Development of a Design Tool for Aerodynamic Shape Optimization of Airfoils

by

Marc Secanell Gallart
Bachelor in Engineering, Technical University of Catalonia (UPC), 2002

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF APPLIED SCIENCE
in the
Department of Mechanical Engineering.

University of Victoria

Supervisors: Dr. Afzal Suleman

Abstract

A design tool for aerodynamic shape optimization is developed by coupling a CFD solver and a gradient-based optimization package. The aerodynamic solver is a parallel viscous Navier-Stokes solver with a Spallart-Allmaras one-equation turbulence model to account for turbulence. The optimization package contains three optimization algorithms: the modified method of feasible directions, sequential linear programming and sequential quadratic programming. The developed tool is used to obtain minimum drag airfoils subject to a minimum lift requirement. The results show a 20% reduction in drag with respect to the initial airfoil. The same optimization problem is solved using the three optimization algorithms. The sequential quadratic programming algorithm is found to outperform the other two algorithms, even though they all converge to a similar solution. Finally, the developed design tool is used for the preliminary design of a set of airfoils for an airfoil aircraft.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank my supervisor Dr.Afzal Suleman for giving me a chance to be part of his research group and for his support and encouragement throughout my graduate studies, especially during the initial stages of my thesis. It is difficult to transition from electrical to mechanical engineering, and his support was crucial in making this transition a successful one. I would also like to thank Dr. Ned Djilali for his many insightful comments about computational fluid dynamics (CFD); they were essential to my understanding of CFD. Finally, I would like to thank Dr. W.-S. Lu; his course in optimization motivated me to work in this challenging area, and he helped to set a solid foundation for further understanding of the subject.

Among my fellow graduate students, I would like to give special thanks to Gonçalo Pedro for his helpful comments and discussions concerning the computational fluid dynamics solver, SPARC, and on the mesh deformation algorithm. I would also like to thank Bart Stockdill for his useful insights on grid generation, and Pedro Gamboa for the many helpful discussions on aircraft morphing and design. Finally, thanks to all members of our research group: Dr. Suleman, Gonçalo Pedro, Bart Stockdill, Pedro Gamboa, Diogo Santos, Luís Falcão, David Cruz, Joana Rocha, Scott Bumpus, Ernest Ng, Sandra Makosinski, Geoff Woods, and, Ahmad Kermani. Thank you for sharing your enthusiasm for research and your expertise with me. Your support and companionship made these two years of graduate school an amazing experience.

Last but not least, I thank my wife, Tabitha Gillman, for her emotional support and my parents, Josep and Concepció, for giving me the encouragement and opportunities to learn. Without you this thesis would never have been written.

To my family

Chapter 1

Introduction

In the past, the design process for engineering systems such as aircraft was a trial-and-error process based on the experience of the designers. As designed systems have become more complex, the trial-and-error process has become an expensive and tedious task. To aid in the design of complex engineering systems, new design methods are necessary that rely on numerical tools to select the most efficient parameters for a desired design.

In the last two decades, increases in computing power, and new advances in the areas of computational fluid dynamics (CFD) and computational structural dynamics (CSD) have allowed engineers to model and analyze complex systems in a reasonable amount of computational time. Numerical methods for optimization have also emerged that are able to optimize a certain performance index with respect to a specified set of parameters. As a result of these advances it is now possible to couple an analysis tool, such as CFD and CSD, with a numerical optimization technique in order to obtain engineering design tools for optimal design.

In this thesis, a numerical optimization technique is coupled with a CFD solver

to develop a design tool for aerodynamic shape optimization. The motivation of the thesis is described in detail in the next section. The most recent advances in the areas of numerical optimization and aerodynamic shape optimization are then reviewed in sections 1.2.1 and 1.2.2. The scope of the thesis is presented in section 1.3. Finally, a description of the forthcoming chapters is presented in 1.4.

1.1 Motivation

The demand for surveillance unmanned aerial vehicles (UAVs) during the last few years has increased the amount of research being done in the development of efficient low Reynolds number airfoils, since most UAVs fly at a Reynolds number in the range of 100,000 to 1,000,000 [1]. In the past two decades, airfoil shape optimization using CFD codes has been applied to transonic airfoils and wings [2, 3]. However, shape optimization using CFD has seldom been applied to airfoils at low Reynolds numbers because of the complexity of the fluid flow at low Reynolds numbers [4]. Low Reynolds number airfoils have different aerodynamic characteristics than transonic and supersonic airfoils because the viscous forces have a larger impact on the aerodynamics of the airfoils. Therefore, in order to accurately predict the aerodynamic characteristics, an aerodynamic solver that takes into account the viscosity of the fluid is necessary in order to properly predict the aerodynamic characteristics of such airfoils. In this thesis, a fully viscous solver is used to compute the aerodynamic characteristics so that aerodynamic shape optimization can be performed at this low Reynolds number.

Surveillance UAVs have a large flight envelope. They are expected to: fly at high speeds in order to arrive at the surveillance area in the shortest amount of time possible, fly at low speed in the surveillance area and, takeoff and land within

the minimum amount of space possible. These requirements are difficult to meet efficiently with a conventional single airfoil configuration, and it is necessary to achieve a compromise between the different stages of flight. A possible solution to increase the efficiency of UAVs in all stages of flight is to develop an aircraft with a morphing airfoil. The morphing airfoil would be able to adapt its shape to the various mission requirements [5, 6]. To achieve this goal, a design tool must be developed to obtain optimal airfoils for each of the different stages of flight, so that the systems to deform the airfoil and a prototype aircraft can be designed and tested.

In the area of aerodynamic shape optimization, research has centered on reducing the amount of computational time needed to evaluate the functions and gradients for optimization. This is because the evaluation of aerodynamic objective function and constraints involves the solution of a set of partial differential equations (PDE) and therefore, it is the most time consuming task during the optimization process. Usually an optimization method is chosen a priori, and all the results are reported using this algorithm. A good selection of the optimization algorithm can reduce the amount of iterations necessary to obtain the optimum and, thereby, reducing the number of function and gradient evaluations and as a result reducing the computational time. However, the study of different optimizaiton algorithms used to solve aerodynamic shape optimization problems has not received the necessary attention. Only in [7] is an aerodynamic shape optimization problem solved using genetic algorithms and quasi-Newton method and the performance is compared. It is not known by the author that rates of convergence of different first-order optimization methods have been compared when solving an aerodynamic shape optimization problem using an accurate CFD analysis. The comparison of several first-order optimization methods is problem dependent and has yielded interesting results when applied to structural optimization [8]. This issue will also be addressed in this thesis.

Finally, aerodynamic shape optimization is an essential part of a multidisciplinary design optimization (MDO) tool for aerospace applications [9]. The development of an aerodynamic shape optimization tool in this thesis is an important step toward the understanding and development of an MDO application for aircraft design. This thesis has provided the necessary background for a future PhD thesis in MDO, which will involve the coupled optimization of aerodynamics and structures to obtain more realistic results.

1.2 Background

1.2.1 Optimization Theory

Advances in digital computer technology during the early fifties led to an incredible advance in the area of numerical methods for optimization. Since then, active research has produced a variety of methods for unconstrained and constrained optimization [10, 11, 12].

Engineering applications for optimization usually involve solving a nonlinear constrained optimization problem. Nonlinear constrained optimization problems involve the search for a minimum of a nonlinear objective function subject to a set of nonlinear constraints, and it is common for this optimization problem to have multiple extrema. Due to this difficulty, two different approaches have emerged in the area of nonlinear constraint optimization: local methods and global methods. Local methods aim to obtain a local minimum, and they cannot guarantee that the minimum obtained is the absolute minimum. These methods are usually first-order methods, i.e. they require information about the gradient of the objective function and the constraints. On the other hand, global methods aim to obtain the absolute minimum

of the function. They do not need any information about the gradient, and they are mostly based on stochastic procedures.

Local constrained methods can be classified into sequential methods and transformation-based methods. Sequential methods aim to solve the nonlinear constrained problem by iteratively solving a more simple constrained optimization problem. The most commonly used local sequential methods are: the method of feasible directions (MFD) and modified method of feasible directions (MMFD) [10, 13, 14], sequential linear programming (SLP) [13, 10, 15], sequential quadratic programming (SQP) [11, 16], and response surface approximation methods (RSM) [17, 18].

The MMFD is based on obtaining a sequence of feasible directions, i.e. directions that reduce the objective function and satisfy the constraints. Then, the design is moved in these directions until convergence to the optimum is achieved. The main drawback of this method is that it performs poorly if the constraints are highly nonlinear or discontinuous. The SLP method solves iteratively a linear programming subproblem obtained by linearizing the objective function and the constraints. Because linear approximations are only valid in the neighborhood of the linearization point, the norm of the search vector used to improve the design needs to be constrained. This constraint is achieved by imposing limits to the maximum allowable change on the design variables. These limits are known as move limits. The main drawback of SLP methods is the choice of the move limits; if the move limits are large, the method leads to oscillations on the convergence and it may not converge. On the other hand, if the move limits are too small, the SLP presents a low convergence rate. The main advantages of SLP methods are: they are simple to implement because they only involve the solution of a linear programming problem (LP) and, they are proved to yield good results if the move limits are properly adjusted [19]. Similarly, SQP methods are based on a second-order approximation of the objective function

and a linearization of the constraints [10] or on a second-order approximation of the Lagrangian of the original problem [11]. SQP methods are robust, have a fast convergence rate and are the most frequently used local nonlinear constrained optimization method. Finally, response surface approximation methods use interpolation models to model the objective function and constraints of the original problem. The interpolation model, usually a quadratic model, is then used to optimize the problem. The problem is solved iteratively and the approximation model is updated with the last solution.

Local transformation-based methods transform the original nonlinear optimization problem into an unconstrained optimization problem by adding a penalty function to the objective function. When the constraints are not satisfied, the penalty function increases its value thereby increasing the value of the objective function. Once the constrained problem has been transformed into an unconstrained problem, any unconstrained optimization algorithm can be used to solve the transformed problem. For example, a Quasi-Newton method or a conjugate-gradient method can be used [13, 10, 14, 11]. The most commonly used local transformation-based methods are: penalty methods [13, 10] and augmented Lagrangian methods [13, 10, 14]. The former eliminates the constraints by adding a penalty function to the objective function. The penalty function increases the value of the objective function when the constraints are violated. The main drawback with these methods is that the penalty functions are dependent on the problem at hand and, thereby making it difficult to generalize. On the other hand, Lagrangian methods solve the optimization problem by introducing a set of Lagrange multipliers that control the penalty function and make the Lagrange multipliers variables in the optimization program. All penalty methods have a main drawback; due to the introduced penalty, the objective function becomes highly nonlinear and this makes it difficult for the unconstrained methods to obtain

the minimum.

To conclude this description on local constrained methods, it is important to note that, although local methods do not aim for the global optima, they can be used to obtain such global optima. Several approaches can be used to continue searching once a local minimum has been obtained, thereby enabling the identification of all local minimum and, therefore, also the global minimum. Some of these methods based on a stochastic approach are: random multi-start methods [20, 21] and ant colony searches [22]. In the former method, once a minimum has been obtained, it restarts the optimizer with a new, randomly generated initial point. The second method uses the information from search agents (ants) in order to find the global minimum. Some other methods introduce a deterministic approach. For example, in the local-minimum penalty method [23] the objective function is penalized if the algorithm tends to go to an already known local minima.

The other group of constrained methods, the global methods, can be classified as direct or transformation-based. Direct methods solve the problem without transforming it into a simple problem. Transformation-based methods transform the initial constrained optimization problem into an unconstrained problem. Direct methods include covering methods and pure random searches. Covering methods follow a deterministic approach where regions of the design space are tested and eliminated if specific design criteria are not met. The most common of these methods are the interval methods [24]. Pure random searches evaluate randomly generated points until a minimum is obtained. The main drawback of both these methods is that they require a large number of function evaluations and are therefore computationally expensive.

Global transformation-based methods start by transforming the original problem into an unconstrained problem. Then, global unconstrained techniques are used to obtain the global minima. Popular unconstrained global methods are: genetic algo-

rithms [25], evolutionary algorithms [26] and simulated annealing [27]. These methods have the same drawback as the global direct methods; they require a large number of objective function evaluations, therefore the computational cost of the method becomes excessive when the evaluation of objective function and constraints is time consuming.

1.2.2 Aerodynamic Shape Optimization

During the late seventies Hicks et al. published one of the first papers on aerodynamic shape optimization [28]. In [28], Hicks used a first-order optimization technique in conjunction with a fully-potential inviscid flow solver in order to determine the optimal shape of a wing with respect to a certain performance criteria. Since [28], aerodynamic shape optimization has become an increasingly active area of research and several innovative methods for aerodynamic shape optimization of airfoils and wings [29, 30, 31], full aircraft configurations [2, 3], and even aero-structural optimization of full aircraft configurations [32] have been published.

In general, to solve an aerodynamic or airfoil shape optimization problem it is necessary to:

- Develop a set of parameters that represents the airfoil shape.
- Develop tools that, given a set of parameters, will compute the objective function and constraints of the optimization problem. This involves an aerodynamic analysis tool and algorithms to transform the set of parameters into the input necessary for the analysis tool.
- Develop tools that, given the design variables, objective function and constraints, will obtain a new aerodynamic shape close to the optimal or, the optimal shape itself.

- Develop tools that, given a set of design parameters, will compute the gradients of objective function and constraints with respect to these parameters.

The way each one of these problems is solved will influence the optimization process and the results. Therefore, in the following paragraphs, the most used techniques for items one, two and four are described. Item three was the focus of attention in the previous subsection.

Some of the parametrization techniques to represent an aerodynamic shape that have been applied to optimal shape optimization are: discrete parametrization, analytical parametrization, polynomial parametrization, spline parametrization and, CAD-based parametrization [33, 34]. A description of these methods is undertaken in section 3.1. In general, the ideal airfoil shape representation method is one that, with a small set of parameters can define any possible airfoil shape. If the parameterization technique is not able to represent all possible airfoil shapes, then the optimal solution is constrained by the shape parametrization, and a true optimal shape cannot be obtained [35]. On the other hand, if the number of parameters is large, the optimization problem becomes unnecessarily large and this will result in the need for excessive computational time in order to obtain the optimum.

Given an airfoil shape, the solution of the flow field yields the objective function and the constraints of the optimization problem. There are mainly four types of analysis codes used to solve the fluid flow: the fully potential flow solvers [36], the coupled boundary-layer potential flow solvers [37], Euler solvers [38], and the viscous Navier-Stokes flow solvers [38, 39]. Potential flow and Euler solvers solve the fluid flow by assuming that the viscosity effects are negligible. The coupled boundary-layer potential flow solvers assume that the viscous effects are only important in the neighborhood of the aerodynamic shape. Finally, the viscous Navier-Stokes flow solvers consider the viscous effects to be everywhere in the fluid, and they solve the Navier-

Stokes equations or the Reynolds Averaged Navier-Stokes (RANS) equations of the fluid flow. Viscous Navier-Stokes flow solvers are the most accurate solvers of those mentioned above. However, they are computationally more expensive than the other analysis codes and, for this reason, their use in aerodynamic shape optimization has been limited [31]. In aerodynamic optimization Euler solvers are the most commonly used.

Euler and viscous Navier-Stokes flow solvers use a discrete decomposition of the fluid domain, called fluid mesh, to solve the fluid flow. During an optimization process, the optimizer will change the aerodynamic shape at each iteration. Assuming that an initial mesh has been determined for the initial aerodynamic shape, when the shape of the body changes, the mesh must also change in order to adapt to the new shape. This is the third part of the optimization algorithm. A methodology is needed that will modify or reconstruct the fluid mesh everytime the aerodynamic shape changes. This is a one way fluid-structure interaction problem. Basically, two solutions have been applied to this problem: the mesh can be rebuilt each time the shape changes [40] or, the mesh is deformed with the body [41, 42]. In aerodynamic shape optimization, the second method is the one most commonly used.

Finally, if a first-order optimization algorithms is used, the gradients of objective function and constraints are needed for the optimization process. Since most of the computational time is used to obtain the gradients, this is one of the most critical parts of the optimization algorithm. In order to avoid having to find the gradients, non-gradient based methods can be used. However, the amount of function evaluations that these algorithms require in order to find the optimum, defeats the initial purpose. Several methods are available to obtain the gradients: analytical differentiation [43, 32, 44], finite-differences [32], complex-step derivative [32, 45], and automatic differentiation [46]. These methods are described in detail in section 3.4. Analytical

differentiation, also known as the adjoint method, is the most computationally efficient. The solution of the flow field and the gradient of the objective function are obtained in approximately twice the amount of time used to solve the flow field alone, independent of the number of design variables. On the other hand, finite-differences, complex-step derivatives and forward automatic differentiation are all more expensive methods of computing the derivatives because the computational time depends on the number of design variables. However, these methods are easier to implement and can be used as a black-box.

1.3 Scope of the Thesis

In this thesis, a code for two-dimensional, single element, aerodynamic shape optimization is developed. Given the fluid flow characteristics, the developed code is able to obtain an airfoil that optimizes certain performance criteria while satisfying certain constraints. For example, it can minimize the airfoil drag subject to a certain minimum lift requirement. The obtained airfoil has optimal aerodynamic performance for the fluid flow conditions specified; however, there is no guarantee that the airfoil will perform optimally away from the specified flow characteristics. The design of optimal airfoils for multiple fluid flow conditions, usually called robust airfoil optimization, is outside of the scope of this thesis, and it is part of the future work in our research group. The design tool has the capabilities to optimize airfoils at subsonic, transonic and supersonic speeds, however, only subsonic airfoils are of interest in this thesis.

As will be described in chapter 3, the design tool uses a spline representation of the airfoil, a viscous Navier-Stokes solver, a mesh adaptation method based on deforming the initial grid, and forward-differentiation to compute the gradients. The use of a spline representation is chosen because it reduces the number of design variables

necessary to represent the airfoil and it guarantees an almost free-form airfoil. The viscous Navier-Stokes solver is used because it guarantees accurate solutions at low Reynolds numbers where the viscous effects are important. Notice that the design tool has the capabilities of also using an Euler solver. Finally, forward-difference is used because of its ease of implementation.

The gradient-based sequential local constrained methods implemented in the commercial package DOT [47] are used for the optimization. The optimization methods used are: the modified method of feasible direction, sequential linear programming and sequential quadratic programming. These three methods were proven to yield good results in the area of structural optimization [8]. In this thesis, the three methods are analyzed to evaluate their performance and to determine the optimization algorithm that better adapts to aerodynamic shape optimization problems. These methods are local methods, therefore the design tool does not guarantee that the optimal shape is a global optimum.

1.4 Structure of the Thesis

A variety of algorithms have been selected, coupled and implemented in order to develop a design tool for aerodynamic shape optimization. In what follows, the rationale for these selections and the algorithms to be used in the design tool are described. Chapter 2 describes in further detail the optimization algorithms used to solve the aerodynamic shape optimization problem. An accurate understanding of the optimization algorithms allows for a better set up of the optimization problem and will be essential to understanding the performance of each one of the methods when solving the optimization problem. Once the optimization algorithms are described, chapter 3 describes: the method used to represent the airfoil and why it is

used, the viscous Navier-Stokes solver, the mesh deformation technique, the method used to compute the gradients and, finally, the implementation of the aerodynamic shape optimization design tool. Once the design tool has been described, chapter 4 shows several applications of the design tool. The design tool is first applied to solve a constrained drag minimization problem. To solve the optimization problem the three optimization algorithms are used. Then, the results are used to validate the program and to compare the different optimization algorithms and their performance. The design tool is then applied to the design of an airfoil morphing aircraft. Finally, chapter 5 contains concluding remarks and points out areas for future development in the areas of aerodynamic shape optimization and multidisciplinary design optimization.

Chapter 2

Optimization Theory

As with most engineering applications, the aerodynamic shape optimization problem is a nonlinear constrained optimization problem, also called a nonlinear programming problem (NLP). Therefore, most of this chapter focuses on methods used to solve these types of problems. Additionally, the chapter focuses on gradient-based optimization algorithms because these are the methods used to solve the problems in this thesis. Although gradient-based methods do not guarantee a global optimum, they are used because they are more efficient than non-gradient based methods at reducing the number of function evaluations needed to achieve the optimum. In this case, the reduction of the number of function evaluations is extremely important, since each function evaluation involves the solution of a computational fluid dynamics (CFD) problem to find the aerodynamic characteristics of a specified shape and this task is computationally extremely expensive. The three gradient-based constrained optimization methods implemented in the optimization package DOT are used to solve the aerodynamic shape optimization problem: the modified method of feasible directions, sequential linear programming and sequential quadratic programming [47].

In section 2.1, the general nonlinear constraint problem is mathematically formu-

lated and all its various elements are described. A description follows of the local optimality conditions for the general nonlinear constraint problem and also a general discussion of how such problems are solved. The succeeding sections are devoted to the discussion of the different algorithms that are used in this thesis. Section 2.2 describes the modified method of feasible directions algorithm. Section 2.3 describes the sequential linear programming (SLP) algorithm. Finally, section 2.4 describes sequential quadratic programming (SQP) algorithms.

2.1 Preliminaries

In general, a nonlinear unconstrained optimization problem aims to

$$\text{Minimize } f(\mathbf{x}) \tag{2.1a}$$

$$\text{w.r.t. } x_k \qquad \text{for } k = 1, 2, \ldots, n \tag{2.1b}$$

where the function to be minimized, $f(\mathbf{x})$, is known as the objective function. The objective function depends on a set of variables, $\mathbf{x}$, that can take arbitrary values. These variables are known as the design variables. The aim of the optimization algorithm is to obtain the value of the variables, $\mathbf{x}$, that makes the objective function minimal. This point is known as the solution of the optimization problem and is represented by $\mathbf{x}^*$. It is important to notice that maximizing a function, $m(\mathbf{x})$, is equivalent to minimizing the function $f(\mathbf{x}) = -m(\mathbf{x})$. Therefore any unconstrained maximization problem can be solved using an algorithm to solve the standard optimization problem in (2.1). The same argument holds for constrained optimization. For this reason, it is assumed in that follows that the optimization problem is a minimization problem.

The nonlinear constraint optimization problem is slightly different than (2.1) be-

cause the design variables cannot be arbitrarily chosen. The design variables must satisfy certain constraints. In general,the aim in a nonlinear constraint optimization problem is

$$\begin{aligned} \text{Minimize } & f(\mathbf{x}) && && \text{(2.2a)} \\ \text{w.r.t. } & x_k && \text{for } k = 1, 2, \ldots, n && \text{(2.2b)} \\ \text{subject to: } & h_i(\mathbf{x}) = 0 && \text{for } i = 1, 2, \ldots, p && \text{(2.2c)} \\ & g_j(\mathbf{x}) \leq 0 && \text{for } j = 1, 2, \ldots, q && \text{(2.2d)} \end{aligned}$$

In (2.2), $f(\mathbf{x})$ is the objective function, $\mathbf{x}$ is the vector of design variables, n is the number of design variables, $\{h_i(\mathbf{x}) = 0 \text{ for } i = 1, 2, \ldots, p\}$ are the equality constraints and $\{g_j(\mathbf{x}) \leq 0 \text{ for } j = 1, 2, \ldots, q\}$ are the inequality constraints. Furthermore, it is assumed that functions $f(\mathbf{x})$, $\mathbf{h}(\mathbf{x})$, $\mathbf{g}(\mathbf{x})$ are nonlinear, continuous and have continuous first and second order derivatives.

As discussed, the design variables must satisfy equations (2.2c) and (2.2d). Then, the design space or feasible region of (2.2), $\mathcal{R}$, is defined as

$$\mathcal{R} = \{\mathbf{x} : h_i(\mathbf{x}) = 0 \text{ for } i = 1, 2, \ldots, p \quad \text{and} \quad g_j(\mathbf{x}) \leq 0 \text{ for } j = 1, 2, \ldots, q\} \tag{2.3}$$

A point inside, $\mathbf{x} \in \mathcal{R}$ is considered a feasible point. In a constrained optimization problem the minimum must be in the feasible region. An unconstrained problem can be understood as a constrained problem with an unbounded or infinite feasible region.

Equality constraints are a set of equations that explicitly or implicitly relate some design variables with other design variables. Therefore, the number of equality constraints must be smaller than the number of design variables, $p \leq n$. Furthermore, if the number of design variables and the number of equality constraints are the same,

the design space will be a finite set of points and the solution of the problem is trivial. Since equality constraints introduce a relation between design variables, in an equality constrained problem only $n - p$ design variables are arbitrarily chosen. The rest of the variables are obtained from the value of the arbitrarily chosen variables and the equality constraints.

Inequality constraints are a set of equations that impose bounds on the design variables. Unlike equality constraints, the number of inequality constraints is unlimited. Inequality constraints can be active or inactive. For a feasible point, $\mathbf{x}$, if $g_k(\mathbf{x}) = 0$ the inequality constraint $g_k(\mathbf{x})$ is considered to be active. Equality constraints must be considered as active constraints. The feasible point $\mathbf{x}$ satisfying an active constraint is at a limit of the design space and not all its neighboring points are in the feasible region. On the other hand, for a feasible point $\mathbf{x}$, if $g_k(\mathbf{x}) < 0$ the inequality constraint is inactive. In this case, all its neighboring points are feasible and this inequality constraint does not need to be considered when looking for a new design point from $\mathbf{x}_1$. As an example, consider the problem in figure 2.1. The dashed region corresponds to the feasible region where $g_1(\mathbf{x}) \leq 0$, $g_2(\mathbf{x}) \leq 0$ and $g_3(\mathbf{x}) \leq 0$. At the feasible point $\mathbf{x}_1$, only constraint $g_2(\mathbf{x}_1)$ is equal to zero. Therefore, only this constraint is active. The other constraints are inactive.

So far the constraint optimization problem and its components have been described. However, it still remains to known how to solve the optimization problem in (2.2). Before the problem can be solved, it is necessary to know the properties of a minimizer. Minimizers can be classified as local minimizers and global minimizers.

Definition 2.1.1 (Global minimizer) $\mathbf{x}^*$ *is a global minimizer of problem* (2.2) *if* $f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \forall \mathbf{x} \in \mathcal{R}$.

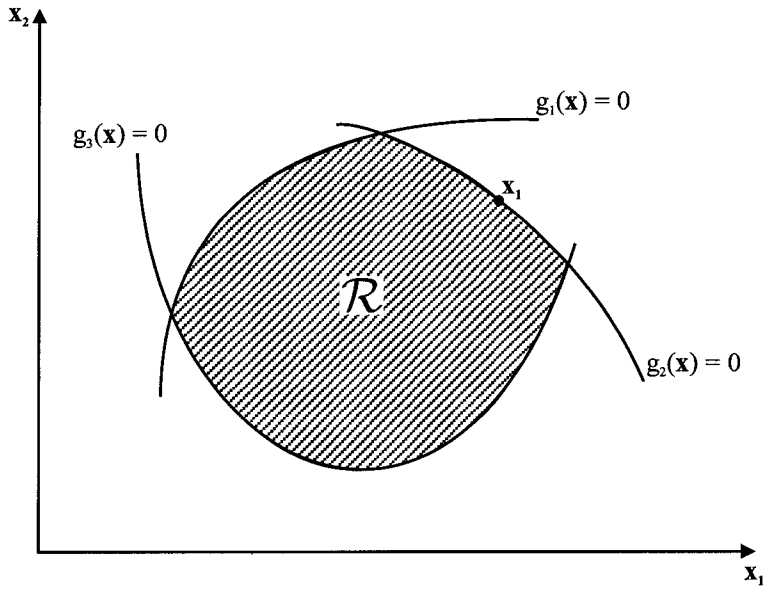

Figure 2.1: Design space, active and inactive constraints

Definition 2.1.2 (Local minimizer) $\mathbf{x}^*$ *is a local minimizer of problem* (2.2) *if* $f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \forall \mathbf{x} \in \mathcal{R}$ *with* $\|\mathbf{x}^* - \mathbf{x}\| < \delta$.

For a point $\mathbf{x}^*$ to be a local minimizer, it needs to satisfy the Karush-Kuhn-Tucker (KKT) conditions. These conditions are outlined here without proof, because they are the basis of all constrained optimization methods. For proof see [11].

Theorem 2.1.1 (Karush-Kuhn-Tucker (KKT) conditions) *If* $\mathbf{x}^*$ *is a local minimizer of the optimization problem* (2.2) *and the gradient of all active constraints at*

this point are linearly independent, then the following relations hold

$$h_i(\mathbf{x}^*) = 0 \text{ for } i = 1, \ldots, p \tag{2.4a}$$

$$g_j(\mathbf{x}^*) \leq 0 \text{ for } j = 1, \ldots, q \tag{2.4b}$$

$$\nabla_x \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \nabla_x f(\mathbf{x}^*) + \sum_{i=1}^{p} \lambda_i^* \nabla_x h_i(\mathbf{x}^*) + \sum_{j=1}^{q} \mu_j^* \nabla_x g_j(\mathbf{x}^*) = 0 \tag{2.4c}$$

$$\lambda_i^* h_i(\mathbf{x}^*) = 0 \text{ for } i = 1, \ldots, p \tag{2.4d}$$

$$\mu_j^* g_j(\mathbf{x}^*) = 0 \text{ for } j = 1, \ldots, q \tag{2.4e}$$

$$\mu_j^* \geq 0 \text{ for } j = 1, \ldots, q \tag{2.4f}$$

where

$$\mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = f(\mathbf{x}^*) + \sum_{i=1}^{p} \lambda_i^* h_i(\mathbf{x}^*) + \sum_{j=1}^{q} \mu_j^* g_j(\mathbf{x}^*) \tag{2.4g}$$

is the Lagrangian, $\nabla_x \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ is the gradient of the Lagrangian with respect to $\mathbf{x}$ and λ_i^* for $i = 1, \ldots, p$ and μ_j^* for $j = 1, \ldots, q$ are the Lagrange multipliers associated with the local optimum.

