{\partial X}$ is a $[1 \times N_G]$ row vector, $\frac{\partial X}{\partial z}$ a $[N_G \times N_C]$ matrix, $\frac{\partial J}{\partial U}$ a $[1 \times N_v]$ row vector and $\frac{\partial U}{\partial z}$ a $[N_v \times N_C]$ matrix. This ultimately leads to a gradient vector having $N_C$ components. In eq. 3.6, the main difficulty resides in obtaining $\frac{\partial U}{\partial z}$, the state sensitivities. A popular way of dealing with this term is to proceed by introducing an adjoint state. Since future work should concentrate towards obtaining accurate gradients efficiently, adjoint methods are described in section 3.2.3. For the moment however, only an approximation by finite differences has been implemented.

### 3.2.2 Finite differences

Finite differences are the simplest approach to obtain a reasonably accurate estimate of the sensitivities with respect to control parameters. An approximation of the gradient vector $\frac{\partial J}{\partial z}$
can be obtained by evaluating $\mathcal{J}$ at $N_C + 1$ points located close, in the design space, to the actual solution.

The one-sided-difference formula is defined, in the present context, as:

$$\frac{d\mathcal{J}}{dz_i} \approx \frac{\mathcal{J}(X(z_i + \epsilon e_i), U(z_i + \epsilon e_i)) - \mathcal{J}(X(z_i), U(z_i))}{\epsilon}$$

(3.7)

with $e_i$ a unit vector chosen to be perpendicular to the surface at the location of every control point. By applying eq. 3.7 for $i = 1, 2, \ldots, N_C$, an approximation of the gradient vector can be built. The evaluation procedure consists of perturbing, one at a time, the $N_C$ control points, moving them by $\epsilon$. Then, after obtaining the new locations of the internal grid points with the deformation law of section 2.4, the solution to the state equations is re-computed for the altered geometry. The objective function’s value is then re-evaluated based on this new solution. Applying eq. 3.7, the gradient components can finally be obtained. The main advantage of this method is that the flow solver can be treated as a black box. This makes finite differences attractive to use in conjunction with commercial software. However, some clear difficulties can be associated with this method.

Firstly, efficiency issues come with the need to compute the solution of the state for every gradient component. Thus, the number of control parameters must be limited to a small number. For two dimensional problems, generally requiring few parameters to control $\Omega$, the burden of computational work is still within reasonable tolerances. However, it quickly becomes impractical to use finite differences when the number of control variables increases, for example when dealing with complex three dimensional geometries.

Furthermore, the choice of $\epsilon$ can be difficult. Recognizing that eq. 3.7 is based on Taylor’s theorem \cite{48} and assuming the objective function to be twice continuously differentiable, then

$$\mathcal{J}(z + \delta) = \mathcal{J}(z) + \nabla \mathcal{J}(z)^T \delta + \frac{1}{2} \delta^T \nabla^2 \mathcal{J}(z + t\delta) \delta, \quad \text{for some } t \in (0, 1)$$

(3.8)

If a value $L$ is chosen to bound the size of the Hessian $\|\nabla^2 \mathcal{J}\|$ in eq. 3.8 in the neighborhood of $z$, then the last term is bounded by $\frac{L}{2} \|\delta\|^2$ leading to

$$\|\mathcal{J}(z + \delta) - \mathcal{J}(z) - \nabla \mathcal{J}(z)^T \delta\| \leq \frac{L}{2} \|\delta\|^2$$

(3.9)

If the vector $\delta$ is chosen to be equal to $\epsilon e_i$, then by rearranging eq. 3.9 and by observing that
\[ \nabla \mathcal{J}(z)^T \delta = \nabla \mathcal{J}(z)^T \epsilon e_i = \epsilon \frac{\partial \mathcal{J}}{\partial z_i}, \] it is possible to rewrite eq. 3.7 as

\[ \frac{\partial \mathcal{J}}{\partial z_i} = \frac{\mathcal{J}(z + \epsilon e_i) - \mathcal{J}(z)}{\epsilon} + \zeta, \quad \text{where } |\zeta| \leq \frac{L}{2} \epsilon \quad (3.10) \]

and to remark that the error \( \zeta \) decreases as \( \epsilon \) approaches zero. This suggests that \( \epsilon \) should be chosen as small as possible. Unfortunately, this expression does not take into account the roundoff errors that are introduced when \( \mathcal{J} \) is evaluated on a computer using floating point arithmetics. In double-precision arithmetics, the unit roundoff, \( \tilde{u} \), is of the order of \( 10^{-16} \).

If it is assumed that \( \mathcal{J} \) is bounded by \( L_{\mathcal{J}} \) in the neighborhood of \( z \), then the differences between exact values \( \mathcal{J} \) and computed values \( \mathcal{J}_C \) must obey the following relations:

\[
\begin{align*}
|\mathcal{J}_C(z) - \mathcal{J}(z)| & \leq \tilde{u} L_{\mathcal{J}} \\
|\mathcal{J}_C(z + \epsilon e_i) - \mathcal{J}(z + \epsilon e_i)| & \leq \tilde{u} L_{\mathcal{J}}
\end{align*} 
\]

Using the computed values in eq. 3.10, instead of the exact values, leads to the error term

\[ \zeta \leq \frac{L}{2} \epsilon + 2\tilde{u} \frac{L_{\mathcal{J}}}{\epsilon} \]

Naturally, the goal is to select \( \epsilon \) that gives the smallest error possible. The minimizing value is

\[ \epsilon^2 = \frac{4\tilde{u} L_{\mathcal{J}}}{L} \]

If it is assumed that the problem is well-scaled, meaning that the ratio of the objective function values to the second derivative values remains modest, choosing the perturbation size to be equal to the square-root of machine precision is close to optimal.

\[ \epsilon = \sqrt{\tilde{u}} \quad (3.11) \]

This last equation gives an idea of the order of magnitude for \( \epsilon \). However, it is impossible to guarantee that the assumption leading to eq. 3.11 will hold. Furthermore, since the objective functions are usually highly non-linear, the estimated gradient value might depend on the choice of \( \epsilon \) [37]. Typically, its value, in practical applications, should be in the range of \( 10^{-5} \) to \( 10^{-8} \). The use of a large step size will introduce significant truncation errors, while a small one will cause subtractive cancellation errors.
CHAPTER 3. OPTIMIZATION ALGORITHM

The last limitation, and perhaps the most important one, is inherent in the form of eq. 3.7. It is easy to observe that the numerator is obtained from the difference of two nearly identical terms, which is then divided by a very small value. This leads to conditioning issues. The state equation must therefore be solved to the smallest possible residuals to ensure that the objective function evaluation contains as many significant figures as possible. This is why, in future investigations, it will be crucial to find a better approach to compute the objective function's gradient.

The methods presented in sections 3.2.3 and 3.2.4 were not implemented for this research project. A detailed introduction is however necessary to guide the direction of future work.

3.2.3 Adjoint methods

As mentioned earlier, the state sensitivity term, \( \frac{\partial \mathcal{J}}{\partial x} \), is the most difficult term to evaluate in eq. 3.6. To overcome this difficulty, the introduction of an adjoint state can be contemplated. The idea is to consider the state equation \( \mathcal{F}(X, U) = 0 \) as an additional constraint to the optimization problem and to introduce its Lagrange multipliers \( \psi \) [48]. The Lagrangian can be written as

\[
\mathcal{L}(z, U, \psi) = \mathcal{J}(X(z), U(z)) + \psi^T \mathcal{F}(X(z), U(z)).
\]

(3.12)

The adjoint state variables \( \psi \) associated with \( U \) are obtained by solving the linear adjoint state equation [37]:

\[
\frac{\partial \mathcal{L}}{\partial U} = 0
\]

(3.13)

which after development is written as

\[
\frac{\partial \mathcal{F}^T}{\partial U} \psi = -\frac{\partial \mathcal{J}}{\partial U}.
\]

(3.14)

The derivative term on the left-hand side comes from differentiating the state equations, while \( \frac{\partial \mathcal{F}}{\partial U} \) can either be computed by automatic differentiation or by using a centered difference formula. The expression on the left-hand side of eq. 3.14 can be obtained by either differentiating the state equations on the continuous level or by starting from the discretized version of this same state equation.

The first option is known as the continuous approach [32]. It aims to find the analytical expression of \( \frac{\partial \mathcal{F}}{\partial U} \) for a given state equation. Then by discretizing the resulting expression,
CHAPTER 3. OPTIMIZATION ALGORITHM

eq. 3.14 can be solved numerically to obtain the adjoint variables $\psi$. This technique has been used in aerodynamic shape optimization [33]. However, in the context of generic optimization, this method is not applicable since the differentiation of $F$ is problem dependent. An approach better suited to the current research would consist of constraining the optimization problem with the discretized version of the state equation $R(X, U) = 0$. The adjoint equation would therefore be written as

$$\frac{\partial R^T}{\partial U} \psi = -\frac{\partial F}{\partial U}.$$  \hspace{1cm} (3.15)

The term $\frac{\partial R}{\partial U}$ in eq. 3.15 is the same Jacobian matrix as in eq. 2.18. The Newton-Krylov GMRES algorithm could therefore be used to efficiently solve the discrete adjoint state. Nemeč and Zingg [47] present an implementation of a preconditioned Newton-Krylov GMRES algorithm computing the discrete adjoint state on structured grids. The solver presented in section 2.5.4 is a similar one, but is designed to work with unstructured grids. It is reasonable to expect that, in the future, it will be possible to obtain the solution to eq. 3.15 using this solver.

