An Adjoint-Based Shape-Optimization Method for Aerodynamic Design

An Adjoint-Based Shape-Optimization Method for Aerodynamic Design

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. dr. ir. J.T. Fokkema,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op woensdag 9 september 2009 om 12.30 uur

door

Giampietro CARPENTIERI

Ingegnere meccanico, Università degli Studi Roma Tre, Italië,
geboren te Rome, Italië

Dit proefschrift is goedgekeurd door de promotoren:

Prof. dr. ir. M.J.L. van Tooren
Prof. dr. ir. B. Koren

Samenstelling promotiecommissie:

Rector Magnificus,	Voorzitter
Prof. dr. ir. M.J.L. van Tooren,	Technische Universiteit Delft, promotor
Prof. dr. ir. B. Koren,	CWI / Universiteit Leiden, promotor
Prof. dr. ir. C.W. Oosterlee,	CWI / Technische Universiteit Delft
Prof. dr. O. Pironneau,	Université Paris VI, France
Prof. dr. M. Berggren,	Umeå Universitet, Sweden
Dr. O. Amoignon,	Swedish Defence Research Agency, Sweden
Dr. A. Guardone,	Politecnico di Milano, Italy

This research was supported by the Dutch Technology Foundation STW, applied science division NWO and the technology program of the Ministry of Economic Affairs. The project was granted under number DLR.6054.

ISBN 978-90-6464-352-1

Keywords: Euler equations, Computational Fluid Dynamics, Adjoint equations, Finite-Volume method, Shape Optimization, Shape Parameterization

Printed in the Netherlands

Summary

This thesis presents a shape optimization framework for problems that are encountered routinely in Aerodynamic Design. The nature of the framework is numerical. Its focus is wide as different aspects of the shape optimization practice are treated, e.g., the solution of the flow equations, the sensitivity analysis and the parameterization of the shape. The framework components are not taken as black-boxes but are conceived and implemented within the present work. A considerable part of the thesis describes the characteristics and the implementation of those components. Additional work on unsteady flows, which may find applications in aeroelastic analysis, is presented in the appendix.

The thesis has been compiled using material that has been produced during the course of the PhD research. A large part of this material has been published and presented at conferences [19, 20, 22, 21, 79, 16, 17, 18] whereas some of it is presented here for the very first time. The structure of the manuscript is as follows:

Chapter 1 introduces the facts and the literature that are relevant to the thesis; gives a general description of the shape optimization framework; and provides an overview of the thesis, emphasizing distinctive and innovative aspects of the work. The chapter is meant for those readers who are more interested in the framework as a whole, and in the final application, rather than in the numerical details of the single implementations. For those readers it should be possible to read Chapter 1 together with the chapter on shape optimization, Chapter 5, in order to have a good understanding of the work.

Chapter 2 describes the unstructured finite-volume formulation used to discretize the Euler equations. All aspects of the solution process are discussed: the discretization

of the equations; the evaluation of the numerical fluxes; and the solution scheme for the steady equations. In the final part of the chapter results are presented for airfoil sections in subsonic, transonic and supersonic flows, and for a wing and a wing-body configuration in transonic flow. Comparisons with the literature and with another solver are also provided.

Chapter 3 describes the discrete adjoint formulation for the sensitivity analysis. Since the adjoint method is a central topic in the thesis, this chapter is the most dense of information. In the first part the adjoint method is introduced. In the second part the derivation and the implementation of the discrete adjoint equations is discussed in detail. All the important aspects of the procedure are covered, from the exact/approximate differentiation of the flow solver to the on-the-fly assembly of the matrix-vector products that appear in the equations. The fifth part addresses the solution process and discusses the advantages of treating the adjoint equations as non-linear. Also, a convenient method to solve multiple adjoint problems simultaneously is presented, which can be very useful for constrained design problems. Finally, the last part discusses the testing of the adjoint codes and shows numerical solution of the sensitivity variables.

Chapter 4 describes the parameterization of the shape using Chebyshev polynomials. The requirements for the shape parameterization are introduced first. Then, the least-squares method used for the extraction of the shape parameters is described. Results are included that show how the parameterization can approximate very different airfoil shapes. An extension of the parameterization to wing-like shape is also presented. It uses coefficients for the Chebyshev representation that vary along the span and only parameterize the displacements of the wing, rather than the wing itself. Results show that wings with complex curvatures can be generated.

Chapter 5 is the part of the thesis in which applications of the design method are presented. First, the shape optimization problems are defined and suitable algorithms for their solution are briefly described. Then, the design applications are presented, first in 2D and next in 3D. The 2D applications focus mainly on shock-free airfoil design, although inverse design is also presented. Some of the applications are repeated using different adjoint approximations with the purpose of identifying the effective ones. The first 3D application focuses on the drag minimization of a wing. The second 3D application is divided in two phases. In the first one the shape is optimized in order to generate twice as much lift as the original wing. In the second phase, the wing obtained in the first phase is optimized in order to reduce the drag. Finally, the last part of the chapter presents an interesting method to compute the sensitivity of the shock position with respect to the shape parameters.

Appendix A deals with unsteady flows on deforming meshes. It is divided in two parts. In the first part the extension of the steady solver to unsteady problems on deforming meshes is discussed. In the second part computations are presented that show how the solver can be applied to certain aeroelastic problems. The behavior of airfoils/wings in transonic unsteady flow is discussed from the point of view of system response.

Samenvatting

Deze dissertatie beschrijft een raamwerk voor het oplossen van vormoptimalisatievraagstukken die veelvuldig voorkomen bij aerodynamische vormgeving. Dit raamwerk heeft een numeriek karakter. Het toepassingsgebied is breed: diverse aspecten van vormoptimalisatie worden behandeld, bijvoorbeeld de oplossing van stromingsvergelijkingen, gevoeligheidsanalyse en de parametrisatie van vleugelprofielen. De onderdelen van het raamwerk worden niet als black box gezien, maar worden uitvoerig beschreven. Een groot deel van deze dissertatie beschrijft de eigenschappen en de toepassing van deze onderdeler. Verder onderzoek aan instationaire stromingen, welke toepassingen binnen aero-elastisch onderzoek kunnen hebben, wordt in de appendix besproken.

Het proefschrift is samengesteld uit publicaties geschreven tijdens het promotieonderzoek. Een groot deel van dit werk is gepubliceerd en gepresenteerd op conferenties [19, 20, 22, 21, 79, 16, 17, 18], maar een ander deel wordt hier voor het eerst besproken. De indeling van deze dissertatie is als volgt:

Hoofdstuk 1 introduceert de feiten en literatuur die belangrijk zijn voor dit proefschrift; het geeft een algemene beschrijving van het vormoptimalisatie-raamwerks; en biedt een overzicht over het proefschrift, met de nadruk op de onderscheidende en innovatieve aspecten. Het hoofdstuk is een handige leidraad voor lezers die meer in het systeem als geheel zijn geïnteresseerd en in de uiteindelijke toepassing, dan in de numerieke details van de afzonderlijke implementaties. Voor die lezers zou het mogelijk moeten zijn om hoofdstuk 1 tezamen met het hoofdstuk over vormoptimalisatie, hoofdstuk 5, te lezen en zo een duidelijk beeld van het werk te krijgen.

Hoofdstuk 2 beschrijft de ongestructureerde eindige-volume formulering, die gebruikt

wordt om de Euler-vergelijkingen te discretiseren. Alle aspecten van het oplossingsproces worden besproken: de discretisatie van de vergelijkingen; de evaluatie van de numerieke fluxen, en de oplossingsmethode voor de stationaire vergelijkingen. In het laatste deel van het hoofdstuk worden resultaten gegeven voor vleugelprofielen in subson, transson en superson stromingen, en voor een vleugel en een vleugel-romp configuratie in transson stroming. Vergelijkingen met literatuur en met een andere oplossingsmethode worden hier ook gemaakt.

Hoofdstuk 3 beschrijft de discrete geadjungeerde vergelijking-formulering voor de gevoeligheidsanalyse. Daar de geadjungeerde vergelijking-methode het centrale onderwerp in deze dissertatie is, bevat dit hoofdstuk de meeste informatie. In het eerste deel wordt de geadjungeerde vergelijking-methode geïntroduceerd. In het tweede deel worden de afleiding en de implementatie van de discrete geadjungeerde vergelijkingen in detail behandeld. Alle belangrijke aspecten van de procedure staan beschreven, van de exacte/benaderde differentiatie tot het dynamisch samenstellen van de matrix-vectorproducten die in de vergelijkingen voorkomen. In het vijfde deel worden de voordelen besproken van het niet-lineair beschouwen van de geadjungeerde vergelijkingen. Daarnaast wordt een methode voor het simultaan oplossen van meerdere geadjungeerde grootheden gegeven. De methode erg handig kan zijn bij ontwerpvraagstukken met lijbehorende ontwerpeisen. Het laatste deel, tenslotte, bespreekt het testen van de geadjungeerde vergelijking-programmatuur en geeft de numerieke oplossing voor de gevoeligheidsvariabelen.

Hoofdstuk 4 beschrijft de parametrisatie van de vorm met behulp van Chebyshev polynomen. De eisen ten aanzien van de vormparametrisatie worden eerst geïntroduceerd. Daarna wordt de kleinste kwadratenmethode gebruikt om de beschreven vormparameters te extraheren. Resultaten worden getoond die laten zien hoe de parametrisatie zeer verschillende vleugelprofielen kan benaderen, alsmede een vleugelachtige vorm. Hierbij worden coëfficiënten voor de Chebyshev representatie gebruikt die alleen in de vleugelspanwijaterichting variëren en alleen verplaatsingen van de vleugel parametriseren in plaats van de vleugel zelf. De resultaten laten zien dat vleugels met complexe krommingen gegenereerd kunnen worden.

Hoofdstuk 5 is het deel van de dissertatie waarin toepassingen van de ontwerpmethode worden gepresenteerd. Allereerst worden vormoptimalisatievraagstukken gedefinieerd en worden geschikte algoritmen voor het oplossen kort beschreven. Vervolgens worden de vormgevingstoepassingen gepresenteerd, eerst in 2D en dan in 3D. De 2D toepassingen zijn vooral gericht op schokvrij vleugelprofielontwerp, hoewel invers ontwerp ook beschreven wordt. Enkele toepassingen worden opnieuw gebruikt met andere geadjungeerde vergelijking benaderingen, om zo de effectieve benaderingen te identificeren. De eerste 3D toepassing is gericht op weerstandsminimalisatie van de vleugel. De tweede 3D toepassing is onderverdeeld in twee fasen. In de eerste fase wordt het profiel geoptimaliseerd zodat de draagkracht verdubbelt ten opzichte van de originele vleugel. In de tweede fase wordt de vleugel uit de vorige fase geoptimaliseerd voor weerstandsreductie. Tenslotte wordt in het laatste deel van dit hoofdstuk een interessante methode getoond om de gevoeligheid van de schokpositie voor de vormparameters te berekenen.

Appendix A behandelt instationaire stromingen op vervormende roosters. Het is verdeeld in twee stukken. In het eerste deel wordt de uitbreiding van de stationaire oplossingsmethode naar instationaire vraagstukken op vervormende roosters besproken. In het tweede deel worden berekeningen getoond die laten zien hoe de oplossingsmethode toepast kan worden op bepaalde aero-elastische vraagstukken. Het gedrag van vleugelprofielen/vleugels in transson instationaire stromingen wordt besproken vanuit het systeemtheoretisch oogpunt.