Notice that for unconstrained optimization problems, the KKT conditions become a single condition, the gradient of the objective function at the local minimizer must be zero. The KKT conditions must be satisfied for a point to be a local minimizer. However, this does not guarantee that the point is a local minimizer. To guarantee that a point is a local minimizer, another condition must be added to the KKT conditions.

Theorem 2.1.2 (Sufficient conditions for a local minimizer) *A point $\mathbf{x}^*$ is a local minimizer of the problem (2.2) if it satisfies the Karush-Kuhn-Tucker conditions*

and the following relation holds

$$\mathbf{N}^T(\mathbf{x}^*)\nabla_x^2 L(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)\mathbf{N}(\mathbf{x}^*) \text{ is positive definite} \tag{2.5}$$

where $\mathbf{N}^T(\mathbf{x}^*)$ *is a matrix whose columns are the basis of the subspace* $\mathcal{N}$*. Furthermore,* $\mathcal{N}$ *is the null space of the space whose basis is formed by the gradient of all active constraints.*

All constrained optimization algorithms are obtained from the definitions and theorems described above. Therefore, in the following sections, the concepts discussed above are used to derive the algorithms implemented in the optimization package DOT [47]. To derive the algorithms in DOT, the nonlinear optimization problem below will be considered because this was the problem used to derive the equation implemented in the optimization package.

$$\text{Minimize } f(\mathbf{x}) \tag{2.6a}$$

$$\text{w.r.t. } x_k \qquad \text{for } k = 1, 2, \ldots, n \tag{2.6b}$$

$$\text{subject to: } g_j(\mathbf{x}) \leq 0 \qquad \text{for } j = 1, 2, \ldots, q \tag{2.6c}$$

where $f(\mathbf{x})$ is the objective function, $\mathbf{x}$ is the vector of the design variables, n is the number of design variables and $\{g_j(\mathbf{x}) \leq 0 \text{ for } j = 1, 2, \ldots, q\}$ are the inequality constraints. It is also assumed that functions $f(\mathbf{x})$ and $\mathbf{g}(\mathbf{x})$ are nonlinear, continuous and have continuous first and second order derivatives. The equality constraints are not taken into consideration in the optimization problem because the authors of the optimization package considered them uncommon in engineering applications. Equality constraints can be introduced into the problem as two inequality constraints and the code is proved to yield good results. However, this method is not efficient compared with working with equality constraints directly inside the code.

2.2 Modified Method of Feasible Directions

The modified method of feasible directions is a robust nonlinear optimization algorithm. This method appeared in 1983 as a combination of the method of feasible direction [48, 13] and the reduced gradient method of Wolfe [48, 13, 49].

Using this method and assuming a feasible initial point, the optimization problem in (2.6) is solved by first finding a search direction, $\mathbf{d}$, that reduces the objective function and also guarantees the satisfaction of the constraints. Then, a one-dimensional search is performed to minimize the objective function in the search direction. This will result in a step size parameter, α. Finally, the design variables are updated using $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha \mathbf{d}_k$. This process is repeated until a solution is found. If the initial point is infeasible, an initial subproblem must be solved prior to the application of this procedure, to obtain an initial feasible point.

Assuming the current design point, $\mathbf{x}$, is a feasible point of the problem in (2.6), then the desired search direction should rapidly reduce the objective function while maintaining a feasible design. In order to reduce the objective function, the search direction should be at an angle of 90° to 270° with respect to the gradient of the objective function. Mathematically this condition is equivalent to

$$\boldsymbol{\nabla}^T f(\mathbf{x})\mathbf{d} \leq 0 \tag{2.7}$$

where $\mathbf{x}$ is the actual design point and $\mathbf{d}$ is the search direction. A search direction satisfying (2.7) is called a usable direction.

The search direction must maintain the design in the feasible region, away from the constraints. In order to guarantee this condition, the search direction should also be at an angle of 90° to 270° with respect to the gradient of the constraints.

Therefore, the search direction must also satisfy

$$\nabla^T g_i(\mathbf{x})\mathbf{d} \leq 0 \tag{2.8}$$

where $g_i(\mathbf{x}) \in J$, $J = \{g_i(\mathbf{x}) : CT \leq g_i(\mathbf{x}) \leq CTMIN \text{ for } i = 1, 2, \ldots, q_a\}$ is the set of active inequality constraints and, CT is a small negative number (for example -0.0001), $CTMIN$ is a small positive number (for example 0.0001) and q_a is the number of active constraints. A search direction, $\mathbf{d}$, satisfying (2.8) is called a feasible direction. In the case of (2.8) being smaller than zero, the search direction points toward the feasible region. If (2.8) is exactly zero, the search direction is tangent to the constraint. Then, it is necessary to take special care during the one-dimensional search, because if the constraint is convex, any design point following the search direction would become infeasible.

In conclusion, the desired search direction should be a usable and feasible direction. Since the main goal is to obtain the maximum possible reduction of the objective function, the desired search direction can be obtained by solving the optimization problem

$$\text{Maximize } -\nabla^T f(\mathbf{x})\mathbf{d} \tag{2.9a}$$

$$\text{subject to: } \nabla^T g_i(\mathbf{x})\mathbf{d} \leq 0 \tag{2.9b}$$

$$\mathbf{d}^T\mathbf{d} \leq 1 \tag{2.9c}$$

where $g_i \in J$, J is the set of active constraints. In matrix form and transforming the maximization problem into a minimization problem the aforementioned problem

becomes

$$\text{Minimize } \mathbf{c}^T\mathbf{d} \tag{2.10a}$$

$$\text{subject to: } \mathbf{A}\mathbf{d} \leq 0 \tag{2.10b}$$

$$\mathbf{d}^T\mathbf{d} \leq 1 \tag{2.10c}$$

where

$$\mathbf{c} = \begin{pmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_n} \end{pmatrix} \quad \mathbf{d} = \begin{pmatrix} d_1 \\ \vdots \\ d_n \end{pmatrix} \quad \mathbf{A} = \begin{pmatrix} \frac{\partial g_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial g_1(\mathbf{x})}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_{q_a}(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial g_{q_a}(\mathbf{x})}{\partial x_n} \end{pmatrix} \tag{2.10d}$$

$\mathbf{c} \in R^{n\times 1}$, $\mathbf{d} \in R^{n\times 1}$, $\mathbf{A} \in R^{q_a \times n}$, n is the number of design variables and q_a is the number of active inequality constraints.

Since only the active constraints are used to compute the search direction in (2.9), bounds on the design variables are introduced to guarantee a bounded solution to the problem. In this case, quadratic constraints are used to bound the solution instead of a linear constraint. The reason can easily be seen in figure 2.2 where the search direction is obtained using linear and quadratic constraints. If linear constraints are used for each design variable, the optimization algorithm will look for a direction that ends at one of the edges of the square (in n dimensions, hypersquare). This allows for a longer vector, resulting in a greater reduction in the objective function. Therefore, the constraints added to bound the problem would affect the solution. In order to guarantee that the constraints added to bound the solution do not affect the solution of the problem, the constraints should guarantee that the vectors are all the same length in all directions. This is achieved with a quadratic constraint. Then, all vectors are inscribed in a sphere (or a hypersphere in n dimensions).

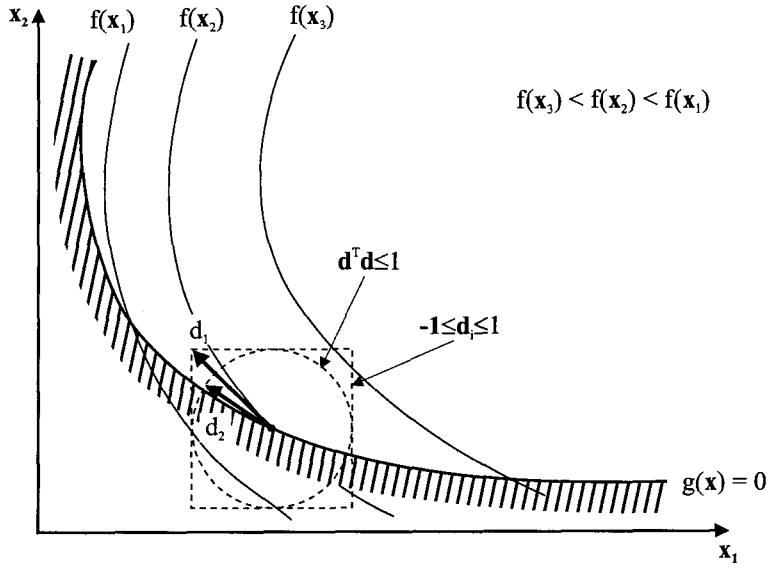

Figure 2.2: Obtained search direction using $-1 \leq \mathbf{d} \leq 1$ as a constraing, $\mathbf{d}_1$, or , $\mathbf{d}^T\mathbf{d} \leq 1$, $\mathbf{d}_2$

The introduction of quadratic constraints to bound the solution changes the optimization problem from a linear programming problem to a more complex problem. To solve the problem, let us consider the Karush-Kuhn-Tucker conditions. The KKT conditions for (2.10) require that,

$$\mathbf{Ad} \leq 0 \tag{2.11a}$$

$$\mathbf{d}^T\mathbf{d} \leq 1 \tag{2.11b}$$

$$\mathbf{c} + \mathbf{A}^T\boldsymbol{\mu} + \hat{\mu}\mathbf{d} = \mathbf{0} \tag{2.11c}$$

$$\boldsymbol{\mu}^T\mathbf{Ad} = \mathbf{0} \tag{2.11d}$$

$$\hat{\mu}\mathbf{d}^T\mathbf{d} = 0 \tag{2.11e}$$

$$\boldsymbol{\mu} \geq \mathbf{0} \qquad \hat{\mu} \geq 0 \tag{2.11f}$$

where $\boldsymbol{\mu} \in R^{q_a \times 1}$ and $\hat{\mu} \in R$ are the Lagrange multipliers for the inequality constraints of the problem. Multiplying (2.11c) by $\mathbf{A}$,

$$\mathbf{Ac} + \mathbf{AA}^T\boldsymbol{\mu} + \hat{\mu}\mathbf{Ad} = \mathbf{0} \tag{2.12}$$

and defining $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{b}$ as,

$$\mathbf{u} = \boldsymbol{\mu} \qquad \mathbf{v} = -\hat{\mu}\mathbf{Ad} \qquad \mathbf{b} = \mathbf{Ac} \tag{2.13}$$

equation (2.12) can be reformulated as,

$$\begin{pmatrix} \hat{\mathbf{A}} & I \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} = \mathbf{b} \tag{2.14a}$$

$$\mathbf{u} \geq \mathbf{0} \qquad \mathbf{v} \geq \mathbf{0} \qquad \mathbf{u}^T\mathbf{v} = 0 \tag{2.14b}$$

$$\text{where} \quad \hat{\mathbf{A}} = -\mathbf{AA}^T \tag{2.14c}$$

and I is the identity matrix. To find the optimal solution, a vector that satisfies the system of equations (2.14) needs to be found.

If $b_i \geq 0 \quad \forall i \in 1, \ldots, q_a$, then the vector created with

$$\mathbf{u} = \mathbf{0} \qquad \mathbf{v} = \mathbf{b} \tag{2.15}$$

is a solution of the system of equations in (2.14). However, if any $b_i < 0$, then the vector in (2.15) does not satisfy that $\mathbf{v} \geq \mathbf{0}$ and therefore it is not a solution. In this case, a solution is found by obtaining

$$\max \left. \frac{b_i}{\hat{A}_{ii}} \right|_{b_i<0} \tag{2.16}$$

and letting $k = i$ for i, which satisfies (2.16), and replace v_k by u_k. This is done by pivoting on $\hat{A}_{kk}$. Repeat the process until all $b_i \geq 0$. Notice that if $b_i \leq 0$, then by the definition of $\hat{A}_{kk}$ in (2.14c), $\hat{A}_{kk} \leq 0$.

Once a solution of (2.14) is obtained, the search direction is found using (2.11c),

$$\hat{\mu}\mathbf{d} = -(\mathbf{c} + \mathbf{A}^T\boldsymbol{\mu}) = -(\mathbf{c} + \mathbf{A}^T\mathbf{u}) \tag{2.17}$$

Since only the direction of $\mathbf{d}$ is of interest, because a line search will be performed in the search direction to find the optimal magnitude of the vector, $\hat{\mu}$ can be set to unity. In this way, the search direction is obtained.

Once a search direction has been found, the next step is to find the parameter α^* such that $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha^*\mathbf{d}_{k+1}$ with $f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k)$ and $\mathbf{x}_{k+1}$ is a feasible point. If the set $T = \{g_i : \boldsymbol{\nabla}^T g_i\mathbf{d} = 0 \text{ for } i = 1, \ldots, t\}$ is null, then the line search is straightforward. However, if the set T is not empty, the search direction is tangent to the constraints inside the set. This means that there is a risk of obtaining an infeasible design point, $\mathbf{x}_{k+1}$. To prevent this, in the line search the constraint is forced to also be active in the next iteration.

Assuming that the set T is empty. Then, before performing any one-dimensional search in parameter α, upper and lower bounds for the parameter are needed. Since, the search direction satisfies that $\boldsymbol{\nabla}^T f(\mathbf{x}_k)\mathbf{d}_{k+1} \leq 0$, the lower bound for α must be zero so that the inequality does not change sign. The upper bound is more difficult to obtain. If in the $k+1$ iteration the objective function is reduced by 10% and linear approximation of $f(\mathbf{x}_{k+1})$ holds,

$$f(\mathbf{x}_{k+1}) = 0.9f(\mathbf{x}_k) = f(\mathbf{x}_k) + \alpha\boldsymbol{\nabla}^T f(\mathbf{x}_k)\mathbf{d}_{k+1} \tag{2.18}$$

then, the parameter α should have the value

$$\alpha = \frac{-0.1|f(\mathbf{x}_k)|}{\nabla^T f(\mathbf{x}_k)\mathbf{d}_{k+1}} \tag{2.19}$$

where the absolute value guarantees that α is positive. Following the same idea, the gradients of the constraints can be used to find the α that will make the inactive constraints active at the next iteration. Notice that, since the constraints that are inactive are not part of the optimization in (2.9), the search direction can move toward them, i.e. it is possible that $\nabla^T g_i(\mathbf{x}_k)\mathbf{d}_{k+1} \geq 0$. For a constraint $g_i(\mathbf{x}_{k+1})$, the value of alpha that makes the constraint active at the next iteration can be found using a linear approximation of the constraint,

$$\begin{aligned} g_i(\mathbf{x}_{k+1}) &= g_i(\mathbf{x}_k) + \alpha \nabla^T g_i(\mathbf{x}_k)\mathbf{d}_{k+1} = 0 \\ \alpha &= \frac{-g_i(\mathbf{x}_k)}{\nabla^T g_i(\mathbf{x}_k)\mathbf{d}_{k+1}} \end{aligned} \tag{2.20}$$

Since it is desired to reduce the function by 10% and at the same time obtain a feasible point, the upper limit for α is obtained using

$$\alpha_u = \min\left(\frac{-0.1|f(\mathbf{x}_k)|}{\nabla^T f(\mathbf{x}_k)\mathbf{d}_{k+1}}\ ,\ \frac{-g_i(\mathbf{x}_k)}{\nabla^T g_i(\mathbf{x}_k)\mathbf{d}_{k+1}}\right) \text{ for } i = 1, \ldots, q \tag{2.21}$$

Once the upper and lower bounds have been obtained, a quadratic or cubic polynomial interpolation is used to approximate the objective function and the constraints inside the bounds. Then, using basic calculus, a set of values of the parameter α is obtained. One value, α_f minimizes the objective function and the other values α_i for $i = 1, ..., q$ make each one of the constraints active. Finally, the optimum value of α

that reduces the objective function as much as possible and creates a feasible point is

$$\alpha^* = \min(\alpha_f, \alpha_i) \quad i = 1, \ldots, q \tag{2.22}$$

where α^* is the optimal parameter for α. Now, the design variables are updated and the algorithm can proceed to the next iteration.

To end this discussion, the case where the set $T = \{g_i : \boldsymbol{\nabla}^T g_i \mathbf{d} = 0 \text{ for } i = 1, \ldots, t\}$ is not empty must be studied. The constraints inside the set T are tangent to the search direction. This means that if the design variables are updated following the search direction, there is a risk of obtaining an infeasible design point at the next iteration, $\mathbf{x}_{k+1}$, that will violate the constraints in T. To prevent constraint violation, in the line search the constraints in T are forced to be active in the next iteration, thereby, introducing a set of equality constraints into the line search. Because equality constraints are equivalent to relations between design variables, taking a first-order Taylor expansion of the constraints in T and taking into account that they are active at iteration k and must be active at iteration $k+1$, an approximate relation between design variables can be obtained,

$$g_i(\mathbf{x}_{k+1}) = g_i(\mathbf{x}_k) + \boldsymbol{\nabla}^T g_i(\mathbf{x}_k)\hat{\mathbf{d}}_{k+1} = \boldsymbol{\nabla}^T g_i(\mathbf{x}_k)\hat{\mathbf{d}}_{k+1} = 0 \tag{2.23}$$

where $g_i(\mathbf{x}_k) \in T$ and $\hat{\mathbf{d}}_{k+1} = \alpha \mathbf{d}_{k+1}$. Using the relations between variables obtained from (2.23), the search vector can be divided into dependent and independent

components. Therefore, the last equation can be written as

$$\mathbf{G}\hat{\mathbf{d}}_{k+1} = \begin{pmatrix} \mathbf{N} & \mathbf{D} \end{pmatrix} \begin{pmatrix} \hat{\mathbf{d}}_I \\ \hat{\mathbf{d}}_D \end{pmatrix}_{k+1} = 0 \tag{2.24}$$

$$\mathbf{G} = \begin{pmatrix} \boldsymbol{\nabla}^T g_1(\mathbf{x}_k) \\ \vdots \\ \boldsymbol{\nabla}^T g_t(\mathbf{x}_k) \end{pmatrix} \tag{2.25}$$

where t is the total number of constraints in T, $\mathbf{G} \in R^{t \times n}$, $\mathbf{N} \in R^{t \times n-t}$, $\mathbf{D} \in R^{t \times t}$, $\hat{\mathbf{d}}_D \in R^{t \times 1}$ and $\hat{\mathbf{d}}_I \in R^{n-t \times 1}$. Then, if $\mathbf{D}$ is nonsingular, the dependent variables can be obtained from the independent variables as

$$\hat{\mathbf{d}}_D = \mathbf{D}^{-1}\mathbf{N}\hat{\mathbf{d}}_I \tag{2.26}$$

where $\hat{\mathbf{d}}_D = \alpha_D \mathbf{d}_D$, $\hat{\mathbf{d}}_I = \alpha_I \mathbf{d}_I$. Notice that since $\mathbf{G}$ has more columns than rows, it is usually easy to obtain a nonsingular matrix $\mathbf{D}$. The matrix $\mathbf{D}$ can be obtained using row operations to transform $\mathbf{G}$ into its Row-Echelon Normal Form as described in [50]. $\mathbf{G}$ will be divided into a set of columns that form an identity matrix and other columns. The columns that form the identity matrix can be used as $\mathbf{D}$. The remaining columns will be $\mathbf{N}$.

Then, given the update vector of the independent design variables, $\hat{\mathbf{d}}_I$, the relationship in equation (2.26) can be used to obtain the update vector of the dependent design variables that guarantees that the constraints in T are active at the next iteration, $k+1$. First, α_I is computed as in the case of T being a null set. The update vector for the independent variables and the independent design variables are

obtained using,

$$\hat{\mathbf{d}}_I^{(k+1)} = \alpha_I^* \mathbf{d}_I^{(k+1)} \tag{2.27}$$

$$x_I^{(k+1)} = x_I^{(k)} + \hat{\mathbf{d}}_I^{(k+1)} \tag{2.28}$$

Then, the value $\hat{\mathbf{d}}_D^{(k+1)}$ is found using equation (2.26). Since the relation between variables given by equation (2.26) is derived from a first-order approximation of $g_i(\mathbf{x}_{k+1})$, the relation cannot be used to find an accurate value for $\hat{\mathbf{d}}_D^{(k+1)}$, since the constraints are nonlinear. Therefore, the values from equation (2.26) are used as an initial guess and a method similar to the Newton-Raphson method [51] is used to obtain the exact value for $\hat{\mathbf{d}}_D^{(k+1)}$. The procedure to obtain $\hat{\mathbf{d}}_D^{(k+1)}$ is,

Step 1 Determine $\hat{\mathbf{d}}_I^{(k+1)}$ using the method described in the first part with T null.

Step 2 Determine initial guess for $\hat{\mathbf{d}}_D^{(k+1)}$ using equation (2.26)

Step 3 Update design variables using $\mathbf{x}^{(k+1)} = \begin{pmatrix} \mathbf{x}_I^{(k)} + \hat{\mathbf{d}}_I^{(k+1)} \\ x_D^{(k)} + \hat{\mathbf{d}}_D^{(k+1)} \end{pmatrix}$ and compute $g_i(\mathbf{x}^{(k+1)})$. If $g_i(\mathbf{x}^{(k+1)}) = CTMIN$ stop. If not continue.

Step 4 Update $\hat{\mathbf{d}}_D^{(k+1)}$ using

$$\hat{\mathbf{d}}_D = \mathbf{D}^{-1}\left(-\mathbf{N}\hat{\mathbf{d}}_I^{(k+1)} - \mathbf{g}_T\right) \tag{2.29}$$

$$\mathbf{g}_T = \begin{pmatrix} g_1(\mathbf{x}^{(k+1)}) \\ \vdots \\ g_t(\mathbf{x}^{(k+1)}) \end{pmatrix} \tag{2.30}$$

and go to step 3.

At this point, if an initial feasible point is known, the optimization problem in

(2.6) can already be solved. The only problem that remains to be solved is how to obtain a feasible initial point. This will be discussed in the next section.

2.2.1 Infeasible Initial Point

The method of feasible direction assumes that an initial feasible point is known. The optimization process then searches for new feasible points that reduce the objective function until the optimal is found. However, it is sometimes difficult to obtain a feasible point; therefore, a technique for obtaining a feasible point is described in what follows.

To obtain a feasible point, a optimization problem similar to (2.9) can be formulated in order to obtain a search direction pointing toward the feasible region.

$$\text{Minimize } -\eta w + \frac{\boldsymbol{\nabla}^T f(\mathbf{x})}{\|\boldsymbol{\nabla}^T f(\mathbf{x})\|}\mathbf{d} \tag{2.31a}$$

$$\text{subject to: } \frac{\boldsymbol{\nabla}^T g_i(\mathbf{x})}{\|\boldsymbol{\nabla}^T g_i(\mathbf{x})\|}\mathbf{d} + \theta_i w \leq 0 \tag{2.31b}$$

$$\mathbf{d}^T\mathbf{d} + w^2 \leq 1 \tag{2.31c}$$

where $g_i(\mathbf{x}) \in J$, J is the set of active and violated constraints, η and θ_i are positive constants. If the value of the constant η is large, the first term of the objective function dominates the minimization, since the vector $\boldsymbol{\nabla}^T f(\mathbf{x})$ is normalized. To minimize the objective function w must be as large as possible. On the other hand, from (2.31b), if w is large, $\boldsymbol{\nabla}^T g_i(\mathbf{x})\mathbf{d}$ should be a negative large number, thereby obtaining a direction that satisfies the constraints. The second term in the objective function, (2.31a), is introduced to try to obtain a direction that will also reduce the objective function of the original problem $f(\mathbf{x})$; but as discussed earlier, this is not

the main goal. The constants η and θ_i are user defined.

If η is large, the second term in (2.31a) influences the optimization problem marginally, thereby giving total priority to obtaining a direction that will reach the feasible region, i.e. a direction perpendicular to the constraints. However, if η is small, then the first term will have a greater influence on the optimization problem, and the obtained direction will be a compromise between a direction that reduces the objective function and that points toward the feasible direction. In DOT, η is initialized with the value 5.0. If the obtained search direction does not return a feasible point, η is increased by a factor of 10 and a new direction is obtained. η is allowed to increase up to 1000 where an upper limit is set to avoid numerical ill-conditioning of the problem.

The selection process for the parameters θ_i follows a similar idea. These parameters are known as push-off factors. For large values of θ_i, the dot product $\boldsymbol{\nabla}^T g_i(\mathbf{x})\mathbf{d}$ has to be large and negative. Because of this requirement, the search direction is more perpendicular to the constraint, i.e. the search direction points more toward the feasible direction. In DOT, the parameters θ_i are chosen as

$$\theta_i = \theta_0 \left(1 - \frac{g_i(\mathbf{x})}{CT}\right)^2 \tag{2.32}$$

with $\theta_i \leq 50$ and where CT is a small negative constant that represents the minimum requirement to make the constraint inactive.

To solve the optimization problem in (2.31), the problem is first written in matrix

form

$$\text{Minimize } \mathbf{c}^T\hat{\mathbf{d}} \tag{2.33a}$$

$$\text{subject to: } \mathbf{A}\hat{\mathbf{d}} \leq 0 \tag{2.33b}$$

$$\hat{\mathbf{d}}^T\hat{\mathbf{d}} \leq 1 \tag{2.33c}$$

where

$$\mathbf{c} = \begin{pmatrix} \frac{1}{\|\nabla^T f(\mathbf{x})\|}\frac{\partial f(\mathbf{x})}{\partial x_1} \\ \vdots \\ \frac{1}{\|\nabla^T f(\mathbf{x})\|}\frac{\partial f(\mathbf{x})}{\partial x_n} \\ -\eta \end{pmatrix} \quad \hat{\mathbf{d}} = \begin{pmatrix} d_1 \\ \vdots \\ d_n \\ w \end{pmatrix} \tag{2.33d}$$

$$\mathbf{A} = \begin{pmatrix} \frac{1}{\|\nabla^T g_1(\mathbf{x})\|}\frac{\partial g_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{1}{\|\nabla^T g_1(\mathbf{x})\|}\frac{\partial g_1(\mathbf{x})}{\partial x_n} & \theta_1 \\ \vdots & \ddots & \vdots & \\ \frac{1}{\|\nabla^T g_{q_a}(\mathbf{x})\|}\frac{\partial g_{q_a}(\mathbf{x})}{\partial x_1} & \cdots & \frac{1}{\|\nabla^T g_{q_a}(\mathbf{x})\|}\frac{\partial g_{q_a}(\mathbf{x})}{\partial x_n} & \theta_{q_a} \end{pmatrix} \tag{2.33e}$$

$\mathbf{c} \in R^{n+1\times 1}$, $\hat{\mathbf{d}} \in R^{n+1\times 1}$, $\mathbf{A} \in R^{q_a \times n+1}$, n is the number of design variables of the initial problem and q_a is the number of active and violated inequality constraints. The problem in (2.33) has the same structure as (2.10). Therefore, the method outlined in the last section can be used to solve the optimization problem in (2.33).

Once the search direction has been obtained, a line search using the parameter α is performed in the search direction. As in the last section, the first step is to determine the bounds of the parameter α. Again, the lower bound is zero because the search direction points toward the feasible direction, and a negative sign in α would result in a change in the direction of the vector. The upper bound, however, is obtained differently here respect to the feasible design case. In this case, the value of α needs to guarantee that all constraints are satisfied. Therefore, it must guarantee that the violated constraints will not be violated in the next step and, at the same

time, it must guarantee that none of the satisfied constraints becomes violated. This can be achieved using

$$\alpha_u = \min\left(\max\left(\frac{-g_i(\mathbf{x})}{\nabla^T g_i(\mathbf{x}_k)\mathbf{d}}\right),\ \min\left(\frac{-g_j(\mathbf{x})}{\nabla^T g_j(\mathbf{x})\mathbf{d}}\right)\right) \tag{2.34}$$

where $i = 1, \ldots, q_v$; $j = 1, \ldots, q_s$; q_v is the number of violated constraints, and q_s is the number of active and satisfied constraints.

If the objective function increases in the search direction, the value of α given by equation (2.34) is the final upper bound. However, if the objective function decreases in the search direction, a larger value may exist for the upper bound of α than the one given by (2.34), that will satisfy the constraint and decrease the objective function. Then, α_u is found using

$$\alpha_u = \min\left(\max\left(\frac{-g_i(\mathbf{x})}{\nabla^T g_i(\mathbf{x}_k)\mathbf{d}},\ \frac{-0.1|f(\mathbf{x})|}{\nabla^T f(\mathbf{x})\mathbf{d}}\right),\ \min\left(\frac{-g_j(\mathbf{x})}{\nabla^T g_j(\mathbf{x})\mathbf{d}}\right)\right) \tag{2.35}$$

where again $i = 1, \ldots, q_v$; $j = 1, \ldots, q_s$; q_v is the number of violated constraints and q_s is the number of active and satisfied constraints.

Once the upper and lower bounds have been obtained, a quadratic or cubic polynomial interpolation is used to approximate the objective function and the constraints inside the bounds. Then, using basic calculus, optimal values of the parameter α are obtained for the objective function and each one of the constraints. The optimum value of α that reduces the objective function as much as possible and creates a feasible points is

$$\alpha^* = \begin{cases} \min\left(\alpha_j, \max\left(\alpha_f, \alpha_i\right)\right) & \text{if } \nabla^T f(\mathbf{x}) d < 0 \\ \min\left(\alpha_j, \max\left(\alpha_i\right)\right) & \text{if } \nabla^T f(\mathbf{x}) d \geq 0 \end{cases} \tag{2.36}$$

where α_f minimizes the objective function, α_i for $i = 1, ..., q_v$ makes the violated

constraints active the next iteration, and α_j for $j = 1, ..., q_s$ makes the active and inactive constraints active the next iteration. Then, using the optimal parameter α^* the design variables are updated. If the design point is infeasible, the process described above is repeated until a feasible point is obtained. Once a feasible design is obtained the algorithm proceeds as described in 2.2.