Once the adjoint variables have been calculated, the gradient with respect to the control points can directly be computed using

$$\frac{dJ}{dz} = \left( \frac{\partial F}{\partial X} + \frac{\partial R^T}{\partial X} \psi \right) \cdot \frac{\partial X}{\partial z}. $$  \hspace{1cm} (3.16)

The approach leading to the expression of eq. 3.16 is presented in detail in [37]. The residual sensitivities, $\frac{\partial R}{\partial X}$, can be computed by automatic differentiation or by the following centered difference formula:

$$\frac{\partial R}{\partial X} \approx \frac{R(X(z + \epsilon e_i), U(z)) - R(X(z - \epsilon e_i), U(z))}{2\epsilon} \hspace{1cm} (3.17)$$

Since the state solution does not need to be recomputed, the use of eq. 3.17 is not expensive. The terms $\frac{\partial F}{\partial X}$ and $\frac{\partial X}{\partial z}$ can be obtained in the same way.

By using an adjoint state, the cost of computing the objective function's gradient becomes almost independent of the number of control parameters, making this method very attractive for large-scale problems. Furthermore, compared to finite-differences, the state solution does not necessarily need to be computed to tolerances as close to machine precision. This can drastically accelerate the optimization process since every flow solution will require less effort.
3.2.4 Automatic differentiation

Automatic differentiation is another technique for gradient evaluation. The principle behind automatic differentiation is that a function, having its value calculated by a computer program, can be differentiated by differentiating every single line of this program. No matter how complicated its definition is, this function will always be formed from basic mathematical operations including, but not limited to, addition, subtraction, multiplication, division and trigonometric functions. There are various methods to perform automatic differentiation. They mostly differ in the way they implement this central idea. A very complete review is made in [24].

The various implementations can be divided into two families: the forward mode and the reverse (or adjoint) mode. The forward mode is simpler to implement, but can be computationally expensive. On the other hand, the adjoint mode allows computation of the flow sensitivities at a cost independent of the number of control parameters. Improperly implemented, it might lead to the use of prohibitive amounts of memory because of the required storage. The adjoint mode is usually the preferred choice because of its higher efficiency.

To illustrate the principles behind automatic differentiation in reverse mode, an example can be used. Let’s assume a simple computer program performing the following operations to evaluate the function $f$.

\[
\begin{align*}
  u_1 &= x \\
  u_2 &= x^2 + 2u_1^2 \\
  f &= u_1 + u_2
\end{align*}
\]

As was done with adjoint methods, a Lagrangian formulation is used, but now, every line of the program is considered as a constraint

\[ L = u_1 + u_2 + \psi_1 (u_2 - x^2 - 2u_1^2) + \psi_2 (u_1 - x) \]

where $\psi_i$ are the Lagrange multipliers. The adjoint state system (see eq. 3.13) therefore takes the form

\[
\begin{align*}
  \frac{\partial L}{\partial u_1} &= 1 - 2\psi_1 u_1 + \psi_2 = 0 \\
  \frac{\partial L}{\partial u_2} &= 1 + \psi_1 = 0
\end{align*}
\] (3.18)
To find the values for $\psi_i$, the system of eq. 3.18 must be solved in reverse order. A great deal of data therefore has to be kept in memory. This is why an incorrect usage can become an issue. A careful implementation requires the process to be split in many parts.

Once $\psi_i$ are known, then

$$\frac{\partial L}{\partial x} = \frac{df}{dx} = -2\psi_1 x - \psi_2$$

More details concerning this technique can be found in [48, 43].

It would be doubtful that, in its current form, ANSLib could be automatically differentiated. For instance, because they are not differentiable, boolean operations could not appear in the solver code. This is primarily why this technique is not explored further for this project. Some independent pieces of the code, such as the functions called to compute the mesh deformation or to parameterize the shape, could however be differentiated. The sensitivities $\frac{\partial J}{\partial x}$ and $\frac{\partial X}{\partial z}$ could then be directly obtained and used in eq. 3.16. Among all differentiators available, ADOL-C [27] should be considered because it is designed to work in conjunction with computer codes written in the C/C++ language.

### 3.2.5 Incomplete sensitivities

Based on various numerical experiments using automatic differentiation, Mohammadi [41] observed that the variations of internal mesh nodes had no effect on an objective function evaluated through a boundary integral. In such problems, most of the contributions to the gradient only come from the first few cells located close to the boundary. This observation has important implications. In fact, it implies that the sensitivity of the objective function with respect to the solution of the state equation can be neglected.

$$\frac{\partial J}{\partial U} \approx 0$$

Eq. 3.6 can therefore be reduced to

$$\frac{dJ}{dz} \approx \frac{\partial J}{\partial X} \frac{\partial X}{\partial z}.$$  (3.19)
CHAPTER 3. OPTIMIZATION ALGORITHM

This result applies when the objective function is defined as a contour integral of the geometry to be optimized (or at least some parts of it) and is of the form

\[ J(z) = \int_{\partial \Omega} f(z) g(U) \, ds \]  

(3.20)

where \( \partial \Omega \) represents the contour of the shape to be optimized. This result is particularly important as it allows substantial reduction in the computational effort required for gradient evaluation. In fact, the state sensitivities \( \frac{\partial U}{\partial z} \) no longer need to be evaluated. Both the objective function value and its gradient can therefore be obtained in virtually the same amount of time necessary to calculate the solution to the state equation. The terms in eq. 3.19 can be individually evaluated using automatic differentiation and finite differences. The whole gradient expression can also be approximated using

\[ \frac{dJ}{dz} \approx \frac{J(X(z_i + \epsilon e_i), U(z_i)) - J(X(z_i), U(z_i))}{\epsilon} \]  

(3.21)

The shape needs to be modified through the control points. The mesh is then deformed accordingly. However, the solution does not need to be recomputed as is the case in section 3.2.2. It is simply reconstructed on the deformed mesh, assuming that the value at the cell’s centroids remain unchanged. By applying eq. 3.21, an approximation of the gradient can be obtained.

Aerodynamic shape optimization is the principal field of application of incomplete sensitivities. In fact, many problems such as lift-constrained drag minimization, lift enhancement or the maximization of lift-to-drag ratio, have objective functions that share the form of eq. 3.20. Using this method, Dadone et al. [16] were able to perform overnight optimization on three dimensional wings in transonic regime, using the Euler equations. Other applications are also presented in [42, 39].

3.3 Descent direction

Once the gradient vector is obtained, a local approximation of the objective function can be made using a Taylor series expansion. This allows computation of a search direction \( q_k \), for the current iterate \( k \). Since the line search algorithms used in this work require progress
towards the minimum, at every iteration \cite{48,18} it is expected that \( q_k \) be a descent direction\(^\text{2}\). Referring to eq. 3.1, if \( a \) is small enough, it is easy to see that \( q_k \) will be a descent direction if
\[
\nabla J(z_k)^T q_k < 0. \tag{3.22}
\]

If the search direction is assumed to have the form
\[
q_k = -B_k^{-1} \nabla J(z_k), \tag{3.23}
\]
where \( B_k \) is a symmetric and nonsingular matrix, then it is easy to see that \( B_k \) must be positive definite to satisfy eq. 3.22.

\[
\nabla J(z_k)^T q_k = -\nabla J(z_k)^T B_k^{-1} \nabla J(z_k) < 0
\]

Once the search direction vector is known, the update of the design variables is made according to:
\[
z_{k+1} = z_k + a_k q_k \tag{3.24}
\]
where \( a_k \) is the step length obtained through a line search procedure.

To obtain global convergence, not only must the descent direction be chosen carefully, but the step length \( a \) to be taken in this direction must also obey a set of conditions. These are presented in detail in section 3.4. If the conditions on the line search are respected, then the Zoutendjik's theorem holds\(^\text{3}\):
\[
\sum_{k \geq 0} \cos^2 \theta_k \| \nabla J(z_k) \| < \infty \tag{3.25}
\]
where \( \theta_k \) is the angle between \( \nabla J(z_k) \) and \( q_k \). Inequality 3.25 implies that
\[
\cos^2 \theta_k \| \nabla J(z_k) \| \rightarrow 0. \tag{3.26}
\]

If the method for choosing the search direction \( q_k \) ensures that \( \theta_k \) is bounded away from 90°, it follows that
\[
\lim_{k \rightarrow \infty} \| \nabla J(z_k) \| = 0. \tag{3.27}
\]
\(^\text{2}\)The term “descent direction” will be used to describe a search direction that provides a decrease in the objective function for a small enough step size.
\(^\text{3}\)The proof of this theorem can be found in \cite{48}. 

\textit{CHAPTER 3. OPTIMIZATION ALGORITHM}
In other words, provided that the search directions are never too close to orthogonality with the gradient, the gradient norm converges to zero, satisfying the first-order necessary optimality condition.