Contents

Chapter 1

Introduction

1.1 Motivation

Designing an aerodynamically efficient aircraft is of paramount importance in the aviation market. Any gain in terms of aerodynamic efficiency translates directly into a reduction in fuel costs, which makes the aircraft appealing to airlines. A reduced amount of fuel burned also implies a reduced impact on the environment, of which the public is increasingly concerned. In general it is very easy to see the benefits of improved aerodynamic efficiency but it is not that easy to achieve.

Aircraft design is multidisciplinary and as such is a very complex task. Aerodynamics plays an important role in the design but it is not the only discipline that requires attention. Stability, structural analysis and aeroelasticity are some of the other disciplines that must be taken into account. As a result, the aerodynamic designer is faced with a constrained design problem, in which the optimum design point must be found subject to several constraints. For instance, consider the wing: its volume may be constrained by the requirement of storing the fuel; its thickness may be constrained by structural considerations; and its planform may be constrained by stability requirements. When trying to improve the aerodynamics of the wing, the designer must ensure that these constraints are satisfied.

Still, the multidisciplinary nature of the problem is not really an obstacle to the designer. In fact, provided that suitable and inexpensive computational tools are available, the designer can learn how to successfully solve the design problem by trial-and-error. An inexpensive tool is one that is not time-consuming, i.e., one that features very limited execution times on the computers that are available today. Usually, an inexpensive tool is based on algebraic relations or numerical models of low complexity. Unfortunately, inexpensive computational tools are not always available, especially for flow problems. In fact, the solution of flow problems usually involves time-consuming numerical methods that are characterized by iterative processes of large systems of equations.

Consider the transonic flow regime at which most commercial aircraft fly at cruise conditions. The flow is subsonic almost everywhere, except in some regions around the aircraft surface in which pockets of supersonic flow appear. Shock waves may eventually mark the passage from supersonic to subsonic flow, causing a loss in dynamic pressure. In these cases the designer wants to reduce the strength of the shock as much as possible. In practice, modeling transonic flow is already a challenging task on its own, which requires sophisticated and time-consuming computer codes. Finding the optimal shape requires sensitivity information, e.g., the gradient of the drag coefficient with respect to the design parameters that describe the shape. That information must be used to determine directions of improvement along which the design should be driven. However, the computation of the sensitivity of an expensive model is usually as expensive as the model itself. Keeping into account that such sensitivity must be computed for all the design parameters, and that the number of such parameters can be very high, one understands that the problem may easily become prohibitively expensive.

A common practice in engineering optimization is to perform the design using low-fidelity modeling. Instead of modeling the relevant physics, the designer makes certain approximations that simplify the solution of the problem. Low-fidelity models are usually very fast and make it possible to compute sensitivity information in a very efficient way. However, the problem occurs that the low-fidelity sensitivity information may not bring the design towards the optimum of the original design problem. The result is that the designer must use the original model to check that the optimum suggested by the low-fidelity model is valid and often additional iterations are required. Usually, low-fidelity models are not very effective for complex problems and in certain situations can be inadequate. In fact, the complexity of the design problem may not be captured at all and consequently the optimum provided by the low-fidelity model may be meaningless.

Because of the aforementioned reasons, the present thesis does not support the use of low-fidelity models for the design and suggests that the original model should be used for the evaluation as well as for the design. The thesis advocates the use of the adjoint method for the sensitivity analysis. Distinguishing feature of the method is that the cost of the sensitivity analysis is independent of the number of design variables and is almost of the same order as that of the flow solution. The method clearly removes the obstacle represented by the excessive costs of more traditional sensitivity analysis and therefore makes the design process feasible when the original model is used. The solver that implements the adjoint method is part of a shape-optimization framework that is very effective and efficient for aerodynamic design. The framework contains all the components that are necessary to perform aerodynamic shape optimization.

1.2 Background

The present section provides an account of the relevant literature on the subject of computational aerodynamics and shape design.The reader should be aware that such account is far from being a complete review. It only mentions the literature that has had a major impact on the present work.

1.2.1 Numerical methods in aerodynamics

Potential flows, Theodorsen and the panel method

In the early days of aviation, engineers needed to evaluate the characteristics of the low-speed airfoil sections they designed. Since the field was just born they were practically in a knowledge vacuum. At that time computers were not there to help. And good numerical methods for partial differential equations were not available either. Therefore, apart from performing wind-tunnel tests, devising analytical methods was the only option available in order to gain some insight in the flow. Theodorsen et al. [120] introduced a theory that used complex potential flow to compute velocities around arbitrary airfoil sections. Core of the theory was a mapping in the complex space, which allowed circles to be mapped into arbitrary airfoil shapes. The method was relatively simple to implement and produced velocity distributions in agreement with those obtained by the experiments.

The complex method could not be extended to three dimensions because a complex space is not available there. Eventually the first computers became available and numerical solution of the three-dimensional potential flow equations by means of integral methods could be attempted. Given the low computational resources integral methods appeared very convenient because only the body surface had to be discretized, and not the field. The panel methods of Morino [89] and Hess [45] covered the body with panels and then computed the so-called influence coefficients, which were integrals evaluated on the panels. In terms of implementation, once the influence coefficients were available, the numerical solution simply required the inversion of a matrix. An interesting description of the panel method suitable for practitioners is given in [62].

The panel method could also work in the transonic regime, which is the regime in which commercial aircrafts began to fly. The method needed to include field points in certain regions, specifically those mostly affected by compressibility. However, including those points considerably increased the level of complexity of the method, which was, and still is, widely used because of its simplicity. Eventually as computer power increased much of the research focused on numerical methods based on the direct solution of partial differential equations.

Full potential flows and the advent of Computational Fluid Dynamics

With aircraft flying at transonic speed, the capability of the flow model to capture shock waves appeared to be paramount. The Euler equations are a complete model of inviscid flow and can capture shock waves. However, the limited computer power, as well as the complex nature of the Euler equations, made the full potential flow equation a better candidate to model the flow at more affordable costs.

Solving the full potential equation in the transonic regime appeared to be a difficult task, mainly because of the mixed elliptic-hyperbolic nature of the equation. Finite-difference solutions with central schemes were usually hampered by instabilities. Murman and Cole [91] discovered that central differences are unsuitable and devised the upwinding concept, which evaluated the differences according to the flow direction. They solved a simpler version of the full potential model, which is known as transonic small disturbance equation. Later, Jameson [51] devised the rotated difference scheme, with which he solved the full potential equation.

The finite-difference method did not appear to be the best choice for discontinuous flow problems [49]. The finite-volume method looked better suited for such problems. In fact, the flow equation contains the divergence operator. Thus the equation can be cast in the form of a conservation law if integrated in a volume. According to the Gauss's theorem, the integral of the divergence may be transformed into a flux integral over the surface around the volume. Since the integral is used, the method can formally handle discontinuities. Application of the method demonstrated its high potential and computations of the flow around wings appeared. A description of the method suitable for practitioners, completed with several applications, can be found in [49]. A more rigorous exposition is found in [71].

Considering the knowledge vacuum that researchers had to face, one can think of the aforementioned findings as major breakthroughs. In the period that followed those findings several researchers started to investigate better numerical schemes, faster solution methods for the equations and flexible ways of generating computational meshes. A new field of research was born. The proceedings of the first AIAA Computational Fluid Dynamics (CFD) conference, which was held in Palm Springs, California, in 1973, give a good description of the enthusiasm generated around the new field. As pointed out by Jameson in [55], that conference signified the emergence of CFD as an accepted tool for airplane design.

Euler equations, artificial dissipation and reconstruction schemes

A combination of improved numerical knowledge and increased computational power gave researchers the opportunity to solve the Euler equations. Jameson, Schmidt and Turkel devised a numerical scheme [58], which is known as JST scheme, that uses a central discretization plus artificial dissipation, and a Runge-Kutta pseudo-time marching procedure to reach steady state. The scheme is very effective and is still in use today.

Other researchers preferred to further the idea of Godunov [39], who introduced a very innovative concept. His method considers the flux between two cells to be similar to that obtained by solving the Riemann problem of gas-dynamics. I.e., a problem in which two inviscid gases, each with different properties, are separated by an interface, which is suddenly removed. The method naturally provided the required upwinding, without the need to introduce artificial dissipation. However, the method needed improvements because it is only first-order accurate, thus not very attractive in practice. Second-order accuracy was achieved by Van Leer, who introduced a reconstruction procedure that was named MUSCL [124]. Van Leer's scheme still uses a Riemann solver, but performs a linear reconstruction of the primitive variables at the interface between the cells. The reconstruction is an additional step, which can be performed prior to the flux evaluation.

Reconstruction-based schemes allowed the extension of Godunov's method but also introduced unwanted oscillations in the solution. Since it appeared that the reconstructed variables violated monotonicity, limiting procedures had to be introduced, which "limited" the value of the variables in order to enforce a monotonic behavior. Amongst the several limiters that have been introduced by researchers, the Van Albada limiter [123] is widely used.

Another issue that prevented the application of the original Godunov method was the time required for the solution of the Riemann problem. In fact, an iterative Newton solution was required, which was time-consuming. Researchers investigated approximate Riemann solvers, which did not require any iterative solution. Amongst the several solvers that have been introduced, Roe's approximate Riemann solver [108] is widely used.

Along the years, several improvements have been devised for both reconstruction-based schemes as well as for dissipation-based schemes. Various applications in science and engineering have arisen. A detailed account of the different methods, suitable for practitioners, can be found in [70].

Unstructured solvers

Initially the finite-volume method became widespread because it is indeed the natural framework for the discretization of conservation laws. However, the finite-volume method also has another interesting feature, which is the capability of discretizing the equations on a mesh with different types of elements and without any particular ordering of the nodes. That is, the finite-volume method may be used for the implementation of unstructured flow solvers. Other methods exist that may be used for the same purpose, e.g., the finite-element method and the spectral element method.

Usually, generating a structured mesh may be extremely time consuming and eventually impossible, especially around complex configurations such as complete aircraft. The limitations of structured solvers were already clear in the past and more flexibility was desirable. Still, the possibility of implementing unstructured flow solvers was not exploited, mainly because of the lack of computational power and memory. Compared to structured solvers, unstructured solvers usually require a much larger amount of memory and more computational power.

The memory is needed for the storage of several quantities, which cannot be computed on-the-fly as in the case of structured solvers. E.g., the computation of the pressure gradient requires knowledge of the pressure at the nearest neighbors of the node, which are usually known in the case of a structured solver. For instance, for a 2D structured solver the nearest neighbors of the node (i,j) are the nodes $(i,j+1)$, $(i+1,j)$, $(i,j-1)$ and $(i-1,j)$. Instead, in the case of an unstructured solver, the nearest neighbors are not implicitly known, and retrieving them may be costly. The computational power is needed in a larger amount because an unstructured solver performs in general more operations than a structured one. E.g., the matrix of an unstructured discretization has a sparse structure, which is due to the lack of ordering of the nodes. The solution of sparse linear systems requires the use of preconditioning techniques, which are not used for structured solvers. Preconditioning techniques require a substantial amount of work for the preparation and for the application of the preconditioner.