2.2.2 Implementation

The modified method of feasible directions described in the preceding two sections can be simplified using the following set of steps:

Step 1 Start $k = 0$ and $\mathbf{x}_k = \mathbf{x}_0$

Step 2 Evaluate $f(\mathbf{x}_k)$ and $\mathbf{g}(\mathbf{x}_k)$

Step 3 If $\mathbf{g}(\mathbf{x}_k) \leq 0$, continue. If not, $\mathbf{x}_k$ is an infeasible design and the steps in section 2.2.1 must be followed until a feasible design is obtained.

Step 4 Identify set of active constraints, i.e. $J = \{g_i(\mathbf{x}) : CT \leq g_i(\mathbf{x}) \leq CTMIN \text{ for } i = 1, 2, \ldots, q_a\}$

Step 5 Evaluate gradient of objective function and active constraints

Step 6 Determine search direction by solving the optimization problem in (2.10)

Step 7 Identify the constraints in the set $T = \{g_i : \nabla^T g_i \mathbf{d} = 0 \text{ for } i = 1, \ldots, t\}$. Perform the appropriate one-dimensional search to find α^* depending on if set T is empty or not.

Step 8 Update the design point, $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha^* \mathbf{d}_k$

Step 9 Check for convergence. If $\frac{f(\mathbf{x}_{k+1})-f(\mathbf{x}_k)}{f(\mathbf{x}_k)} \leq 0.001$ in two consecutive iterations or $\mathbf{d}_i \leq 0.001 \quad \forall i = 1, \ldots, n$ convergence is achieved, STOP.

Step 10 Update the iterations counter, $k = k + 1$. Go to step 2.

The main advantages of the modified method of feasible directions are: all the designs after a full cycle are feasible, gradient calculations are infrequent and the gradient calculations involve only active constraints. Its main disadvantages are: infeasible designs occur during line search calculations and, the Newton's method used to bring the design to the feasible region during a line search sometimes does not converge. Note that, if an efficient way to compute the gradients exists, the aforementioned advantage of infrequent gradient calculation becomes a drawback.

2.3 Sequential Linear Programming

Sequential Linear Programming is considered to be one of the simplest methods to implement; the only necessary requirement is an efficient linear programming solver. To solve the nonlinear programming problem in (2.6), a linear programming subproblem is created by linearizing the original objective function and constraints at each iteration. The LP problem is then solved using a linear programming method to obtain an increment of the design variables that moves toward the optimal solution. Then, the design variables are updated. This process is repeated iteratively until the solution is found.

To transform the nonlinear programming problem (2.6) into a LP problem a Taylor series expansion of the objective function and constraints is used. In the neighborhood of the design variables at the k iteration, $\mathbf{x}_k$, the objective function and constraints

can be approximated as

$$f(\mathbf{x}_{k+1}) = f(\mathbf{x}_k) + \boldsymbol{\nabla}^T f(\mathbf{x}_k)\mathbf{d} + \mathcal{O}(\|\mathbf{d}\|^2) \tag{2.37a}$$

$$g_i(\mathbf{x}_{k+1}) = g_i(\mathbf{x}_k) + \boldsymbol{\nabla}^T g_i(\mathbf{x}_k)\mathbf{d} + \mathcal{O}(\|\mathbf{d}\|^2) \tag{2.37b}$$

where $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}$ and $i = 1, \ldots, q$.

Using the linear approximation in (2.37), a linear programming problem can be formulated to obtain a vector, $\mathbf{d}$, that achieves a new design point, $\mathbf{x}_{k+1}$, which reduces the objective function $f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k)$ and satisfies the constraints in (2.37b). Unfortunately, there is a problem, the error produced by using the approximation in (2.37) is proportional to $\|\mathbf{d}\|^2$. Therefore, the norm of $\mathbf{d}$ must be limited to small values. Constraining the norm of $\mathbf{d}$ introduces second order equations to the optimization problem. These new equations defeat the purpose of the linear approximation, since an LP subproblem cannot be created. Therefore, in order to keep the optimization problem linear, instead of limiting the norm of the update vector, $\mathbf{d}$, the components of the vector are limited. The constraints on the components of vector $\mathbf{d}$ are known as move limits. Their selection is critical, and will be discussed below.

Finally, the LP subproblem at iteration k can be formulated as

$$\text{Minimize } \boldsymbol{\nabla}^T f(\mathbf{x}_k)\mathbf{d} \tag{2.38a}$$

$$\text{subject to: } g_i(\mathbf{x}_k) + \boldsymbol{\nabla}^T g_i(\mathbf{x}_k)\mathbf{d} \leq 0 \tag{2.38b}$$

$$\mathbf{d}_L \leq \mathbf{d} \leq \mathbf{d}_U \tag{2.38c}$$

where $\mathbf{d}_L$ and $\mathbf{d}_U$ are the vectors of lower and upper limits of the components of $\mathbf{d}$. i.e. the move limits. Notice that the term $f(\mathbf{x}_k)$ is omitted from the objective function because to minimize $f(\mathbf{x})$ is the same as minimize $f(\mathbf{x}) + K$ where K is a

constant because the constant does not depend on the design variables, and therefore it cannot be minimized.

Once the problem is stated, it is necessary to obtain a method to compute the move limits $\mathbf{d}_L$ and $\mathbf{d}_U$ to be used in the LP problem. This is the most critical step of the SLP algorithm. A correct value for the move limits should be as large as possible, and, at the same time, should limit the error of the approximations to say, 1%. If the move limits chosen are too small, the algorithm will converge to the optimum slowly. On the other hand, if the move limits are too large, the algorithm may never converge.

If correct move limits are chosen, sequential linear programming has proven to be a robust and efficient method for nonlinear optimization, [8]. Efficient ways to determine move limits have been described in [52, 19, 53, 54]. In [15] several of these methods are compared. In DOT, the move limits are computed using the technique in [52]. Therefore, this is the technique that will be discussed here.

The approach in DOT is as follows [52, 47]. At the first iteration, the move limits for the ith component are chosen as

$$(\mathbf{d}_U)_i = -(\mathbf{d}_L)_i = \max\left(k|(\mathbf{x}_0)_i|, k\right) \tag{2.39}$$

where k is a user defined constant, in this work $k = 0.05$. Then, in the following iteration, if the maximum constraint violation increases, the move limits are reduced by 50% . This is done in order to reduce the design space when the approximations yield infeasible results. However, this can result in premature convergence if the move limits are reduced too quickly. To correct for this problem, individual move limits are increased by 33% for all variables that reach its upper or lower bound in two consecutive iterations. The 33% is used based on the experience of the designers of

DOT.

Once the move limits are obtained, the problem in (2.38) can be solved using any LP algorithm, such as the simplex method or an interior-point method. In DOT, the modified method of feasible directions is used instead of a LP algorithm. This has the advantage that if the move limits are too small and no feasible solution exists, the MMFD code will yield a solution that minimizes the constraints while an interior-point method or the simplex method may yield inaccurate results. However, if an optimization problem with a large number of design variables was to be solved, using the modified method of feasible directions to solve the LP problem would reduce the computational efficiency of the algorithm. In those cases, a more efficient LP solver, such as a simplex method or an interior-point method for LP would increase the efficiency of the method. These methods for LP are described in [51, 11]. If an interior-point method or the simplex methods are used, special care should be taken in the cases where no feasible solution exists for the LP problem.

Finally, the design variables are updated using

$$\mathbf{x}_{k+1} = \begin{cases} \mathbf{x}_U & \text{if } \mathbf{x}_k + \mathbf{d} > \mathbf{x}_U \\ \mathbf{x}_k + \mathbf{d} & \text{if } \mathbf{x}_L \leq \mathbf{x}_k + \mathbf{d} \leq \mathbf{x}_U \\ \mathbf{x}_L & \text{if } \mathbf{x}_k + \mathbf{d} < \mathbf{x}_L \end{cases} \tag{2.40}$$

where $\mathbf{x}_U$ and $\mathbf{x}_L$ are the upper and lower bounds for the design variables if they exist.

2.3.1 Implementation

The sequential linear programming method in DOT can be summarized as follows:

Step 1 Start $k = 0$ and $\mathbf{x}_k = \mathbf{x}_0$

Step 2 Evaluate $f(\mathbf{x}_k)$ and $\mathbf{g}(\mathbf{x}_k)$

Step 3 Evaluate gradient of objective function and constraints

Step 4 Determine the move limits $\mathbf{d}_L$ and $\mathbf{d}_U$. If it is the first iteration use (2.39), if not use the method described.

Step 5 Create the LP problem as described in (2.38) using information from steps 3 and 4

Step 6 Update the design point using (2.40)

Step 7 Check for convergence. If $\frac{f(\mathbf{x}_{k+1})-f(\mathbf{x}_k)}{f(\mathbf{x}_k)} \leq 0.001$ in two consecutive iterations or $\mathbf{d}_i \leq 0.001 \quad \forall i = 1, \ldots, n$ convergence is achieved, STOP. Notice $f(\mathbf{x}_{k+1})$ will need to be evaluated.

Step 8 Update the iterations counter, $k = k + 1$. Go to STEP 2.

2.4 Sequential Quadratic Programming

Sequential Quadratic Programming is considered to be one of the most efficient methods for solving nonlinear constraint optimization problems. In order to implement this method, the main requirement is an efficient quadratic programming solver. To solve the nonlinear programming problem in (2.6), a quadratic programming subproblem is formulated using the KKT conditions of the original problem to obtain the direction that will lead the current design variables to the optimum. Then, a line search is performed in this direction in order to obtain the next design. The quadratic subproblem is then updated and the process is repeated until convergence is achieved.

The optimal solution of the optimization problem in (2.6) must satisfy the KKT conditions

$$g_j(\mathbf{x}^*) \leq 0 \text{ for } j = 1, \ldots, q \tag{2.41a}$$

$$\boldsymbol{\nabla}_x \mathcal{L}(\mathbf{x}^*, \boldsymbol{\mu}^*) = \boldsymbol{\nabla}_x f(\mathbf{x}^*) + \sum_{j=1}^{q} \mu_j^* \boldsymbol{\nabla}_x g_j(\mathbf{x}^*) = 0 \tag{2.41b}$$

$$\mu_j^* g_j(\mathbf{x}^*) = 0 \text{ for } j = 1, \ldots, q \tag{2.41c}$$

$$\mu_j^* \geq 0 \text{ for } j = 1, \ldots, q \tag{2.41d}$$

where

$$\mathcal{L}(\mathbf{x}^*, \boldsymbol{\mu}^*) = f(\mathbf{x}^*) + \sum_{j=1}^{q} \mu_j^* g_j(\mathbf{x}^*) \tag{2.41e}$$

is the Lagrangian, $\boldsymbol{\nabla}_x \mathcal{L}(\mathbf{x}^*, \boldsymbol{\mu}^*)$ is the gradient of the Lagrangian and $\{\mathbf{x}^*, \boldsymbol{\mu}^*\}$ is the optimal design variable-multipliers pair.

In a nonlinear problem and, even more so, when numerical methods are used to compute the objective function or constraints, the equations in the KKT conditions can be quite tedious to obtain analytically. Furthermore, once obtained, it is difficult to solve the complex system of equations generated. Therefore, it is necessary to approximate the KKT conditions using simpler equations and then solve the problem sequentially. Given a set of design variables and Lagrangian multipliers in the kth iteration, $\{\mathbf{x}_k, \boldsymbol{\mu}_k\}$, the goal is to obtain an increment, $\{\boldsymbol{\delta}_x, \boldsymbol{\delta}_\mu\}$, so that the next design variables will satisfy the approximated KKT conditions at point $\{\mathbf{x}_k + \boldsymbol{\delta}_x, \boldsymbol{\mu}_k + \boldsymbol{\delta}_\mu\}$. The approximate KKT conditions at the kth iteration are obtained using a first order Taylor expansion of the equations in (2.41)

$$g_j(\mathbf{x}_k + \boldsymbol{\delta}_x) \approx g_j(\mathbf{x}_k) + \boldsymbol{\delta}_x^T \boldsymbol{\nabla}_x g_j(\mathbf{x}_k) \leq 0 \text{ for } j = 1, \ldots, q \tag{2.42a}$$

$$\nabla_x \mathcal{L}(\mathbf{x}_k + \boldsymbol{\delta}_x, \boldsymbol{\mu}_k + \boldsymbol{\delta}_\mu) \approx$$

$$\approx \nabla_x \mathcal{L}(\mathbf{x}_k, \boldsymbol{\mu}_k) + \nabla_x^2 \mathcal{L}(\mathbf{x}_k, \boldsymbol{\mu}_k)\boldsymbol{\delta}_x + \nabla_{x\mu}^2 \mathcal{L}(\mathbf{x}_k, \boldsymbol{\mu}_k)\boldsymbol{\delta}_\mu = 0 \quad (2.42b)$$

$$(\boldsymbol{\mu}_k + \boldsymbol{\delta}_\mu)_j g_j(\mathbf{x}_k + \boldsymbol{\delta}_x) = (\boldsymbol{\mu}_{k+1})_j g_j(\mathbf{x}_k + \boldsymbol{\delta}_x) \approx$$

$$\approx (\boldsymbol{\mu}_{k+1})_j (g_j(\mathbf{x}_k) + \boldsymbol{\delta}_x^T \nabla_x g_j(\mathbf{x}_k)) = 0 \text{ for } j = 1, \ldots, q \quad (2.42c)$$

$$(\boldsymbol{\mu}_k + \boldsymbol{\delta}_\mu)_j = (\boldsymbol{\mu}_{k+1})_j \geq 0 \text{ for } j = 1, \ldots, q \quad (2.42d)$$

where

$$\nabla_x \mathcal{L}(\mathbf{x}_k, \boldsymbol{\mu}_k) \approx \nabla_x f(\mathbf{x}_k) + \sum_{j=1}^{q} (\mu_k)_j \nabla_x g_j(\mathbf{x}_k) \quad (2.42e)$$

$$\nabla_x^2 \mathcal{L}(\mathbf{x}_k, \boldsymbol{\mu}_k) \approx \nabla_x^2 f(\mathbf{x}_k) + \sum_{j=1}^{q} (\mu_k)_j \nabla_x^2 g_j(\mathbf{x}_k) \quad (2.42f)$$

$$\nabla_{x\mu}^2 \mathcal{L}(\mathbf{x}_k, \boldsymbol{\mu}_k) \approx \sum_{j=1}^{q} \nabla_x g_j(\mathbf{x}_k) \quad (2.42g)$$

The equations above can be rewritten in matrix form as

$$\mathbf{b}_k + \mathbf{A}_k \boldsymbol{\delta}_x \leq 0 \quad (2.43a)$$

$$\mathbf{H}_k \boldsymbol{\delta}_x + \mathbf{p}_k + \mathbf{A}_k^T \boldsymbol{\mu}_{k+1} = 0 \quad (2.43b)$$

$$\boldsymbol{\mu}_{k+1}(\mathbf{b}_k + \mathbf{A}_k \boldsymbol{\delta}_x) \quad (2.43c)$$

$$\boldsymbol{\mu}_{k+1} \geq 0 \quad (2.43d)$$

where

$$\mathbf{b}_k = \begin{pmatrix} g_1(\mathbf{x}_k) \\ \vdots \\ g_q(\mathbf{x}_k) \end{pmatrix} \qquad \mathbf{A}_k = \begin{pmatrix} \nabla_x^T g_1(\mathbf{x}_k) \\ \vdots \\ \nabla_x^T g_q(\mathbf{x}_k) \end{pmatrix} \tag{2.43e}$$

$$\mathbf{p}_k = \nabla_x f(\mathbf{x}_k) \qquad \mathbf{H}_k = \nabla_x^2 f(\mathbf{x}_k) + \sum_{j=1}^{q} (\mu_k)_j \nabla_x^2 g_j(\mathbf{x}_k) \tag{2.43f}$$

$\boldsymbol{\delta}_x \in R^{n\times 1}$, $\boldsymbol{\mu}_k \in R^{q\times 1}$, $\mathbf{b}_k \in R^{q\times 1}$, $\mathbf{A}_k \in R^{q\times n}$, $\mathbf{p}_k \in R^{n\times 1}$, and $\mathbf{H}_k \in R^{q\times n}$.

The approximation of the KKT conditions yields the new set of equations in (2.43). This new set of equations is equivalent to the KKT conditions of the QP problem,

$$\text{Minimize } \frac{1}{2}\boldsymbol{\delta}_x^T \mathbf{H}_k \boldsymbol{\delta}_x + \boldsymbol{\delta}_x^T \mathbf{p}_k \tag{2.44a}$$

$$\text{subject to: } \mathbf{A}_k \boldsymbol{\delta}_x + \mathbf{b}_k \leq 0 \tag{2.44b}$$

which can be solved easily. In DOT, the QP problem is solved using the MMFD. By solving the QP problem in (2.44), the design variables increment $\boldsymbol{\delta}_x$ is obtained. Notice that the new optimization problem minimizes a second order approximation of the Lagrangian of the original problem, and not just the objective function. Therefore, the design variable increment reduces the objective function and violated constraints, while at the same time obtaining the minimum Lagrangian possible.

Two main problems arise from the above formulation: how to compute the Hessian matrix and how to check that the approximation of the KKT conditions holds. To compute the matrix $\mathbf{H}_k$, second order derivatives of the objective function and constraints are necessary, and these computations are computationally expensive. Therefore, a method is necessary to obtain an approximation of $\mathbf{H}_k$ that will converge to the real Hessian matrix as the optimization problem evolves, and, that will

always be positive definite - in order to guarantee the existence of a solution for the optimization problem in (2.44). The method used to obtain an approximation of the matrix $\mathbf{H}_k$ will be discussed later in this section.

Assuming that a value for $\mathbf{H}_k$ exists, and that a solution of the problem in (2.44) has been obtained, it then becomes necessary to check the accuracy of the solution. If $\boldsymbol{\delta}_x$ is large, then the approximation of the KKT conditions that led to the optimization problem in (2.44) may not be accurate. In order to test the accuracy of the approximation and to reduce the value of $\boldsymbol{\delta}_x$ if necessary, a parameter α is used to control the dimensions of vector $\boldsymbol{\delta}_x$. Then, once the QP problem in (2.44) is solved, the vector of the design variables is updated using

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha \boldsymbol{\delta}_x \tag{2.45}$$

and the parameter α is obtained by solving the one-dimensional optimization problem

$$\underset{\text{r.t. } \alpha}{\text{Minimize}} \; f(\mathbf{x}_k + \alpha \boldsymbol{\delta}_x) + \sum_{j=1}^{q} (r_{k+1})_j \max(0, g_j(\mathbf{x}_k + \alpha \boldsymbol{\delta}_x)) \tag{2.46a}$$

where

$$(r_{k+1})_j = \begin{cases} |(\mu_{k+1})_j| & j = 1, \ldots, q \quad \text{if first iteration} \\ \max(|(\mu_{k+1})_j|, \frac{1}{2}((r_k)_j + |(\mu_{k+1})_j|) & j = 1, \ldots, q \quad \text{else} \end{cases} \tag{2.46b}$$

and $(\mu_{k+1})_j$ are the Lagrangians from the approximated problem computed using

$$\boldsymbol{\mu}_{k+1} = (\mathbf{A}_k \mathbf{A}_k^T)^{-1} \mathbf{A}_k (\mathbf{H}_k \boldsymbol{\delta}_x + \mathbf{p}_k) \tag{2.47}$$

This problem is solved using the line search with $\alpha \in [0, 1]$ and with an initial α of

one. Finally, once α has been found the Lagrange multipliers are also updated using

$$\boldsymbol{\mu}_{k+1} = (\mathbf{A}_k \mathbf{A}_k^T)^{-1} \mathbf{A}_k (\alpha \mathbf{H}_k \boldsymbol{\delta}_x + \mathbf{p}_k) \tag{2.48}$$

At the first iteration, the Hessian matrix of the Lagrangian, $\mathbf{H}_k$, is approximated by the identity matrix. In subsequent iterations, the approximation of the Hessian matrix of the Lagrangian denoted by $\mathbf{B}_k$, is obtained using the BFGS formula [16, 13, 47], commonly used in quasi-Newton methods and written here for completeness without proof

$$\mathbf{B}_{k+1} = \mathbf{B}_k - \frac{\mathbf{B}_k \mathbf{d}_k \mathbf{d}_k^T \mathbf{B}_k}{\mathbf{d}_k^T \mathbf{B}_k \mathbf{d}_k} + \frac{\boldsymbol{\eta}_k \boldsymbol{\eta}_k^T}{\mathbf{d}_k^T \boldsymbol{\eta}_k} \tag{2.49a}$$

where

$$\mathbf{d}_k = \alpha \boldsymbol{\delta}_x = \mathbf{x}_{k+1} - \mathbf{x}_k \tag{2.49b}$$

$$\boldsymbol{\eta}_k = \theta \mathbf{y}_k + (1 - \theta) \mathbf{B}_k \mathbf{d}_k \tag{2.49c}$$

$$\mathbf{y}_k = \boldsymbol{\nabla}_x \mathcal{L}(\mathbf{x}_{k+1}, \boldsymbol{\mu}_{k+1}) - \boldsymbol{\nabla}_x \mathcal{L}(\mathbf{x}_k, \boldsymbol{\mu}_{k+1}) \tag{2.49d}$$

$$\theta = \begin{cases} 1.0 & \text{if } \mathbf{d}_k^T \mathbf{y}_k \geq 0.2 \mathbf{d}_k^T \mathbf{B}_k \mathbf{d}_k \\ \frac{0.8 \mathbf{d}_k^T \mathbf{B}_k \mathbf{d}_k}{\mathbf{d}_k^T \mathbf{B}_k \mathbf{d}_k - \mathbf{d}_k^T \mathbf{y}_k} & \text{if } \mathbf{d}_k^T \mathbf{y}_k < 0.2 \mathbf{d}_k^T \mathbf{B}_k \mathbf{d}_k \end{cases} \tag{2.49e}$$

There are other SQP methodologies available in the literature [16]. However, the SQP discussed here, and implemented in DOT, is one of the most popular versions of SQP. The DOT implementation has one main disadvantage with respect to other SQP software; the QP problem is solved using the MMFD, which is computationally inefficient compared to the simplex method for QP or interior-point methods. However, since the number of design variables in this thesis is quite small, this problem is minimal.

2.4.1 Implementation

The sequential quadratic programming method in DOT can be summarized in the following steps

Step 1 Set $k = 0$, $\mathbf{x}_k = \mathbf{x}_0$ and $\mathbf{B}_k = I$ where I is the identity matrix

Step 2 Evaluate $f(\mathbf{x}_k)$ and $\mathbf{g}(\mathbf{x}_k)$

Step 3 Evaluate gradient of objective function and constraints

Step 4 Solve the QP problem in (2.44).

Step 5 Compute the Lagrange multipliers using (2.47) and compute α by doing the line search in (2.46)

Step 6 Update the design point and the Lagrange multipliers using (2.45) and (2.48) respectively

Step 7 Update the Hessian matrix approximation, $\mathbf{B}_k$, using (2.49)

Step 8 Check for convergence. If $\frac{f(\mathbf{x}_{k+1})-f(\mathbf{x}_k)}{f(\mathbf{x}_k)} \leq 0.001$ in two consecutive iterations or $\mathbf{d}_i \leq 0.001 \quad \forall i = 1, \ldots, n$ convergence is achieved, STOP. Notice $f(\mathbf{x}_{k+1})$ will need to be evaluated.

Step 9 Update the iterations counter, $k = k + 1$. Go to STEP 2.

Chapter 3

Aerodynamic Optimization

In engineering design by optimization, the first step towards the optimal solution of the design problem is to formulate the problem in the general optimization form described in chapter 2

$$\text{Minimize } f(\mathbf{x}) \tag{3.1a}$$

$$\text{w.r.t. } x_k \qquad \text{for } k = 1, 2, \ldots, n \tag{3.1b}$$

$$\text{subject to: } g_j(\mathbf{x}) \leq 0 \qquad \text{for } j = 1, 2, \ldots, q \tag{3.1c}$$

where, from an engineering perspective, $\mathbf{x}$ is the design parameters that can be changed in the design, $f(\mathbf{x})$ is a measure of the performance of the design and $g_j(\mathbf{x})$ are a set of equations used to guarantee that the design meets the necessary requirements.

In aerodynamic shape optimization, the design variables, $\mathbf{x}$, in the aforementioned problem are a set of parameters that represents the shape of the airfoil. The objective function, $f(\mathbf{x})$, is a combination of aerodynamic characteristics. Finally, the constraints, $g_j(\mathbf{x})$, are aerodynamic and geometric constraints. An example of an

aerodynamic shape optimization problem is

$$\text{Minimize } c_d(\mathbf{x}) \tag{3.2a}$$

$$\text{w.r.t. } x_k \qquad \text{for } k = 1, 2, \ldots, n \tag{3.2b}$$

$$\text{subject to: } c_l^{desired} - c_l(\mathbf{x}) \leq 0 \tag{3.2c}$$

$$g_j(\mathbf{x}) \leq 0 \qquad \text{for } j = 1, 2, \ldots, q \tag{3.2d}$$

where $\mathbf{x}$ represents an airfoil shape, $c_l(\mathbf{x})$ and $c_d(\mathbf{x})$ are the lift and drag coefficient for the airfoil shape represented by $\mathbf{x}$, $c_l^{desired}$ is a minimum lift requirement and $g_j(\mathbf{x})$ are a set of linear geometric constraints to constrain the shape of the airfoil. In this case, the objective function and the aerodynamic constraint, (3.2c), are obtained numerically by solving a set of partial differential equations that model the flow around the airfoil. In particular, the Navier-Stokes equations and a transport equation to model the turbulence are solved iteratively using a numerical method to obtain the value of $c_l(\mathbf{x})$ and $c_d(\mathbf{x})$. The evaluation of these two terms is the most time consuming part of the optimization process taking more than 95% of the computational time.

To write and solve the problem in (3.2) it is necessary to:

- Develop a set of parameters, $\mathbf{x}$, that represents the airfoil shape.
- Develop tools that, given a set of parameters, $\mathbf{x}$, will compute the objective function and constraints. Namely, an analysis tool, and an algorithm to transform the set of parameters into the input necessary for the analysis tool.
- Develop tools that, given design variables, $\mathbf{x}$, objective function, $f(\mathbf{x})$, and constraints, $g_j(\mathbf{x})$, will obtain a new shape closer to the optimal. These tools were discussed in chapter 2.

- Develop tools that, given a set of parameters, $\mathbf{x}$, will compute the gradients of the objective function and constraints with respect to the parameters $\mathbf{x}$.

In this chapter, all the components listed above are discussed in detail. Section 3.1 focuses on the different methods used to represent the airfoil and, in particular, it focuses on the method finally chosen in this work. Section 3.2 describes the computational fluid dynamics solver used to compute the aerodynamic characteristics of the given shapes. CFD solvers need a discretization of the flow domain in order to obtain the characteristics of the airfoil. Such discretization is called the fluid mesh. In section 3.3, the method used to translate the set of parameters, $\mathbf{x}$, that defines the airfoil into a fluid mesh is discussed. Section 3.4 describes the existing methods used to compute the gradients. The method chosen in this thesis to obtain the gradients is contrasted with other methods. Finally, section 3.5 focuses on the implementation of aerodynamic shape optimization and the coupling of all different elements described in order to create an efficient and robust program for airfoil design.

3.1 Shape Representation

To formulate an aerodynamic shape optimization problem in the form of (3.1), a shape representation for the airfoil shape is necessary. This representation should be able to represent the airfoil with a small set of parameters, and it should also be able to represent a wide variety of shapes. The former property is necessary in order to reduce the computational time required to solve the optimization problem, and in order to reduce the amount of time necessary to compute the gradient of objective function and constraints. The latter property is important in order to guarantee that the optimal shape is not restricted by limitations of shape representation capabilities. Additionally, the shape representation needs to be able to be converted into the

necessary input for the analysis tools used to solve the analysis problem - in this case a CFD solver.

Shape parametrization is an active area of research. In the literature, several methods have been suggested for representing an aerodynamic shape [33, 34]. The main methods are

- Analytic Representation
- Partial Differential Equation Representation
- Discrete Representation
- Polynomial Representation
- Spline Representation
- CAD Representation

In the analytic representation, given an original shape, a set of functions are used to deform the original shape. The parameters that determine the value of the functions are used as the design variables. Therefore, the design variables represent the deformations added to the original shape in order to create the new shape. This method was used in [28] to optimize the shape of several airfoils and wings. This method has the advantage of reducing the necessary number of design variables to a small set. It also gives the airfoil an smooth surface. On the other hand, it is only applicable to simple geometries and the deformations are dependent on the shape functions used.

The second method, the partial differential equation representation, generates the shape by solving a boundary-value problem of an elliptic partial differential equation.

This method has two main drawbacks: it is time consuming to implement and it is only applicable to simple geometries.

The discrete representation method can only be used when using computational methods to solve the analysis problem. If a computational method is used, the boundary nodes of the fluid flow mesh at the surface of the airfoil can be used as the design variables. This method is easy to implement and can be used with any geometry. However, it requires a large number of design variables and the final shape can have high frequency oscillations. In [55, 56], this method is used with a smoothing function to prevent the high frequency oscillation of the shape.