Two different ways of computing a descent direction (choosing $B_k^{-1}$) are implemented. These are the steepest descent method, presented in section 3.3.1, and the BFGS method\(^4\), which is outlined in section 3.3.2.

### 3.3.1 Steepest descent method

The steepest descent method chooses the simplest symmetric positive-definite matrix to obtain a descent direction. For $B_k = I$, then

$$q_k = -\nabla J(z_k)$$

(3.28)

The convergence rate for this method is linear. Unfortunately, a problem arises when the Hessian matrix ($\nabla^2 J(z_k)$) has widely varying eigenvalues (if its condition number is large). For a two dimensional case, this would result in very elongated level sets as illustrated in figure 3.1. It can be observed that consecutive steepest descent directions tend to zig-zag, leading to extremely slow convergence. For an idealized method, with a quadratic objective function and an exact line search, it can be shown that

$$J(z_{k+1}) \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^2 J(z_k)$$

where $\kappa$ is the condition number of the Hessian. If $\kappa \gg 1$, then it is easy to observe that the convergence to the minimum will be slow. This situation is very common in practical applications and can be observed when the sensitivity with respect to certain control variables is much greater than to others.

To accelerate the optimization process, a better technique to compute the descent direction must be implemented.

\(^4\)Named after Broyden, Fletcher, Goldfarb and Shanno who independently discovered this method.
3.3.2 BFGS method

To improve convergence rates, instead of using a linear approximation of $\mathcal{J}$, as was the case for the steepest descent method, a quadratic model of the objective function can be utilized [48]:

$$m_k (q) = \mathcal{J}(z) + q^T \nabla \mathcal{J}(z_k) + \frac{1}{2} q^T H_k q \approx \mathcal{J}(z_k + q)$$  \hspace{1cm} (3.29)

where $H_k = \nabla^2 \mathcal{J}(z_k)$ is the Hessian matrix. The minimum of $m_k$ is found when

$$\nabla m_k = 0 = \nabla \mathcal{J}(z_k) + H_k q.$$  \hspace{1cm} (3.30)

Solving this system for $q$, the search direction of eq. 3.31, known as the Newton’s direction, is obtained:

$$q_k = -H_k^{-1} \nabla \mathcal{J}(z_k).$$  \hspace{1cm} (3.31)

It is easy to verify that eq. 3.31 only yields a descent direction if $H_k$ is positive definite. From eq. 3.29, it is also possible to see that the step length $a = 1$ is a natural one for this method. However, there are many shortcomings to this method. Most importantly, it requires the knowledge of the Hessian matrix, which is, in shape optimization, very costly to compute. Furthermore, the solution of the linear system of eq. 3.30 must be obtained at every iteration. Finally, this Hessian matrix is not guaranteed to be positive definite. The descent direction condition required for line search algorithms would therefore not be respected.

The BFGS method, which can be categorized among quasi-Newton methods, aims to compute a descent direction respecting the model of eq. 3.29 without having to explicitly compute the Hessian. It looks for a matrix $B_k$ that is symmetric positive definite, is easily invertible and somehow approximates the Hessian. If the notation $\mathcal{H}_k = B_k^{-1}$ is used, so that:

$$q_k = -\mathcal{H}_k \nabla \mathcal{J}(z_k)$$  \hspace{1cm} (3.32)
then, starting from $H_0 = I$, the BFGS update formula can be written as

$$
H_{k+1} = \left( I - \frac{s_k y_k^T}{y_k s_k} \right) H_k \left( I - \frac{y_k s_k^T}{y_k^T y_k} \right) + \frac{s_k y_k^T}{y_k^T y_k} \frac{s_k s_k^T}{y_k^T y_k}
$$

(3.33)

where

$$
s_k = z_{k+1} - z_k
$$

$$
y_k = \nabla J(z_{k+1}) - \nabla J(z_k).
$$

(3.34)

The BFGS formula of eq. 3.33 will yield a positive-definite $H_k$ matrix that approximates the inverse Hessian if the curvature condition

$$
\nabla^T y_k > 0
$$

(3.35)

is respected. This condition imposes a restriction on the choice of $z_{k+1}$ and therefore, on the step length $a_k$. Equation 3.35 is guaranteed to hold if the Wolfe conditions on the line search, presented in the next section, are respected. The BFGS method allows for super-linear rates of convergence in the neighborhood of the minimum if a full Newton step ($a = 1$) is used. Further details on how this formula is obtained are given in [10, 18, 48].

### 3.4 Line search procedure

It has been mentioned in the previous section that global convergence to the minimum depends on the step length $a_k$. Ideally, $a_k$ should be chosen to provide a substantial reduction of $J$. Choosing the global minimum of the one-dimensional function defined by

$$
\phi(a) = J(X(z + aq), U(z + aq)) \quad a > 0
$$

(3.36)

would be ideal. Finding this minimum, which amounts to performing an exact line search, can be very expensive and is not necessary to guarantee global convergence. Practically, an inexact line search identifying a step length providing an adequate reduction of $J$ is more appropriate.

For the algorithm to be globally convergent, the step lengths must be chosen to respect
certain conditions. For example, it is not sufficient to require a simple decrease in the objective function: $J(z_k + a_k q_k) < J(z_k)$. The strong Wolfe conditions [48, 10] are a popular set of conditions for performing inexact line search. They can be written as

$$
J(z_k + a_k q_k) \leq J(z_k) + c_1 a_k \nabla J(z_k)^T q_k
$$

$$
|\nabla J(z_k + a_k q_k)^T q_k| \leq c_2 |\nabla J(z_k)^T q_k|
$$

(3.37)

where $0 < c_1 < c_2 < 1$. These two equations are respectively known as the sufficient decrease condition and the curvature condition. They ensure a sufficient decrease in the objective function for a sufficiently large step length. Furthermore, if the BFGS method is used, respecting eq. 3.37 guarantees that the approximate inverse Hessian $H_k$ obtained with the BFGS update, remains positive definite.

Given that $q_k$ is a descent direction, the sufficient decrease condition can always be satisfied with a small step length. The curvature condition exists to prevent selection of a small step length, which would not make enough progress towards the minimum. Unfortunately, the curvature condition requires the knowledge of the gradient at every tested step length. It would be too costly to enforce it explicitly when using finite-differences gradients. A backtracking approach can allow one to dispense with the curvature condition and only use the sufficient decrease condition to terminate the line search.

Backtracking proceeds by first selecting a long enough step size. If it satisfies the sufficient decrease condition, then it is accepted; if not, it is multiplied by a contraction factor $\rho$ (in this work, a value of $\rho = 0.5$, is used). The process is repeated until a satisfactory step length is found. It is important to note that the curvature condition can always be checked a posteriori. If it was not satisfied, then the initial step size of the backtracking process is increased for the next iteration.

To limit the number of state solutions required and to prevent the solver from crashing because of badly shaped geometries, it is important to provide the backtracking routine with a good guess for the initial step size. For methods that do not produce well-scaled search directions, such as the steepest descent method, current information from the problem can be used. The guess can be made by assuming that the change in the objective function will
be the same as the one obtained for the previous iteration.

\begin{equation}
 a_k^0 = a_{k-1} \frac{\nabla J(z_{k-1})^T q_{k-1}}{\nabla J(z_k)^T q_k}.
\end{equation}

For quasi-Newton methods, a full Newton step \( a = 1 \), should always be tried and will eventually be accepted, thereby producing super-linear rates of convergence. As mentioned in section 3.3.2, the initial approximation for the inverse Hessian is set such that \( \mathcal{H}_0 = I \). This choice, equivalent to doing a steepest descent iteration, is however not well-scaled to the problem and might prevent usage of \( (a = 1) \) in subsequent iterations. To scale \( \mathcal{H}_0 \), eq. 3.39 is used after the first step has been computed, but before the first BFGS update is performed.

\begin{equation}
 \mathcal{H}_0 = \frac{y_k^T s_k}{y_k^T y_k} I.
\end{equation}

This equation approximates an eigenvalue of the exact initial inverse Hessian scaling \( \mathcal{H}_0 \) to the size of \([\nabla^2 J(z_0)]^{-1}\). For more details, see [48]. Being able to scale the search direction vector makes the BFGS method even more attractive when compared to the steepest descent method. This generally allows to decrease the number of step lengths tried during the backtracking process, thus resulting in fewer state solutions. This leads to a noticeable reduction in the computational time.

If the cost of computing the gradient is of the order of obtaining the solution to the state equation as would be the case when using an adjoint method, the line search procedure could be accelerated by building a cubic interpolant of eq. 3.36 [48]. This technique should be considered in future work.

All the elements required to perform unconstrained optimization have now been described. The next section provides an insight on the details concerning the implementation of these strategies.

### 3.5 Implementation details

This section presents some of the details concerning the implementation of the various elements presented in chapter 2 and 3. As illustrated in figure 3.2, the optimization process can
be subdivided into three parts: the preparation of the computational domain which includes the definition of the grid function, the numerical solution of the partial differential equations describing the physics of the problem and finally the optimization algorithm itself.