In the mid 80s sufficiently powerful computers became available and researchers thought that the time was right to start investigating unstructured solvers. Désidéri and Dervieux [29] solved the Euler equations on an unstructured median-dual mesh, which is a mesh obtained by the union of the nodes with the center of the elements. They devised the so-called upwind triangle scheme, which was the first attempt to implement reconstruction schemes on unstructured meshes. Barth presented a comprehensive work, which is summarized in [9], and which was cardinal in the further progress of unstructured solvers. He investigated the implementation of reconstruction schemes and solution methods suitable for median-dual meshes. He has also shown the equivalence of the finite-volume method on the median-dual mesh to one type of finite-element method. The finite-element method was already used in structural analysis. It uses support functions, linear in its simple form, and integrates the PDEs by parts. A brief introduction to the method may be found in [49] and a very detailed account may be found in [136]. The method is very flexible but has the disadvantage of becoming slightly involved in certain formulations. Usually, the finite-volume method is more intuitive and easy to assimilate.

It is also in the 80s that the first ever computations of the flow around a complete aircraft configuration appeared. Dassault, which pioneered the use of CFD very early, computed the 3D transonic potential flow around the complete Dassault Falcon by means of finite elements [14]. Also Jameson [56] used an unstructured solver based on the finite-element method to compute the transonic Euler flow around the Boeing 747. The numerical scheme was based on an adapted version of his dissipation scheme and the finite-element method he used was equivalent to the finite-volume method on the median-dual mesh.

During the 90s unstructured methods reached a considerable level of maturity, especially for the solution of the RANS equations. Several solution techniques, which were previously used on structured solvers, have been successfully applied to unstructured solvers [82]. Examples are the multigrid method [134, 66, 44, 118, 90, 58] and the lower upper symmetric Gauss-Seidel method, LU-SGS [60]. The former method has been adapted to unstructured solvers mainly by Marviplis, who has produced a large amount of work on the subject [80]. The latter method has been refined for application on unstructured solvers by several authors and two interesting works on the subject are described in [25, 41]. Two interesting review articles on unstructured solvers (written in the mid 90s) that contain a comprehensive list of literature are those by Venkatakrishnan [129] and Mavriplis [81].

Nowadays several unstructured implementations are available worldwide. They are used routinely within the aeronautical industry and within research centers. The unstructured formulation based on the median-dual mesh has been particularly successful and it is widely used. To name but a few of those solvers: the Funcode of NASA Langley, the Edge code of the FOI, the TAU code of the DLR and the NSU code of Mavriplis. There are also plenty of other industries that use CFD for their products. E.g., hydraulic constructions, combustion engines, ventilation systems and many biomedical apparatus are products that take advantage of CFD to a certain extent. It is no coincidence that commercial CFD software appeared and that this software is unstructured. Fluent and NUMECA are examples of commercial software. They are both based on a cell centered discretization, as most commercial software is. Within the aeronautical industry, in spite of the level of maturity reached by unstructured solvers, which would justify a more widespread use of commercial software, in-house developments are still very common [61]. The reason is that flow problems are still very complex and demanding. Thus, tailored implementations can still make a difference [59]. Different is the story in structural analysis, which is apparently a mature field, and which has seen commercial software like NASTRAN and ABAQUS becoming standards.

1.2.2 Aerodynamic Shape Optimization

The work of Lighthill

In the early days of computational aerodynamics, engineers and scientists were mainly focused on devising methods to solve the flow equations. They were not particularly concerned with design methods. The first of such methods was proposed by Lighthill [75]. It is an inverse design method for incompressible potential flow and is named inverse because it attempts to find the shape satisfying a pre-specified pressure distribution. Such a shape does not always exist.

Lighthill also produced two other works, which were not conceived specifically for design purposes, but which have had a major indirect impact on the design of airfoil sections. His work on the boundary layer thickness [77] has been the foundation on which engineers have developed coupling methods for the boundary layer equations and inviscid external flows at times where Navier-Stokes solutions were very far from being available. Coupling methods have been used constantly to design airfoil sections, and are still in use today. The work on the hodograph transformation [76], which allows the solution of transonic flows by means of coordinate transformations, yielded the tool with which the first shock-free airfoils have been created.

Shock-free airfoils

While computational methods in aerodynamics were in their infancy, the focus of the aviation industry shifted from the subsonic regime to the transonic regime (see [27] for a discussion of the benefits of flying in the transonic regime). The compressible flow methods mentioned in Section 1.2.1 were not yet developed and there was no design method available yet. Nevertheless, the need to design suitable airfoils for transonic flow was high and a solution had to be provided.

It was known that shock-free airfoils existed, which smoothly allow the flow to recompress from supersonic to subsonic without shock waves. A theoretical result from

Morawetz [88] confirmed the possible existence of those airfoils for a single design point, i.e., for only one angle of attack and Mach number.

Nieuwland [98] at the NLR capitalized on the work of Lighthill and devised an inverse design theory based on the hodograph transformation. According to the method one could specify a shock-free pressure in the hodograph plane and find the corresponding shape, which needed to be transformed back in the physical plane. Some of the airfoils generated with Nieuwland's method, amongst them the well known NLR7301, were tested successfully in the NLR wind tunnels and appeared to give shock-free pressure distributions. Garabedian and Korn [12] implemented a similar method. They also included boundary layer corrections according to Lighthill's boundary layer thickness theory. In a series of three books, several airfoil designs have been described and program listings were included.

Numerical optimization

The hodograph-based techniques could only be used for the design of shock-free airfoils. They could not be used away from the shock-free design point. Engineers needed design techniques to reduce the shock strength, without necessarily eliminating them completely. They needed techniques that would allow existing designs to be improved. In order to do so it was not possible to start from the pressure distribution but it was necessary to start from a given shape and deform it according to specific design criteria. Numerical models for compressible flow were becoming available and were routinely used to compute transonic flows. However, there were no specific methods available in order to use them for design purposes.

Meanwhile in the field of structural engineering gradient-based optimization techniques became available. These techniques use the gradient of the objective function to drive the design into a direction of improvement. Some algorithms allowed the specification of constraints and bounds on the design variables. Vanderplaats, Hicks and coworkers proposed the use of optimization algorithms for black-box aerodynamic design [125, 47, 48]. Black-box refers to the fact that the flow solver is considered as such. It receives an input geometry and computes the corresponding functions, e.g., lift, drag and pitching moment coefficients. The optimization algorithm drives the process and calls the solver several times to compute the functions and to evaluate the gradient by finite differences.

Black-box gradient-based optimization methods appeared to be effective. Nevertheless, they also appeared to be extremely expensive, impractical for computing-intensive cases such as the flow around 3D wings. The computation of the gradient was and is the cause of the large cost. It is usually computed by finite differences, and requires at least one additional evaluation for each design variable. In the case of N design variables, N flow computations are required to compute the gradient. A rule of thumb says that an optimization problem converges in N steps, which means that the overall cost of the optimization scales with N^2. This explains why such an approach is impractical for 3D wings, which may require hundreds of design variables, if not thousands.

The parameterization of the shape

In applying the black-box method researchers were immediately faced with the need to properly represent the airfoil shape. The design variables for an aerodynamic shape are often not identified intuitively. The concept of parameterization was introduced, which links the shape to a certain number of parameters, the design variables, via a mapping function. Probably, the simplest parameterization is that obtained by using the reduced basis concept. According to the latter, several shapes that the designer assumes to be relevant for the design condition of interest are linearly combined. The coefficients of the linear combination are then used as design variables. Other parameterization methods have been proposed, which use basis functions, such as the bump functions of Hicks and Henne [46]. Basis functions are still widely used nowadays to parameterize shapes. A general review on the subject of shape parameterization can be found in [111].

The continuous adjoint method

Since the computation of the gradient represented a major obstacle in using gradient-based optimization methods, researchers tried to find ways of computing the gradient efficiently. Pironneau proposed a method to derive the sensitivity of the flow equations with respect to shape parameters in the context of the finite-element method for incompressible flows [100]. The method is based on control theory [78]. Also Jameson [52] proposed a method, which is based on the same theory, but for applications of the compressible potential flow equation and the Euler equations. The sensitivity analysis of both methods employs dual variables, which are solved by additional equations. The solution method is similar to the one used for the flow equations. The additional equations are called the adjoint equations and the method is referred to as the continuous adjoint method.

Jameson and co-workers [53, 57, 54, 103, 106, 104, 105] developed the continuous adjoint method initially for the potential flow equation, then for the Euler equations and finally for the RANS equations. Airfoils as well as wings have been successfully optimized. The aerospace community was impressed by the applications and several research groups started working on the method.

Two drawbacks of the continuous adjoint method appeared. The differentiation of the equations, followed by the discretization, leads to an inconsistency between the computed gradient and that of the discrete implementation. Thus, the convergence of a gradient-based optimization algorithm could be hampered by the inconsistent gradient. Moreover, boundary conditions for the dual variables, which have to be provided in order to solve the equations, are not very easy to define because the dual variables do not have an immediate physical interpretation.

The discrete adjoint method

An alternative to the continuous adjoint method is the discrete adjoint method, which is obtained by performing the sensitivity analysis directly on the discrete code. The method is relatively straightforward to understand: only basic algebra is involved and the boundary conditions for the dual variables are not an issue because they unfold naturally. Moreover, the direct differentiation of the discretized flow equations, if performed exactly, implies a consistent gradient. In terms of implementation, having a consistent gradient means that the verification of the sensitivity code may be performed unambiguously.

Disadvantages of the method are the storage requirements and the difficulty of differentiating large implementations. The storage requirements have to be carefully considered because several matrices appear in the derivation. In practice not all of these need to be stored and some may be evaluated on-the-fly. Differentiating large implementations may be a huge undertaking, mainly because of the presence of complex numerics. In order to simplify the derivation of the adjoint code, a methodology known as Automatic Differentiation (AD) may be applied. AD uses basic linearization rules to manipulate source code and to create the sensitivity code. Adjoint codes may be obtained by using the so-called reverse mode of automatic differentiation.

Several implementations of the discrete adjoint method have appeared in the literature. Elliott [35] hand-coded a discrete adjoint code and presented a 3D shape optimization application. At NASA Langley, researchers gradually hand-coded the adjoint of a 3D RANS solver [7, 95, 97] and presented applications of industrial interest. Their code was also used for another application that involves the adjoint method: error estimation [127]. Mohammadi [86] pioneered the use of Automatic Differentiation and derived an adjoint code that was used for shape optimization directed towards sonic boom reduction. Giles [37] and Müller [85] performed a thorough investigation of Automatic Differentiation and proposed an efficient use of it, which discarded the practice of differentiating the code as a black-box. Nemec [93], Amoignon [5], Dwight [32] and Mavriplis [83, 84] hand-coded the discrete adjoint of large unstructured solvers and presented interesting applications.

Although Automatic Differentiation may reduce dramatically the amount of work required to differentiate the code, it has to be used carefully, especially in reverse mode. A black-box use of AD may yield extremely inefficient code, which in some cases may require an amount of memory several times that of the original code. A black-box use of AD is for instance the case where the routines are submitted for differentiation to the AD tool without any modification. Performance and efficiency (CPU and memory usage) of the black-box differentiated code may be very poor, depending on how the original code is implemented. Better performance and higher efficiency are achieved when the AD tool is applied at the cell-interface level, as suggested by Giles [37] and Müller [85]. E.g., consider the residual assembly routine, which usually contains a loop on the interfaces for the collection of the fluxes. Differentiation of the routine means dealing with matrices of dimensions equal to the number of interfaces. Instead, one can apply the AD tool to the routine that compute the fluxes within the loop. This way the matrices of the differentiated code are of size equal to the number of equations (for instance 4 for 2D Euler). However, applying AD at the cell-interface level usually requires the original code to be rearranged. Moreover, it requires the assembly of the sensitivity to be implemented. In fact, for the aforementioned residual assembly, the sensitivity of the fluxes must be assembled in order to have the sensitivity of the residual. In general, rearranging the original code and assembling the sensitivity means a considerable amount of human work, which makes the process far from being automatic.