A polynomial can also be used to represent the airfoil. In this method, the coefficients of a polynomial are used as the design variables. An example is the NACA representation for an airfoil. If the NACA representation is used, then the thickness, maximum camber and position of the maximum camber can be used as the design variables to define the airfoil. [57] describes the NACA representation. The main advantage in this method is that a small set of design variables can be used. The main disadvantage is that if a low order polynomial is used some shapes become impossible to represent.

A spline can also be used to represent the airfoil. Spline representations use a sum of weighted polynomials to represent the airfoil. In this case, the set of weighting parameters, called control points, are used as the design variables. There exist several types of splines: Bezier curves, B-splines and non-uniform rational B-spline (NURBS). The most complete spline representations are the NURBS which are able to represent any shape using a small set of parameters, create a smooth shape and offer the possibility of modifying the shape locally. However, they are difficult to implement. B-splines are easier to implement and they offer the same advantages as NURBS, except that they are unable to represent implicit conic sections; however,

these shapes are not common in airfoils. Finally, Bezier curves are the simplest spline to implement. However, in order to represent a complex geometry they require high order polynomials and they do not offer the same advantages as the other two groups.

In this work, B-splines are used to represent the shape of the airfoil. In particular, a uniform cubic B-spline is used [58]. There are several reasons for using a B-spline: it allows for a reduction in the number of design variables required because the control points of the B-spline can be used as the design variables; the perturbation of one control point has only local effects on the design shape; it results in a curve with C^2 continuity, therefore it guarantees a shape that is most likely possible to manufacture; it is easy to implement; and, it generates a smooth airfoil without any high frequency oscillations. In the next subsection, the B-spline representation is described.

3.1.1 Uniform Cubic B-Spline Airfoil Representation

In general, a two-dimensional curve can be represented parametrically as

$$Q(\bar{u}) = (X(\bar{u}), Y(\bar{u})) \tag{3.3}$$

where $X(\bar{u})$ and $Y(\bar{u})$ are single-value functions of the parameter $\bar{u}$. $X(\bar{u})$ and $Y(\bar{u})$ represent the Cartesian coordinates x and y of the points on the curve for any value of $\bar{u}$. In order to be able to represent complex curves, $X(\bar{u})$ and $Y(\bar{u})$ are divided into several pieces, called segments. Each segment is characterized by a different polynomial representation. The different polynomials are then joined together to create a piecewise polynomial curve. The values of $\bar{u}$ where the segments are joined, $\bar{u}_i$, are called knots. Therefore, to represent a curve, a sequence of knots is created and then, at each segment between knots, a polynomial is used to represent the curve.

In a uniform cubic B-spline (B- stands for basis) the curve is created by a sum of

weighted basis functions over a uniform knot sequence

$$Q(\bar{u}) = \sum_i \mathbf{V}_i B_i(\bar{u}) = \sum_i (x_i B_i(\bar{u}), y_i B_i(\bar{u})) \tag{3.4}$$

where $\mathbf{V}_i$ are the control points of the spline and $B_i(\bar{u})$ are the basis functions. The basis functions are defined as

$$B_i(\bar{u}) = \begin{cases} b_{i-1} = 0 & \text{if } -\infty < \bar{u}_i \leq i \\ b_i = \frac{1}{6}u^3 & \text{if } i < \bar{u}_i \leq i+1 \\ b_{i+1} = \frac{1}{6}\left(1 + 3u + 3u^2 - 3u^3\right) & \text{if } i+1 < \bar{u}_i \leq i+2 \\ b_{i+2} = \frac{1}{6}\left(4 - 6u^2 - 3u^3\right) & \text{if } i+2 < \bar{u}_i \leq i+3 \\ b_{i+3} = \frac{1}{6}\left(1 - 3u + 3u^2 - u^3\right) & \text{if } i+3 < \bar{u}_i \leq i+4 \\ b_{i+4} = 0 & \text{if } i+4 < \bar{u}_i \leq \infty \end{cases} \tag{3.5}$$

where $u = \frac{\bar{u}-\bar{u}_i}{\bar{u}_{i+1}-\bar{u}_i}$ and $\bar{u} \in [\bar{u}_i, \bar{u}_{i+1}]$. The composite polynomial in (3.5) is obtained by using a cubic polynomial to represent each segment, requiring that at the joined positions, first derivatives and second derivatives match and requiring that $b_i(0) + b_{i+1}(0) + b_{i+2}(0) + b_{i+3}(0) = 1$. Figure 3.1 shows the shape of the basis function. The uniform knot sequence is a sequence of knots where all the knots are different and a certain distance apart. In this case, the knot sequence considered is $\bar{u}_{i+1} = \bar{u}_i + 1$.

From (3.4) and (3.5), the coordinates of a point in the curve on an knot interval $\bar{u}_i \leq \bar{u} < \bar{u}_{i+1}$ are obtained as

$$Q(\bar{u}) = \sum_i \mathbf{V}_i B_i(\bar{u}) = \mathbf{V}_{i-3}B_i(\bar{u}_{i-3}) + \mathbf{V}_{i-2}B_i(\bar{u}_{i-2}) + \mathbf{V}_{i-1}B_i(\bar{u}_{i-1}) + \mathbf{V}_i B_i(\bar{u}_i) \tag{3.6}$$

This equation is used to compute any point in the curve. Then, in order to be able to

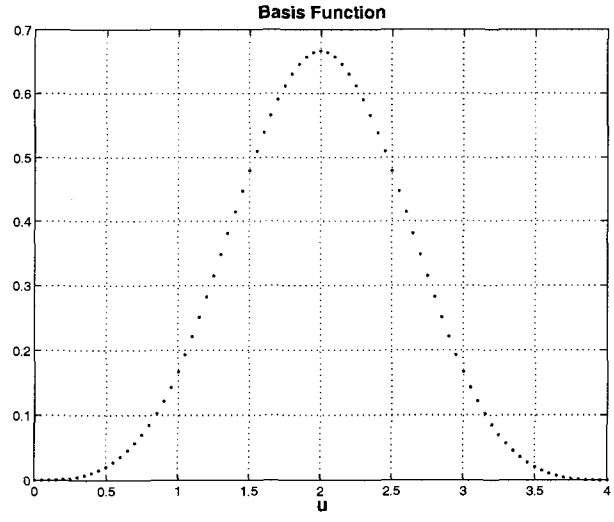

Figure 3.1: B-Spline basis function

use equation (3.6), four basis function segments must exist at any location, including the initial and final curve interval. Therefore, the first curve segment must start at $\bar{u}_3$ and the last segment must be until $\bar{u}_{m+1}$ where m is the number of control points. For example, figure 3.2 shows the four segments used to create the initial segment of the spline curve as well as the control points numbering.

From figure 3.2, it can be observed that the curve will start at the second control point. However, it is sometimes desirable to start the curve at the initial control point. In this case, phantom vertices can be created to generate a new initial or new final control point. Several methods can be used to create phantom vertices [58]. In this thesis, the phantom vertices at the beginning and at the end are created using

$$\mathbf{V}_{-1} = 2\mathbf{V}_0 - \mathbf{V}_1 \tag{3.7}$$

$$\mathbf{V}_{m+1} = 2\mathbf{V}_m - \mathbf{V}_{m-1} \tag{3.8}$$

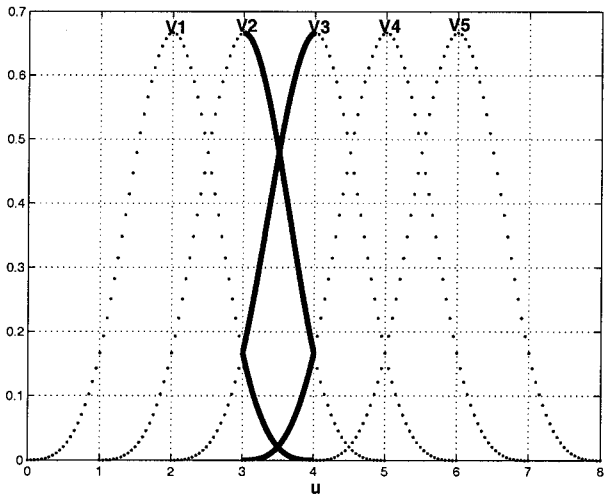

Figure 3.2: Basis functions used to create the initial segment of the curve $Q(\bar{u})$

which guarantees that the curve starts and ends at $\mathbf{V}_0$ and $\mathbf{V}_m$ respectively. This is particularly useful in this case because the initial and final control points are the trailing edge of the airfoil, therefore the curve should start at such points.

Given the above discussion, the main properties of the uniform B-spline can be described as follows. Due to (3.4) and also to the fact that the basis functions are zero everywhere but in the four segments around the control point, moving a control point only affects part of the curve. This gives local control over the B-spline curve generated. Moreover, due to the requirements set to create the composite polynomial in (3.5) the basis functions are C^2 continuous and since the sum of C^2 continuous function is also C^2 continuous, any B-spline is C^2 and therefore, smooth.

To summarize, in this thesis, a uniform cubic B-spline with 15 control points is used to represent the airfoil shapes. From the 15 control points, the y-coordinate of control points numbered 1-5 and 7-11 in figure 3.3 are used as design variables. The three control points aligned at the x position, one at the fixed point $(0, 0)$ and the

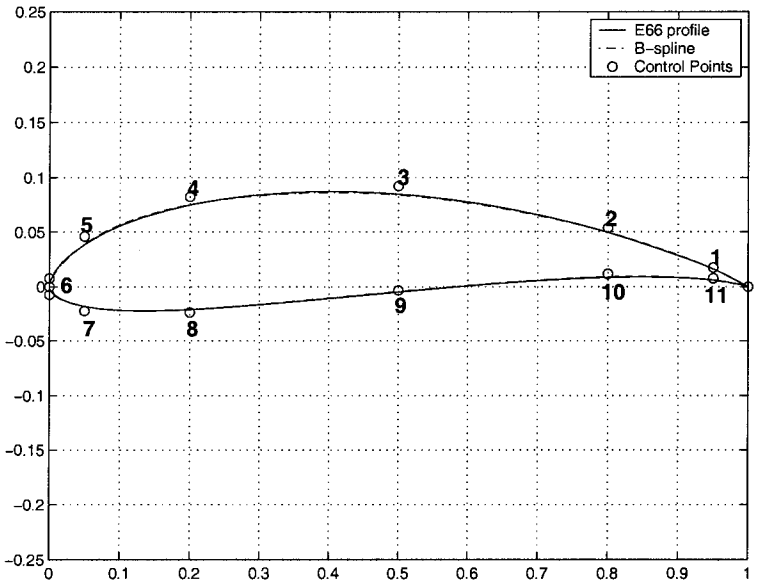

Figure 3.3: B-spline representation of a E66 airfoil using 15 control points

other two symmetrically distributed around $(0, 0)$ in the y direction are used to force the different airfoils to have the same leading edge point. Then, a last design variable is introduced to represent the distance between the two aforementioned points at the leading edge. This is done in order to control the radius of curvature of the leading edge. This B-spline representation can be used to represent a great variety of existing and new airfoil shapes. Its adaptability makes it a good candidate for shape optimization, because it guarantees an almost free-form representation of the airfoil. For example, in figure 3.3 the B-spline is used to represent the Eppler 66 airfoil [59]. It can be observed that the B-spline accurately represents the E66 airfoil.

3.2 Fluid Flow Analysis

Once an airfoil is obtained from the shape representation, the aerodynamic characteristics of this airfoil must be obtained by solving the fluid flow around the airfoil. In this case, the flow around the airfoil is assumed to be steady, viscous and incompressible, and it is solved using a viscous Navier-Stokes CFD code. The CFD code used is the Structured PArallel Research Code (SPARC). SPARC has been developed by Magagnato [39] at the University of Karlsruhe, Germany, and the source code is available free of charge in exchange for further development and debugging. SPARC is implemented in Fortran90 and it is designed to be used in distributed memory parallel architectures, such as a parallel cluster. The parallel capabilities are implemented using the message passing interface (MPI) programs. In this thesis, only small modifications have been made to SPARC, the sole purpose being to debug the code and to ease the interactions between SPARC and the other programs used in the optimization process.

SPARC is a very general code able to solve a large variety of problems: steady and unsteady flows, laminar and turbulent flows, compressible and incompressible flows and, viscid and inviscid flows. Furthermore, it has a large number of turbulence models. Upon solution of the flow, SPARC returns lift, drag and pitch moment coefficients as well as the pressure and velocities of the flow field. Therefore, this code is an excellent choice for solving fluid flow problem because it is able to output to the optimization algorithm, the required aerodynamic characteristics and, enables the optimization code to optimize airfoils for any flow regime. Furthermore, there is the possibility of comparing the behavior of several turbulence models. Finally, because the source code is available, the code may be modified to include the computations of the analytic sensitivities of the design variables.

To obtain the flow field around the airfoil SPARC solves the compressible mass-weighted Reynolds Averaged Navier Stokes (RANS) equations [60, 39]

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho\overline{u})}{\partial x} + \frac{\partial(\rho\overline{v})}{\partial y} + \frac{\partial(\rho\overline{w})}{\partial z} = 0 \qquad \text{compressible continuity} \tag{3.9a}$$

$$\begin{aligned}
\frac{D(\rho\overline{u})}{Dt} &= -\frac{\partial \overline{p}}{\partial x} + \frac{\partial}{\partial x}\left(\mu_l \frac{\partial \overline{u}}{\partial x} - \rho\overline{u'^2}\right) + \frac{\partial}{\partial y}\left(\mu_l \frac{\partial \overline{u}}{\partial y} - \rho\overline{u'v'}\right) + \\
&\quad \frac{\partial}{\partial z}\left(\mu_l \frac{\partial \overline{u}}{\partial z} - \rho\overline{u'w'}\right) \qquad \text{x momentum} \qquad (3.9b) \\
\frac{D(\rho\overline{v})}{Dt} &= -\frac{\partial \overline{p}}{\partial y} + \frac{\partial}{\partial x}\left(\mu_l \frac{\partial \overline{v}}{\partial x} - \rho\overline{u'v'}\right) + \frac{\partial}{\partial y}\left(\mu_l \frac{\partial \overline{v}}{\partial y} - \rho\overline{v'^2}\right) + \\
&\quad \frac{\partial}{\partial z}\left(\mu_l \frac{\partial \overline{v}}{\partial z} - \rho\overline{v'w'}\right) \qquad \text{y momentum} \qquad (3.9c) \\
\frac{D(\rho\overline{w})}{Dt} &= -\frac{\partial \overline{p}}{\partial z} + \frac{\partial}{\partial x}\left(\mu_l \frac{\partial \overline{w}}{\partial x} - \rho\overline{u'w'}\right) + \frac{\partial}{\partial y}\left(\mu_l \frac{\partial \overline{w}}{\partial y} - \rho\overline{v'w'}\right) + \\
&\quad \frac{\partial}{\partial z}\left(\mu_l \frac{\partial \overline{w}}{\partial z} - \rho\overline{w'^2}\right) \qquad \text{z momentum} \qquad (3.9d)
\end{aligned}$$

where ρ is the density of the flow, $\overline{u}$, $\overline{v}$, $\overline{w}$ are the mean x, y and z velocities of the flow, $\overline{p}$ is the mean pressure, and μ_l is the laminar viscosity of the flow. To solve the above equations, the pressure is obtained by using the energy equation and the ideal gas law, [60]. The Reynolds stress terms

$$\tau_{ij}^t = -\rho\overline{u_i' u_j'} \text{ for } i = 1,2,3 \text{ and } j = 1,2,3 \tag{3.10}$$

are not solved and are modeled by the addition of an eddy viscosity term to the laminar viscosity. In the last equation, indexes i, j and k imply summation over all values of repeated subscript and u_1, u_2 and u_3 are equivalent to u, v and w in equation (3.9). The eddy viscosity is obtained using the Spalart-Allmaras one equation turbulence model without the tripping term [61]. The Spalart-Allmaras

model in SPARC has been extensively tested at the University of Victoria to predict the lift and drag of a NACA0012 airfoil and it has yielded good results for steady and unsteady state simulations [62]. The main disadvantage of the Spalart-Allmaras model is that it is unable to properly predict laminar-to-turbulent transition because the tripping term in [61] is not implemented in SPARC. In the future, the Spalart-Allmaras model in SPARC should be improved by introducing a tripping source term to force laminar-to-turbulent transition, as discussed in [61].

SPARC solves the partial differential equations in (3.9) by discretizing the equations using a finite volume formulation and central differences in space. To discretize the equations, SPARC uses a structured multiblock discretization of the fluid flow domain, commonly called fluid mesh. The multiblock mesh enables SPARC to divide the flow domain and use MPI to solve the governing equations using a distributed memory architecture. Once the equations have been discretized in space, a Runge-Kutta ordinary differential equation (ODE) solver is used to solve in time the discretized equations in space. If the flow is steady, a similar procedure is used to solve the equations in (3.9). However, the changes of the velocity and pressure in time are considered to be the residuals of the equations, and are reduced to zero at the steady state solution. Additionally to the ODE solver, SPARC uses a multigrid technique and local stepping to accelerate convergence to the steady state.

Once the fluid flow around the airfoil has been solved the aerodynamic characteristics of the airfoil can be obtained. The aerodynamic characteristics obtained from SPARC are the lift, drag and pitch moment coefficients of the airfoil. These parameters are defined as

$$c_l = \frac{Lift}{\frac{1}{2}\rho U_\infty^2 S} \tag{3.11}$$

$$c_d = \frac{Drag}{\frac{1}{2}\rho U_\infty^2 S} \tag{3.12}$$

$$c_m = \frac{Pitch \quad Moment}{\frac{1}{2}\rho U_\infty^2 Sc} \tag{3.13}$$

where U_∞ is a reference velocity, usually the freestream velocity of the flow, S is the wing area, ρ is the density of the fluid and c is the chord of the airfoil. The chord is defined as the distance between the leading and the trailing edge.

3.3 Fluid Mesh Deformation

Each optimization cycle results in a new airfoil. In order to obtain the aerodynamic characteristics of the new airfoil using SPARC, it is necessary to obtain a new fluid mesh that represents the new airfoil. There are two main methods used to obtain the new fluid mesh: 1) the entirely new mesh can be automatically created from the new shape, 2) the original mesh can be deformed to adapt to the new shape. In this thesis, the second method is used because it is computationally more efficient and because automatic mesh generation is mostly used with unstructured meshes. In our case, the CFD code uses a structured mesh. For structured meshes, typically the generation of the fluid mesh is done by the designer instead of automatically, and it usually takes up a large amount of time; therefore, it is the most natural choice to reuse the original grid by using a deformation technique.

Mesh deformation has been investigated in fluid-structure interaction [63] and in shape optimization problems. To deform an initial mesh there are also two main methods: 1) spring-analogy methods, discussed in [41] and 2) algebraic methods. Spring-analogy methods assume that there is a spring connecting each grid point with its neighboring nodes. Initially, the system of spring is in equilibrium. Then, when a boundary deforms, the position of the mesh nodes are recalculated to set the system in equilibrium. These methods are computationally expensive because the size

of the system of equilibrium equations for the springs is proportional to the number of mesh points. This number is usually large - in our case more than 10,000 nodes. Algebraic methods employ algebraic equations to redistribute the deformations of the airfoil over the mesh.

In this thesis, the deformation technique used is a combination of the spring-analogy method and an algebraic method [42]. Since our problem is two-dimensional, and also for the sake of simplicity, the deformation methodology is discussed for a two-dimensional mesh only. The three-dimensional description of the algorithm can be found in [42].

In a two-dimensional structured multiblock mesh, each block is a quadrilateral element. To create the grid inside the block, the four block edges are divided into smaller segments with opposing edges having the same number of divisions. Then, the mesh inside the block is generated by creating lines connecting the edge nodes of opposing edges and using the points created from the intersection of these lines as the grid nodes. Figure 3.4 shows a multiblock structured grid. In the figure the thick solid lines represent block divisions while the thin lines are the lines connecting the points created by dividing the edges. Each point on the grid is characterized by a set of two indexes. In what follows, the indexes will be i and j, and are either local or global. Global indexes refer to the indexes of the nodes that form the blocks. Local indexes refer to the indexes contained inside a block.

Given an initial two-dimensional structured multiblock mesh and a new airfoil shape, the deformation process can be broken down into the following steps:

- Find a set of parameters that relate the nodes of the mesh that belong to the airfoil boundary to the airfoil representation
- Parametrize all mesh nodes for each specific block

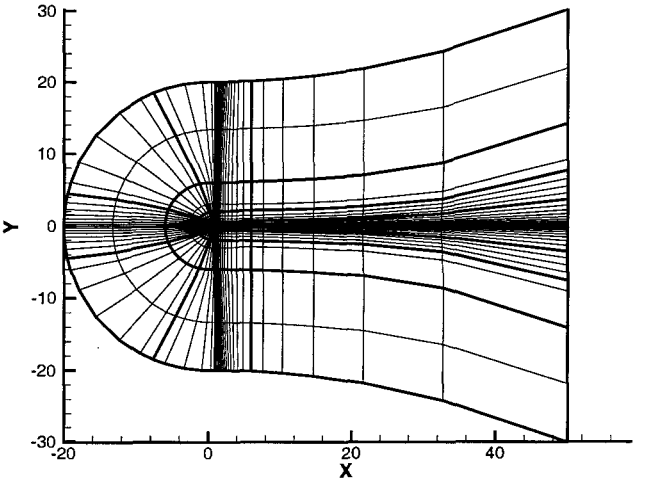

Figure 3.4: Example of a two-dimensional multiblock fluid mesh around an airfoil

- Compute deformations of the mesh points on the airfoil boundary
- Compute deformations of the corner blocks using the spring-analogy method
- Compute the deformations of edges and interior of each specific block independently using the computed corner block deformations and an algebraic model, in this case, transfinite interpolation (TFI)
- Add the computed deformations to the original mesh to obtain the new mesh.

In this list, steps 1 and 2 are only carried out at the beginning of the program.

In step 1, the B-spline parametrization variable, $\overline{u}$, associated with each one of the mesh points on the airfoil boundary is obtained. Since the control point positions of the B-spline defining the initial airfoil are known, and since the boundary nodes of the mesh are placed on this B-spline, the $\overline{u}$ value associated with each node is

obtained by generating points of the B-spline with different values for the $\overline{u}$ variable until a B-spline point is found that matches the coordinates of the mesh point. This process is repeated for each mesh point. Finally, the $\overline{u}$ values for each mesh point are recorded, so that the position of the boundary nodes of the mesh can be obtained from any new B-spline generated by the optimizer. Therefore, the $\overline{u}$ values give the relation between the boundary mesh points and the airfoil shape, so that the fluid mesh can be deformed to adapt to any new airfoil shape possible.

In step 2, the mesh points in each block are parametrized using the normalized arclengths according to the block local indexes i and j. As an example, the normalized arclength parameter of a line with varying i index and fixed j index would be obtained using

$$s_{1,j} = 0 \tag{3.14}$$

$$s_{i,j} = s_{i-1,j} + \sqrt{(x_{i,j} - x_{i-1,j})^2 + (y_{i,j} - y_{i-1,j})^2} \tag{3.15}$$

$$M_{i,j} = \frac{s_{i,j}}{s_{i_{max},j}} \tag{3.16}$$

where $i = 2, \ldots, i_{max}$, $s_{i,j}$ is the arclength of the grid point with indexes (i, j), i_{max} is the total number of grid points in the i direction and $M_{i,j}$ is the normalized arclength of the grid point with indexes (i, j). The normalized arclength parameter in the j direction, $N_{i,j}$, is found using the same procedure.

Using the design variables derived from the optimization process, a new B-spline is defined. Then, in step 3, using the B-spline parametrization and the $\overline{u}$ parameter associated to each grid point at the boundary, the new position of each boundary point is found. Once the grid points on the airfoil have been deformed, each one of the blocks that forms the grid is deformed in order to guarantee that the block size is not greatly reduced. This is done using the segment spring-analogy method described

in detail in [41]. However, here the method is only applied to the block corner nodes instead of all the nodes of the mesh. The corner points of all the blocks in the multiblock mesh are assumed to be connected to their neighboring block corners by springs. The stiffness of the springs are inversely proportional to the original length of the connection edges. For example, the spring connecting corners with global indexes (i, j) and $(i + 1, j)$ is defined as

$$k_{i+\frac{1}{2},j} = \frac{1}{((x_{i+1,j} - x_{i,j})^2 + (y_{i+1,j} - y_{i,j})^2)^{\frac{p}{2}}} \tag{3.17}$$

where the indexes are global and p is a parameter that can be used to increase the stiffness of the springs. The higher the value of p, the stiffer the spring. In our case, p is set to 1. Once the stiffness of the springs have been calculated, the new position of the corner blocks is calculated by solving the static equilibrium equations of the spring system. At node (i, j) the two-dimensional equilibrium equations are

$$\boldsymbol{F}_{i,j} = k_{i+\frac{1}{2},j}(\boldsymbol{\delta}_{i,j} - \boldsymbol{\delta}_{i+1,j}) + k_{i-\frac{1}{2},j}(\boldsymbol{\delta}_{i,j} - \boldsymbol{\delta}_{i-1,j}) + k_{i,j+\frac{1}{2}}(\boldsymbol{\delta}_{i,j} - \boldsymbol{\delta}_{i,j+1}) + k_{i,j-\frac{1}{2}}(\boldsymbol{\delta}_{i,j} - \boldsymbol{\delta}_{i,j-1}) = 0 \tag{3.18}$$

where $\boldsymbol{F}_{i,j}$ is the vector of x and y forces on the (i, j) node and $\boldsymbol{\delta}_{i,j}$ is the vector of deformations of the (i, j) node.

The system of equations generated by the equilibrium equations of each corner block is solved iteratively. In this case, the system is solved using a predictor-corrector iterative process. The predictor step is computed using

$$\bar{\boldsymbol{\delta}}_{i,j} = \boldsymbol{\delta}^n_{i,j} + (\boldsymbol{\delta}^n_{i,j} - \boldsymbol{\delta}^{n-1}_{i,j}) \tag{3.19}$$

where n indicates the iteration number. Note that when $\boldsymbol{\delta}_{i,j}$ approaches the solution,

$\bar{\boldsymbol{\delta}}_{i,j}$ approaches $\boldsymbol{\delta}_{i,j}$ since $\boldsymbol{\delta}^{n}_{i,j} \simeq \boldsymbol{\delta}^{n-1}_{i,j}$. Then, from equation (3.18) the corrector step is

$$\boldsymbol{\delta}^{n+1}_{i,j} = \frac{k_{i+\frac{1}{2},j}\bar{\boldsymbol{\delta}}_{i+1,j} + k_{i-\frac{1}{2},j}\bar{\boldsymbol{\delta}}_{i-1,j} + k_{i,j+\frac{1}{2}}\bar{\boldsymbol{\delta}}_{i,j+1} + k_{i,j-\frac{1}{2}}\bar{\boldsymbol{\delta}}_{i,j-1}}{k_{i+\frac{1}{2},j} + k_{i-\frac{1}{2},j} + k_{i,j+\frac{1}{2}} + k_{i,j-\frac{1}{2}}} \tag{3.20}$$

After several iterations the solution of the system is achieved. Note that by using the spring-analogy method only at the corner of the blocks, instead of in all grid nodes, the system of equations to solve is greatly reduced.

Once the corner block deformations are obtained transfinite interpolation is used to interpolate the corner block deformations into the edges of the block that are not solid boundaries, and also into the interior of the block. The one-dimensional transfinite interpolation in an edge with free i index is

$$\boldsymbol{\Delta E}_{i,1} = (1 - M_{i,j})\boldsymbol{\Delta P}_{1,1} + M_{i,j}\boldsymbol{\Delta P}_{i_{max},1} \tag{3.21}$$

where $i = 1, \ldots, i_{max}$, $\boldsymbol{\Delta E}_{i,j}$ are the deformation of the node (i,j), $\boldsymbol{\Delta P}_{1,1}$ and $\boldsymbol{\Delta P}_{i_{max},1}$ are the apriori computed deformations of the corner points and $M_{i,j}$ is the normalized arclength of node (i,j) computed in step 2. The interpolation in the j direction is similar.

Once the edges have been deformed, transfinite interpolation is used again to obtain the deformation of all the grid points in the interior of each block. The deformation of a node (i,j) inside a block is obtained using

$$\begin{aligned}\boldsymbol{\Delta S}_{i,j} = {} & A_{i,j}\boldsymbol{\Delta E}_{i,1} + B_{i,j}\boldsymbol{\Delta E}_{i,j_{max}} + C_{i,j}\boldsymbol{\Delta E}_{1,j} + D_{i,j}\boldsymbol{\Delta E}_{i_{max},j} - A_{i,j}C_{i,j}\boldsymbol{\Delta P}_{1,1} \\ & - B_{i,j}C_{i,j}\boldsymbol{\Delta P}_{1,j_{max}} - A_{i,j}D_{i,j}\boldsymbol{\Delta P}_{i_{max},1} - B_{i,j}D_{i,j}\boldsymbol{\Delta P}_{i_{max},j_{max}}\end{aligned} \tag{3.22a}$$

where

$$A_{i,j} = 1 - \eta_{i,j} \qquad B_{i,j} = \eta_{i,j} \tag{3.22b}$$

$$C_{i,j} = 1 - \xi_{i,j} \qquad D_{i,j} = \xi_{i,j} \tag{3.22c}$$

$$P_{i,j} = 1 - (M_{i,j_{max}} - M_{i,1})(N_{i_{max},j} - N_{1,j}) \tag{3.22d}$$

$$\eta_{i,j} = \frac{M_{i,1} + N_{1,j}(M_{i,j_{max}} - M_{i,1})}{P_{i,j}} \tag{3.22e}$$

$$\xi_{i,j} = \frac{N_{1,j} + M_{i,1}(N_{i_{max},j} - N_{1,j})}{P_{i,j}} \tag{3.22f}$$

$i = 1, \ldots, i_{max}$, $j = 1, \ldots, j_{max}$, $\boldsymbol{\Delta E}_{i,1}$ is the deformation of the edge with index $j = 1$.