![Diagram of the optimization process]

**Figure 3.2: Overview of the optimization process**

The operation of each of these parts are schematically illustrated in figures 3.3 to 3.5. Details about the grid function definition are given in section 3.5.1. The solver's operation is explained in section 3.5.2 and finally, some additional details about the optimization algorithm are presented in section 3.5.3.

### 3.5.1 Grid function definition

Domain preparation is an important part of numerical simulation. When performing shape optimization, an additional difficulty comes from the fact that this domain is deformable. Defining how the boundaries are represented, how they move and how these displacements modify the computational mesh is an important part of the optimization process. Figure 3.3 presents, in detail, the interaction between the elements that are necessary to define the grid function \( X(z) \).
Boundary definition and control points

As it was mentioned in section 2.2.5, the control points are boundary points located on deformable boundaries. By proceeding this way, the informations required for shape parameterization and deformation is contained in the boundary file read at run time.

For the moment, reference curves are defined with analytic functions. However, in the future, when the mesh generator and the solver are well integrated together, this way of defining control points will allow the mesher to automatically take shape parameterization in charge. GRUMMP would be able to handle the boundary deformations by using the exact geometric representations it generates to construct guaranteed-quality meshes. The domain
deformations would then be taken into account by the mesh generator itself.

**Mesh refinement**

The number of cells in the mesh can be critical to the accuracy of the solution to the state equations. Increasing the number of cells, especially in the region where the changes in the solution are important, usually results in improvements in accuracy.

The computational grids are constructed based on the boundary file provided by the user. Refinement is made based on geometric characteristics of the boundaries. The user can also control the refinement and grading of the mesh by setting parameters at run time.

**Reference curve**

Since this optimization framework is still in its primary development phase, only reference curves defined analytically are used in this research. This implies that additional functions defining this curve need to be created for every studied problems. This requirement should disappear in the future.

**Remeshing**

After a few iterations, some cells in the mesh can become too deformed. This can result in a loss of accuracy when solving the state equations. It is sometimes better to create a new mesh based on the actual locations of the control (and boundary) points. The decision to re-create the mesh is made based on the minimal and maximal angles in the triangulation forming the current mesh. These two angles ($\theta_{\text{min}}$ and $\theta_{\text{max}}$) are evaluated during the first iteration. Then, after every geometry modification, all the angles in the mesh are verified. If one of them is smaller than $0.9 \theta_{\text{min}}$ or bigger than $\frac{\theta_{\text{max}}}{0.9}$, then the mesh is re-generated. Other quality measures are available from GRUMMP. Testing more sophisticated remeshing criteria should be part of future research.

It is important to note that, since the objective function depends on the computational grid, the optimization problem needs to be reset when a new mesh is generated. Because
discrete mathematics are used, the desired minimum will be modified. The extent of this modification depends on how well the mesh is adapted to the studied problem.

Mesh deformation

To follow the motion of the boundary, the internal grid points must also be displaced. In this research, an explicit deformation law is used. It is controlled by the parameter $\beta$, which can be set, at run time, by the user. The lower this parameter, the more localized the mesh deformations remain.

3.5.2 Numerical solution of the state

To calculate a numerical solution, a reconstruction-based finite-volume solver repeatedly performs the following operations (illustrated on figure 3.4):

Starting from an initial solution, provided by the physics class or by the solution from the previous iteration,

1. If necessary, set the boundary constraints for the reconstruction.
2. Reconstruct the solution over all control volumes.
3. Evaluate the fluxes along control volume boundaries.
4. Accumulate the fluxes in the proper control volumes.
5. Evaluate the source term in every control volume.
6. Solve eq. 2.18 by minimizing the residuals over the Krylov subspace.
7. Perform the Newton update of the state variables.

Of all these operations, 1, 3 and 5 are specific to each problem. The remaining steps are strictly numerical methods and no knowledge of the problem is necessary to perform them. The idea behind the generic solver is to have all these numerical operations performed by a
finite-volume toolkit. This toolkit then makes calls to an external physics package to retrieve the necessary information to perform the problem-specific calculations.

By using object-oriented programming, these physics packages can all be built using the same standard interfaces. These interfaces are known as classes. This allows simple implementation of physics packages by the user, without having to worry about the numerical issues. A more thorough discussion about the implementation of the physics class and the generic solver is made in [5].

### 3.5.3 Optimization algorithm

Presented in figure 3.5 are the details of the interaction between the various operations used in the optimization algorithm. The main elements required by the gradient-based optimization algorithm are: the evaluation of the objective function, the computation of the gradient, the determination of the search direction and the line search.
Objective function and constraints

For every optimization problem, problem-specific functions have to be defined. They are needed to evaluate the value of the objective function, which includes penalty terms accounting for constraints. Once again, using object oriented programming allows the creation of classes. The creation of new optimization problems can therefore be made in a short amount of time.

Gradient computation and search direction

Currently, the gradient is computed using finite-differences. This implies that, for every gradient component, a complete state solution has to be computed. This technique is adequate for the problems studied in this work. This choice limits the application to simple problems that only require a limited number of control variables.
Once the gradient is known, a search direction, which should yield a decrease in the objective function, can be obtained. Two different techniques are implemented: the steepest descent method and the BFGS method.

**Wolfe conditions**

The Wolfe conditions are used to determine whether a given step length respects the requirements leading to global convergence. If the sufficient decrease condition is satisfied, supposing a long enough step length, the new geometry is accepted, resulting in an improved solution. Otherwise, the step length is reduced by a factor of 2 and a new solution of the state equations is calculated. Because it requires the knowledge of the gradient of the objective function for every step length tried, the curvature condition can only be assessed a posteriori, once the sufficient decrease condition has been verified.

The values $c_1 = 10^{-4}$ and $c_2 = 0.9$ are used in all the validation problems presented in chapter 4. Setting $c_1$ higher (e.g. $10^{-2}$) can accelerate convergence, but can also cause the line search procedure to fail more often.

**Stopping conditions**

The stopping conditions are implemented to determine if either the optimization run has found the desired minimum, or if it has reached a solution from which no further progress can be achieved.

It was said in section 3.1 that the first-order necessary condition for a stationary point is that the gradient of the objective function be equal to zero. The goal of an unconstrained optimization algorithm is therefore to drive the gradient norm as close to zero as possible. In the validation problems presented in the next chapter, convergence is considered to be attained when the gradient $L2$-Norm is reduced by four orders of magnitude.

It might be possible that the algorithm reaches a solution where no further progress towards the minimum can be made. Usually, this situation happens in the neighborhood of the minimum, where the gradient might not be precise enough to compute a feasible search direction. If the step length accepted by the line search is such that the maximum displacement among
all the control points is less than \( \epsilon \) (see section 3.2.2), then the optimization run is stopped. The value of the gradient at this point can be assessed by the user. One can decide whether the actual solution is satisfying or not, in which case one can reformulate the problem by using, for example, a finer mesh or more control points.
Chapter 4

Validation problems

In this chapter, the correct operation of the many elements composing the shape optimization framework is verified. To accomplish this, four test problems are used. The first one, presented in section 4.1, is a strictly geometric problem. The idea is to test components such as the deformation parameterization or the gradient computation without having to solve partial differential equations, obviously allowing for faster optimization runs. This problem is also used to compare two different methods of constraining with penalty terms. In section 4.2, a physics package corresponding to the Laplace equation, is introduced for the second test case. It can be seen as a potential flow problem where no circulation, and therefore no lift, is present. Since this is an inverse problem, the target to reach is already known. Section 4.3 presents another inverse problem, this time using the Poisson equation. A more complex geometry is tested, showing the versatility of the deformation parameterization. The final problem, presented in section 4.4, uses a solid mechanics physics package.

4.1 Constrained geometric problem

For this first test problem, no physics package is used. The objective function is strictly defined based on the domain’s shape and grid function: \( J = J(X(z)) \). The time required for every function evaluation is therefore relatively small since the numerical solver is not utilized. This allows for a quick assessment of the performance of the deformation parameterization, the mesh deformation, the gradient computation and the line search. Furthermore, because
constraints are to be enforced, the use of penalty terms, included directly into the objective function, can also be tested.

The objective of the problem consists of minimizing the perimeter \( P \) of a given shape, such that its area \( A \) is equal to an arbitrary constant \( A_0 \). More formally, this can be written as:

\[
\min P : A = A_0
\]  

(4.1)

The shape resulting from the solution of eq. 4.1 should be a circle. The value of \( A_0 \) is set to be equal to \( \pi \) such that an unit circle should be obtained. The objective function can therefore be expressed as:

\[
J = P + \eta \left( \frac{A - \pi}{\pi} \right)^2
\]  

(4.2)

where \( \eta \) is a penalty coefficient. The solution of eq. 4.1 is not unique, in the sense that every circle satisfying the area constraint constitutes a local minimizer\(^1\). With gradient-based optimization, there is no guarantee that, for instance, the algorithm converges to the unit circle centered at the origin. The final geometry depends on the initial solution. The initial geometry, as well as the shape obtained after convergence, are shown in figure 4.1.

The objective function evaluation is made directly by GRUMMP. Even though no partial differential equation is solved inside the domain, it still needs to be triangulated in order to evaluate the objective function. By summing the areas of these triangles, the total area \( A \) is calculated. The perimeter \( P \) is measured by measuring the length of all boundary edges.