CFD as design tool

Prior to the introduction of sensitivity analysis in the form of the adjoint method, CFD was seen more as an analysis tool rather than as a design one. Obviously it was computationally too expensive to have an active role in the design process. Hence CFD was mainly used to analyze objects created by means of low-fidelity design methods in order to verify if they were behaving as predicted. CFD was involved in the design process, but in a rather passive role

The adjoint method has changed the image of CFD seen by engineers. There is now awareness that CFD can have a very active role in the design process and that its involvement may start much earlier. The past and future role of CFD in aircraft design is summarized by Jameson in [55]. A possible range of applications, in the field of aeronautics as well as in several other fields, are summarized by Mohammadi and Pironneau in [87].

1.2.3 The adjoint method seen from a duality viewpoint

Readers that are unfamiliar with the literature on the adjoint method have sometimes trouble understanding how it works, and why it works. As pointed out in [38] the adjoint method may be seen from a Lagrangian viewpoint or from a dual one. The Lagrangian viewpoint is usually not very easy to grasp for those who are not confident with optimization theory. Instead, the duality viewpoint, which may be explained using only some matrix algebra, is very intuitive and easy to understand. Below the duality viewpoint is discussed for introductory purposes. In the chapter that describes the adjoint formulation an explanation based on the Lagrangian viewpoint is provided.

Single dual vector

Consider the vector $\mathbf{g}$ of length N and the matrix $\mathbf{A}$ of size $N \times N$. Also, consider the rectangular matrices $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_\mathrm{M}]$ and $\mathbf{C} = [\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_\mathrm{M}]$, which are formed by M vectors of length N. Suppose that one has to evaluate

$$\mathbf{q} = \mathbf{g}^T\mathbf{B} \qquad \text{with } \mathbf{B} \text{ satisfying} \qquad \mathbf{AB} = \mathbf{C}\,. \tag{1.1}$$

In order to evaluate the vector of matrix-vector products $\mathbf{q} = [q_1, q_2, ..., q_M]$, one has to compute $\mathbf{B}$, which requires the solution of M linear systems. I.e., one has to solve

$$q_i = \mathbf{g}^T\mathbf{b}_i \qquad \text{with } \mathbf{b}_i \text{ satisfying} \qquad \mathbf{A}\mathbf{b}_i = \mathbf{c}_i\,, \quad (i = 1, M)\,. \tag{1.2}$$

Assuming that the computation of the matrix-vector products is inexpensive in comparison to the solution of the linear systems, the cost of solving the above problem scales linearly with M. The problem defined in (1.2) is referred to as the primal problem. Definitions (1.1) and (1.2) are the same and the former definition is only more compact.

Another problem exists, which is equivalent to the primal problem, but which only requires a single solution of the linear system instead of the M solutions of the primal problem. Such a problem is referred to as the dual problem, and requires the evaluation of

$$\mathbf{q} = \mathbf{s}^T\mathbf{C} \qquad \text{with } \mathbf{s} \text{ satisfying} \qquad \mathbf{A}^T\mathbf{s} = \mathbf{g}\,. \tag{1.3}$$

$\mathbf{s}$ is the vector of dual variables. As shown by the above definition, the cost of solving the dual problem is equal to one linear system solution for $\mathbf{s}$ only, regardless of the dimension M. As for the primal problem, it is assumed that the matrix-vector products are inexpensive to compute in comparison to the solution of the linear system.

Some algebraic manipulation shows that the matrix-vector products obtained by the solution of the two problems are equivalent:

$$\mathbf{q} = \mathbf{s}^T\mathbf{C} = \mathbf{s}^T\mathbf{AB} = (\mathbf{A}^T\mathbf{s})^T\mathbf{B} = \mathbf{g}^T\mathbf{B}\,. \tag{1.4}$$

Multiple dual vectors

Consider the case of more than one vector $\mathbf{g}$. If there are L vectors $\mathbf{g}_j$, there must also be L dual vectors $\mathbf{s}_j$, which implies the solution of L linear systems. Hence one can see that there is a trade-off between the primal and the dual problem in terms of convenience. The dual problem is more convenient when $L \ll M$ whereas the primal problem is more convenient when $M \ll L$.

The adjoint equations

The sensitivity problem of the present work may be cast in the primal/dual form described above. For this purpose consider the functional $J = J(\mathbf{U}, \alpha_i)$, which depends on the flow variables $\mathbf{U}$ and on the shape parameters α_i, $i = 1, M$. In practice J may be the lift, the drag coefficient or some other similar functional. The flow variables are the solution of the steady flow equation $\mathbf{R}(\mathbf{U}, \alpha) = \mathbf{0}$, where $\mathbf{R}$ is the residuals vector, which is also dependent on the same quantities.

Differentiating the residuals and the functional with respect to α_i yields the following problem:

$$\frac{dJ}{d\alpha_i} = \frac{\partial J}{\partial \alpha_i} + \frac{\partial J}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial \alpha_i}\,, \qquad \frac{\partial \mathbf{R}}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial \alpha_i} = -\frac{\partial \mathbf{R}}{\partial \alpha_i}. \tag{1.5}$$

As can be seen the problem is identical to Eq. (1.2). Therefore, by analogy with Eq. (1.3), one can readily write the dual problem as

$$\frac{dJ}{d\alpha} = \frac{\partial J}{\partial \alpha_i} - \mathbf{\Lambda}^T\frac{\partial \mathbf{R}}{\partial \alpha_i}\,, \qquad \frac{\partial \mathbf{R}}{\partial \mathbf{U}}^T\mathbf{\Lambda} = \frac{\partial J}{\partial \mathbf{U}}^T\,, \tag{1.6}$$

where $\mathbf{\Lambda}$ is the vector of dual/adjoint variables. In the above equation the linear system is referred to as the adjoint equation. Equation (1.4) may be used to verify that the above dual problem is equivalent to the primal problem.

In the case of more functionals with respect to which the sensitivity must be computed, more than one adjoint equation must be solved. E.g., in order to compute the lift and drag sensitivity together, two adjoints must be computed, one for the lift and one for the drag. For aerodynamic problems usually there are more variables than functionals and therefore the dual problem is convenient. For other problems, for instance for structural analysis problems, the reverse may be true [126].

1.3 Overview of the thesis

1.3.1 Description of the shape optimization framework

Figure 1.1 shows the diagram of the shape optimization framework implemented in the present work. The diagram illustrates the optimization process, which is repeated a certain number of design iterations in order to improve an existing design. Starting from the optimizer, the process works as follows.

> **Optimizer** - The process is driven by an optimization algorithm. At each design iteration it receives the functionals and their gradients, and computes a new set of design variables $\bar{\alpha}$. The latter variables should give an improvement in the design, which means they should minimize the objective function at convergence while satisfying the design constraints of the optimization problem.
>
> **Shape Parameterization** - The M design variables $\bar{\alpha} = [\alpha_1, ..., \alpha_M]$ define the displacements $\Delta\mathbf{X}_B$ of the boundary surface. After the design variables are computed by the optimizer, the shape parameterization module receives them as input and generates the displacements. The module also generates the geometric functionals $\bar{I}$. For a 2D case the latter functionals may be, e.g., the relative thickness, the nose radius and the trailing edge angle. For a 3D case the same quantities may be evaluated at different sections along the span. Also the wing volume may be evaluated for the purpose of imposing a constraint on it.
>
> **Mesh Deformation** - The mesh deformation module imposes the displacements $\Delta\mathbf{X}_B$ on the wing surface and computes the deformed mesh coordinates $\mathbf{X}$. In the present work the deformation is realized by the spring analogy method, which propagates the surface deformations into the volume mesh by means of Jacobi iterations.
>
> **Flow Solver** - The updated mesh is used by the flow solver to compute the flow field $\mathbf{U}$ and to evaluate the L flow functionals $\bar{J} = [J_1, .., J_L]$ needed by the optimizer. Usually the functionals are the lift, the drag and the pitching moment coefficients. Other functionals that are defined as flow variables (e.g., pressure) integrals over the surface, may also be considered.
>
> **Adjoint Solver** - If the optimization algorithm requires the gradients, the adjoint solver computes the adjoint variables $\mathbf{\Lambda}_j$. The subscript j is necessary because

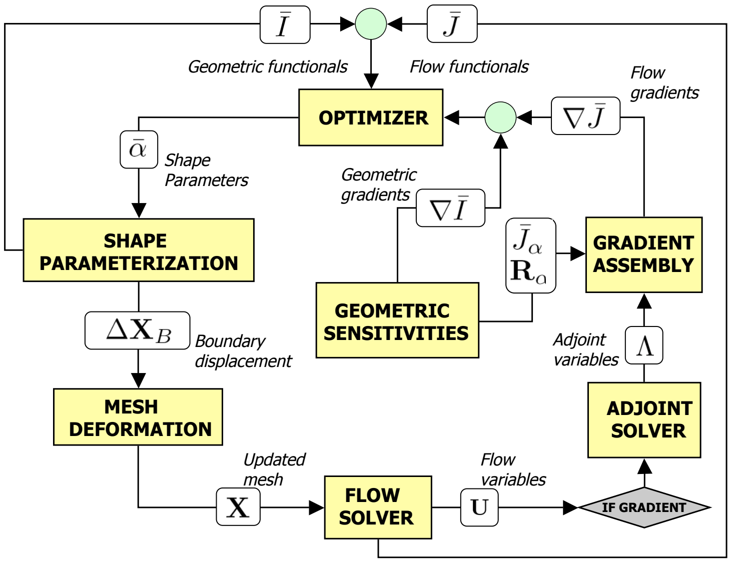


Figure 1.1: *Flow chart of the adjoint-based shape optimization framework*

each functional needs its own adjoint. E.g., if lift, drag and pitching moment sensitivity is needed, $L = 3$ adjoints have to be computed.

Geometric Sensitivities - As shown by the first term of Eq. (1.6), the geometric sensitivities $\bar{J}_\alpha = \partial J_j/\partial\alpha_i$ and $\mathbf{R}_\alpha = \partial\mathbf{R}/\partial\alpha_i$, for $j = 1, L$ and $i = 1, M$, must be computed in order to assemble the gradient. The gradient of the geometric functionals, $\nabla\bar{I}$, is also a geometric sensitivity. The term geometric sensitivities is used because these are partial derivatives with respect to geometric quantities. In the present work they are computed using Automatic Differentiation (in 2D) and finite differences (in 3D). The computation for 2D cases is inexpensive compared to the flow solution process. For 3D cases the computational cost increases considerably because of the iterative mesh deformation algorithm.

Gradient Assembly - Each component of the final gradient $\nabla\bar{J}$ is assembled using the adjoint variables and the geometric sensitivities according to the first formula in Eq. (1.6). As can be seen from the equation, the assembly amounts to the computation of matrix-vector products.