Finally, once all the deformations have been computed, they are added to the original grid and a new grid is obtained.

Using this method to deform the grid, only the movement of the corner block requires the use of information from all blocks. The rest of operations can be performed block by block, and therefore the grid deformation algorithm can be parallelized easily. Furthermore, because the spring-analogy method is only used for the corner block points, the system of equations to be solved is relatively small. Finally, the original mesh is only perturbed, therefore it retains the structure of the initial grid. As an example, figure 3.5 shows a detail of the original and deformed airfoil mesh. It can be seen that the deformed grid retains the structure of the original grid even after a large deformation of the airfoil. The main disadvantage of this method is that some cells of the grid may sometimes overlap, i.e. negative block cells can occur. Negative cells are not allowed by the CFD solver, and will therefore cause the simulation to stop. A case where negative cells may occur is when a block contains an edge at a solid wall. The edge at the solid wall deforms with the wall. It can happen that the wall does

not deform at the corner points of the block, but is greatly deformed in the middle of the block. Then, because only the information used to obtain the deformations of the other edges of the block are from the corner nodes deformation, the other edges are not aware of the deformation of the solid wall and will not expand accordingly. This may result in overlapping edges and negative volume cells.

3.4 Sensitivity Analysis

Sensitivity analysis is concerned with obtaining the gradients or sensitivities of a certain output variable with respect to an input variable. In the case of optimization, sensitivity analysis is used to obtain the derivatives of objective function and constraints with respect to the design variables. Sensitivity analysis for CFD is an active area of research.

In the literature, several methods have been suggested to compute the gradients of the aerodynamic characteristics with respect to the design variables

- Finite-Difference
- Complex-Step Differentiation
- Automatic Differentiation
- Analytical Differentiation, i.e. the adjoint method

In the following, each one of these methods is described in order to show the rationale behind the decision to use finite-differences to compute the gradients in this thesis.

Forward-difference uses a Taylor series expansion of a function around a point, $\mathbf{x}_0$, to obtain an approximation of the gradient. Given the value of a multi-dimensional

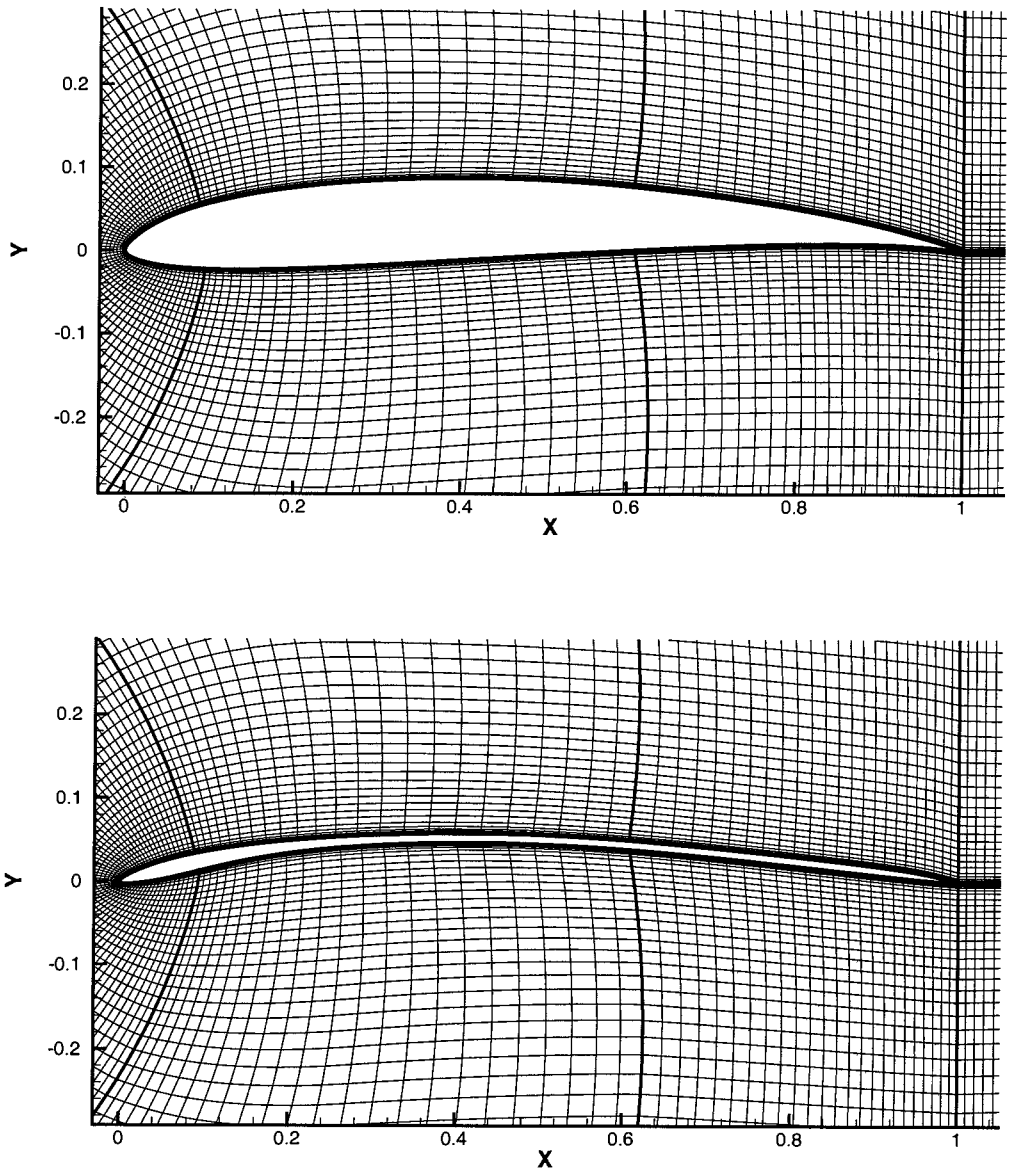

Figure 3.5: Original (above) and deformed (below) mesh around an airfoil

function at $\mathbf{x}_0$, $f(\mathbf{x}_0)$, the value of the function in the neighborhood of $\mathbf{x}_0$ can be expressed using its Taylor series expansion as

$$f(\mathbf{x}_1) = f(\mathbf{x}_0) + \boldsymbol{\delta}^T \boldsymbol{\nabla} f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_0} + \frac{1}{2!}(\boldsymbol{\delta}^2)^T \boldsymbol{\nabla}^2 f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_0} + \ldots + \frac{1}{n!}(\boldsymbol{\delta}^n)^T \boldsymbol{\nabla}^n f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_0} + \mathcal{O}(\|\boldsymbol{\delta}\|^n) \quad (3.23)$$

where $\boldsymbol{\delta} = \mathbf{x}_1 - \mathbf{x}_0$. Taking the first three terms of the Taylor expansion and using a perturbation vector, $\boldsymbol{\delta}_i \in R^{n\times 1}$, with the ith component with a value δ and all other components equal to zero, an expression for the ith component of the gradient of $f(\mathbf{x})$ is obtained

$$\begin{aligned} f(\mathbf{x}_0 + \boldsymbol{\delta}_i) &= f(\mathbf{x}_0) + \boldsymbol{\delta}_i^T \boldsymbol{\nabla} f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_0} + \mathcal{O}(\delta^2) \\ &= f(\mathbf{x}_0) + \delta \frac{\partial f(\mathbf{x})}{\partial x_i}\Big|_{\mathbf{x}=\mathbf{x}_0} + \mathcal{O}(\delta^2) \end{aligned} \quad (3.24)$$

then, rearranging the terms in 3.24, a first-order approximation of the ith component of the gradient can be obtained as

$$\frac{\partial f(\mathbf{x})}{\partial x_i}\Big|_{\mathbf{x}=\mathbf{x}_0} = \frac{f(\mathbf{x}_0 + \boldsymbol{\delta}_i) - f(\mathbf{x}_0)}{\delta} + \mathcal{O}(\delta) \quad (3.25)$$

Equation (3.25) is known as the forward-difference equation to compute the gradients. In a similar fashion to the way the forward-difference equation has been obtained, other higher order approximations of the gradient can also be obtained. Forward difference needs $n + 1$ function evaluations to compute the gradient of a function, with n number of independent variables. Forward difference is easy to implement and is computationally more efficient than complex-step differentiation and automatic differentiation methods [45, 64]. However, forward-difference is also the most inaccurate of all the methods described above. This is because the error is

proportional to the step size. Therefore, to reduce the error, the step size must be reduced. However, if the step size becomes too small, the two terms that are substracted on the numerator become very similar and a numerical error occurs when computing their difference. Therefore, it is necessary to obtain a step size small enough to reduce the error, but not so small that substractive errors occur. This is known as the *step size dilemma.* This problem is also encountered in higher order methods that use the Taylor series to approximate the gradients, e.g. the central-difference method [45, 64]. Furthermore, for higher order methods more function evaluations are necessary to compute the gradients.

Complex-differentiation solves the step size dilemma encountered in the finite-difference methods by using a complex step to compute the gradients. In this case, taking the first five terms of the Taylor expansion and using a perturbation vector, $\boldsymbol{\delta}_i \in C^{n\times 1}$, with the ith component with a complex step value of $i\delta$ and all other components equal to zero an expression for the function at the point $\mathbf{x}_0 + \boldsymbol{\delta}_i$ can be written using a Taylor expansion as

$$f(\mathbf{x}_0 + \boldsymbol{\delta}_i) = f(\mathbf{x}_0) + i\delta \frac{\partial f(\mathbf{x})}{\partial x_i}\Big|_{\mathbf{x}=\mathbf{x}_0} - \frac{1}{2!}\delta^2 \frac{\partial^2 f(\mathbf{x})}{\partial x_i^2}\Big|_{\mathbf{x}=\mathbf{x}_0} - \frac{1}{3!} i\delta^3 \frac{\partial^3 f(\mathbf{x})}{\partial x_i^3}\Big|_{\mathbf{x}=\mathbf{x}_0} + \\ + \frac{1}{4!}\delta^4 \frac{\partial^4 f(\mathbf{x})}{\partial x_i^4}\Big|_{\mathbf{x}=\mathbf{x}_0} + \mathcal{O}(\delta^5) \quad (3.26)$$

and taking the imaginary part of equation (3.26) and rearranging, a value for the ith component of the gradient of the function $f(\mathbf{x})$ at $\mathbf{x}_0$ is obtained

$$\frac{\partial f(\mathbf{x})}{\partial x_i}\Big|_{\mathbf{x}=\mathbf{x}_0} = \frac{Im(f(\mathbf{x}_0 + \boldsymbol{\delta}_i))}{\delta} + \mathcal{O}(\delta^2) \quad (3.27)$$

In this case, there is no substractive term in the equation and therefore, the step size dilemma has disappeared. Furthermore, the approximation is second order instead of

first order as in equation in (3.25). The number of function evaluations necessary to obtain the gradient is still $n + 1$ where n is the number of independent variables of the function. In order to obtain the gradients using complex-step differentiation, the source code of the analysis program has to be changed so that all the real variables become complex variables. Some intrinsic functions such as *max* and *min* must also be redefined. If the designer is really familiar with the analysis code, the required changes to the code can be accomplished in a small amount of time. It is important to notice that because all the variables are complex instead of real, the complexified code requires twice the time of the original code to solve the same problem .

Automatic differentiation (AD) - also known as algorithmic differentiation or computational differentiation - is based on successive application of the chain rule to each operation performed in the analysis computer code [46, 65]. Since the structure of a computer code is basically composed of a successive set of arithmetic operations used to compute the value of a function, successive application of the chain rule to each one of the operations in the code will result in the exact (to machine precision) desired derivatives. For example, imagine the function $f = x_1 * \sin x_2$ is to be computed using a computer code. Defining x_1 and x_2 as the independent variables and f as the dependent variable, the value of the gradient of f with respect to the independent variables can be obtained adding the following equation in the code,

$$\nabla f = \sin(x_2)\nabla x_1 + \cos(x_2)x_1\nabla x_2 \tag{3.28}$$

where

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{pmatrix} \quad \nabla x_1 = \begin{pmatrix} \frac{\partial x_1}{\partial x_1} \\ \frac{\partial x_1}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \nabla x_2 = \begin{pmatrix} \frac{\partial x_2}{\partial x_1} \\ \frac{\partial x_2}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{3.29}$$

Using the same procedure in each line of the code, the gradient of all dependent variables with respect to the independent design variables can be obtained. In this case, the derivatives of intermediate variables with respect to the independent variables propagate throughout the code until, finally, all derivatives of the dependent variables are computed. This type of automatic differentiation is known as the forward mode or tangent linear model. Using this method, the modified code uses approximately $2n$ times more memory and computational time as the original code where n is the number of independent variables. Notice that the computational time and memory of the modified code is independent of the number of function that the gradients need to be obtained for, i.e. dependent variables. For example, to compute the gradient of $f_1(x_1)$ or to compute the gradient of $f_1(x_1)$ and $f_2(x_1)$, automatic differentiation in forward mode will take the same time, 2 times more computational time than the original code. AD in forward mode is found to be similar to complex differentiation, [66].

In contrast to the forward mode of automatic differentiation, there is the reverse mode. In the reverse mode, the derivatives of the final result with respect to intermediate quantities - adjoint quantities - are propagated throughout the code. In order to do so, the flow of the program must be reversed and some variables need to be stored. Using this approach, the computational time is m, the number of dependent design variables, instead of the independent number of design variables, n. However, the memory requirements are larger than using forward mode and depend on the code.

In order to transform a code into a forward or reverse automatic differentiation code, there are several programs that, given a list of the dependent and independent variables, precompile the original code and transform it into a AD code. Some of the codes that can be used to transform FORTRAN source codes to AD codes are: ADIFOR, TAMC, DAFOR, GRESS, Odysee, PADRE2, AD01, ADOL-F, IMAS, Tape-

nade and OPTIMA90. Some of these codes use forward mode, some reverse mode and some, such as ADIFOR, use a hybrid of the two in order to take advantage of the reverse mode efficiency and the lower memory demands of the forward mode. AD is an active area of research and codes to implement this technique are, at this point, not very robust. In the CFD community, ADIFOR, TAF and Tapenade have been used in forward mode to test their ability to obtain sensitivities from simple two-dimensional CFD codes [64, 67] and ADIFOR has also been applied to a three-dimensional CFD code [68, 69]. However, to the knowledge of the author, AD has not yet been used for aerodynamic shape optimization.

Finally, analytical differentiation consists on deriving the analytical expressions for the sensitivities and introducing them to the original analysis code. These methods are the most efficient and accurate, however, they are also the most difficult and time consuming to implement because they require a complete knowledge of the original analysis code and the physics of the flow. There are basically two methods used to compute the sensitivities analytically: direct methods and adjoint methods.

In a CFD solver, the solution of the flow field is obtained when the governing equations of the flow are satisfied, that is when the residuals of the governing equations are equal to zero. In mathematical form

$$\mathcal{R}(x_j, y_k(x_j)) = 0 \text{ for } j = 1, \ldots, n \text{ and } k = 1, \ldots, p \tag{3.30}$$

where $R(x_j, y_k(x_j))$ represents the residuals of the governing equations of the fluid flow, x_j are the independent variables, i.e. the variables that represent the shape of the airfoil and, y_k are the fluid flow variables which depend on x_j.

The sensitivities of a certain function, $f(x_j, y_k(x_j))$, with respect to the indepen-

dent variables can be obtained as

$$\frac{\delta f}{\delta x_j} = \frac{\partial f}{\partial x_j} + \frac{\partial f}{\partial y_k}\frac{\delta y_k}{\delta x_j} \tag{3.31}$$

where indexes j and k imply summation over all values of repeated subscript and $j = 1, \ldots, n$ and $k = 1, \ldots, p$. $\frac{\partial f}{\partial x_j}$ and $\frac{\partial f}{\partial y_k}$ can be obtained from the definition of f. $\frac{\delta y_k}{\delta x_j}$ is the only remaining term to obtain is $\frac{\delta f}{\delta x_j}$. To obtain this last term, equation (3.30) is used. Since at the solution $\mathcal{R}(x_j, y_k(x_j)) = 0$, then at the solution

$$\delta\mathcal{R} = \frac{\partial \mathcal{R}}{\partial x_j}\delta x_j + \frac{\partial \mathcal{R}}{\partial y_k}\delta y_k = 0 \tag{3.32}$$

must also be satisfied. Since $\frac{\partial \mathcal{R}}{\partial x_j}$ and $\frac{\partial \mathcal{R}}{\partial y_k}$ can be obtain, the last equation can be rearranged to obtain

$$\frac{\delta y_k}{\delta x_j} = -\left(\frac{\partial \mathcal{R}}{\partial y_k}\right)^{-1}\frac{\partial \mathcal{R}}{\partial x_j} \tag{3.33}$$

where $j = 1, \ldots, n$, $k = 1, \ldots, p$ and $\frac{\partial \mathcal{R}}{\partial y_k}$ is assumed invertible. Therefore, $\frac{\delta y_k}{\delta x_j}$ can be obtained by solving the system of equations in (3.33) for each independent variable y_k. Then, once the vector $\frac{\delta y_k}{\delta x_j}$ for $j = 1, \ldots, n$ and $k = 1, \ldots, p$ is obtained, it can be used in equation (3.31) to obtain $\frac{\delta f}{\delta x_j}$. The system of equations in equation (3.33) to be solved for each y_k contains the same number of equations as the system of governing equations of the flow. Therefore, obtaining the sensitivities of a function with respect to n independent variables is computationally equivalent to solve n times the flow field. Therefore, the computational expense being similar to finite-differences in forward mode. However, it is necessary to obtain the vector $\frac{\delta y_k}{\delta x_j}$ only once for each shape and it can be used to obtain the gradient of any function with respect to x_j. Therefore, using this method, the necessary time to obtain the gradients of m function is n times the time to obtain the fluid flow solution, where n is the number of independent variables, i.e. design variables.

The method described above is the direct analytical method. In most cases in aerodynamics, there are more independent variables, i.e. the design variables in the optimization problem, than functions for which the gradients are necessary. To eliminate the dependence of the gradient computations on the number of independent variables, the adjoint method was created. Introduced in the CFD community by Jameson [55], the adjoint method differs from the direct method in that equations (3.31) and (3.32) are joined to obtain

$$\delta f = \frac{\partial f}{\partial x_j}\delta x_j + \frac{\partial f}{\partial y_k}\delta y_k + \psi_k^T \left(\frac{\partial \mathcal{R}}{\partial x_j}\delta x_j + \frac{\partial \mathcal{R}}{\partial y_k}\delta y_k \right) \tag{3.34}$$

where ψ_k^T is free to take any value because it is multiplying a zero term. ψ_k^T is known as the adjoint vector. Then, rewriting equation (3.33)

$$\delta f = \left(\frac{\partial f}{\partial x_j} + \psi_k^T \frac{\partial \mathcal{R}}{\partial x_j} \right) \delta x_j + \left(\frac{\partial f}{\partial y_k} + \psi_k^T \frac{\partial \mathcal{R}}{\partial y_k} \right) \delta y_k \tag{3.35}$$

and using an adjoint vector such that

$$\frac{\partial f}{\partial y_k} = \psi_k^T \frac{\partial \mathcal{R}}{\partial y_k} \tag{3.36}$$

equation (3.35) becomes

$$\frac{\delta f}{\delta x_j} = \frac{\partial f}{\partial x_j} + \psi_k^T \frac{\partial \mathcal{R}}{\partial x_j} \tag{3.37}$$

where $\frac{\delta f}{\delta x_j}$ can easily be solved. In this case, to obtain the gradient, the main concern is to obtain the adjoint vector. To obtain the adjoint vector, the system of equations in (3.36) needs to be solved. The system of equations in (3.36) is independent of the independent variables and, therefore, it only needs to be solved once independently of the number of independent variables. Furthermore, the system of equations in (3.36) has the dimensions of the system formed by the governing equations of the flow field.

Therefore, the cost of computing the gradient of f is similar to the computational time to obtain the solution of the flow field and it is independent on the number of independent design variables. On the other hand, an adjoint vector is needed for each function for which the gradients want to be computed. In conclusion, the adjoint method obtains the gradient of m functions respect to n independent design variables in m times the computational time necessary to solve the governing equations of the flow.

In this thesis, for all the problems treated, the number of design variables is larger than the number of functions for which the gradients are necessary. Therefore, the ideal choice would be to use the analytical adjoint method to compute the gradients. However, the development of a CFD code that computes the gradients using the adjoint method requires a thorough knowledge of the CFD code and, additionally, it requires approximately one year to implement for a laminar Navier-Stokes code, [9]. Therefore, the adjoint method was considered to be beyond the scope of this project. The second natural option would be to use AD in reverse mode. However, because of the memory demands, the dimensions of the CFD code being used and the lack of robustness of the AD tools, this method was also considered beyond the scope of this project. Finally, there were four viable options: forward-differences (FD), central-differences, complex-step differentiation and finally, AD using forward mode. Forward-differences is the least accurate of the aforementioned methods, but it is the most efficient and it is easily parallelized. Central differences is more accurate, but it is two times more computationally expensive than FD, and it does not solve the step size dilemma. Complex-step differentiation is accurate, but it is two to three times more computationally expensive than FD [45, 64]. Furthermore, it is necessary to modify the original analysis code. Finally, AD in forward mode gives an exact gradient, but for CFD applications it has been noted that it is approximately as

computationally expensive as complex-step differentiation. It also has higher memory requirements than the other methods, and it requires modifications of the existing analysis source code [64, 67, 68, 69].

In the end, forward-differences was the method chosen to compute the gradient in this thesis. The main reasons for this choice were: faster computation of the gradients compared with complex-differentiation and forward AD, ease of parallelization and, ease of implementation. To reduce the inaccuracy of the gradients, a step size study will be performed to decide the most appropriate step size.

Once the method to compute the gradients has been decided, a code is created to perturb each design variable of the airfoil with a small step. New meshes are created for each perturbed airfoil and the different aerodynamic characteristics are computed using SPARC for the new meshes. Finally, equation (3.25) is used to obtain the gradients of the aerodynamic characteristics with respect to the control points of the B-spline used as design variables. If the angle of attack or the Reynolds number are used as design variables, then the program perturbs such variables in SPARC and also computes the aerodynamic characteristics. For example, the gradient of the lift coefficient with respect to the ith design variable will be obtained as

$$\frac{\partial c_l(\mathbf{x})}{\partial x_i}\bigg|_{\mathbf{x}=\mathbf{x}_0} = \frac{c_l(\mathbf{x}_0 + \boldsymbol{\delta}_i) - c_l(\mathbf{x}_0)}{\delta} \tag{3.38}$$

where in this case $\mathbf{x}_0$ represents the original shape and original mesh and $\mathbf{x}_0 + \boldsymbol{\delta}_i$ represents the new shape when the ith design variable is perturbed. Since for each perturbed shape all aerodynamic characteristics are obtained, the gradient of lift, drag and pitch moment with respect to the n design variables are obtained after $n+1$ analysis runs. Therefore, in this case, the computational cost is only proportional to the number of design variables.

It must be noted that the analysis runs needed to compute the gradient are independent of each other and, therefore, they can be solved in parallel. In order to take advantage of this property and to further reduce the computational time necessary to compute the gradients, a scheduling program for the parallel cluster is used during the gradient calculations, namely Portable Batch System (PBS) [70]. Then, instead of solving the analysis runs one by one, all the analysis runs are sent to PBS at once and PBS allocates the necessary number of processors for each different analysis run until the parallel cluster does not have any more processors free. If all analysis runs are not able to be executed at the same time, they are saved in the PBS queue until more processors are free and then, the remaining analysis runs are executed in those new processors. By using PBS, the processors can be dynamically allocated and deallocated thereby taking full advantage of the cluster capabilities when computing the gradients. In this thesis, each analysis run uses 3 processors and the analysis program is executed in a 16 processors cluster. Therefore, 5 analysis runs can be executed at the same time, reducing by 5 the amount of time necessary to compute the gradients, even though the CPU time is not reduced.

In summary, the gradients are computed following this procedure

Step 1 Given an original shape and grid, set $k = 0$

Step 2 Submit analysis to PBS and set $k = k + 1$

Step 3 If $k \geq 1$, add a step, δ, to the k design variable

Step 4 Obtain the new airfoil shape

Step 5 Deform the original mesh according to the new airfoil shape

Step 6 Submit analysis job to PBS, set $k = k + 1$

Step 7 If all the geometric design variables have been perturbed continue, else go to step 3

Step 8 Wait until all analysis submitted to PBS are done

Step 9 Use c_l, c_d and c_m from all the different analysis and equation (3.25) to obtain gradients

3.5 Implementation

The different algorithms described above have been assembled to create a code for aerodynamic shape optimization of airfoils at any Reynolds number and angle of attack. Figure 3.6 shows a schematic of the implementation. The code can be simplified as follows:

Step 1 Create an airfoil using the B-spline generator of section 3.1

Step 2 Create a structured multiblock grid to solve the airfoil generated in step 1

Step 3 Compute objective function and constraint of the optimization problem using the fluid flow solver in section 3.2

Step 4 Compute gradient of objective function and constraints using forward-differences as described in 3.4

Step 5 Solve the optimization problem using DOT

Step 6 If the optimization problem has converged STOP; if the optimization did not converge yet CONTINUE

Step 7 Using the new design variables create new airfoil and mesh using sections 3.1 and 3.3 and go to step 3

Note that once the aerodynamic characteristics are obtained a subroutine must be created to used this information to obtain the desired objective function and constraints.

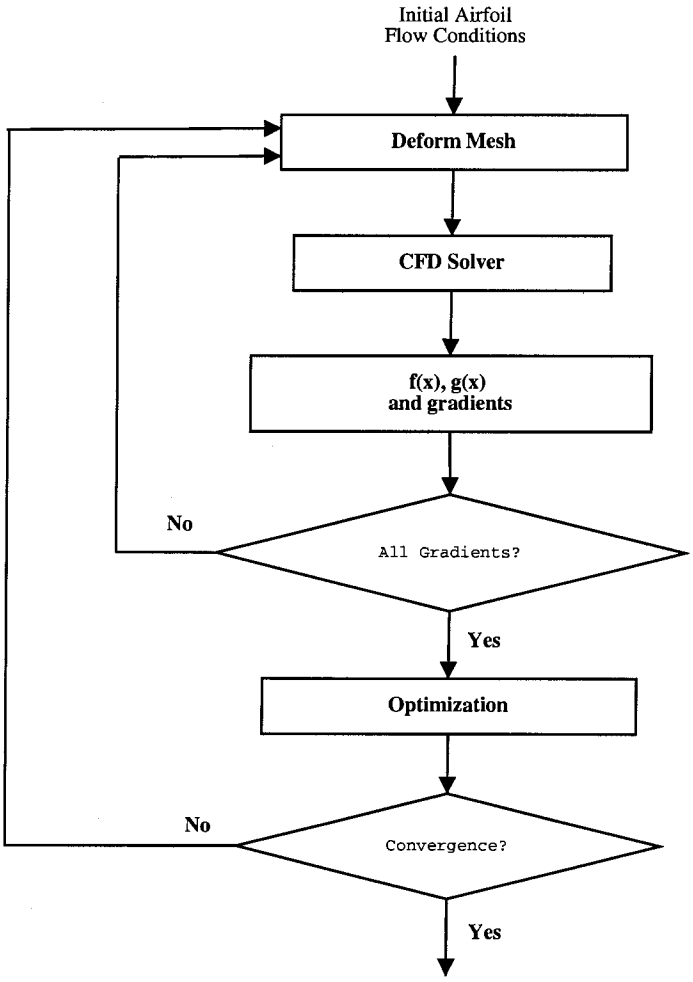

Figure 3.6: Flow chart of the aerodynamic shape optimization design tool

Chapter 4

Applications

Once the computational design tool described in the preceding chapter has been implemented in Fortran90, the design tool can then be used to aid in the design of airfoils for aircraft at any Reynolds number. The only requirement is an initial airfoil shape and a fluid mesh that yields accurate results at the Reynolds numbers being studied. In this chapter, the computational tool is used to solve several optimal design problems for application to the design of unmanned aerial vehicles (UAV) [1, 5]. Section 4.1 focuses on the testing and validation of a fluid mesh. This fluid mesh will then be used for all the following study cases because the flow characteristics are similar for all the cases under consideration. Once a grid has been selected, the optimization process begin. However, because forward-differences are being used to compute the gradient, a parametric study of different step sizes is performed in section 4.2 in order to obtain the most accurate gradient possible. Furthermore, this data could prove useful in the future to validate more advanced methods used to compute the gradients, such as automatic differentiation or the adjoint method. Finally, section 4.3 contains the first test case under study. In this case, a lift-constrained minimum drag airfoil is obtained. This initial example is used to validate the design tool's

ability to obtain minimum drag airfoils, and to study the performance of the different optimization algorithms. Finally, section 4.4 takes advantage of the full capability of the design tool to obtain a set of optimal shapes for several different stages of flight of an airfoil morphing UAV.

4.1 Grid Study

Using the grid in [62] to successfully predict the lift and the drag of a NACA0012 at $Re = 3 \times 10^6$ as a base line, a fluid mesh was generated to predict the lift and drag of an airfoil at the Reynolds numbers of interest in this thesis. In this case, the Reynolds numbers of interest are of the order of $Re = 5 \times 10^5$, smaller than the ones used in [62]. Since experimental data exists for an Eppler 64 airfoil at $Re = 2 \times 10^5$ [59], the grid studies are performed for the Eppler 64 at the aforementioned Reynolds number so that numerical and experimental results can be compared. The grid obtained from this grid study will be used throughout the thesis and it is assumed to be valid at $Re = 5 \times 10^5$ as well as for other airfoils in the same flow regime, because the flow characteristics are similar.