To parameterize the deformation, a reference curve is required. For this problem, it is defined as a circle centered at the origin:

\[
\begin{pmatrix}
x \\
y
\end{pmatrix} = \begin{pmatrix}
R \cos(\pi t) \\
R \sin(\pi t)
\end{pmatrix}
\]

where \( R = 0.5 \) is the radius of the circle and \( t \in [0, 2[ \) is the curvilinear abscissa in the parametric space. The point \( t = 0 \) is located on the \( x- \)axis. The value of \( t \) increases following the reference curve counter-clockwise. A total of 12 control points are used to define the shape. They are approximatively located at every 30 degrees along the reference curve. From these control points, the location of the boundary vertices can be determined with

\(^1\)On the continuous level, every unit circle would actually be a global minimizer of the objective function.
cubic spline interpolation. The interior mesh is deformed using the explicit law, presented in section 2.4, with $\beta = 2$.

Because the evaluation of the objective function can be made quickly, a centered-difference formula, which is second-order accurate, is used to compute the gradient vector. Once the gradient is known, the descent direction is either calculated using the steepest descent method or the BFGS method.

Two different methods are used to set the penalty coefficient $\eta$. The simplest way starts with a large enough $\eta$, keeping it constant throughout the optimization process. A slightly more complicated approach starts with a smaller value and eventually performs updates. The problem is first partially converged (in this case, until the gradient’s L2-Norm is reduced by three orders of magnitude). The coefficient $\eta$ is then multiplied by a factor of 10. This is repeated until $\frac{A-\pi}{\pi} \leq 0.005$. At this point, the problem is fully converged. The starting
values of \( \eta \) are set such that
\[
\eta = \tau \mathcal{J}(X_0)
\]
where \( \tau \) is a constant. For the simple method, \( \tau \) is initially set at 150 while it is set at 15 for the method with updates. Convergence histories of this test problem are presented in figure 4.2.

![Convergence history of the geometric problem](image)

(a) Constant penalty coefficient  
(b) Updated penalty coefficient

Figure 4.2: Convergence history of the geometric problem

These plots clearly indicate that the BFGS method performs better than the steepest descent method. This can be explained by the fact that, in the neighborhood of the minimum, the quadratic penalty term dominates the expression of the objective function. Since the BFGS method builds a local quadratic approximation and then chooses a step length such that the minimum of this approximation is obtained, the minimizer is likely to be reached more quickly. In the results obtained with the steepest descent method, oscillations can be observed in the gradient norm. These are the result of the impossibility to obtain well-scaled
search directions. It therefore becomes difficult to compute a step length that leads close to the minimum. The successive solution updates rather tend to jump from one side to the other of the desired minimum. This causes the progress to be quite slow.

Since all the methods roughly converge to the same point, thereby demonstrating good global convergence behavior, only the results obtained with the BFGS method using constraint coefficient updates are presented in table 4.1. As it can be seen, the geometry obtained after convergence is not centered at the origin. If it were, the values of $z$ would all be equal to 0.5. It is however nearly a perfect circle\(^2\). Depending on the method used, the resulting perimeter vary from 6.28175 to 6.28177 while the areas are contained between 3.1363 and 3.1370. Of course, the target perimeter is $2\pi$, while the target area is $\pi$. This gives a maximum error of 0.023% for the perimeter and 0.168% for the area. The target solution could probably still be better approximated by increasing the value of the constraint coefficient $\eta$.

<table>
<thead>
<tr>
<th>control point</th>
<th>$t$</th>
<th>$z_{init.}$</th>
<th>$z_{fin.}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0.7130</td>
<td>0.5053</td>
</tr>
<tr>
<td>2</td>
<td>0.1667</td>
<td>0.8500</td>
<td>0.4808</td>
</tr>
<tr>
<td>3</td>
<td>0.3333</td>
<td>0.3120</td>
<td>0.4611</td>
</tr>
<tr>
<td>4</td>
<td>0.5</td>
<td>0.2500</td>
<td>0.4520</td>
</tr>
<tr>
<td>5</td>
<td>0.6667</td>
<td>0.4397</td>
<td>0.4547</td>
</tr>
<tr>
<td>6</td>
<td>0.8333</td>
<td>0.5378</td>
<td>0.4691</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0.7655</td>
<td>0.4919</td>
</tr>
<tr>
<td>8</td>
<td>1.167</td>
<td>0.5378</td>
<td>0.5161</td>
</tr>
<tr>
<td>9</td>
<td>1.333</td>
<td>0.4397</td>
<td>0.5358</td>
</tr>
<tr>
<td>10</td>
<td>1.5</td>
<td>0.7000</td>
<td>0.5472</td>
</tr>
<tr>
<td>11</td>
<td>1.667</td>
<td>0.5000</td>
<td>0.5444</td>
</tr>
<tr>
<td>12</td>
<td>1.833</td>
<td>0.3000</td>
<td>0.5291</td>
</tr>
</tbody>
</table>

Table 4.1: Locations of the control points in the parametric space for the geometric problem.

Updating the coefficient appears to yield better performances when using the steepest descent method. However, for the BFGS method, there is no clear advantage for either of the techniques. Fewer iterations are required to reach convergence when the coefficient is kept constant because the initial gradient norm is higher in this case. The problem therefore needs to be converged further in order to reach the stopping criterion, explaining the increased

\(^2\)Obviously, a perfect circle cannot be obtained since the circumference is approximated using a collection of straight boundaries.
number of iterations. Note that the moment at which the constraints are updated can easily
be seen on figure 4.2. At this point, the gradient norms jump dramatically. This is a combined
effect of the objective function suddenly changing and of the mesh being re-generated.

The framework needed for two variants of the penalty method has been implemented and
the choice will be left to the user. It is however important to note that, according to
Fiacco and McCormick [22], updating of the penalty coefficient improves the likelihood of
global convergence to the desired minimum. Because penalty methods can be affected by ill-
conditioning, the domain of attraction may be prohibitively small if the value of \( \eta \) is chosen
to be too large. This is especially the case if many constraints are present.

This problem verified the correct operation of various components of the optimization frame­
work. The results are as expected. The next step is to consider objective functions defined
based on the solution of a state equation.

4.2 Inverse problem using the Laplace equation

Having verified the proper operation of various components of the optimization framework,
the numerical solver can now be added into the process. The second test problem requires the
solution of a partial differential equation, namely the Laplace equation, in order to evaluate
the objective function value. This equation is relatively easy to solve numerically. Even
if its study does not present great practical interest, it is nevertheless an excellent tool for
validation purposes.

The objective of the problem is to find the shape that minimizes the following objective
function:

\[
J (X, U) = \int_{\partial \Omega} (\phi - \phi_0)^2 \, ds .
\]  

The Laplace equation can be used to describe potential flows [60]. The state variable \( \phi \),
representing the potential function, is obtained by solving:

\[
\nabla^2 \phi = 0 .
\]  

To find the expression of the fluxes, eq. 4.4 can be written in the form of eq. 2.10 using:

\[
U = (\phi)
\]
CHAPTER 4. VALIDATION PROBLEMS

\[ F = \left( \frac{\partial \phi}{\partial x} \right) \]

\[ G = \left( \frac{\partial \phi}{\partial y} \right) \]

\[ S = 0. \]

The fluxes can therefore be expressed as:

\[ \vec{F} = \left( \frac{\partial \phi}{\partial x}, \frac{\partial \phi}{\partial y} \right). \] (4.5)

This problem aims at finding the shape of a domain \( \Omega \), such that the contour integral around its boundary \( \partial \Omega \) matches a known potential function \( \phi_0 \). The target geometry for this problem is a circle. To create a potential flow around a circle, a uniform flow of velocity \( \| \vec{V}_\infty \| \) and a doublet can be superposed\(^3\). After simplification, the potential function on the boundary \( \partial \Omega \) of a unit circle is given by:

\[ \phi_0 = 2 \| \vec{V}_\infty \| \cos \theta \] (4.6)

where \( \theta \) is the angle between the x-axis and a segment joining the origin and a point along the circle. The boundary conditions are set such that \( \| \vec{V}_\infty \| = 100 \) and that \( \frac{\partial \phi}{\partial n} = 0 \) on \( \partial \Omega \). They are imposed by constraining the reconstruction.

The reference curve is defined exactly the way it was for the problem of section 4.1. The control points, which are listed in table 4.2, are also defined in the same manner as in the geometric problem. The initial geometry, shown on figure 4.3, is an ellipse centered at the origin having a major axis of 2.2 and a minor axis of 1.8. The Laplace equation is converged until the L2-Norm of the residuals is \( 10^{-13} \). Values of \( \epsilon \) ranging from \( 10^{-5} \) to \( 10^{-7} \) were tested for the one-sided finite-difference formula used to compute the gradient. They all returned similar results. The smallest value of \( 10^{-7} \) is retained for this inverse problem.