1.3.2 Distinctive features and innovative aspects of the work

Solution of the flow equations - Chapter 2

Finite-volume discretization on the median-dual mesh. The flow solver is based upon an unstructured finite-volume formulation that discretizes the Euler equations on the median-dual mesh [9]. The median-dual mesh can be easily drawn on top of the original mesh, connecting each cell center to its edge mid-points. The result of the construction is a mesh where the control volumes can be located on the nodes of the original mesh.

Interesting feature of the median-dual discretization is the mesh transparency [43]. In fact the edge-based data structure used within the solver makes no distinction between 2D and 3D or between different types of elements. There is always a single edge, only with a normal vector linked to it.

Second-order reconstruction-based scheme. The flow solver employs a MUSCL-like reconstruction scheme [124], which linearly reconstructs the primitive variables across the control volume interfaces, i.e., at the mid-point of each edge of the mesh. The gradient is computed using a least-squares or a Green-Gauss formulation [9]. In order to avoid numerical oscillations a multidimensional limiter [128] is used, which enforces monotonicity in the solution.

The reconstructed variables/states have different values across the control volume interfaces and therefore a discontinuity exists. Roe's approximate Riemann solver [108] is used to evaluate the flux given the discontinuous states. For the boundary conditions flux-vector splitting [119] is used for the boundaries at infinity whereas for the wall and symmetry interfaces zero normal-velocity flux is enforced.

Implicit solution The steady solution of the flow equations is found by means of an implicit pseudo-time stepping scheme. The scheme is obtained as an application of the defect correction method to the semi-discrete form of the equations [66]. A local pseudo-time step is added to each control volume and is gradually increased to infinity according to the level of convergence of the solution. In practice, the scheme coincides with a backward Euler method that uses an approximate Jacobian [82].

Symmetric Gauss-Seidel preconditioning At each pseudo-time step the flow solution is updated by solving a sparse linear system of equations. The solution is obtained iteratively by means of a Symmetric Gauss-Seidel procedure. The procedure may be seen as an iterative one in which the residuals of the linear system are preconditioned by the Symmetric Gauss-Seidel matrix. The matrix has a lower-upper structure and therefore may be inverted by a forward sweep on the nodes for the lower part, followed by a backward sweep for the upper part. The linear iterations are stopped when the norm of the linear residuals vector is around one order of magnitude smaller than the norm of the residuals vector.

An interesting feature of the solution method is the possibility of running matrix-free, i.e., without storing the off-diagonal terms of the Jacobian matrix. The matrix-free option requires an amount of memory similar to that of an explicit scheme. However, if quantities are not stored, they must be re-computed at each iteration. Thus, the matrix-free option has higher requirements in terms of CPU-time.

The method has some resemblance with the LU-SGS method [60, 41]. It may be seen as an improved version of LU-SGS.

Sensitivity analysis - Chapter 3

Exact discrete adjoint A hand-coded and exact derivation of the discrete adjoint is presented. The derivation starts with the differentiation of the residuals vector assembly. The result is the assembly of the matrix-vector product of the transposed residuals Jacobian, which is obtained in closed form. The presence of the transposition makes the implementation of the matrix-vector product rather complicated. It is necessary to assemble the matrix-vector product directly, mainly because of storage reasons. First storing the matrix, and then performing the transposition, would require too much memory.

The derivation is exact; each routine is differentiated without introducing any approximation. Exactness is demonstrated by showing that Newton iterations converge quadratically when the differentiated residuals Jacobian is used.

Approximate discrete adjoint The assembly of the exact discrete adjoint is complicated because of the non-linearity of the Euler equations and its coupling, and because of the presence of both the gradient and the limiter in the discretization. To derive a code which is more efficient and relatively easier to implement, approximate derivations are presented.

Approximations consist of neglecting the limiters and details in the numerical fluxes. Using such approximations makes it possible to construct the adjoint by means of a two-pass matrix-free assembly procedure. This will be shown in detail. Memory requirements and execution times of the adjoint code appear to be very similar to those of the original flow solver.

Implicit solution with Symmetric Gauss-Seidel preconditioning The adjoint equations are a linear system of equations. Because of the second-order contribution of the reconstruction operator the Jacobian is not diagonally dominant. Hence an iterative linear solver is not suitable for the solution.

A widely used approach to the solution of the adjoint equations is to use the same solution method as the non-linear flow equations [37, 83, 5]. The approach is also adopted within the present work, i.e., the defect correction method of the flow solver is used. The defect correction shifts the exact Jacobian to the right-hand side and uses the approximate Jacobian, which is diagonally dominant, to drive the iterations. The advantage of using the same flow solution method is twofold. First, the solution process is robust and, second, the implementation is already available from the flow solver.

Simultaneous solution of several adjoint equations As pointed out by Giles [38] the presence of functional constraints in the optimization problem, e.g., a lift or a pitching moment constraint or both, may reduce the attractiveness of the adjoint method. In fact, as already mentioned in Section 1.2.3, each functional constraint requires the solution of an adjoint. The computational time required to evaluate the gradient scales linearly with the number of functional constraints.

An original aspect of the present work is that the solution of multiple adjoints is performed simultaneously. As can be seen from Eq. (1.6) different adjoint equations have the same Jacobian matrix in common. Since the evaluation of the Jacobian terms is relatively expensive, it makes sense that once those terms are available several matrix-vector products are performed simultaneously. Results show that, compared to sequential solutions, the simultaneous solution method is faster at the expense of a relatively small memory overhead.

Shape parameterization - Chapter 4

Chebyshev polynomials for the parameterization of airfoils The parameterization of the shape is based upon Chebyshev polynomials. The shape of an airfoil is represented as a truncated linear expansion of orthonormal basis functions, the coefficients of which are used as design variables (or shape parameters).

The Chebyshev parameterization method has seen very few applications to shape optimization problems, especially those that involve transonic flows, which are the target application of the present thesis. The first work based on Chebyshev polynomials [132] included only a single transonic shape optimization case. A later work [114] did not include any transonic case at all and so far other applications did not appear in the literature. An original aspect of the present work is that it presents several design cases, transonic and supersonic, which all show the great potential of Chebyshev polynomials for shape optimization.

Parameterization of wings A novelty is the extension of the Chebyshev parameterization to wing-like shapes. The extension is relatively simple to implement. It is assumed that the shape parameters change along the span of the wing according to another polynomial law. Moreover, the displacements of the shape, rather than the shape itself, are parameterized. Applications of the method to wing design cases are presented, which shows that the method is promising.

Optimization algorithms and applications - Chapter 5

Two- and three-dimensional design applications Drag minimization of several airfoils with constraints on the lift and on geometric quantities such as the relative thickness and the radii of curvature are presented. The cases are mainly transonic and one is supersonic. A three-dimensional design case is presented, which is split into two phases. In the first phase the lift of an existing wing is doubled. In the second phase the drag of the wing obtained in the first phase is reduced, while its lift is kept fixed.

Constrained optimization In the literature on shape optimization a widespread practice for the solution of constrained problems is to use an unconstrained optimization algorithm and to include the constraints as penalty terms in the objective. The approach is convenient mainly because the adjoints of the constraints do not need to be computed. However it may lead to ill-conditioning of the optimization problem and difficulties in satisfying the constraints [126].

In the present work, as already mentioned, multiple adjoints may be computed conveniently. Therefore the most logical choice is to use constrained optimization algorithms, which are designed specifically for constrained problems. Two algorithms are used. The first one is the widely used Sequential Quadratic Programming algorithm, which is taken from the Matlab library. The second one is the Sequential Linear Programming algorithm known as method of centers. It is based upon a linearization of the optimization problem. The result of the linearization is a linear programming problem that may be solved by a simplex algorithm. The algorithm is implemented using the Simplex method available within Matlab. Although the algorithm is extremely simple it is very effective as shown by the applications.

Sequential Linear Programming does not reflect today's state-of-the-art in engineering optimization. The intention here is to show that design constraints may be taken into account very easily, without the need to use such methods as the penalty function method.

Comparison of exact and approximate adjoint The approximations introduced in the discrete adjoint have a deteriorating effect on the accuracy of the gradient. However, the gradient may not be the best indicator of whether an approximation is suitable or not for design purposes.

An original aspect of the present work is that it considers the effect of the approximations directly on the optimization results. A comparison is made between the results obtained using the exact adjoint and those obtained using approximate adjoint codes. Results show that those approximations that take into account the second order accuracy of the scheme are as effective as the exact adjoint is.

The issue of approximations in the adjoint was raised by the author in an AIAA conference paper [20]. In that paper results were only presented for an inverse design case. For constrained cases results have been presented later [22]. Within the same AIAA conference another author raised the same issues and presented results for RANS computations with similar findings [32].

Sensitivity of shock displacements It may be interesting for a designer to specify the position of the shock along a transonic airfoil section. In order to do so Pironneau suggested a method to compute the sensitivity of the shock position with respect to a design parameter [101]. The sensitivity may be used to drive the shock towards the desired position by means of a gradient-based optimization algorithm.

The method is yet to be investigated thoroughly and complete design applications have yet to appear. The present work contributes to the development of the method by presenting some computations of the shock sensitivity for different cases. From the few results presented it is possible to draw some interesting conclusions.

Additional work on unsteady flows - Appendix A

Unsteady flow computations on deforming meshes An unsteady version of the flow solver is also presented, which is suitable for computations on deforming meshes. The unsteady numerical scheme is second order accurate in time and satisfies the Geometric Conservation Law (GCL). Computations are presented of oscillating airfoils and wings.

Linear behavior of the impulsive response The unsteady flow solver is used to compute the impulsive response of airfoils and wings. The response may be used for the purpose of investigating the aeroelastic stability. A hypothesis was formulated and confirmed [79] that the behavior of airfoils with respect to small perturbations is linear. In the present work results are presented, which show that that hypothesis is also valid for wings.

1.3.3 Implementation

FORmula TRANslation

The solvers within the present framework, except for the optimizer, have been implemented from scratch using the Fortran programming language. The Fortran 90 standard has been adopted. The code has been divided in modules, and dynamic memory allocation and basic data types have been used.

Several reasons motivated the choice for the language. E.g., it is widespread in CFD, at least in research codes. Since its first appearance it has evolved into a very complete language for numerical purposes. Moreover, there are a considerable amount of numerical libraries that are freely available on the web. Those libraries are often validated and tested and are thus reliable.

Another reason for choosing Fortran is that at the time of starting this work reliable and publicly available Automatic Differentiation software was available only in Fortran. In other languages it was available, but was still in a phase of early development. Although Automatic Differentiation has never played a major role in the present work, it was planned to use it at least for testing the hand-coded differentiation.

The different solvers within the shape optimization framework are assembled together with a scripting language, which is available in Matlab. In practice, the Fortran code is compiled and executables are generated, which are run by the Matlab scripts. Note that the scripts only run the executables and do not call Fortran functions in any case.

Mesh data structure

Unstructured meshes require a more complicated data structure than structured meshes. For the latter meshes the grid points can be read from a list, without topology information. Such information is not required because the grid points are ordered. Instead, for an unstructured mesh the grid points are in general not ordered and therefore topology information must be provided. In practice, a list of numbered grid points, in whatever order, must be provided first. Then the topology must be provided in the form of a list of elements/cells, where for each cell/element the corresponding grid points are given.

The format with which an unstructured mesh may be written/read may be complicated if different types of elements are to be included. Efforts have been made to develop general formats. A widely used one is the CGNS format, the library of which may be compiled and included in the solver. In the present work a very basic format has been implemented, which proved usable only for 2D problems. In order to cope with more complex 3D meshes the FFA format [2] implemented by the FOI (Swedish Defence Research Agency) has been included in the code. The FFA format is a very general format, which has been implemented in Fortran, and comes with utilities to convert the data to other major formats, such as CGNS.