The generated grid was refined 5 times in all directions and each refined grid was used to solve the flow field. Tables 4.1 and 4.2 show the total lift, c_l, total drag, c_d, friction drag, c_{df}, and pressure drag, c_{dp}, values for the Eppler 64 airfoil computed using the different grids at an angle of attack of 0 and 4 degrees respectively. In the tables, grid 1 represents the coarsest grid and grid 5 represents the most refined grid.

Comparing the lift and drag coefficient between the different grids gives an estimate of the grid resolution necessary to obtain a grid independent solution. Looking at the evolution of the lift coefficient in table 4.1, the lift coefficient is similar for grid levels 2 to 5 with oscillations of less than 5% around the value of 0.5. The lift

Table 4.1: Lift and drag values for different grid refinements at $\alpha = 0°$, $Re_c = 2 \times 10^5$

Grid	c_l	c_d	c_{df}	c_{dp}
1	0.39400	0.04200	0.00446	0.03754
2	0.48946	0.02760	0.01417	0.01343
3	0.51137	0.01755	0.01207	0.00549
4	0.50472	0.00957	0.00596	0.00360
5	0.47964	0.00780	0.00490	0.00290
Experiments, [59]	0.50	0.0125	-	

Table 4.2: Lift and drag values for different grid refinements at $\alpha = 4°$, $Re_c = 2 \times 10^5$

Grid	c_l	c_d	c_{df}	c_{dp}
1	0.76468	0.05826	0.00434	0.05392
2	0.89256	0.03978	0.01335	0.02644
3	0.92985	0.02408	0.01181	0.01227
4	0.90990	0.01417	0.00558	0.00859
5	div	div	div	div
Experiments, [59]	0.925	0.01450	-	

coefficient of 0.5 coincides with the experimental value for the lift coefficient and the lift coefficient at a 4 degrees angle of attack follows a similar pattern. Table 4.2 shows how grids 2, 3 and 4 have a lift coefficient around 0.91, a value close to the experimental value reported to be 0.925.

The total drag coefficient evolution with respect to the grid refinement in tables 4.1 and 4.2 shows larger changes compared to the lift coefficient changes with different grids. To study the drag, instead of focusing on the total drag, the pressure and the friction drag were studied independently, because SPARC computes the total drag using

$$c_d = c_{df} + c_{dp} \tag{4.1}$$

Looking at the evolution of the pressure drag, it can be observed that it decreases steadily from grid 1 to grid 5 with the largest changes occurring between the first three grids. After grid 3 the pressure drag still decreases, but by small amounts when compared with the initial changes. The friction drag evolution follows a similar pattern from grid 2 to grid 5; however, the changes between grids are more pronounced. The large change in the friction drag from grid 3 to grid 4 is produced by changes in the behavior of the boundary layer as the grid is refined, and also by changes in the characteristics of the laminar-to-turbulent transition as observed in the turbulent to laminar eddy viscosity ratio plot in figure 4.1. At the low Reynolds number under consideration, the laminar-to-turbulent transition happens around the middle of the airfoil [59]. This transition considerably affects the behavior of the boundary layer, and the lift and drag as discussed in [71, 72]. Looking at table 4.1, the large change in the friction drag value does not appear again when the grid is refined further, so it is assumed that the friction drag is also close to its converged value. For an angle of attack of 4 degrees, grid 5 is numerically unstable; therefore, results are not reported. The evolution of the total drag in tables 4.1 and 4.2 shows a strong dependence on the friction drag, which is difficult to predict. However, at zero angle of attack, grids 4 and 5 yield similar results with less than a 20% change in total drag. Furthermore, the evolution of pressure and friction drag at a 4 degree angle of attack is similar. Therefore, it is assumed that grid 4 converged to a satisfactory result in predicting drag.

Comparing the total drag with experimental data, it can be observed that, at a zero degree angle of attack, the total drag is under predicted by 24% using grid 4, while at a four degree angle of attack the total drag is under predicted by 4% using the same grid. Both these errors are considered to be small because of the difficulty of predicting friction drag. The discrepancy between experimental and numerical results

is probably due to an under resolved boundary layer and an inaccurate prediction of the laminar-to-turbulent transition. The results could be improved by introducing a tripping source term to the Spallart-Allmaras model in SPARC in order to accurately predict the correct laminar-to-turbulent transition, as discussed in [61].

In conclusion, since grid 4 yields good results for lift and drag predictions, and since a coarse grid is preferable due to its lower computational cost, grid 4 is chosen to be the base grid for optimization. A study of the computational domain is not undertaken here because the domain size used is the same as the computational domain size used in [62] where it was already proven that the grid was domain independent. Grid 4 is shown in figure 4.2 and the detail of the grid around the airfoil and around the leading edge of the airfoil are shown in figures 4.3 and 4.4. Grid 4 has 36 blocks, 24,396 nodes, 157 nodes around the airfoil and the first node from the boundary of the airfoil is at a distance of 4×10^{-5}, which results in a y^+ value of 0.5.

4.2 Study of the Step Size Used to Compute the Gradient

Once a grid is obtained for optimization, the optimization algorithm can already receive information of objective function and constraints. However, as discussed earlier, the optimization algorithm also needs information on the gradients of objective function and constraints. Such gradients are computed using forward differentiation. The value of the gradient of a function with respect to its ith variable is obtained using forward differentiation as

$$\frac{\partial f(\mathbf{x})}{\partial x_i}\Big|_{\mathbf{x}=\mathbf{x}_0} = \frac{f(\mathbf{x}_0 + \boldsymbol{\delta}_i) - f(\mathbf{x}_0)}{\delta} + \mathcal{O}(\delta) \tag{4.2}$$

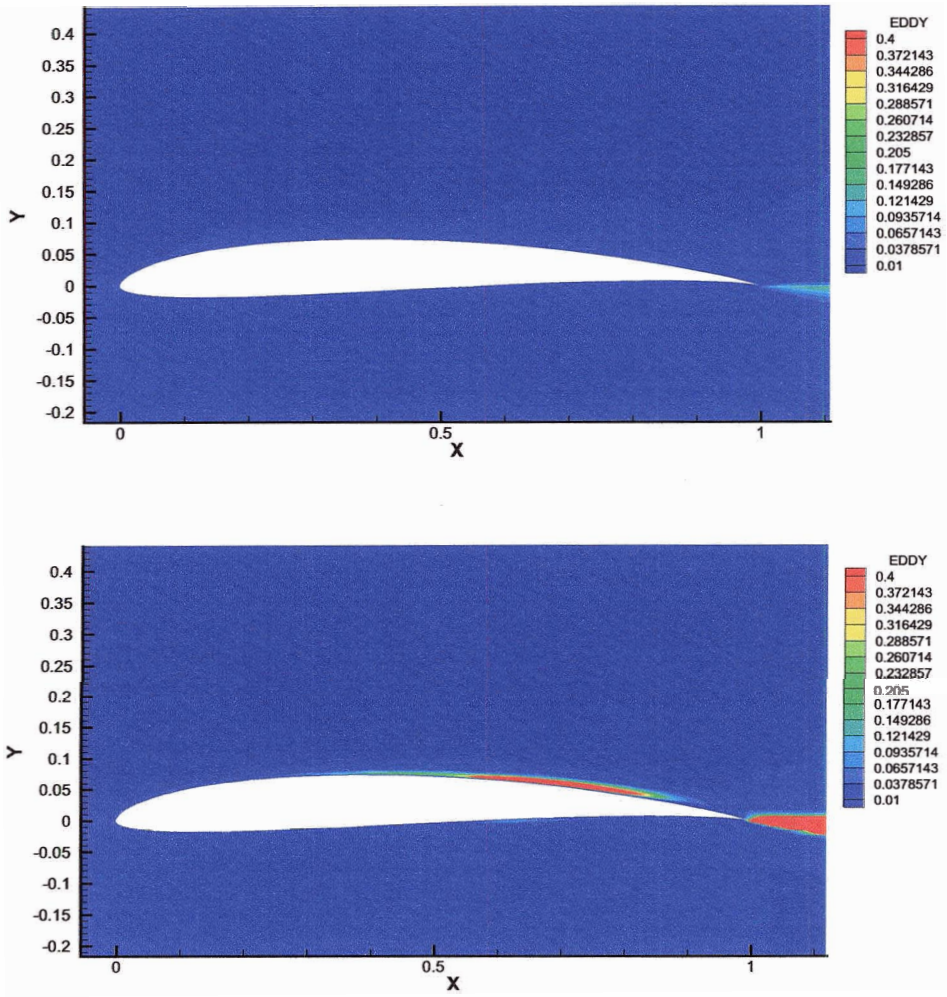

Figure 4.1: Turbulent to laminar eddy viscosity ratio contour plot close to the Eppler 64 airfoil at a 4 degree angle of attack for grids 3 (above) and grid 4 (below)

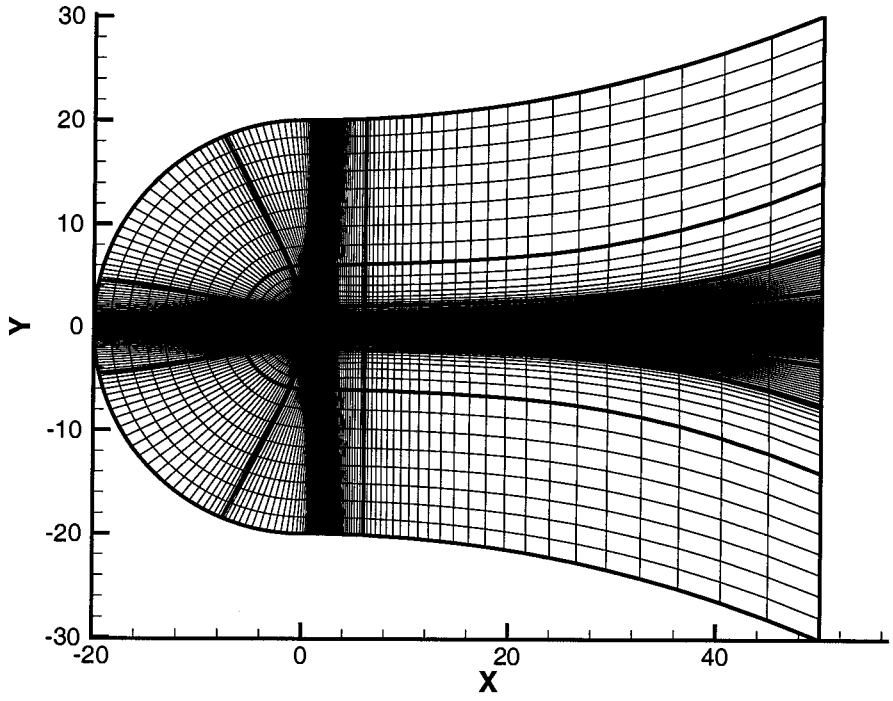

Figure 4.2: Grid 4 around the Eppler 64 airfoil

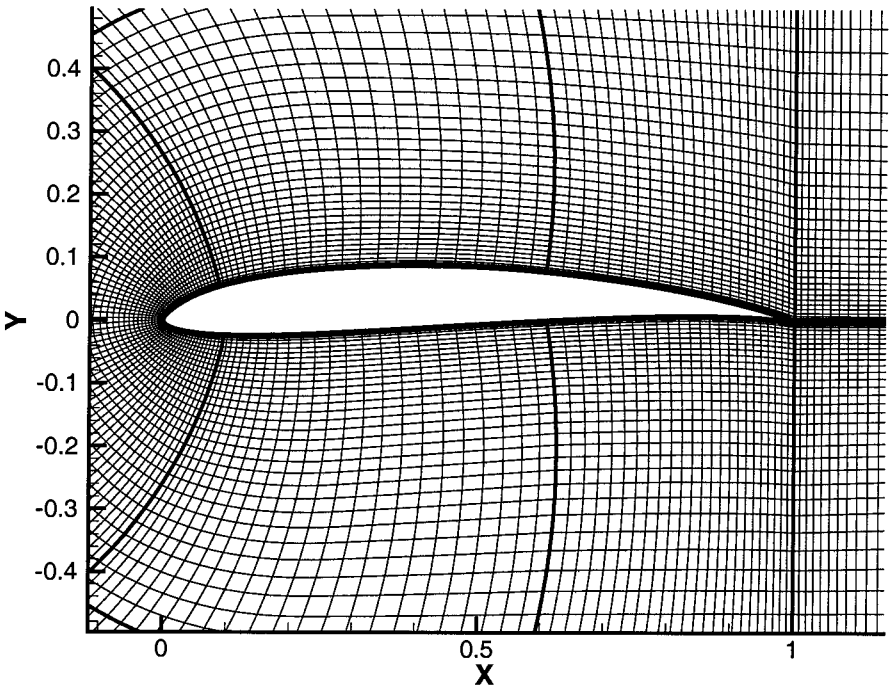

Figure 4.3: Detail of grid 4 around the Eppler 64 airfoil

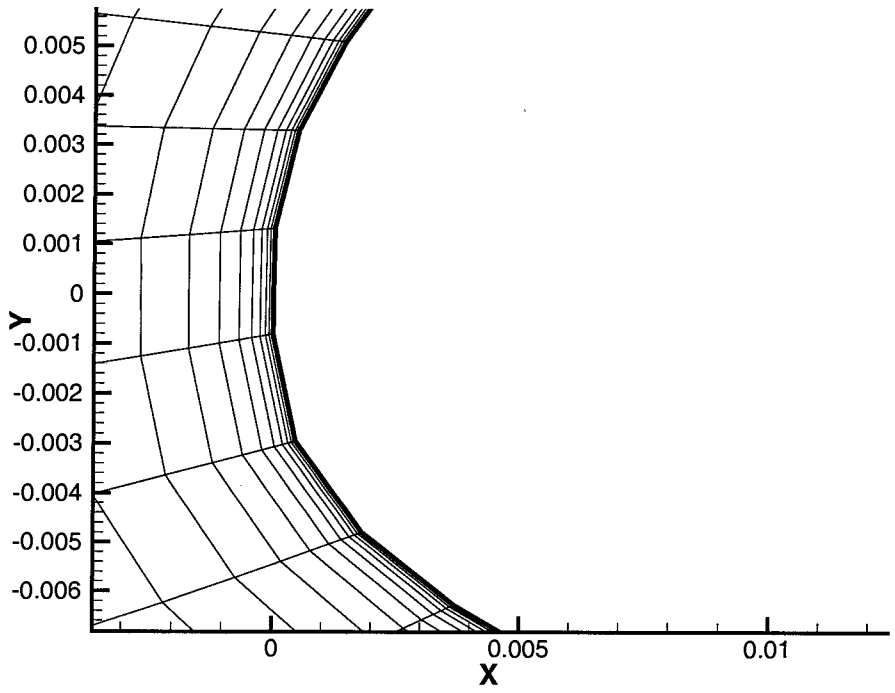

Figure 4.4: Detail of the grid around the leading edge of the Eppler 64 airfoil

where $\boldsymbol{\delta}_i$ is the perturbation vector for the i variable and δ the step size. This method to compute the gradients is subjected to the step size dilemma.

To obtain the optimal step size for the gradient calculations, the lift and drag gradients for each design variable in the optimization problem are plotted versus step sizes from 10^{-2} to 10^{-7} in figures 4.5 and 4.6. The figures show clearly the step size dilemma. Up to a step size of 10^{-5} or 10^{-6}, depending on the variable, the value of the gradients seem to converge to a value for the gradient; then as the step size is reduced further, the value of the gradients start to change again. This is due to numerical errors in the subtraction and the fact that an iterative solver is used to solve the fluid flow. The converged solution of the CFD solver is only accurate to a certain value, in this case 10^{-9}. From figures 4.5 and 4.6 it appears that the most appropriate step size is 10^{-4}, 10^{-5} or 10^{-6} depending on the variable. In this case, a step size of 10^{-5} is chosen for all the variables as the step size for gradient calculations.

4.3 Drag Minimization

An airplane in level flight must satisfy the following conditions,

$$Weight = Lift \tag{4.3a}$$

$$Drag = Thrust \tag{4.3b}$$

Therefore, it is desirable to obtain an airfoil that is able to satisfy a certain lift requirement and, at the same time, has a minimum drag. From equation (4.3b), minimum drag will result in a minimum thrust requirement, which in turn will result in a more efficient airplane.

To test the performance of the design tool, the program is used to solve a drag

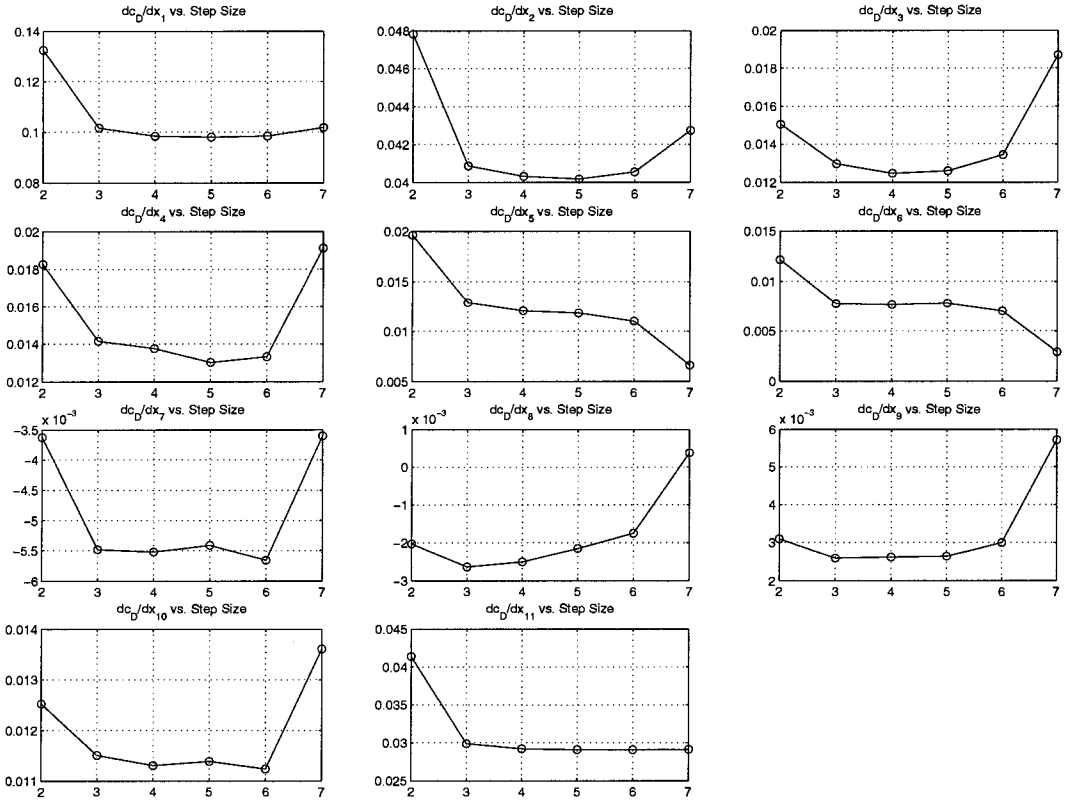

Figure 4.5: Value of the drag coefficient gradient with respect to the decimal logarithm of the step size used to compute the gradient using forward differences

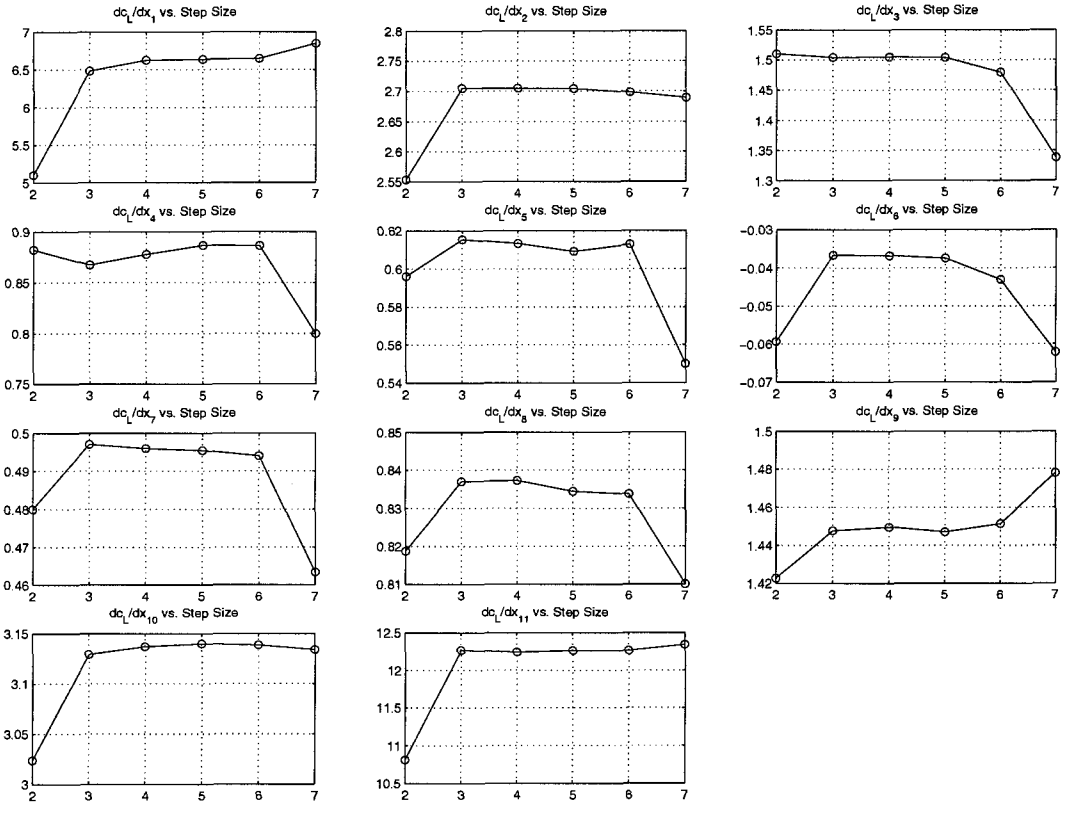

Figure 4.6: Value of the lift coefficient gradient with respect to the decimal logarithm of the step size used to compute the gradient using forward differences

coefficient minimization problem subject to a minimum lift coefficient requirement. The design problem is to obtain an airfoil with minimum drag and a minimum lift coefficient of 0.8, at a $Re = 500,000$, and with a 2 degree angle of attack. Thickness constraints are imposed on the geometry of the airfoil and bounds are also imposed on the design variables. In particular, geometrical constraints are imposed to obtain a minimum thickness of 1% of the chord as described in table 4.3. The bounds of the design variables are presented in table 4.4, where the design variables are the y coordinate of the control points of the B-spline that represents the airfoil, and the numbering corresponds to the numbering in figure 4.7 with x_{LE} representing the distance between leading edge points. Notice that the variables x_5 and x_7 have different bounds than the other variables. This is due to the method used to deform the grid. If the same bounds are used, the mesh deformation algorithm creates a fluid mesh with negative cells, and the analysis program is unable to solve the flow around the airfoil. x_{LE} also has different bounds. This is because of the different nature of the variable in that x_{LE} represents the distance between the two points at the leading edge and, therefore, must always be positive.

Table 4.3: Geometric constraints of the design problem

Constraint	Value
1	$x_1 - x_{11} \geq 0.01$
2	$x_2 - x_{10} \geq 0.01$
3	$x_3 - x_9 \geq 0.01$
4	$x_4 - x_8 \geq 0.01$
5	$x_5 - x_7 \geq 0.01$

To solve the design problem, an Eppler 66 airfoil is used as the initial airfoil for the optimization procedure. This airfoil was chosen because it is one of the airfoils recommended for the design of low Reynolds number aircraft [59]. Previous to the

Table 4.4: Lower and Upper bounds of the design variables

	x_1	x_2	x_3	x_4	x_5	x_{LE}	x_7	x_8	x_9	x_{10}	x_{11}
Lower Bound	-0.2	-0.2	-0.2	-0.2	0.0	0.005	-0.2	-0.2	-0.2	-0.2	-0.2
Upper Bound	0.2	0.2	0.2	0.2	0.2	0.05	0.0	0.2	0.2	0.2	0.2

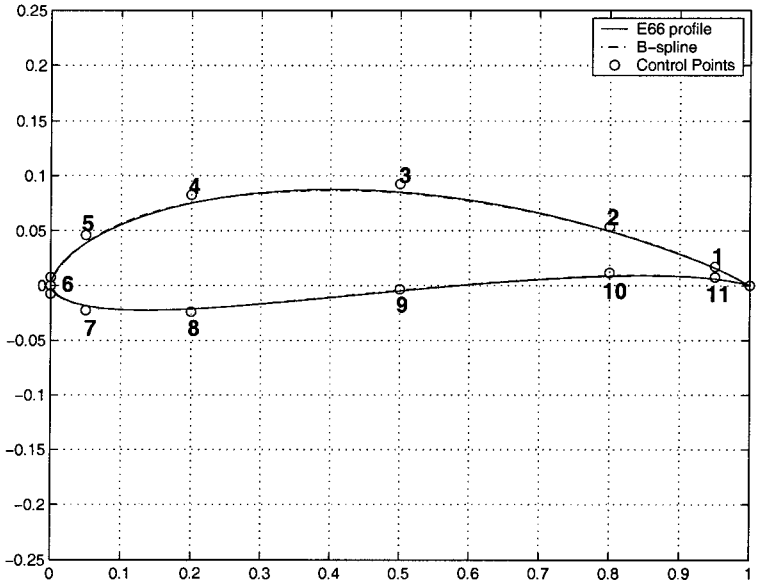

Figure 4.7: B-spline representation of the Eppler 66 airfoil used at the initial airfoil for the optimization algorithm

optimization, the lift and drag coefficients for the intial airfoil were computed to be 0.864 and 1.009×10^{-2} respectively. The lift and the drag were obtained using SPARC with 3 processors on a 22 processor Linux cluster. The lift and drag coefficients were obtained after approximately 45 minutes of CPU time. During the optimization, the lift and drag coefficients are obtained to compute the objective function and aerodynamic constraint using the same grid and number of processors. However, the flow field is restarted from the last flow field solution, thereby enabling a reduction in the number of iterations prior to convergence. This results in a reduction of 15

minutes of CPU time. It can be observed from this discussion that most of the time during the optimization will be spent in the evaluation of the objective function and constraints.

Starting with the Eppler 66 airfoil as the initial design, the design problem was solved using the three optimization algorithms described in chapter 2. The three optimization algorithms converged to a solution with similar aerodynamic characteristics as shown in table 4.5, with a discrepancy of less than 1%. The solution with the smallest drag is obtained using the modified method of feasible directions (MMFD), followed by the sequential linear programming (SLP) algorithm, and finally, the sequential quadratic programming (SQP) algorithm. All three methods obtain a similar airfoil shape as illustrated in figure 4.8 and, in table 4.7, where the value of all design variables at the optimal solution obtained with the different optimization algorithms are shown. The optimal solution obtained with all three methods satisfies all aerodynamic and geometric constraints as shown in tables 4.5 and 4.6. From table 4.5, it can be observed that the lift constraint is active, i.e. the lift coefficient is 0.8. This was expected, since it is well known in the aerodynamic community that lift and drag are opposing goals. From table 4.6, it can be observed that all geometric constraints are active or near active. Therefore, it can be concluded that in order to obtain an airfoil with minimum drag, the airfoil needs to be extremely thin. Note that, from a structural point of view, this presents a challange and could prove to be problematic.