In the convergence histories presented on figure 4.4, it is again possible, as expected, to see that the BFGS method performs better than the steepest descent method. This is especially true when the solution approaches the minimum. It is also important to notice that a sharp increase in the gradient norm can be observed when the mesh is re-generated. Since

\[^3\text{For potential flows, the velocity vector is given by: } \vec{V} = (u, v)^T = \left( \frac{\partial \phi}{\partial x}, \frac{\partial \phi}{\partial y} \right)^T\]
the objective function depends on the grid, the new mesh yields a high value for the term \( \frac{\partial z}{\partial x} \) of eq. 3.6. The value of the objective function also gets modified. However since the grid refinement is chosen such that the solution is grid-converged\(^4\), the effect can hardly be noticed.

The absolute minimum of the objective function is equal to 0. As it can be seen on figure 4.4, this ideal value is not reached by either of the methods. It was observed that refining the mesh further helps to approach this ideal value. This is a sign that these are mostly due to discretization errors.

The parametric locations for the results obtained with the BFGS method and given in table 4.2 are very close to the target \( z = 0.5 \). This demonstrates that the addition of the numerical solver into the optimization framework gives valid results. The next step is to test the framework with a more complicated shape.

\(^{4}\text{Grid convergence analysis is made by evaluating the objective function using the target geometry.}\)
Figure 4.4: Convergence history of the inverse Laplace problem

<table>
<thead>
<tr>
<th>control point</th>
<th>( t )</th>
<th>( z_{\text{init.}} )</th>
<th>( z_{\text{fin.}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0.6</td>
<td>0.4996</td>
</tr>
<tr>
<td>2</td>
<td>0.1667</td>
<td>0.5380</td>
<td>0.5007</td>
</tr>
<tr>
<td>3</td>
<td>0.3333</td>
<td>0.4397</td>
<td>0.5009</td>
</tr>
<tr>
<td>4</td>
<td>0.5</td>
<td>0.4</td>
<td>0.5011</td>
</tr>
<tr>
<td>5</td>
<td>0.6667</td>
<td>0.4397</td>
<td>0.5013</td>
</tr>
<tr>
<td>6</td>
<td>0.8333</td>
<td>0.5380</td>
<td>0.5010</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0.6</td>
<td>0.5004</td>
</tr>
<tr>
<td>8</td>
<td>1.167</td>
<td>0.5380</td>
<td>0.5026</td>
</tr>
<tr>
<td>9</td>
<td>1.333</td>
<td>0.4397</td>
<td>0.5013</td>
</tr>
<tr>
<td>10</td>
<td>1.5</td>
<td>0.4</td>
<td>0.5003</td>
</tr>
<tr>
<td>11</td>
<td>1.667</td>
<td>0.4397</td>
<td>0.5015</td>
</tr>
<tr>
<td>12</td>
<td>1.833</td>
<td>0.5380</td>
<td>0.5005</td>
</tr>
</tbody>
</table>

Table 4.2: Locations of the control points in the parametric space for the Laplace problem.

### 4.3 Inverse problem using the Poisson equation

In the previous problems, the optimization framework has been shown to work with simple geometries. However, one of the future objectives will be to solve design problems in two dimensional aerodynamics. It is therefore fitting to verify the framework operation when dealing with airfoil-like geometries. For this problem, instead of solving a PDE describing the flow around the airfoil, the Poisson equation characterizes a diffusion phenomenon inside the domain \( \Omega \). The inverse Poisson problem presented in this section is inspired by a test case discussed in [37].
CHAPTER 4. VALIDATION PROBLEMS

The objective function for this problem is defined as:

\[ J(X, U) = \int_{\Omega} (u - u_0)^2 \, dA \]  

where \( u \) is the steady-state solution of the Poisson equation

\[ \nabla^2 u = S \text{ in } \Omega \]

\[ u = 0 \text{ on } \partial \Omega \]  

This is virtually the same state equation as the one used in section 4.2, except that the source term \( S \) is no longer equal to 0. The expression for the fluxes is exactly the same. The target solution \( u_0 \) is given by

\[ u_0(x, y) = y^2 - 0.0169 (x - 1)^2 \]  

For this target solution to be achieved on \( \Omega \), \( S \) must be defined as

\[ S(x, y) = -0.1014x + 2.0676 \]  

and \( \partial \Omega^* \), corresponding to the target domain's boundary (and to the reference curve) is expressed as

\[
\begin{pmatrix}
  x \\
  y 
\end{pmatrix} = \begin{pmatrix}
  (t - 1)^2 \\
  0.13t (t - 1) (t - 2) 
\end{pmatrix}, \quad t \in [0, 2] 
\]  

The boundary \( \partial \Omega \) is interpolated with a total of 16 control points, of which 14 are allowed to move (see table 4.3). The points located at the leading \( (t = 1) \) and trailing \( (t = 0) \) edges are fixed and are assigned a parameter \( z = 0 \). The normal vector to the reference curve is given by:

\[
\vec{n} = \begin{pmatrix}
  0.13 (3t^2 - 6t + 2) \\
  -2 (t - 1) 
\end{pmatrix}, \quad \forall t \in ]0, 2[ 
\]  

The initial, final and target geometries are illustrated on figure 4.5. The initial meshed domain \( \Omega_0 \) on which the state equation is solved is shown on figure 4.6.

Since the values of the largest components of the state variable vector are \( \mathcal{O}(10^{-3}) \), the state equation can be converged to a L2-Norm of residuals of \( \mathcal{O}(10^{-19}) \). For finite-difference gradient calculation, \( \epsilon \) is set to be equal to \( 5 \cdot 10^{-11} \). Because the previous problems have
clearly illustrated that the BFGS method offers better performance, the steepest descent search direction is not used for this test case. The convergence history of the inverse Poisson problem is shown on figure 4.7.

The first thing that can be noticed from figure 4.7 is that the gradient cannot be reduced by four orders of magnitude. The algorithm fails to compute a search direction providing a decrease in the objective function before this stopping condition is reached. The optimization process is therefore stopped. The fact that the gradient magnitude reaches a plateau around iteration 30 is a sign that the limit of application of finite-difference gradients has been attained. At this point, the error can be larger than the difference between two solutions separated by $\epsilon$. This is largely due to the fact that, when the minimum is approached, the objective function value eventually becomes quite small. From eq. 3.18, it is possible to see that, when the algorithm stops, the numerator is $O(\epsilon \cdot 10^{-8})$. This is a value approaching the residuals' L2-Norm to which the state equation is solved. There is therefore no possibility of continuing the optimization process. Larger $\epsilon$ have been tried, leading to even worse
CHAPTER 4. VALIDATION PROBLEMS

Figure 4.6: Meshed initial domain for the inverse Poisson problem

results. This is probably due to the introduction of truncation errors in the solution. This is consistent with what was said about the theoretical optimal choice for $\epsilon$ described in section 3.2.2. To improve the performance of the optimization algorithm, a better technique to compute the gradient will need be implemented.

It is also possible to see, on figure 4.7, that the objective function value increases when the mesh is re-created. The absolute change is however quite small and is a good sign that the mesh used provides a grid-converged solution.

As it could be expected, by looking at the control points’ final locations in table 4.3, the problem is not fully-converged. It is safe to say that, with the possibility to evaluate the gradient more accurately, the final results would be much closer to desired values. This example shows that the algorithm performs well with more complex geometries.
\begin{center}
<table>
<thead>
<tr>
<th>control point</th>
<th>\textit{t}</th>
<th>\textit{z}_{\text{init.}}</th>
<th>\textit{z}_{\text{fin.}}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.07849</td>
<td>0.007619</td>
<td>-0.0004189</td>
</tr>
<tr>
<td>3</td>
<td>0.1940</td>
<td>0.002626</td>
<td>0.0001149</td>
</tr>
<tr>
<td>4</td>
<td>0.2921</td>
<td>-0.02331</td>
<td>0.001328</td>
</tr>
<tr>
<td>5</td>
<td>0.5524</td>
<td>0.005805</td>
<td>0.0003469</td>
</tr>
<tr>
<td>6</td>
<td>0.6812</td>
<td>0.01170</td>
<td>0.0008219</td>
</tr>
<tr>
<td>7</td>
<td>0.7729</td>
<td>0.006730</td>
<td>0.0005648</td>
</tr>
<tr>
<td>8</td>
<td>0.8566</td>
<td>0.001452</td>
<td>-0.0003425</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1.146</td>
<td>0.003330</td>
<td>0.002609</td>
</tr>
<tr>
<td>11</td>
<td>1.226</td>
<td>0.004786</td>
<td>0.00006255</td>
</tr>
<tr>
<td>12</td>
<td>1.319</td>
<td>0.01170</td>
<td>0.0008952</td>
</tr>
<tr>
<td>13</td>
<td>1.448</td>
<td>0.005805</td>
<td>0.0001185</td>
</tr>
<tr>
<td>14</td>
<td>1.708</td>
<td>-0.01732</td>
<td>0.001476</td>
</tr>
<tr>
<td>15</td>
<td>1.806</td>
<td>-0.003350</td>
<td>0.0006108</td>
</tr>
<tr>
<td>16</td>
<td>1.922</td>
<td>0.004011</td>
<td>0.001255</td>
</tr>
</tbody>
</table>
\end{center}

Table 4.3: Locations of the control points in the parametric space for the Poisson problem.