Chapter 2

Solution of the flow equations

2.1 Flow equations

2.1.1 Modeling the flow: different level of approximation

The Navier-Stokes equations provide the most complete mathematical description of flows of aerodynamic interest.The equations state the conservation of mass, momentum and energy of the fluid. Unfortunately, the solution for cases of practical interest is still unfeasible due to the multi-scale nature of the turbulence that characterizes the flow. In fact, the required computational capabilities are still unavailable these days. Thus, in order to gain some insight in the behavior of the flow, different levels of approximation have been devised during the years.

A valuable approximation is obtained by averaging the Navier-Stokes equations in time, which gives the so-called Reynolds-Averaged Navier-Stokes (RANS) equations. The averaging gives rise to additional terms that depend on the averaged behavior of the turbulence. Turbulence models have been introduced for those terms, providing closure to the RANS equations. Usually, the turbulence models are cast in the form of algebraic relations, or in the more complex form of transport equations, which must be added to the original system of equations [135]. Nowadays the RANS equations are routinely used by the aircraft industry and they are considered reliable to address transonic high-reynolds number flows including regions affected by separation [61].

Neglecting the viscosity and thermal conductivity leads to the Euler equations, which form the most complete representation of an inviscid fluid. Compared to the RANS equations the complexity is reduced dramatically, not only in terms of the equations, but also in terms of the required computational mesh. In fact, the RANS equations usually require very fine meshes, especially in boundary layer regions, which are characterized by a huge increase of the mesh density approaching the wall. As the viscosity is neglected, the Euler equations can only predict the wave and the induced component of the drag. As such, their use may be limited because crucial information such as the efficiency and the range of an aircraft cannot be evaluated. Fortunately, the Euler equations can be efficiently coupled to Boundary Layer codes, which solve the boundary layer for attached flows [24]. Thus, for cruise conditions, the coupled approach can still provide useful information.

Adding further approximations to the Euler equations leads to the full potential flow equation first and then to the potential flow equation. Both have been mentioned in the Introduction. A detailed description of those equations can be found in [49]. In the present work the Euler equations are the model of choice. Only the inviscid solution is considered, without boundary layer coupling. Below the Euler equations in the integral form are described.

2.1.2 Euler equations in integral form

The Euler equations are a system of conservation laws [71, 70, 49, 134] for the conservative variables $\mathbf{u}$. The integral form of the equations, which must hold for any volume V contained in the domain Ω, reads

$$\frac{d}{dt}\int_V \mathbf{u}\, d\Omega + \oint_{\partial V} \mathbf{F}(\mathbf{u}) \cdot \mathbf{n}\, d\Gamma = \mathbf{0}\,, \tag{2.1}$$

where $\mathbf{F}(\mathbf{u})$ is the inviscid flux and $\mathbf{n}$ is the outward unit normal on the boundary ∂V of V. This equation states that the rate of change of the total amount of $\mathbf{u}$ contained in V, plus the net flux of $\mathbf{F}$ through the boundary ∂V, must be equal to zero. The conservative variables are defined as

$$\mathbf{u} = [\rho\,, \rho\mathbf{w}\,, \rho e_t]^T\ , \tag{2.2}$$

where ρ is the density, $\mathbf{w}$ is the velocity vector and e_t is the total specific energy. The flux vector is defined as

$$\mathbf{F}(\mathbf{u}) = \left[\rho\mathbf{w}^T\,, \rho\mathbf{w}\mathbf{w}^T + p\mathbf{I}\,, (p+\rho e_t)\mathbf{w}^T\right]^T\ , \tag{2.3}$$

where p is the static pressure. The latter must be linked to the conservative variables in order to provide closure of the system. The link is provided by the perfect gas equation, which reads

$$p = (\gamma - 1)\rho\left(e_t - \frac{|\mathbf{w}|^2}{2}\right)\ , \tag{2.4}$$

where γ is the ratio of specific heats, which for air is about 1.4.

The Euler equations may also be expressed in terms of primitive variables, which are defined as

$$\mathbf{v} = [\rho\, , \mathbf{w}\, , p]^T \ . \tag{2.5}$$

However, in this case the Euler equations are not in conservative form. The lack of conservation has negative implications for the numerical solution [71, 70]. Therefore the primitive form is rarely used in practice. Nevertheless, the primitive variables themselves are often used in the numerical solution of the Euler equations as working or intermediate variables. A link between $\mathbf{u}$ and $\mathbf{v}$ is provided by the transformation matrix $\partial\mathbf{v}/\partial\mathbf{u}$, which reads

$$\frac{\partial\mathbf{u}}{\partial\mathbf{v}} = \left[\begin{array}{ccc} 1 & \mathbf{0}^T & 0 \\ \mathbf{w} & \rho\mathbf{I} & \mathbf{0} \\ |\mathbf{w}|^2/2 & \rho\mathbf{w}^T & 1/(\gamma-1) \end{array} \right] , \tag{2.6}$$

and by its inverse, which reads

$$\frac{\partial\mathbf{v}}{\partial\mathbf{u}} = \left[\begin{array}{ccc} 1 & \mathbf{0}^T & 0 \\ -\mathbf{w}/\rho & \mathbf{I}/\rho & \mathbf{0} \\ (\gamma-1)|\mathbf{w}|^2/2 & -(\gamma-1)\mathbf{w}^T & (\gamma-1) \end{array} \right] . \tag{2.7}$$

The two matrices can be easily obtained by the variables definition and by taking into account Eq. (2.4).

Dimensionless variables

Dimensionless variables are used for the solution of the equations. They are obtained by means of dimensional analysis. The dimensionless primitive variables, i.e., density, velocity and pressure, may be written respectively as

$$\rho' = \frac{\rho}{\rho_\infty} , \qquad \mathbf{w}' = \frac{\mathbf{w}}{U_\infty} , \qquad p' = \frac{p}{\rho_\infty U_\infty^2} , \tag{2.8}$$

where ρ_∞ is the free-stream density and U_∞ is the magnitude of the free-stream velocity.

Other dimensionless quantities can be derived using the above variables. For instance, the total specific energy can be made dimensionless after dividing by U_∞^2:

$$e_t = c_v T + \frac{|\mathbf{w}|^2}{2} \qquad \rightarrow \qquad {e_t}' = \frac{e_t}{U_\infty^2} = \frac{c_v T}{U_\infty^2} + \frac{|\mathbf{w}|^2}{2U_\infty^2} \, .$$

c_v is the specific heat at constant volume. The above equation shows that the dimensionless temperature is $T' = c_v T / U_\infty^2$.

In the following, all the variables are assumed to be dimensionless. However, the prime is not used since it can be safely dropped without causing confusion.

2.2 Discretization of the Euler equations

2.2.1 Finite-volume discretization

Equation (2.1) may be discretized using a finite-volume formulation [71, 70, 49, 134]. The domain Ω must be partitioned entirely in a set of non-overlapping control volumes V_i and Eq. (2.1) must be satisfied in each of them. Consequently, a system of partial differential equations is obtained, which reads

$$V_i \frac{d\mathbf{u}_i}{dt} + \oint_{\partial V_i} \mathbf{F} \cdot \mathbf{n} \, d\Gamma = \mathbf{0}, \qquad (i = 1, N)\,, \tag{2.9}$$

where

$$V_i = \int_{V_i} d\Omega \qquad \text{and} \qquad \mathbf{u}_i = \frac{1}{V_i} \int_{V_i} \mathbf{u} \, d\Omega \tag{2.10}$$

are the control volume area and the cell average value of $\mathbf{u}$ in V_i, respectively. The flux integral in Eq. (2.9) can be split into a number of contributions N_i, which corresponds to the number of the nearest-neighbors of the ith control volume. For this purpose, the boundary of V_i has to be divided into an equal number N_i of portions ∂V_{ij}, i.e.,

$$\partial V_i = \sum_{j=1,N_i} \partial V_{ij}\,,$$

where ∂V_{ij} is the portion that separates the two adjacent control volumes V_i and V_j and is called the control-volume or cell interface ij. Therefore, for the generic cell i, Eq. (2.9) becomes

$$V_i \frac{d\mathbf{u}_i}{dt} + \sum_{j=1,N_i} \int_{\partial V_{ij}} \mathbf{F} \cdot \mathbf{n} \, d\Gamma + \int_{\partial V_i \cap \partial \Omega} \mathbf{F} \cdot \mathbf{n} \, d\Gamma = \mathbf{0}\,. \tag{2.11}$$

The last term in Eq. (2.11) accounts for the boundary conditions and is different from zero only if the boundary ∂V_i of cell i is a segment of the domain boundary $\partial\Omega$. Both flux integrals appearing in Eq. (2.11) can be evaluated by means of a mid-point quadrature. Thus, Eq. (2.11) can be written as

$$V_i \frac{d\mathbf{u}_i}{dt} + \sum_{j=1,N_i} \mathbf{F}_{ij} \cdot \int_{\partial V_{ij}} \mathbf{n} \, d\Gamma + \mathbf{F}_i \cdot \int_{\partial V_i \cap \partial \Omega} \mathbf{n} \, d\Gamma = \mathbf{0}\,, \tag{2.12}$$

where $\mathbf{F}_{ij}$ is the constant approximation of the flux along the control-volume interface ∂V_{ij}, and $\mathbf{F}_i$ is the constant approximation of the flux along the boundary $\partial V_i \cap \partial\Omega$. The two integrals appearing in Eq. (2.12) are of geometric nature since they contain only information about the geometry of the domain Ω. Two integrated normal vectors can be defined,

$$\mathbf{n}_{ij} = \int_{\partial V_{ij}} \mathbf{n} \, d\Gamma \qquad \text{and} \qquad \mathbf{n}_{\mathrm{B}i} = \int_{\partial V_i \cap \partial \Omega} \mathbf{n} \, d\Gamma\,. \tag{2.13}$$

One has that $\mathbf{n}_{ij} = -\mathbf{n}_{ji}$. Moreover, the following relations must be satisfied:

$$\sum_{j=1,N_i} \mathbf{n}_{ij} = \mathbf{0}\,, \quad i \in \mathcal{I} \quad \text{and} \quad \sum_{j=1,N_i} \mathbf{n}_{ij} = -\mathbf{n}_{Bi}\,, \quad i \in \mathcal{B}\,, \tag{2.14}$$

where $\mathcal{I}$ and $\mathcal{B}$ are the sets that contain the internal and the boundary nodes, respectively. Substituting the integrated normals into Eq. (2.12) gives

$$V_i \frac{d\mathbf{u}_i}{dt} + \sum_{j=1,N_i} \mathbf{F}_{ij} \cdot \mathbf{n}_{ij} + \mathbf{F}_i \cdot \mathbf{n}_{\mathrm{B}i} = \mathbf{0}\,. \tag{2.15}$$

In order to solve this equation, the evaluations of the internal flux $\mathbf{F}_{ij}$ and of the boundary flux $\mathbf{F}_i$ are needed. These evaluations are not straightforward since particular care is required to make sure that the hyperbolic nature of the equations is properly taken into account. Also, care is needed to ensure that spatial accuracy is achieved.