The three methods reduce the drag by almost 20% with respect to the original airfoil. This reduction in the total drag is achieved by a reduction of 7% in the friction drag and a reduction of approximately 32% in the pressure drag with respect to the original airfoil. To analyze how the optimization algorithms achieve these reductions, the pressure and friction coefficient are used. The pressure and friction coefficients

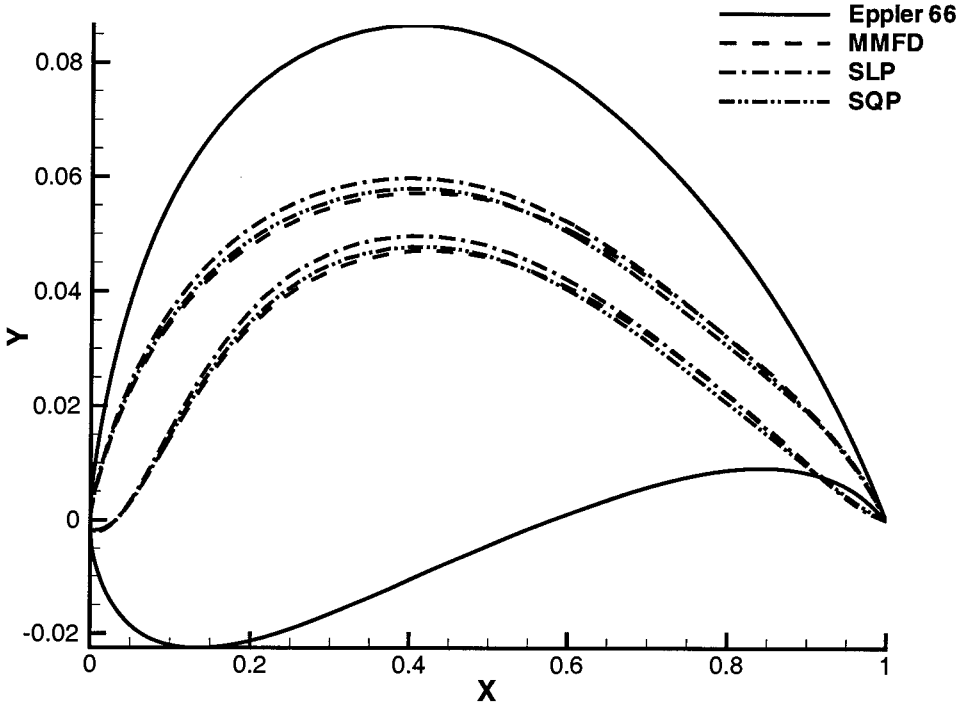

Figure 4.8: Initial and optimal airfoil shapes for MMFD, SLP and SQP

Table 4.5: Aerodynamic characteristics of the initial and optimal solution at $Re = 500,000$ and $\alpha = 2$

	Eppler 66	MMFD	SLP	SQP
c_l	0.864278	0.799998	0.799934	0.800065
$c_d \times 10^2$	1.008671	0.811443	0.812351	0.812701
$c_{df} \times 10^2$	0.509051	0.471875	0.474183	0.470825
$c_{dp} \times 10^2$	0.499620	0.339568	0.338168	0.341876
L/D	85.68	98.59	98.47	98.45

Table 4.6: Value of the geometric constraint at the optimal solution

Constraint	1	2	3	4	5
Eppler 66	2.5000E-04	-3.2000E-02	-8.6000E-02	-9.6500E-02	-5.8500E-02
MMFD	9.3132E-10	1.8626E-09	1.8626E-09	-1.8626E-09	-1.7426E-02
SLP	2.3283E-10	2.2408E-06	-2.3115E-06	-6.6701E-06	-1.8815E-02
SQP	-1.1642E-09	0.0000	1.8627E-09	-1.1317E-04	-1.7713E-02

are defined respectively as,

$$c_p = \frac{p - p_\infty}{\frac{1}{2}\rho U_\infty^2} \tag{4.4}$$

$$c_f = \frac{\tau_w}{\frac{1}{2}\rho U_\infty^2} \tag{4.5}$$

where τ_w is the shear stress at the surface of the airfoil. Pressure and friction coefficients over the initial and the final airfoil surfaces of the three optimization algorithms are plotted in figures 4.9 and 4.11. From the figures, it can be observed that the three optimal solutions have almost the same pressure and friction coefficient distributions over the airfoil surface.

Figure 4.10 shows the pressure coefficient distribution in the flow field around both the initial airfoil and the airfoil obtained using the SQP method. From the figure, it can be observed how the maximum pressure at the leading edge is reduced.

Table 4.7: Value of the design variables at the optimal solution

	x_1	x_2	x_3	x_4
Eppler 66	1.72500E-02	5.35000E-02	9.25000E-02	8.25000E-02
MMFD	1.32334E-02	3.33933E-02	6.21168E-02	5.30299E-02
SLP	1.27780E-02	3.29492E-02	6.42786E-02	5.68418E-02
SQP	1.37749E-02	3.06921E-02	6.29416E-02	5.43026E-02
	x_5	x_{LE}	x_7	x_8
Eppler 66	4.60000E-02	1.50000E-02	-2.25000E-02	-2.40000E-02
MMFD	2.74260E-02	5.30234E-03	-4.52642E-10	4.30299E-02
SLP	2.88151E-02	6.00970E-03	0.00000	4.68352E-02
SQP	2.77129E-02	6.26530E-03	-2.84868E-13	4.41894E-02
	x_9	x_{10}	x_{11}	
Eppler 66	-3.50000E-03	1.15000E-02	7.45000E-03	
MMFD	5.21168E-02	2.33934E-02	3.23338E-03	
SLP	5.42763E-02	2.29514E-02	2.77802E-03	
SQP	5.29416E-02	2.06921E-02	3.77485E-03	

The reduction of the maximum pressure, together with a sharper leading edge, are the responsible for the reduction in pressure drag. Since the leading edge is sharper, the projection of the pressure in the x-direction which contributes to the pressure drag is reduced. In addition, the pressure at the leading edge that contributes to the pressure drag is further reduced by the reduction of the maximum pressure.

From figure 4.11, it can be seem that the friction drag is reduced in both upper (upper curve) and lower surfaces. The reduction in friction drag is due to: a smaller wetted surface of the airfoil, and a reduction in the pressure gradient in the x-direction. The new airfoil is thiner and therefore, has less area in contact with the fluid flow. The reduction of the pressure gradient in the x-direction observed in figure 4.9 produces a reduction of the adverse pressure which makes the laminar boundary layer more stable. A laminar boundary layer produces less friction drag.

The optimization problem outlined above was solved using three different opti-

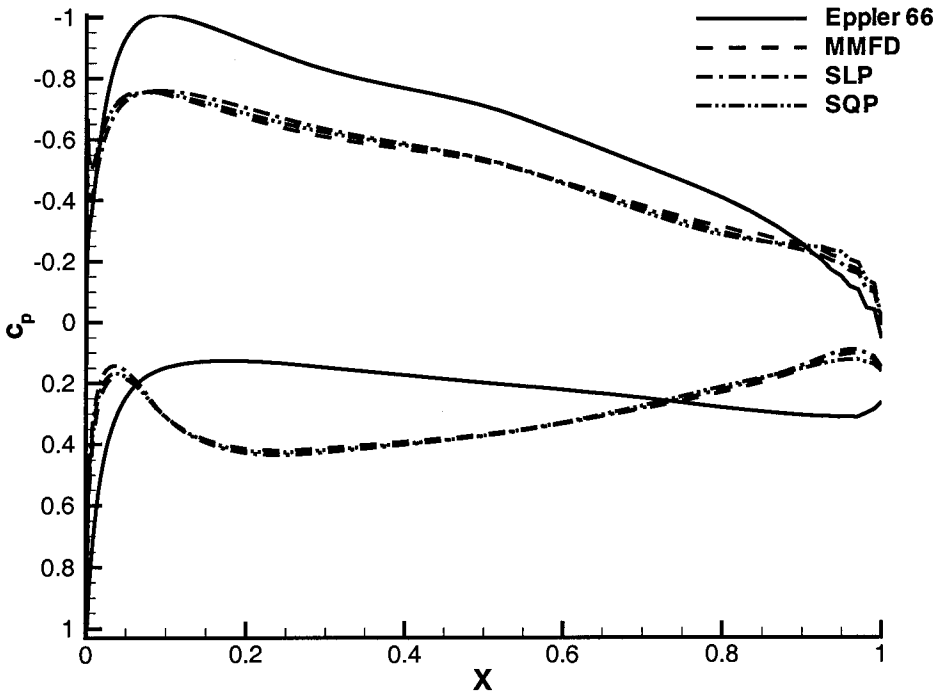

Figure 4.9: Pressure coefficient distribution at $Re = 500,000$ and $\alpha = 2$ over the surface of the Eppler 66 and optimal airfoils using MMFD, SLP and SQP

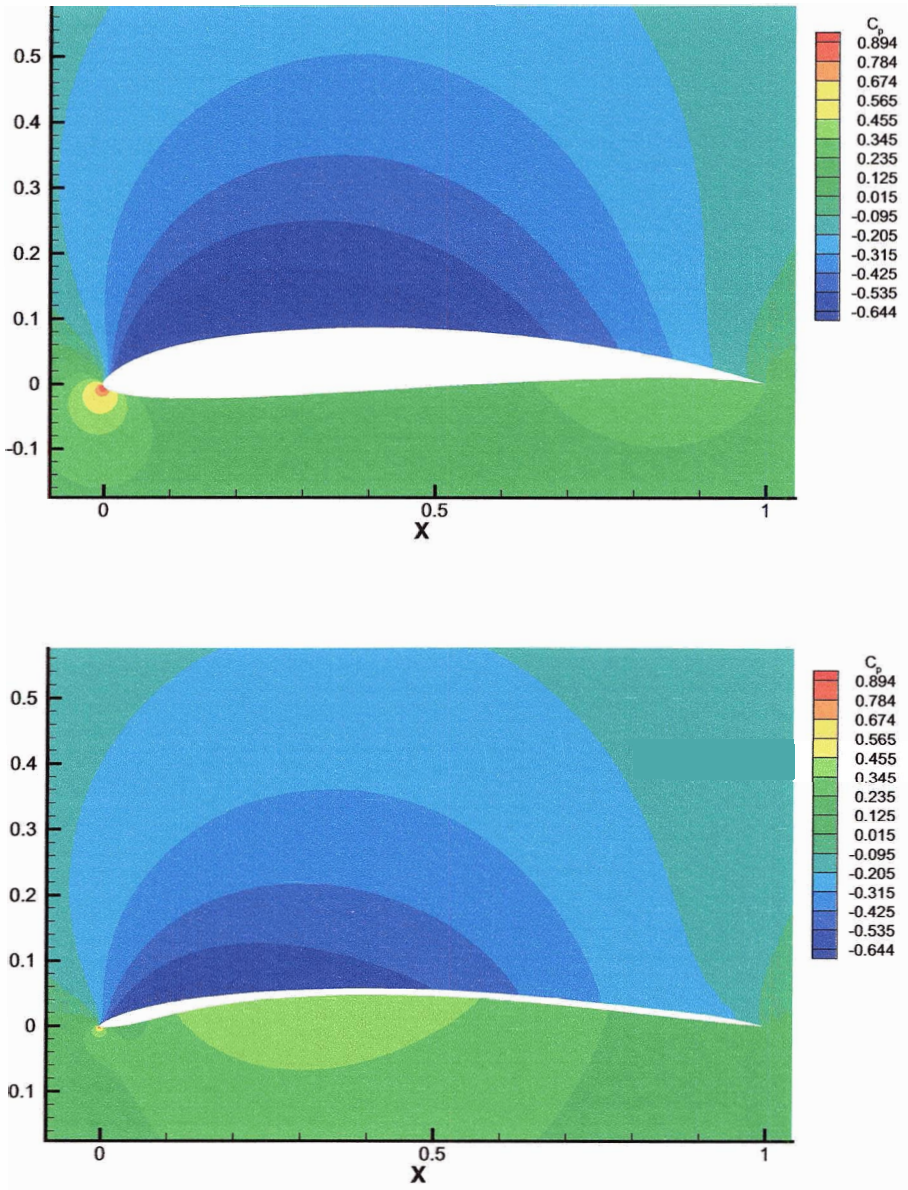

Figure 4.10: Contour plot of the pressure coefficient distribution at $Re = 500,000$ and $\alpha = 2$ over the Eppler 66 airfoil (above) and the optimal SQP airfoil (below)

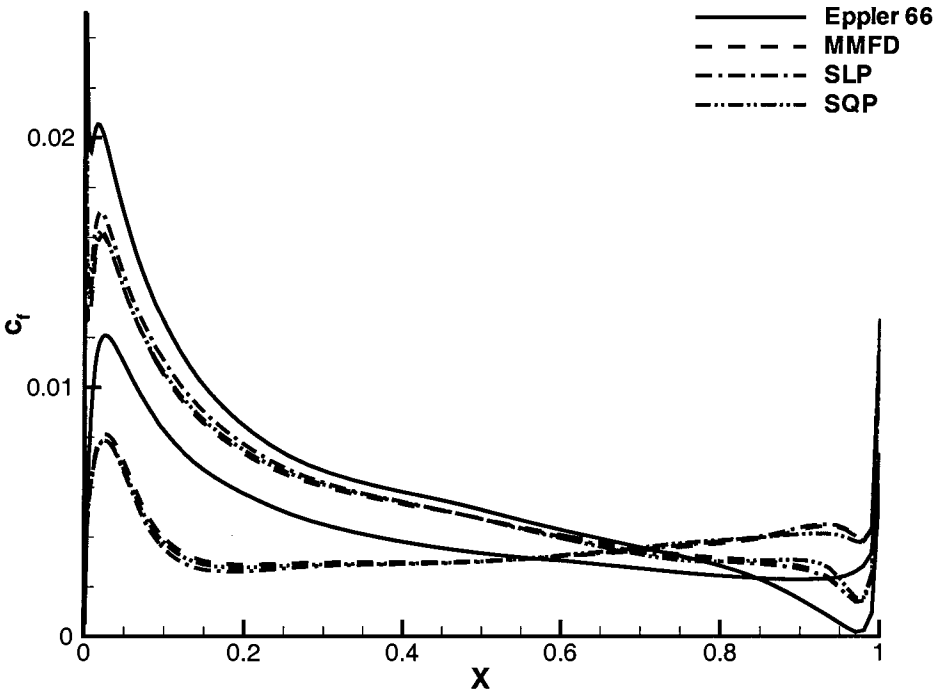

Figure 4.11: Friction coefficient distribution at $Re = 500,000$ and $\alpha = 2$ over the surface of the Eppler 66 and the optimal airfoils using MMFD, SLP and SQP

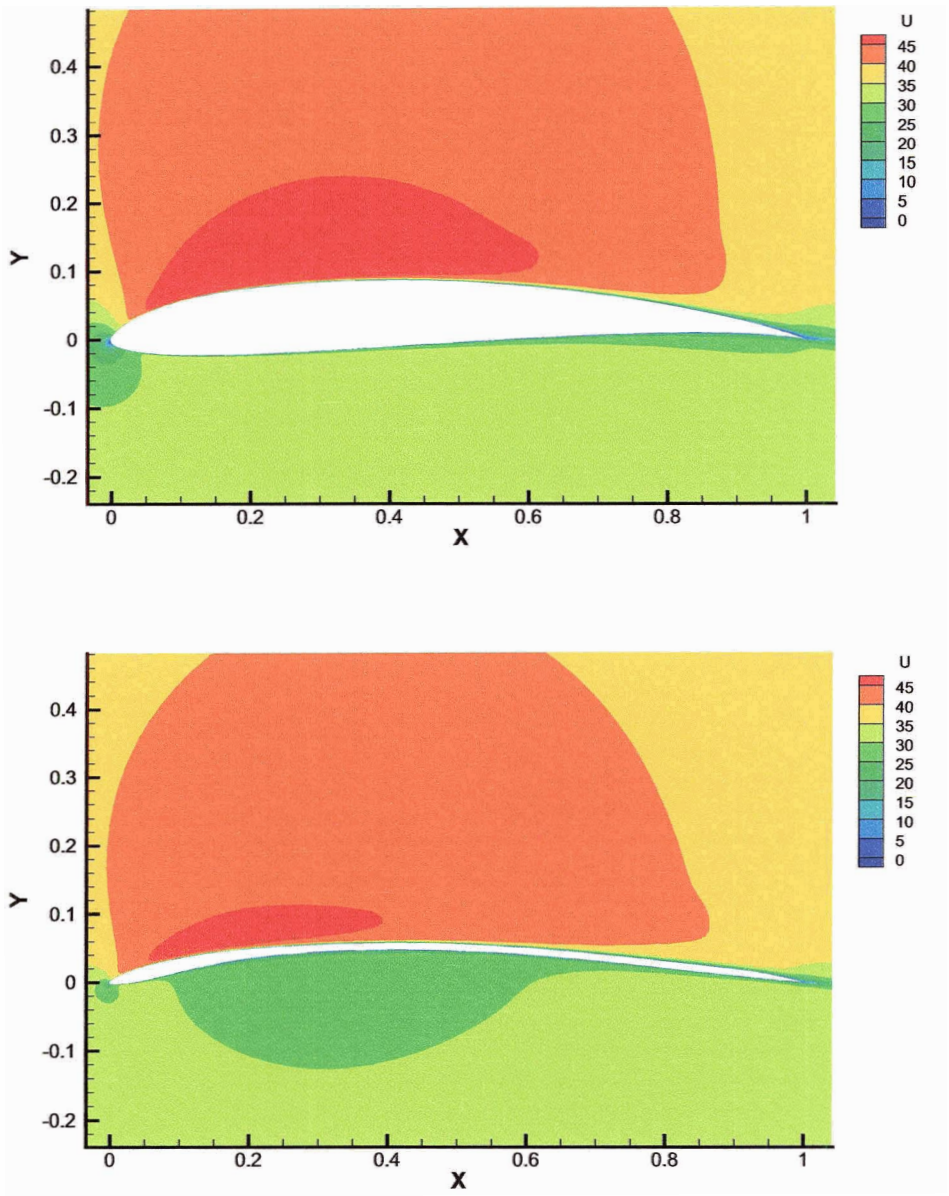

Figure 4.12: Contour plot of the velocity distribution at $Re = 500,000$ and $\alpha = 2$ over the Eppler 66 airfoil (above) and the optimal SQP airfoil (below)

mization algorithms in order to compare the computational efficiency of each one of these methods. The computational efficiency in this case is measured by the number of function evaluations, i.e. calls to the analysis program, that are necessary in order to achieve the optimum solution. A reduction in the number of calls to the analysis program is considered to be the best measure of efficiecy in this case because the analysis program used to evaluate objective function and aerodynamic constraints is the most computationally expensive part of the process taking more thatn 95% of the total computational time.

The SQP algorithm was shown to be the most efficient optimization algorithm due to the fact that it required the lowest number of function evaluations, as shown in table 4.8 and in the convergence plot in figure 4.13. The SQP method obtained the optimum airfoil shape using the least number of iterations, i.e. gradient evaluations, and without relying heavily on the line search - it required only 26 internal function evaluations. In this thesis, the term internal function evaluaions refers to the number of individual function evaluations performed during the optimization. This individual function evaluations are performed directly from the optimization algorithm and they can not be parallelized, therefore the optimization algorithm should reduce the number of internal function evaluations as much as possible.

Table 4.8: Number of function and gradient evaluations before optimum

	MMFD	SLP	SQP
Iterations	26	17	10
Gradient evaluations	26	17	10
Individual function evaluations	231	25	26
Total function evaluations	543	229	146

The MMFD was executed for 26 iterations. It needed to perform 26 gradient evaluations, and 231 internal or individual function evaluations during the line search.

The number of internal function evaluations considerably reduced the efficiency of the method, since these cannot be parallelized. The reason for the large number of internal function evaluations in each iteration is that the lift coefficient constraint must be satisfied at each iteration. As described in section 2, the MMFD algorithm has to solve a Newton-Raphson problem in order to guarantee that the lift constraint is active in the next iteration. This results in a large number of internal function evaluations.

The SLP algorithm converged to the solution after 17 iterations. It needed to perform 17 gradient evaluations and only 25 internal function evaluations. Therefore, in this case, the SLP method is less efficient than the SQP, but more efficient than the MMFD method. Even though the SLP is less efficient than the SQP method, it is important to note the small number of internal function evaluations required. The SLP method used 7 iterations more than the SQP method; however, the number of internal evaluations is 25, one internal function evaluation less than the SQP method. The reduction in the number of internal function evaluations is achieved by using moving limits instead of a line search. Taking into account that more efficient ways exist to compute the gradients of the objective function and constraints as discussed in section 3.4 and that function evaluations are always expensive, the SLP has to be considered as a good candidate for aerodynamic shape optimization.

Through careful observation of the convergence history in figure 4.13, it can be noted that the SLP algorithm increases the objective function in the first iterations instead of decreasing it. Then, from iteration 9 to iteration 17 the algorithm starts to reduce the objective function until it reaches the optimum. The initial behavior is due to a poor selection of the move limits. Initially, the move limits chosen are too large to guarantee an accurate linear approximation of the objective function and constraints. However, as the algorithm evolves, the move limits are reduced

to the appropriate values. Therefore, even though the move limits techniques are responsible for a reduction in the number of internal function evaluations, it is also responsible for an increase in number of iterations required before the optimum is reached. Therefore, the SLP method could be improved by implementing an efficient method to obtain the initial move limits, for example the method discussed in [19]. The method in DOT used to reduce the move limits after the initial move limits are selected seems to perform well in this problem. Once this technique is implemented, the SLP algorithm could be as efficient as the SQP algorithm. Furthermore, to reduce the number of function evaluations in the SQP algorithm, the line search could be substituted with a moving limits methodology.

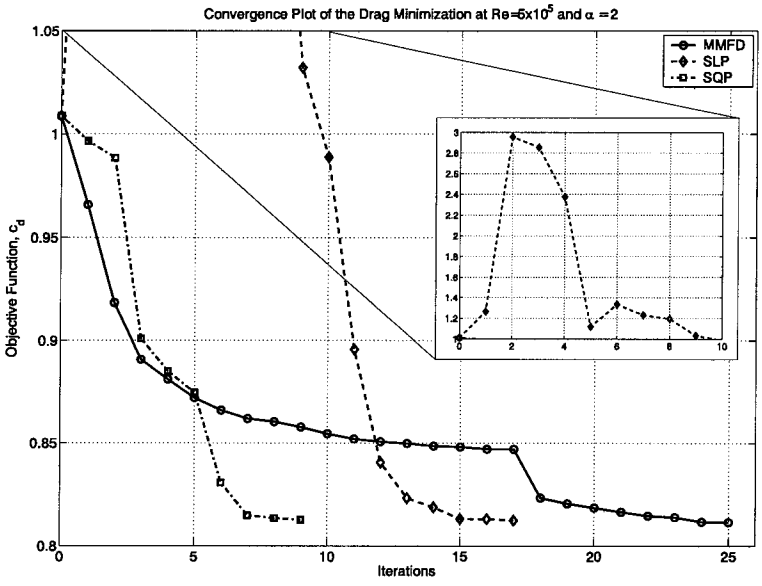

Figure 4.13: Convergence history plot of the drag minimization problem solved using MMFD, SLP and SQP algorithms

In conclusion, in this case, the SQP method outperformed the SLP and MMFD methods in computational efficiency. Therefore, among the three existing method

the SQP method is the method recommended in this thesis. The SLP method is also considered a good candidate for shape optimization if an efficient method to compute the gradients exists. In case of using the SLP method, special care must be taken when selecting the initial move limits. Finally, the MMFD is not recommended, because even though it is capable of reaching the optimal solution, it requires a large amount of internal function evaluations which cannot be parallelized, and this makes the method highly computationally inefficient.

The results obtained from this study are not conclusive, given the small number of cases studied, however they are useful to highlights the main advantages and some improvements on the methods. To obtain more conclusive results, more cases should be studied: a case where the initial design is infeasible, cases with different objective function and constraints, etc. However, due to the computational expense of each one of the optimization methods, the SQP method took 3 days and the MMFD took 10 days using a 22 processor Linux cluster to obtain a solution, more studies were not undertaken. Finally, in order to test that the shape is a global optimum, a global optimization algorithm should be used to solve the problem, or the problem should be solved starting from different initial design.

4.4 Airfoil Morphing

To conclude the applications section of this thesis, the design tool is used to design a set of airfoils for an airfoil morphing UAV for surveillance applications. The UAV has the following characteristics: a takeoff weight of $400N$, a chord length of $0.50m$ and a wing area of $1.4m^2$. This aircraft will morph its airfoil shape in order to increase its performance in each one of the different stages of flight. In general, the main requirements for a surveillance UAV are: a rapid deployment, fast cruise from the

deployment area to the surveillance area (and viceversa) and finally, low speed loiter in the surveillance area. The design program created in this thesis is used here to obtain the optimal airfoil shape at the two main stages of flight: loiter and cruise. The velocity of the UAV, the angle of attack of the airfoil, and the minimum lift coefficient at each one of the stages of flight are shown in table 4.9.

Table 4.9: Characteristics and requirements at each stage of flight

	Velocity $[m/s]$	Air Density $[kg/m^3]$	Re $[-]$	α $[^\circ]$	c_l^{min} $[-]$
Cruise	50.0	1.006	1,450,000	2	0.2272
Loiter	20.0	1.006	582,000	2	1.4201

For loiter and cruise flight, the airfoil shape is obtained by solving a minimum drag airfoil optimization problem subject to a minimum lift coefficient requirement. This method is used because minimizing the drag and keeping the lift equal to the weight of the aircraft will minimize the necessary thrust from the propulsive system. This will increase the range and efficiency of the airplane. To make lift equal to weight, a minimum lift requirement is used instead of an equality constraint to make lift equal to weight. This is because, in order to obtain minimum drag, the final design will most probably be close to the lift constraint, since lift and drag are opposing goals. Additionally, the DOT package is design for inequality constraints and introducing equality constraints will reduce the computational efficiency of the optimization algorithms.

To solve the two problems, the Eppler 66 airfoil is used as the initial airfoil. The solution of these two optimization problems is performed using the SQP method, since it was proven to be the most computationally efficient method in section 4.3. Tables 4.10 and 4.11 show the aerodynamic characteristics of the initial airfoil and the

Table 4.10: Aerodynamic Characteristics of the initial and optimal airfoils at cruise conditions, $Re = 1,450,000$ and $\alpha = 2$

	Eppler 66 (Cruise)	Cruise
c_l	0.89792	0.23154
$c_d \times 10^2$	0.73174	0.43632
$c_{df} \times 10^2$	0.35605	0.32511
$c_{dp} \times 10^2$	0.37569	0.11121
L/D	122.7103	53.0666

Table 4.11: Aerodynamic characteristics of the initial and optimal airfoils at loiter conditions, $Re = 582,000$ and $\alpha = 2$

	Eppler 66 (Loiter)	Loiter
c_l	0.87129	1.42011
$c_d \times 10^2$	0.96626	1.28366
$c_{df} \times 10^2$	0.48720	0.47715
$c_{dp} \times 10^2$	0.47906	0.80651
L/D	90.1703	110.6298

optimal airfoils for cruise and loiter flight conditions. Table 4.13 shows the value of the design variables at the beginning and at the optimal solution for both problems.

In the first case, the design of the airfoil for cruise, the design tool obtained the optimal airfoil after 9 iterations: 9 gradient evaluations and 24 internal function evaluations, adding up to a total of 132 function evaluations. The design tool obtained an airfoil with a drag of 0.43632×10^{-2}; this is a reduction in drag of 57% with respect to the initial design. The drag reduction is obtained mainly from a reduction of almost 80% in the pressure drag. The large reduction in pressure drag is obtained by reducing the pressure peak over the airfoil as can be seen in figure 4.15. The friction drag is also reduced as observed in figure 4.16, but by a smaller amount, because the shear stress at the airfoil boundary is more difficult to reduce. The shape of the airfoil is

plotted in figure 4.14. Comparing the original and final airfoil, it can be observed that the reduction in drag is obtained from: a reduction in camber, a reduction in the thickness of the airfoil and changes in the leading edge shape. By looking at the value of the geometric constraints at the optimal solution in table 4.12, it can be observed that all the constraints are active except close to the leading edge. This proves that the optimization algorithm seeks a reduction in drag by reducing the thickness of the airfoil, except at the leading edge where the reduction of the pressure gradient is more important to guarantee that the boundary layer remains attached.

In the second case under study, the design of the airfoil for loiter, the design tool obtained the optimal airfoil after 16 iterations: 16 function evaluations and 47 internal function evaluations, adding up to a total of 192 function evaluations. In this case, the number of iterations has increased from 9 in the last case to 16. This is due to starting the optimization with an infeasible initial design. At the end of the optimization, the airfoil satisfies all aerodynamic and geometric constraints as shown in tables 4.11 and 4.12, therefore proving that the design tool is able to solve the design problem starting from any design, feasible or infeasible. The optimal airfoil has the required lift coefficient of 1.42011 and a drag coefficient of 1.28366×10^{-2}. This represents an increase of 63% in the lift coefficient and an increase in drag of 33%. In this case, the drag is increased. This was necessary in order to guarantee the lift constraint, because lift and drag are opposing goals and the initial airfoil has a smaller lift than the minimum lift. However, the drag is increased by a relatively small amount while the lift is inceased by a substantial 63%. Looking at table 4.11, it can be observed that the increase in drag is mainly due to an increase in pressure drag. This increase in pressure drag is due to the change in geometry and an also to an increase in the pressure peak as shown in figure 4.17. On the other hand, the friction drag remains fairly constant and it is even slighly reduced respect to the

original airfoil, as shown in figure 4.18.

From the aforementioned results, several conclusions about the characteristics of the set of optimal airfoil for an airfoil morphing UAV and the mechanisms necessary to morph the airfoil for the morphing UAV can be reached. From figure 4.14 and table 4.12, it can be observed that cruise and loiter optimal airfoils share a common characteristic; they are both thin airfoils. In both cases, the thickness of the airfoil is close to the minimum thickness constrained. Only at the leading edge, the airfoils have a larger thickness than the minimum thickness constraint. The main difference between the two airfoils is its camber. While, the cruise airfoil has an almost inexistent camber, the loiter airfoil has a maximum camber of around 10% of the chord. Therefore, the airfoil morphing UAV will have an extreamly thin airfoil - almost a plate - and two morphing mechanics. The first mechanism will be used to control the camber of the airfoil by deforming the airfoil as necessary. The second morphing mechanism will be used to control the thickness of the leading edge and it could be, for example, an inflatable system.