4.4 Optimization problem in solid mechanics

The final test problem is an attempt at verifying the optimization framework capability of solving a direct problem (as opposed to the inverse problems of sections 4.2 and 4.3). To achieve this, a physics package designed to solve solid mechanics equations is used. These equations are most often solved with the finite element method. However, ANSLib allows the solution of plane-stress problems, given that the necessary description of the problem’s physics are provided.

The differential equations of motion of a deformable solid are [5]

\[ \frac{\partial \sigma_{ij}}{\partial x_j} + B_i = 0 \] \hspace{1cm} (4.13)

where \( B_i \) are the body forces in the \( i \) direction. Using the small displacement theory, the strains \( \varepsilon \), for an isotropic material can be expressed using the following relationship:

\[ \varepsilon_{ij} = \frac{1}{2} \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \] \hspace{1cm} (4.14)
where \( u \) is the displacement of the material. The plane stress assumption dictates that:

\[
\begin{pmatrix}
\sigma_{xx} \\
\sigma_{yy} \\
\tau_{xy}
\end{pmatrix}
= \frac{E}{1-\nu^2}
\begin{pmatrix}
\varepsilon_{xx} + \nu \varepsilon_{yy} \\
\varepsilon_{yy} + \nu \varepsilon_{xx} \\
\frac{1-\nu}{2} \varepsilon_{xy}
\end{pmatrix}
\tag{4.15}
\]

where \( E \) is the Young’s Modulus and \( \nu \), the Poisson’s ratio. By combining eqs 4.13, 4.14 and 4.15, the following can be obtained:

\[
-\frac{\partial}{\partial x} \left( \frac{E}{1-\nu^2} (\varepsilon_{xx} + \nu \varepsilon_{yy}) \right) - \frac{\partial}{\partial y} \left( \frac{E}{2(1+\nu)} \varepsilon_{xy} \right) = \begin{pmatrix} -B_x \\ -B_y \end{pmatrix}.
\tag{4.16}
\]

Respectively noting \( u \) and \( v \) the material displacements in \( x \) and \( y \), the PDE of eq. 4.16 can be written in the form of eq. 2.10 using:

\[
U = \begin{pmatrix} u \\ v \end{pmatrix}
\]

\[
F = - \begin{pmatrix} \frac{E}{1-\nu^2} (\frac{\partial u}{\partial x} + \nu \frac{\partial u}{\partial y}) \\ \frac{E}{4(1+\nu)} (\frac{\partial u}{\partial y} + \frac{\partial u}{\partial x}) \end{pmatrix}
\]

\[
G = - \begin{pmatrix} \frac{E}{4(1+\nu)} (\frac{\partial u}{\partial y} + \frac{\partial u}{\partial x}) \\ \frac{E}{1-\nu^2} \end{pmatrix}
\]

\[
S = - \begin{pmatrix} B_x \\ B_y \end{pmatrix}
\]

The domain on which eq. 4.16 is applied can be seen in figure 4.8. For this problem, the Young’s modulus is \( E = 2.1 \cdot 10^6 \) while Poisson’s ratio is \( \nu = 0.3 \).

The normal and shear stress at the boundaries can be enforced using a boundary flux. The flux is expressed as:

\[
\vec{F}_b = \begin{pmatrix}
-\sigma_b \cdot \hat{n}_x + \tau_b \cdot \hat{n}_y \\
-\tau_b \cdot \hat{n}_x - \sigma_b \cdot \hat{n}_y
\end{pmatrix}
\tag{4.17}
\]

The signs are chosen to agree with the sign convention for stresses (tensile stress is positive, compressive stress is negative). The boundary conditions for the current problem impose a tensile stress at the left and the right of the domain while the top and the bottom are left
Figure 4.8: Meshed domain for the solid mechanics test case free of stresses.

The objective of the problem is to minimize the Von Mises stresses while keeping the area of the domain constant. The Von Mises stress is defined, at every point, as:

\[ \sigma_{VM} = \sqrt{\sigma_{xx}^2 - \sigma_{xx}\sigma_{yy} + \sigma_{yy}^2 + 3\tau_{xy}^2} \]  

(4.18)

and therefore, the objective function takes the form

\[ J(X, U) = \max(\sigma_{VM}) + \eta \left( \frac{A - A_0}{A_0} \right)^2 \]  

(4.19)

where \( A \) is the area of the meshed domain. The constraint is enforced by adding a penalty term to the objective function as seen in eq. 4.19. The reference curve for this problem is defined as the initial circle that can be seen in figure 4.8. Twelve control points, located at every thirty degrees along this circle are used. The design space was also limited to insure that the hole remains defined within the domain.

Unfortunately, the L2-Norm of the residuals cannot be converged to less than \( O(10^{-10}) \) with the solver in its current state. Since the components of the displacement vector are \( O(10^{-3}) \) and the objective function is computed based on these displacements, there is no hope of obtaining a valid gradient using finite-differences. This test problem has identified certain shortcomings with the implicit solver. These are in the process of being corrected. However, at the moment, they limit the application of the optimization framework to problems using
the Laplace and Poisson equations as exposed in sections 4.2 and 4.3.

Furthermore, the shape optimization technique used in the current framework is not the most appropriate for practical structural optimization. Topology optimization [30] is surely a better choice than shape optimization when it comes to structural design. The fundamental concept of topology optimization consists of dividing the design space, where the structure should exist, into many smaller entities. Each of these entities can be removed or added onto the structure, allowing for a far greater range of possible designs. Deterministic and heuristic optimization schemes can be used to obtain the optimal design. These techniques might become useful in the future when dealing with multi-physics problems where the aerodynamic loads on a wing are considered. However, for the moment, efforts should be concentrated on making the current shape optimization framework more efficient.
Chapter 5

Conclusions and recommendations

A generic numerical shape optimization framework using triangular unstructured meshes has been developed. It integrates four distinct modules: a mesh generator, a generic numerical solver, a deformation parameterization routine coupled with a mesh deformation algorithm and an optimizer.

Starting from the computational domain's boundary definition, which includes a list of the control points, the mesh generator creates a computational grid composed of triangular elements. The quality of the triangulation is guaranteed by the meshing algorithm. The solutions to the partial differential equations, describing the physical aspects of a given problem, are obtained with a Newton-Krylov GMRES implicit solver. Inside this solver, the physical and numerical aspects of the problem are completely de-coupled, allowing for generic applications. The gradient of the objective function with respect to the shape is computed with finite-differences. Based on this gradient, the geometry is deformed such that the objective function's value decreases. The deformation is defined through a set of control points. The position of every boundary vertex is then calculated, based on the locations of these control points, by cubic spline interpolation. With the new locations of these boundary vertices, the interior mesh is modified with an explicit deformation law. The constrained optimization problem is cast as an unconstrained problem by adding quadratic penalty terms to the objective function. Both the steepest descent method and the BFGS quasi-Newton method are used to solve this unconstrained problem.

On the basis of the validation problems presented in chapter 4, several conclusions concerning
the performance and accuracy of the optimization framework can be drawn:

- An efficient gradient-based optimization method has been implemented. It allows us, at least in principle, to tackle any optimization problem whose physics is described by partial differential equation solvable in ANSLib and whose design space can be described by continuous variables. Some validation will still need to be performed to confirm the performance and accuracy of the algorithm on different problems.

- The creation of a computational grid is the initial step in the solution of the shape optimization problem. In this research, the mesh generator GRUMMP is used to create an unstructured triangular mesh of the domain. Because it utilizes algorithms guaranteeing the quality of the triangulation, it lends itself very well to automatic optimization applications. If mesh elements become too distorted, new grids can be created on-the-fly, without any intervention from the user. Unstructured triangular meshes are an excellent choice for applications where moving boundaries are involved. The algorithms to deform unstructured meshes are, in general, simpler, faster and easier to implement than with their structured counterparts. It should also be noted that the process of creating an unstructured grid is fully-automated, resulting in an overall time gain for domain preparation.

- Even though generic applications could not be explored in this work, ANSLib currently allows the solution of many partial differential equations. The implicit Newton-Krylov GMRES solver cannot, at the present moment, converge the solution's residuals sufficiently for the objective function evaluation to be sufficiently accurate. This, in turn, also causes the finite-difference gradient approximation to be inaccurate. The convergence issues are probably the result of an ill-conditioned Jacobian matrix. Preconditioning is currently being implemented and should lead to more accurate and efficient solutions. Since many state equation solutions are needed to solve a shape optimization problem, the improved efficiency brought by an implicit solver is essential. The current research cannot make further progress until the solver's development is complete.

- Being able to correctly simulate the physics involved in an optimization problem only constitutes a fraction of the task at hand. Therefore, having the possibility of using a generic solver does not automatically make the optimization framework applicable
to every conceivable problem. Many problems may require special assumptions. For example, in structural design, many holes may be needed in a given beam to reach optimality. Design variables other than control points defining the contours of a shape must then be considered. This is one of the reasons why creating a truly generic shape optimization framework is impossible. The current implementation has been developed with applications in aerodynamics in mind. It is believed that future work should concentrate on possible improvements in this area. Design variables such as the angle of attack or the space between the elements of an airfoil in high-lift configuration could then be considered. The versatility of the generic solver would still be exploited. It would allow, for example, to quickly switch from the Euler to Navier-Stokes equations depending on the desired type of analysis.