In practice, numerical fluxes are introduced in both cases: the internal flux is replaced by the numerical flux $\mathbf{\Phi}_{ij} \approx \mathbf{F}_{ij} \cdot \mathbf{n}_{ij}$ and the boundary flux by the numerical flux $\mathbf{\Phi}_i^{bc} \approx \mathbf{F}_i \cdot \mathbf{n}_{\mathrm{B}i}$. Substituting these numerical fluxes into Eq. (2.15) gives

$$V_i \frac{d\mathbf{u}_i}{dt} + \sum_{j=1,N_i} \mathbf{\Phi}_{ij} + \mathbf{\Phi}_i^{bc} = \mathbf{0}\,. \tag{2.16}$$

Several types of numerical fluxes are available to solve this equation. The ones preferred for this work are presented in the next section. The numerical fluxes can be collected into the residual, which is defined as

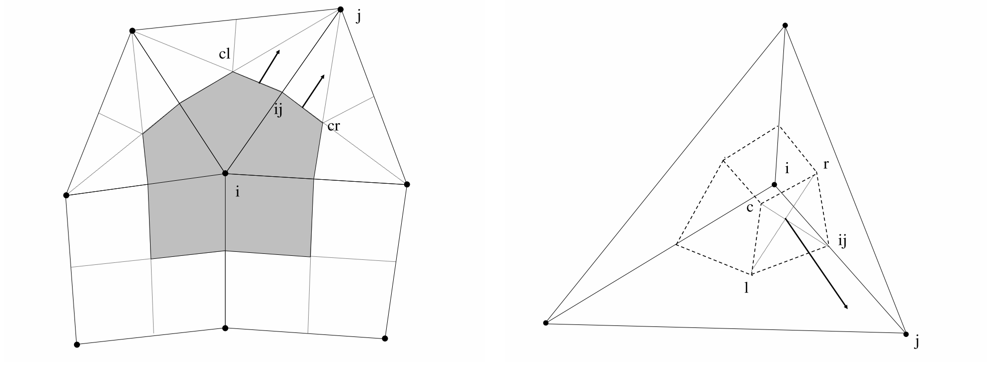
$$\mathbf{r}_i \equiv \sum_{j=1}^{N_i} \mathbf{\Phi}_{ij} + \mathbf{\Phi}_i^{bc}\,. \tag{2.17}$$

Once Eq. (2.16) has been driven to steady state, if one exists, the residual is expected to vanish. For this reason the residual may be used as a measure of convergence of the solution.

2.2.2 Median-dual mesh

Median-dual construction

Equation (2.16) can be discretized on an unstructured type of mesh, e.g., a Delaunay triangulation or a hybrid mesh made of different type of elements. In the present work, a node-centered approach is employed. The approach stores the unknowns at the nodes of the mesh, around which the control volumes are located. It is realized by means of a median-dual type of mesh [29], which has interesting features. For instance, it is mesh transparent [43], that is, after the median-dual mesh is generated, an edge-based data structure is all that is needed to solve the equations both in 2D and 3D and for all types of elements. Thus, the elements information can be discarded. Moreover, when the equations are discretized on the median-dual mesh, the resulting discretization



(a) (b)

Figure 2.1: *Median-dual construction for (a) two-dimensional and (b) three-dimensional elements.*

can be shown to be equivalent to a finite-element Galerkin discretization over the same grid [113, 9].

The median-dual mesh is obtained with a construction process that starts from the original mesh. The process is briefly described in the following.

2D case: Consider a two-dimensional element, i.e., a triangle or a quad. The element is cut into a number of regions equal to the number of its nodes, e.g., three for a triangle and four for a quad. The regions are obtained by connecting the barycenter of the element to its edge mid-points. If the process is performed on all the elements in the mesh, at the end each node in the mesh is surrounded by a number of these regions, the union of which gives the control volume for the node. An example of this construction is depicted in Fig. 2.1a, where the shaded region indicates the control volume of the generic node i. An unstructured mesh around the RAE2822 airfoil, which has been generated by the Delaunay triangulation (a commercial software package has been used), is shown in Fig. 2.2a. Its corresponding dual mesh is shown in Fig. 2.2b. A mesh of triangles around the NACA0012 airfoil, which has been generated by the splitting of a structured mesh of quads, and its dual, are shown in Fig. 2.3a and Fig. 2.3b, respectively.

3D case: More types of elements are available in this case, e.g., tetrahedra, pyramids, prisms and hexahedra. The faces of such elements are either triangles or quads. The construction process described above may be repeated here for each face of the element. In addition, lines can be drawn that connect the center of each face to the barycenter of the element. The latter lines, together with the lines created on the faces, define a number of surfaces internal to the element. The surfaces cut the element itself in a number of regions equal to the number of its nodes. Again, after visiting all the elements of the mesh, one can construct the control volumes by performing the union of all the corresponding regions. An example of this construction is depicted in Fig. 2.1b. There,

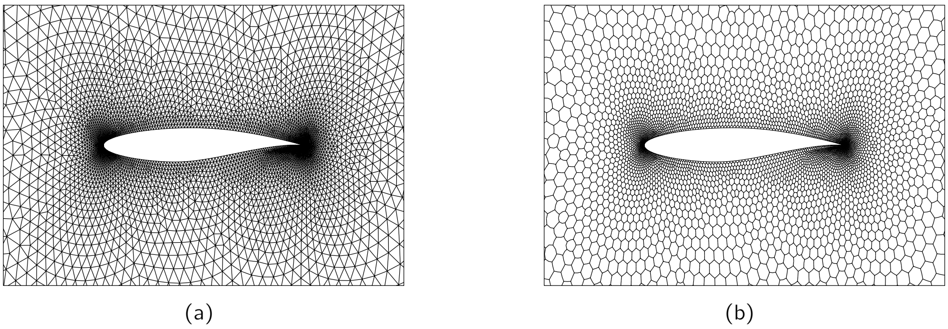


Figure 2.2: *(a) Unstructured mesh and (b) median-dual mesh around the RAE2822 airfoil.*

the region of the cell that belongs to the control volume of the generic node i is shown with dashed lines.

Metrics computation

Metrics is the term used to indicate the geometric quantities that appear in the discretized Euler equations, i.e., the control volumes V_i and the integrated normals $\mathbf{n}_{ij}$ and $\mathbf{n}_{\mathrm{B}i}$ of Eq. (2.13). The control volumes are computed by taking into account that, during the construction of the median-dual, each cell is always divided into a number of regions of equal volumes.

The integrated normals in 2D are computed as

$$\mathbf{n}_{ij} = (\mathbf{x}_{\mathrm{cl}} - \mathbf{x}_{ij}) \times \hat{\mathbf{z}} - (\mathbf{x}_{\mathrm{cr}} - \mathbf{x}_{ij}) \times \hat{\mathbf{z}}\,, \tag{2.18}$$

where $\hat{\mathbf{z}} = (0,0,1)^T$ is the unit direction, $\mathbf{x}_{ij}$ is the edge mid-point and $\mathbf{x}_{\mathrm{cl}}$ and $\mathbf{x}_{\mathrm{cr}}$ are the barycenter of the left and of the right cells, respectively. The contribution of the two terms in the above equation is depicted in Fig. 2.1a. In the case of a boundary edge, the second term on the right-hand side is absent since there is no cell on the right of the edge. In the case of boundary nodes, the integrated boundary normal of Eq. (2.13) is computed as

$$\mathbf{n}_{Bi} = \frac{(\mathbf{x}_{\mathrm{r}} - \mathbf{x}_i)}{2} \times \hat{\mathbf{z}} - \frac{(\mathbf{x}_{\mathrm{l}} - \mathbf{x}_i)}{2} \times \hat{\mathbf{z}}\,, \tag{2.19}$$

where $\mathbf{x}_{\mathrm{r}}$ and $\mathbf{x}_{\mathrm{l}}$ are the nodes that lie respectively at the right and at the left of the boundary node i.

In 3D the integrated normals are computed as

$$\mathbf{n}_{ij} = \sum_{k \in \mathcal{E}_{ij}} \mathrm{sign}_{ij} (\mathbf{x}_{\mathrm{r}k} - \mathbf{x}_{\mathrm{l}k}) \times (\mathbf{x}_{\mathrm{c}k} - \mathbf{x}_{ij})\,, \tag{2.20}$$

where: $\mathcal{E}_{ij}$ is the set that, for a given edge ij, contains the elements that share the edge itself; sign_{ij} defines the orientation of the edge; $\mathbf{x}_{\mathrm{ck}}$ is the barycenter of the kth element

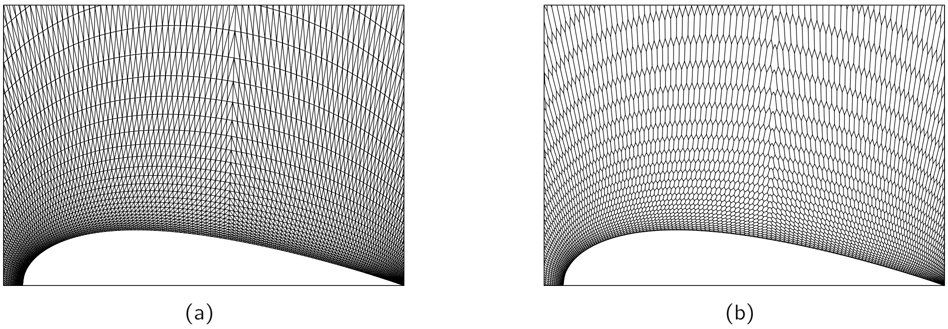


Figure 2.3: *(a) Mesh of triangles and (b) median-dual mesh around the NACA0012 airfoil. The mesh is obtained by splitting the element of a structured mesh.*

in $\mathcal{E}_{ij}$; $\mathbf{x}_{\mathrm{lk}}$ and $\mathbf{x}_{\mathrm{rk}}$ are the center of the left and of the right face of the element k in $\mathcal{E}_{ij}$, respectively. Left and right is intended with respect to the orientation of the edge ij. The integrated boundary normals in 3D are computed as

$$\mathbf{n}_{B\,i} = \sum_{k \in \mathcal{E}_{B\,i}} \frac{\mathbf{S}_k}{n_k}\,, \tag{2.21}$$

where $\mathcal{E}_{B\,i}$ is the set that contains all the boundary elements that share the node i, $\mathbf{S}_k$ is the area of the kth element and n_k is the number of nodes of the kth element.

Figure 2.1b shows the contribution of the kth element to the integrated normal $\mathbf{n}_{ij}$. The other contributions are obtained by visiting all the elements that have the edge ij in common.

Data structure and implementation of conservation laws

All the edges that are present in the mesh are stored in a list during the preprocessing of the mesh. For each edge ij, the list stores the left node i and the right node j of the edge. The left and the right directions are defined by the condition $i < j$, which gives a way to orientate the edge. The list constitutes the data structure that is used throughout all the code. The use of the edge-based data structure makes the implementation of conservation laws very efficient. For instance, consider the residual in Eq. (2.17). Conservation implies that $\mathbf{\Phi}_{ij} = -\mathbf{\Phi}_{ji}$. Therefore, the residual on each node can be assembled by a loop on the edges, where the control volume interfaces are located. The flux computed on each edge is accumulated on the left and on the right of the edge, with opposite signs. The pseudo-code for the residual assembly can be written as

$$\begin{aligned} \mathbf{r}_i &= \mathbf{r}_i + \mathbf{\Phi}_{ij}\,, \\ \mathbf{r}_j &= \mathbf{r}_j - \mathbf{\Phi}_{ij}\,, \qquad (ij = 1, E)\,, \end{aligned} \tag{2.22}$$

where E is the number of edges. Note that the pseudo-code notation will be used throughout the present and the next chapter. In the next chapter, which introduces the sensitivity analysis, the pseudo-code will be very useful for the derivation of several assemblies.