Table 4.12: Value of the geometric constraint at the optimal solution

Constraint	1	2	3	4	5
Eppler 66	2.5000E-04	-3.2000E-02	-8.6000E-02	-9.6500E-02	-5.8500E-02
UAV Cruise	-3.8883E-08	-1.5466E-04	-4.1525E-04	-3.1111E-03	-9.9205E-03
UAV Loiter	0.0000	1.8627E-09	1.8626E-09	-1.4236E-04	-3.7638E-02

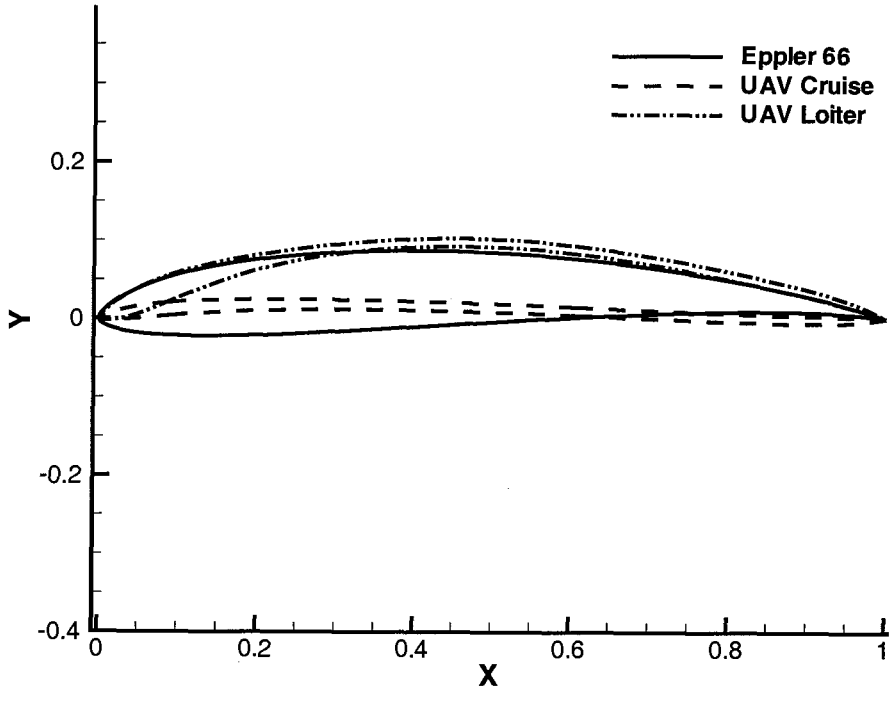

Figure 4.14: Shape of the initial design and the optimal cruise and loiter airfoils

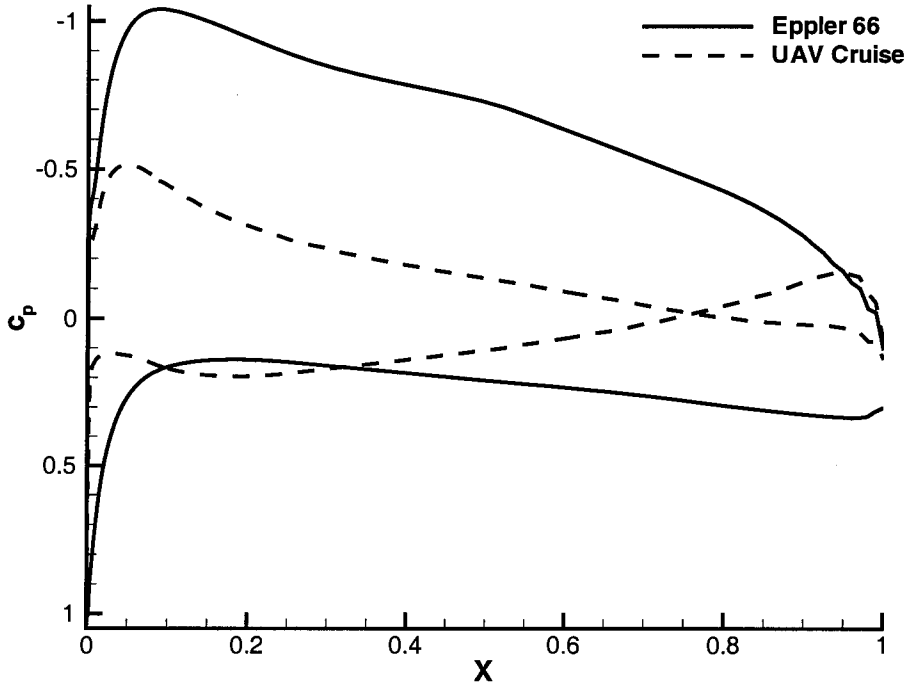

Figure 4.15: Pressure coefficient over the initial and optimal UAV cruise airfoil surface at $Re = 1,450,000$ and $\alpha = 2$

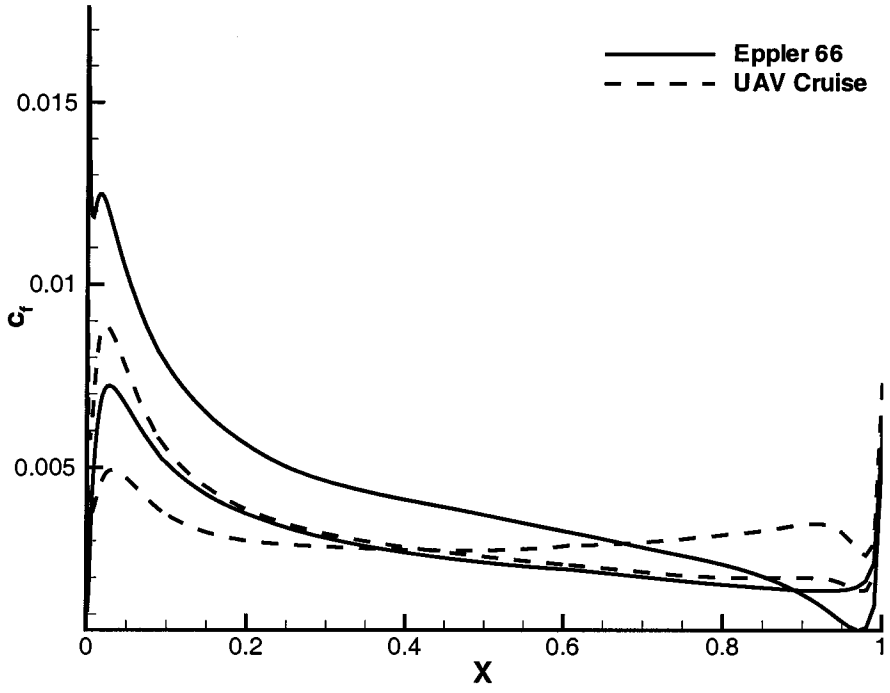

Figure 4.16: Friction coefficient over the initial and optimal UAV cruise airfoil surface at $Re = 1,450,000$ and $\alpha = 2$

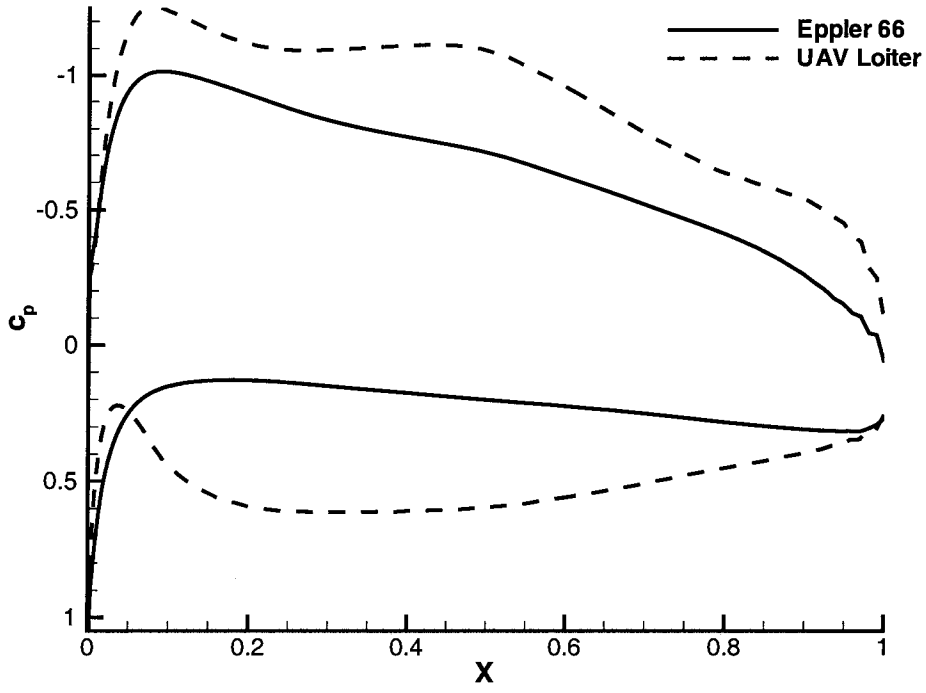

Figure 4.17: Pressure coefficient over the initial and optimal UAV loiter airfoil surface at $Re = 582,000$ and $\alpha = 2$

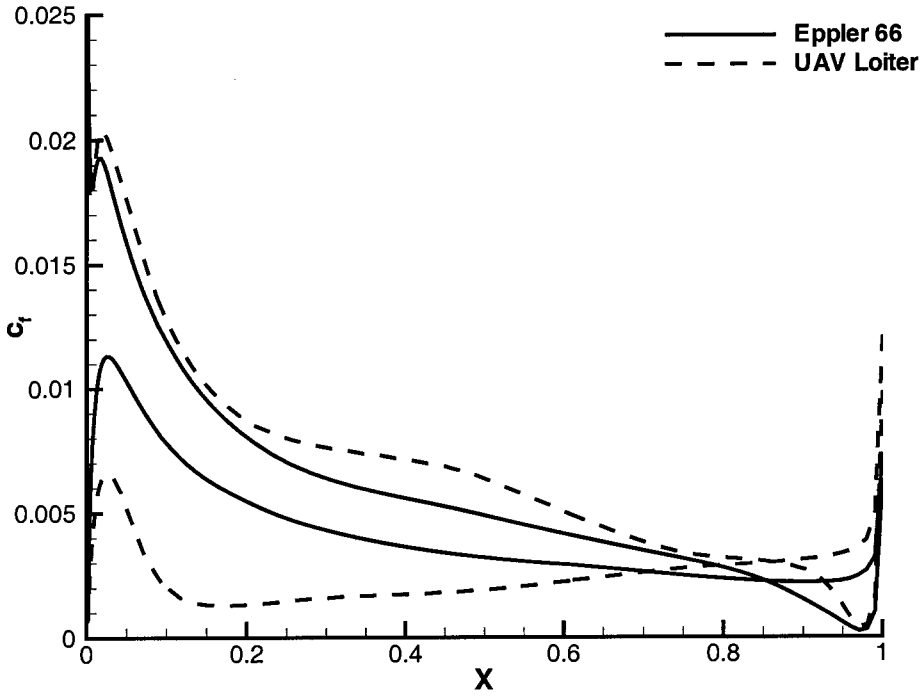

Figure 4.18: Friction coefficient over the initial and optimal UAV loiter airfoil surface at $Re = 582,000$ and $\alpha = 2$

Table 4.13: Value of the design variables at the optimal solution

	x_1	x_2	x_3	x_4
Eppler 66	1.72500E-02	5.35000E-02	9.25000E-02	8.25000E-02
UAV Cruise	2.38709E-03	5.32888E-03	2.00629E-02	2.66050E-02
UAV Loiter	2.65407E-02	6.29888E-02	0.11499	8.67088E-02
	x_5	x_{LE}	x_7	x_8
Eppler 66	4.60000E-02	1.50000E-02	-2.25000E-02	-2.40000E-02
UAV Cruise	1.96137E-02	5.09408E-03	-2.78720E-04	1.34881E-02
UAV Loiter	4.76382E-02	7.03649E-03	-2.85208E-13	7.65665E-02
	x_9	x_{10}	x_{11}	
Eppler 66	-3.50000E-03	1.15000E-02	7.45000E-03	
UAV Cruise	9.64760E-03	-4.82578E-03	-7.61295E-03	
UAV Loiter	0.10499	5.29888E-02	1.65407E-02	

Chapter 5

Conclusions and Future Work

In this thesis, a design tool to optimize airfoil shapes for any flow regime and any stage of flight has been developed. The design tool is developed by coupling: an optimization package, DOT; a shape parametrization algorithm; a fully viscous Navier-Stokes aerodynamic analysis code, SPARC; a mesh deformation algorithm, and; an algorithm to compute the gradient of the objective function and constraints with respect to the design variables. This tool was then used to solve several design problems and to compare several optimization algorithms. A drag minimization problem subject to minimum lift requirements was solved. In this case, the design tool was able to obtain a 20% drag reduction with respect to the initial airfoil and was also able to guarantee the lift requirements. The design tool was then applied to the design of an aircraft with a morphing airfoil.

The drag minimization problem subject to a minimum lift requirement was used to compare the performance of three gradient-based methods: the modified method of feasible directions, sequential linear programming and, sequential quadratic programming. The comparison of the performance of each one of these methods when solving this problem led to the conclusion that all methods achieve the same optimum

and that the SQP algorithm is the most efficient method in reducing the number of iterations necessary to solve aerodynamic shape optimization problems. The SLP is less efficient than SQP in reducing the number of iterations; however, it does not need to perform a line search at each iteration thereby reducing the number of function evaluations necessary at each iteration. Finally, the modified method of feasible directions is not recommended, because it requires a large number of iterations and a large number of function evaluations at each iteration to obtain the optimum. In the MMFD, the large number of function evaluations at each iteration is due to the nature of the algorithm used, which requires that all active constraints in the current iteration remain active during the next iteration. Therefore, it is concluded that sequential quadratic programming algorithms are recommended for aerodynamic shape optimization. SLP algorithms could also be used, if an efficient method to obtain the gradients exist. The SLP algorithm could be improved by introducing an efficient methodology to obtain the initial move limits, since inaccurate initial move limit were the cause of the computational inefficiency of this method. In the future, given that the sequential linear programming algorithm achieved the optimum without using a line search during the iteration, a sequential quadratic programming algorithm without line search should be implemented and its performance should be compared to the sequential quadratic programming algorithm with line search and the new SLP algorithm with a methodology to obtain accurate initial move limits.

The design tool that has been developed has proven its efficiency in optimal airfoil design by obtaining large improvements in performance over initial airfoil shapes. However, the design tool is only able to optimize single element airfoil shapes, i.e. airfoils without flaps or slots. Furthermore, the computational efficiency of the design tool must be improved if more complex objects, such as wings, are to be optimized in a feasible amount of time. In the future, several improvements to the programs

should be undertaken to allow the developed design tool to be able to optimize: multiple element airfoils, wings and, full aircraft configurations. Some of the possible improvements to each of the algorithms used in the developed tool will be outlined in the paragraphs that follow.

A B-spline is used to represent the shape of the airfoil. The B-spline representation is used because it has the ability to represent a large variety of airfoils within a small set of parameters. However, the shape representation method must be reformulated if multiple element airfoils and three-dimensional bodies are to be optimized. In future work, a CAD based parametrization technique should be used that will be able to link the CAD model of the wing or aircraft to the design variables.

The design tool is used for the design of UAVs which fly at low Reynolds Numbers. At low Reynolds numbers, viscous effects contribute greatly to the performance of the airfoil, accounting for almost 50% of the drag. Therefore, a fully viscous Navier-Stokes solver was used in this thesis in order to solve the flow; in this case SPARC [39]. At low Reynolds numbers, the boundary layer characteristics are affected by the laminar-to-turbulent transition of the flow. A suitable turbulence model must be used to account for this phenomenon. In the developed tool, a Spallart-Allmaras one-equation turbulence model is used to account for the turbulence of the flow. However, the laminar-to-turbulent transition is not properly predicted using this model. In the future, the Spallart-Allmaras model should be improved by adding a tripping term. Furthermore, the prediction of laminar-to-turbulent transition using other turbulence models should also be investigated.

Further improvements of the analysis tool could include the implementation of a new fluid flow solver. The fluid flow solver currently used, SPARC, uses a structured multi-block mesh in order to solve the fluid flow. However, structured meshes are difficult to create when working with complex geometries. In the future, to solve the

fluid flow around more complex geometries, such as multi-element airfoils or wings, a CFD code that uses an unstructured mesh should be used. The implementation of an unstructured CFD code, will also affect the mesh deformation algorithm, which will have to be implemented to suit the unstructured grid characteristics. Furthermore, an adaptive discretization technique could be used to deform and reconstruct the mesh. This technique is most suitable for unstructured meshes and could help to reduce the dependence of the results on the quality of the computational mesh as noted in [73].

In the design tool of this thesis, forward-differentiation was used to obtain the gradients. Forward-differentiation is easy to implement and yielded good gradient approximations. However, forward-differences are computationally inefficient and inaccurate compared to adjoint formulations. If computationally expensive shape optimization problems are to be solved - for example optimization of wings or a full aircraft - an adjoint formulation needs to be implemented in the aerodynamic CFD code. The adjoint formulation will increase the accuracy and efficiency of the gradient calculations.

The goal of this project was to obtain a set of optimal airfoils for the different stages of flight of an airfoil morphing aircraft. For this reason, the design tool was developed to perform single point optimization, i.e. the airfoil is optimal only at the specified flight conditions and, there is no guarantee of optimal performance at other flight conditions. In conventional aircraft, where the airfoils do not morph, it is desirable to design an airfoil that has the best possible performance at all the stages of flight. In the future, the option of obtaining an optimal airfoil for a range of flow characteristics could be implemented.

Finally, to design a truly optimal aircraft, all interacting disciplines of the aircraft should be taken into account when the optimization process is performed. In order to fully optimize an aircraft, a multidisciplinary design tool should be developed that

takes into account not only aerodynamic considerations, but also those of structure, propulsion and control.

References

[1] Thomas J. Muller and James D. DeLaurier. Aerodynamics of small vehicles. *Annual Review in Fluid Mechanics*, 35:89–111, 2003.

[2] J. Reuther, J.J. Alonso, A. Jameson, M. Rimlinger, and D. Saunders. Constrained multipoint aerodynamic shape optimization using an adjoint formulation and parallel computers: Part i. *Journal of Aicraft*, 36(1):51–60, 1999.

[3] J. Reuther, J.J. Alonso, A. Jameson, M. Rimlinger, and D. Saunders. Constrained multipoint aerodynamic shape optimization using an adjoint formulation and parallel computers: Part ii. *Journal of Aicraft*, 36(1):61–74, 1999.

[4] M. Drela. Low-Reynolds number airfoil design for the MIT Daedalus prototype: A case study. *Journal of Aircraft*, 25(8):724–732, August 1988.

[5] Michael T. Rusell, Shawn E. Gano, Victor M. Pérez, John E. Renaud, and Stephen M. Batill. Morphing UAV pareto curve shift for enhanced performance. In *45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, Palm Springs, California, 19-22 April 2004.

[6] Shawn E. Gano, Victor M. Pérez, John E. Renaud, Stephen M. Batill, and Brain Sanders. Multilevel variable fidelity optimization of a morphing unmanned

aerial vehicle. In *45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, Palm Springs, California, 19-22 April 2004.

[7] T.H. Pulliam, M. Nemec, T. Holst, and D.W. Zingg. Comparison of evolutionary (genetic) algorithms and adjoint methods for multi-objective viscous airfoil optimizations. In *41st Aerospace Science Meeting and Exhibit*, Reno, NV, 6-10 January 2003.

[8] Schittowski K., Zillober C., and Zotemantel R. Numerical comparison of nonlinear programming algorithms for structural optimization. *Structural Optimization*, 7:1–28, 1994.

[9] Thomas A. Zang and Lawrence L. Green. Multidisciplinary design and optimization techniques: Implications and opportunites for fluid dynamics research. In *30th AIAA Fluid Dynamics Conference*, Norfolk, VA, June 28 - July 1 1999.

[10] J.S. Arora. *Introduction to Optimum Design.* McGraw-Hill, 1989.

[11] A. Antoniou and W.-S. Lu. *Optimization: Methods, Algorithms, and Applications.* Kluwer Academic, 2003.

[12] P.E. Gill, W. Murray, and M.H. Wright. *Practical Optimization.* Academic Press, 1981.

[13] G.N. Vanderplaats. *Numerical Optimization Techniques for Engineering Design with Applications.* McGraw-Hill, 1984.

[14] D.P. Bertseks. *Constrained Optimization and Lagrange Multiplier Methods.* Academic Press, 1982.

[15] L. Lamberti and C. Pappalettere. Comparison of the numerical efficiency of different sequential linear programming based algorithms for structural optimization problems. *Computers and Structures*, 76(6):713–728, July 2000.

[16] P.T. Boggs and J.W. Tolle. Sequential quadratic programming. *Acta Numerica*, 4:1–51, 1995.

[17] G.G. Wang, Z. Dong, and P. Aitchison. Adaptive response surface method - a global optimization scheme for approximation-based design problems. *Engineering Optimization*, 33:707–733, 2001.

[18] J.F. Rodríguez, J.E. Renaud, B.A. Wujek, and R.V. Tappeta. Trust region management is multidisciplinary design optimization. *Journal of Computational and Applied Mathematics*, 124:139–154, 2000.

[19] T.-Y. Chen. Calculation of the move limits for the sequential linear programming method. *International Journal for Numerical Methods in Engineering*, 36:2661–2679, 1993.

[20] F.Schoen. Stochastic techniques for global optimization: A survey on recent advances. *Journal of Global Optimization*, 1(3):207–228, 1991.

[21] L. He and E. Polak. Multistart method with estimation scheme for global satisfying problems. *Journal of Global Optimization*, 3:139–156, 1993.

[22] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 1996.

[23] R.P. Ge and Y.F. Qin. A class of filled functions for finding global minimizers of a function of several variables. *Journal of Optmization Theory and Applications*, 54(2):241–252, 1987.

[24] E.R. Hansen. *Global Optimization Using Interval Analysis.* Dekker, 1992.

[25] D.E. Golberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[26] Z. Michalewicz. *Genetic Algorithms + Data Structure = Evolution Programs.* Springer-Verlag, 1994.

[27] E. Aarts and J. Korst. *Simulated Annealing and Boltzman Machines.* J. Wiley and Sons, 1989.

[28] R.M. Hicks and P.A. Henne. Wing design by numerical optimization. *Journal of Aircraft*, 15(7):407–413, July 1978.

[29] James M. Lacasse and Oktay Baysal. Shape optimization of single- and two-element airfoils on multiblock grids. In *AIAA/USAF/NASA/ISSMO 5th Synopsium on Multidisciplinary Analysis and Optimization*, Panama City Beach, FL, 7-9 September 1994.

[30] L. Huyse, S.L. Padula, R.M. Lewis, and W. Li. Probabilistic approach to free-form airfoil shape optimization under uncertainty. *AIAA Journal*, 40(9):1764–1772, September 2002.

[31] M. Nemec and D. Zingg. Multi-point and multi-objective aerodynamic shape optimization. In *9th AIAA/ISSMO Synopsium on Multidisciplinary Analysis and Optimization*, 4-6 September 2002.

[32] J.R.R.A. Martins. *A Coupled-Adjoint Method for High-Fidelity Aero-structural Optimization.* PhD thesis, Standford University, November 2002.

[33] J.A. Samareh. A survey of shape parameterization techniques. In *CEAS/AIAA/ICASE/NASA Langley International Forum on Aeroelasticity and Structural Dynamics*, pages 333–343, June 22-25 1999.

[34] J.A. Samareh. Status and future of geometry modelling and grid generation for design and optimization. *AIAA Journal*, 36(1):97–104, January-February 1999.

[35] M. Drela. Pros and cons of airfoil optimization. In D.A. Caughey and M.M. Hafez, editors, *Frontiers of Computational Fluid Dynamics*, pages 363–381. World Scientific, 1998.

[36] Joseph Katz and Allen Plotkin. *Low-Speed Aerodynamics: From Wing Theory to Panel Methods.* McGraw-Hill, Inc, 1991.

[37] Mark Drela and Harold Youngren. *Xfoil 6.94 User Guide.* MIT Aeronautics and Astronautics Department, December 2001.

[38] J.H. Ferziger and M. Perić. *Computational Methods for Fluid Dynamics.* Springer Verlag New York, third edition, 2002.

[39] Franco Magagnato. *SPARC Manual (Structured PArallel Research Code).* Department of Fluid Machinery, University of Karlsruhe, Germany.

[40] Susan E. Cliff, Scott D. Thomas, Timothy J. Baker, Antony Jameson, and Raymond M. Hicks. Aerodynamic shape optimization using unstructured grid methods. In *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Atlanta, Georgia, 4-6 September 2002.

[41] Frederic J. Blom. *Investigations on Computational Fluid-Structure Interaction.* PhD thesis, Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland, 1998.

[42] H. M. Tsai, A. S. F. Wong, J. Cai, Y. Zhu, and F. Liu. Unsteady flow calculations with a parallel multiblock moving mesh algorithm. *AIAA Journal*, 6(39):1021–1029, June 2001.

[43] Marian Nemec. *Optimal Shape Design of Aerodynamic Configurations: A Newton-Krylov Approach.* PhD thesis, Univerity of Toronto, 2003.

[44] Siva Kumaran Nadarajah. *The Adjoint Discrete Approah to Aerodynamic Shape Optimization.* PhD thesis, Standford University, 2003.

[45] J.R.R.A. Martins, I.M. Kroo, and J.J. Alonso. An automated method for sensitivity analysis using complex variables. In *38th Aerospace Sciences Meeting and Exhibit*, Reno, NV, January 10-13 2000.

[46] C. Bischof, A. Carle, G. Corliss, A. Griewank, and P. Hovland. ADIFOR - Generating derivative codes from fortran programs. *Scientific Programming*, 1(1):1–29, 1992.

[47] Vanderplaats Research & Development, Inc. *DOT: Design Optimization Tools Users Guide. Version 5.0*, 2001.

[48] Mokhtar S. Bazaraa and C.M. Shetty. *Nonlinear Programming: Theory and Algorithms.* John Wiley & sons, New York, 1979.

[49] J. Abadie, editor. *Nonlinear Programming.* John Wiley & Sons, New York, 1967.

[50] Ben Noble. *Applied Linear Algebra.* Prentice-Hall, 1969.

[51] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in FORTRAN. The Art of Scientific Computing.* Cambridge University Press, second edition, 1994.

[52] H.L. Thomas, G.N. Vanderplaats, and Y-K Shyy. A study of move limit adjustment strategies in the approximation concepts approach to structural synthesis. In *Proceeding 4th AIAA/USAF/NASA/OAI Synopsium on Multidisciplinary Analysis and Optimization*, Holiday Inn, Cleveland, OH, September 21-23 1992.

[53] L. Lamberti and C. Pappalettere. Move limits definition in structural optimization with sequential linear programming. part i. *Computers and Structures*, 81(3):197–213, February 2003.

[54] L. Lamberti and C. Pappalettere. Move limits definition in structural optimization with sequential linear programming. part ii. *Computers and Structures*, 81(3):214–238, February 2003.

[55] Antony Jameson. Aerodynamic design via control theory. *Journal of Scientific Computing*, 3(3):233–260, 1988.

[56] A. Jameson and J.C. Vassberg. Computational fluid dynamics for aerodynamic design: Its current and future impact. In *39th AIAA aerospace sciences meeting & exhibit*, number AIAA Paper 2001-0538, Reno, NV, 8-11 January 2001.

[57] I.H.Abbott and A.E. Von Doenhoff. *Theory of Wing Sections.* Dover Publications, New York, 1959.

[58] Richard H. Bartels, John C. Beatty, and Brain A. Barsky. *An Introduction to Splines for use in Computer Graphics and Geometric Modeling.* Morgan Kaufmann Publishers, Inc., Los Altos, California, 1987.

[59] Richard Eppler. *Airfoil Design and Data.* Springer-Verlag, Berlin, 1990.

[60] H. K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics. The Finite Element Volume Method.* Longman Group Ltd., 1995.

[61] P.R. Spalart and S.R. Allmaras. A one-equation turbulence model for aerodynamic flows. In *30th AIAA Aerospace Sciences Meeting & Exhibit*, number AIAA Paper 92-0439, 1992.

[62] Barton Thomas Stockdill. Turbulence modelling of unsteady separated flow over an airfoil. Master's thesis, Department of Mechanical Engineering, University of Victoria, Canada, 2003.

[63] G. Pedro. Hydrodynamics and fluid-structure interactions in swimming propulsion. Master's thesis, University of Victoria, 2001.

[64] W. Keyle Anderson, James C. Newman, David L. Whitfield, and Eric J. Nielsen. Sensitivity analysis for Navier-Stokes equations in unstructured meshes using complex variables. *AIAA Journal*, 39(1):56–63, January 2001.

[65] Laura L. Sherman, Arthur C. Taylor III, Larry L. Green, Perry A. Newman, Gene W. Hou, and Vamshi Mohan Korivi. First- and second-order aerodynamic sensitivity derivatives via automatic differentiation with incremental iterative methods. *Journal of Computational Physics*, 129:30–331, 1996.

[66] Joaquim R.R.A. Martins, Peter Sturdza, and Juan J. Alonso. The connection between the complex-step derivative approximation and algorithmic differentiation. In *39th Aerospace Sciences Meeting and Exhibit*, Reno, NV, January 8-11 2001.

[67] P. Cusdin and J.-D. Müller. Automatic differentiation and sensitivity analysis methods for computational fluid dynamics. Technical report, School of Aeronautical Engineering. Queen's University, Belfast, May 2003.

[68] C. Bischof, G. Corliss, L. Green, A. Griewank, K. Haigler, and P. Newman. Automatic differentiation of advanced CFD codes for multidisciplinary design.

Computer Systems Engineering, 3(6):625–637, 1993. Presented at the Symposium on High-Performance Computing for Flight Vehicles, Arlington, Virginia, 7-9 December 1992.

[69] Lawrence L. Green, Perrry A. Newman, and Kara J. Haigler. Sensitivity derivatives for advanced CFD algorithm and vicous modeling parameters via automatic differentiation. *Journal of Computational Physics*, (125):313–324, 1996.

[70] Altair Grid Technologies, LLC. *Portable Batch System User Guide*, March 7 2003.

[71] Mohamed Gad el Hak. Control of low-speed airfoil aerodynamics. *AIAA Journal*, 28(9):1537–1552, September 1990.

[72] P.B.S. Lissaman. Low-Reynolds-number airfoils. *Annual Review in Fluid Mechanics*, 15:223–239, 1983.

[73] Emmanuel Laporte and Patric Le Tallec. *Numerical Methods in Sensitivity Analysis and Shape Optimization*. Birkhäuser, Boston, 2003.