• Gradient evaluation through finite-differences does not provide sufficient accuracy. This issue is inherent to the form of the one-sided difference equation (eq. 3.7) used to approximate the gradient. Because the numerator is composed of the difference of two values that, as the minimum is approached, become very similar, a point is eventually reached where this difference is no longer significant. When this situation occurs, as with the Poisson problem of section 4.3, a search direction providing a decrease in the objective function can no longer be computed. The optimization process is therefore brought to a halt.

• The cubic spline deformation parameterization is working particularly well and is also quite versatile. Typical contours only require 12 to 20 control points for accurate representation, while still providing sufficient shape flexibility. The design space is therefore relatively compact, leading to better convergence rates. On the other hand, since this feature is implemented outside of the mesh generator, crucial information such as the local curvature is not accessible. Curved boundaries are therefore only approximated by a collection of straight segments. This has the consequence of preventing the use of the high-order capabilities of ANSLib. Currently, only second-order accurate solutions can be obtained.

• Once the locations of the boundary vertices is known, the internal mesh is deformed. The implicit law appears to be an appropriate choice. Even though the required number of operations is $O(n\sqrt{n})$, where $n$ is the number of mesh nodes, deforming the mesh is far less time consuming than evaluating the objective function. Until applications in three dimensions are developed, the current technique should be satisfactory.
• The search direction computation is done with both the steepest descent method and the BFGS method. As it was expected, the performance of the latter is better. Not only does this method requires fewer iterations to converge (and therefore fewer state evaluations), but the fact that it produces well-scaled descent direction also accelerates the line search procedure. As a matter of fact, because a full Newton step (a step length of 1) will eventually satisfy the Wolfe conditions, it should always be tried as the initial guess of the Armijo backtracking process. This step length is most often accepted. Only one state solution is then required by the line search.

• The line search procedure using Armijo backtracking allowed global convergence for the validation problems presented in chapter 4. However, since Wolfe's curvature condition is not enforced explicitly, there is always a chance that it could be violated. In the worst case, this can cause the BFGS search direction to fail to provide a decrease in the objective function. Because the cost of computing the objective function's gradient is currently very expensive, the Armijo backtracking procedure was the only choice available.

• The penalty formulation, which is currently used to take constraints into account, appears to be performing well. In the problem of section 4.1, convergence to the desired minimum was observed while the constraint was respected. Of the two variants implemented (constant penalty coefficients and updated penalty coefficients), no important differences in performance were observed. However, in the studied problem, only one constraint was present. As the number of constraints increase, it might become more difficult to set constant penalty coefficient while preserving global convergence properties. The method with updates in constraints should therefore be utilized to ensure that the domain of attraction of the local minimum does not shrink and to ensure better convergence behavior.

• Convergence to local minima is a well-known limitation of gradient-based algorithms. This is well illustrated by the geometric optimization problem, where all circles having the correct area and located inside the design space are local minima. For the two inverse problems, the optimal solution is unique within the design space and convergence to the target geometry was expected. Since algorithms converging to a global minimum, such as stochastic methods, require a very large number of state solutions, it is often more efficient to obtain many gradient-based optimal solutions, all starting from different geometries, and then to pick the best design among these. These
optimizations runs can be done concurrently, on many processors, without having to parallelize the solver.

In summary, the interactions between the various elements of the optimization framework were verified. The results to the studied constrained and unconstrained optimization problems illustrate the validity of these elements, both as individual entities and as a whole. There are still some unsolved difficulties that will need to be addressed in order to make further progress. Future work should be initially focused on correcting these problems. Then, improvements of the framework's performance and accuracy will need to be considered. The next section discusses some possible paths that could be explored in the future.

5.1 Future work

Even though the optimization framework has shown promising results when dealing with simple test cases, major improvements will have to be made to tackle practical design problems. These improvements can be divided into two categories: the immediate requirements, without which no further progress can be made, and the desirable elements that will improve the framework's accuracy and range of application.

Immediate requirements

By looking at the test problems presented in chapter 4, it is easy to see that the current range of application is quite limited. This is mainly due to the fact that this research was conducted concurrently with the development of the implicit Newton-Krylov GMRES solver. A broader spectrum of possible problems will be available once this solver is complete.

Gradient calculation is another area where modifications will have to be made. Computing gradients using finite-differences was a good choice for the development phase of the optimization algorithm. However, it has many shortcomings, especially when it comes to efficiency. It is believed that discrete adjoint methods would make the best choice for future applications. The adjoint equation would be solved using the same GMRES algorithm that is utilized to obtain the state solution. Nemec [46] uses this technique on structured grids and
CHAPTER 5. CONCLUSIONS AND RECOMMENDATIONS

reports that solving the adjoint problem takes one fifth to one half of the time required to solve the direct problem. This makes this technique very attractive, especially since its cost is virtually independent of the number of control variables (see section 3.2.3). Furthermore, the Jacobian matrix present in the adjoint equation is readily available from the direct problem. Finite-differencing will most certainly still have a role to play in the implementation of an alternative gradient computation algorithm as it can be used for validation purposes.

Another efficient gradient computation technique is already implemented, but still requires validation. Once the implicit solver works with additional physics package, it will be easy to compare incomplete sensitivities (section 3.2.5) with classic finite-differences. This technique is however bound to introduce accuracy errors in the gradient which could have an adverse effect on the convergence behavior.

With the ability to compute gradients quickly, some modifications to the line search procedure are in order. As mentioned earlier, the curvature condition cannot be enforced because it requires the knowledge of the gradient for every step length tested. However, with rapid gradient computation comes the possibility of explicitly enforcing this condition, making the optimizer more robust. Even though backtracking should implicitly enforce the curvature condition, there is no guarantee that the accepted step length will be long enough. This might cause the BFGS update to fail to obtain a descent direction or, even worse, to cause the whole optimization process to converge to a value that is not the desired minimum. Nocedal and Wright [48] propose a line search algorithm that builds a cubic interpolant of the objective function in the direction in which the line search is performed. This interpolant only requires the knowledge of the objective function's value and gradient at two points along this direction. The step length should then be chosen to coincide with the minimum of this cubic function. This procedure should accelerate the line search procedure while strictly enforcing the strong Wolfe condition.

Another shortcoming of the current implementation is the inability to use the high order capabilities of the solver. This is due to the choice of the deformation parameterization that is implemented outside the mesh generator. Even though it works very well with the test cases, replacing it would be advisable. Two reasons motivated the current choice for deformation parameterization: firstly, it suited well the planned inverse validation problems and secondly, if automatic differentiation were to be used for gradient calculation, it would have eliminated the need to differentiate the mesh generator. As adjoint methods now appear
CHAPTER 5. CONCLUSIONS AND RECOMMENDATIONS

to be the method of choice for future gradient calculation, active use of the mesh generator can be made. Since GRUMMP utilizes exact geometric representations of the domain's boundaries, it could easily be used for shape parameterization. As a matter of fact, all the information necessary is already available in the mesh generator. The only task remaining would be its integration into the optimization loop. This choice presents many advantages over the current parameterization. Firstly, it is more versatile since it can take into account every possible geometry that GRUMMP can produce. It is also a fully-automated process that does not require any intervention from the user.

Desirable features

Adaptive meshing is a feature that could improve both the accuracy of solutions and their evaluation time. Currently, GRUMMP refines meshes based on the geometric characteristics of the domain's boundaries. Usually, this implies that high curvature areas will contain more cells. This, of course, is adequate for geometric representation. However, the physical aspect of the problem might require a high cell concentration in sections where no particular geometric features are present. For example, to be accurately represented, shocks appearing in transonic regimes require a high mesh density on the top of the airfoil. Typically, this area does not have a high curvature compared to the leading edge. There is therefore a relatively low number of cells at this location (see figure 2.12). Modifying the global refinement and grading factors when creating the mesh can add cells in the critical areas and improve accuracy. The drawback is that it also refines the mesh in unnecessary zones, slowing the solution process considerably. Solution adapted refinement increases the number of cells only in critical areas of the domain, based on error estimators. The mesh is usually changed until no more variations in the solution can be observed. This results in an optimal spatial discretization for a given problem, greatly improving the accuracy, certainly improving the performance of the optimization process.

Taking into account multiple objective and multiple point design problems are other features that will be worth exploring in the future. As indicated by their name, multiple objective problems use more than one competing objective functions. For example, one might want to maximize the lift coefficient of an airfoil while minimizing the drag coefficient. The geometries achieving the optimal performance for each of these objectives can obviously be different. Many techniques can be used to solve such problems. A common approach is
the weighted-sum method. A weight is accorded to each objective which are then summed together. The same approach can be used to solve multiple points design problems. This type of problem can be encountered when the performance of a design needs to be optimized over a range of operating conditions. Designing a transonic airfoil required to fly between a minimum and a maximum Mach number is a good example of such problem.

Other desirable elements will surely be identified as the framework’s development continues. Future work will likely be driven by the needs of particular application problems. Until then, efforts should be concentrated on developing reliable applications in two dimensional aerodynamics.
Bibliography