The residual assembly in Eq. (2.22) must be completed by adding the boundary contribution. A possibility is to perform an additional loop on the boundary nodes,

$$\mathbf{r}_i = \mathbf{r}_i + \mathbf{\Phi}_i^{bc} \,, \qquad (i = 1, N_B) \,, \tag{2.23}$$

where N_B is the number of nodes that are lying on the boundary. In the following, in order to simplify the exposition, the loop on the boundary nodes will be neglected when dealing with similar type of assemblies.

Another list, required for the solution of the sparse linear system, is also generated during the construction phase. This list stores the set $\mathcal{N}_i$ for each node. This set, which will be referred to as the stencil, contains the node i and all its distance-one neighbors j. For instance, in the case of Fig. 2.1a, the set would contain the eight nodes marked by the black dots. The nodes in the set $\mathcal{N}_i$ are increasingly ordered,

$$\mathcal{N}_i = [\mathcal{L}_i, i, \mathcal{U}_i] \quad \text{where} \quad \forall j \in \mathcal{L}_i : j < i \quad \text{and} \quad \forall j \in \mathcal{U}_i : j > i \,. \tag{2.24}$$

This list is only necessary to deal with the solution process implemented here and described in Section 2.4. If another solution process would be used, e.g., external libraries for sparse linear systems, this list could be unnecessary. In fact, external libraries usually come with their own data structure.

2.3 Space discretization of the flow equations

As already mentioned in the previous section, particular care is required for the evaluation of the numerical fluxes in the discretized equations. One has to take into account the hyperbolic nature of the Euler equations and the need to achieve second-order spatial accuracy.

Numerical fluxes that are suitable for wave propagation phenomena are used, both for the internal fluxes and for the boundary fluxes. For the boundary, flux vector splitting is employed, which is capable of distinguishing between in- and out-going waves. For the internal flux, an approximate Riemann solver is employed. The latter solver, given the discontinuous left and right states, approximates the flux that would be obtained by solving the well-known Riemann problem of gas dynamics.

Although the Riemann solver is suitable for wave phenomena, it yields only first-order spatial accuracy. A reconstruction scheme is used to improve the accuracy. The scheme performs a linear extrapolation of the states across the interfaces and achieves second-order spatial accuracy. Spurious oscillations are taken care of by means of a limiting procedure, which is suitable for unstructured meshes.

2.3.1 Numerical fluxes

Roe's approximate Riemann solver

The numerical flux in Eq. (2.17) is evaluated using Roe's approximate Riemann solver [108]. The latter is capable of computing the flux across the interface between the two nodes i and j, in spite of the discontinuity between the states. The problem is solved in a one-dimensional fashion, i.e., the numerical flux approximates the Euler flux projected along the integrated normal, $\mathbf{F}_{ij} \cdot \mathbf{n}_{ij}$. The flux is based upon the mean-value theorem. It reads

$$\boldsymbol{\Phi}_{ij} = \boldsymbol{\Phi}(\mathbf{u}_i, \mathbf{u}_j, \mathbf{n}_{ij}) = \frac{\mathbf{F}(\mathbf{u}_i) + \mathbf{F}(\mathbf{u}_j)}{2} \cdot \mathbf{n}_{ij} - \frac{1}{2} |\mathbf{A}(\bar{\mathbf{u}}_{ij}, \mathbf{n}_{ij})| (\mathbf{u}_j - \mathbf{u}_i) \,, \tag{2.25}$$

where $\mathbf{A}(\mathbf{u}, \mathbf{n}) = d(\mathbf{F} \cdot \mathbf{n})/d\,\mathbf{u}$ is the Jacobian of the flux vector projected along the normal. The Jacobian must be evaluated with the variables $\bar{\mathbf{u}}_{ij}$ that satisfy

$$[\mathbf{F}(\mathbf{u}_i) - \mathbf{F}(\mathbf{u}_j)] \cdot \mathbf{n}_{ij} = \mathbf{A}(\bar{\mathbf{u}}_{ij}, \mathbf{n}_{ij})(\mathbf{u}_j - \mathbf{u}_i) \,.$$

Variables that satisfy the above equation are the Roe averages:

$$\rho_{ij} = \sqrt{\rho_i \rho_j} \,, \quad \mathbf{w}_{ij} = \frac{\sqrt{\rho_i}\,\mathbf{w}_i + \sqrt{\rho_j}\,\mathbf{w}_j}{\sqrt{\rho_i} + \sqrt{\rho_j}} \quad \text{and} \quad h_{ij} = \frac{\sqrt{\rho_i}\,h_i + \sqrt{\rho_j}\,h_j}{\sqrt{\rho_i} + \sqrt{\rho_j}} \,,$$

where $h = e_t + p/\rho$ is the specific total enthalpy. The absolute value of the flux Jacobian $|\mathbf{A}(\bar{\mathbf{u}}_{ij}, \mathbf{n}_{ij})|$ is computed as

$$|\mathbf{A}| = \mathbf{X} |\boldsymbol{\Lambda}| \mathbf{X}^{-1} \,, \tag{2.26}$$

where $\mathbf{X}$ and $\boldsymbol{\Lambda}$ are the eigenvectors and the eigenvalues of $\mathbf{A}$, respectively. The eigenvectors, which have lengthy expressions, can be found in [50]. The eigenvalues read

$$\lambda_{1,2,3} = \mathbf{w} \cdot \mathbf{n} \,, \quad \lambda_4 = \mathbf{w} \cdot \mathbf{n} + a|\mathbf{n}| \quad \text{and} \quad \lambda_5 = \mathbf{w} \cdot \mathbf{n} - a|\mathbf{n}| \,,$$

where $a = \sqrt{\gamma p/\rho}$ is the speed of sound.

Note that the Roe flux, see Eq. (2.25), is constructed as a central term plus a matrix dissipation term. And if the eigenvalues happen to be zero, the dissipation vanishes and non-physical solutions may appear. It is a recommended practice to use a so-called entropy fix [42], which modifies the eigenvalues to ensure that they are never zero. For the computations performed in the present work, the entropy fix was found to be unnecessary, probably because unstructured grids tend to be not aligned with the flow and therefore the eigenvalues are not very likely to be zero.

Flux-vector splitting

The far-field boundary fluxes are computed using flux vector splitting [119], which allows to distinguish between in-going and out-going information. The numerical flux for a far-field boundary point is evaluated as

$$\boldsymbol{\Phi}_i^{bc} = \boldsymbol{\Phi}_i^{\infty} = \mathbf{A}^{+}(\mathbf{u}_i, \mathbf{n}_{Bi})\,\mathbf{u}_i + \mathbf{A}^{-}(\mathbf{u}_i, \mathbf{n}_{Bi})\,\mathbf{u}_\infty \,, \tag{2.27}$$

where $\mathbf{u}_\infty$ is the vector of free-stream conservative variables and where $\mathbf{A}^+$ and $\mathbf{A}^-$ are the plus and the minus splittings of the flux Jacobian. The latter are defined as

$$\mathbf{A}^+ = \mathbf{X}\,\frac{\mathbf{\Lambda} + |\mathbf{\Lambda}|}{2}\mathbf{X}^{-1}\,, \qquad \mathbf{A}^- = \mathbf{X}\,\frac{\mathbf{\Lambda} - |\mathbf{\Lambda}|}{2}\mathbf{X}^{-1}\,. \tag{2.28}$$

The splitting of Eq. (2.27) takes advantage of the remarkable property of the flux vector of being a homogeneous function of order one of $\mathbf{u}$, i.e.,

$$\mathbf{F}(\mathbf{u}) \cdot \mathbf{n} = \mathbf{A}(\mathbf{u}, \mathbf{n})\mathbf{u}\,.$$

Inviscid wall and symmetry

In the case of a symmetry boundary or an inviscid wall boundary, the velocity component normal to the boundary is zero, $\mathbf{w}_i \cdot \mathbf{n}_{B\,i} = 0$. A boundary condition may be imposed by setting the normal velocity to zero in the normal projection of the Euler flux of Eq. (2.3), i.e.,

$$\mathbf{\Phi}_i^{bc} = \mathbf{\Phi}_i^{w} = [0\,, p\,\mathbf{n}_{B\,i}\,, 0]^T\ . \tag{2.29}$$

This way of imposing the boundary condition on the wall is referred to as weak because the condition is imposed on the flux rather than on the variables themselves. In contrast, a strong enforcement of the boundary condition assigns the boundary velocity without considering it as unknown in the solution process.

2.3.2 MUSCL-like reconstruction scheme

Second-order spatial accuracy in region of smooth flow, i.e., flow with no discontinuities, is achieved by means of a MUSCL-like scheme [9]. The scheme is an unstructured-grid version of the original MUSCL scheme [124]. In essence, the method evaluates the numerical flux in Eq. (2.25) with reconstructed variables, i.e.,

$$\mathbf{\Phi}_{ij} = \mathbf{\Phi}(\hat{\mathbf{u}}_i, \hat{\mathbf{u}}_j, \mathbf{n}_{ij})\,, \tag{2.30}$$

where the hat on the conservative variables stands for reconstruction. However, the primitive variables $\mathbf{v}$ are reconstructed rather than the conservative variables $\mathbf{u}$. For a generic primitive variable v, the reconstruction across the mid-point of the edge ij reads

$$\hat{v}_i = v_i + \frac{\sigma_i}{2}\nabla v_i^T \Delta\mathbf{x}_{ij} \quad \text{and} \quad \hat{v}_j = v_j - \frac{\sigma_j}{2}\nabla v_j^T \Delta\mathbf{x}_{ij}\,, \tag{2.31}$$

where σ is the slope limiter, ∇v is the gradient of the primitive variable at the node and $\Delta\mathbf{x}_{ij}$ is the distance between i and j. Note that for median-dual meshes the interface between the two control volumes of i and j is located at the edge mid-point. For the latter reason the distance $\Delta\mathbf{x}_{ij}$ is halved in the above equation.

The reconstruction is performed just before the evaluation of the numerical flux. Therefore, the slope limiter and the gradient must be precomputed and stored in order to be used. In fact, on-the-fly gradient computation cannot be performed using the edge-based data structure. Instead, it could be performed using a data structure that contains the stencil of each node, like the one in Eq. (2.24). However, because of the larger number of operations required using the latter data structure, the computation would be inefficient (for the same reason the fluxes of an unstructured solver are computed looping on the interfaces of the cells rather than on the cells). Different is the situation for a structured solver for which nearest-neighbors are directly available and therefore it is easy and convenient to compute the gradient on-the-fly.

In the present work two types of gradients are available: the Green-Gauss and the Least-squares gradient. Below a description is given.

Green-Gauss gradient

The Green-Gauss gradient is based upon the Green formula

$$\int_{\Omega} \nabla v \, d\Omega = \oint_{\partial\Omega} v\mathbf{n} \, d\Gamma \, ,$$

which can be discretized [9] as

$$\nabla v_i = \frac{1}{2V_i} \sum_{j=1}^{N_i} (v_i + v_j)\, \mathbf{n}_{ij} + v_i \frac{\mathbf{n}_{Bi}}{V_i} \, . \tag{2.32}$$

The second term on the right-hand side is only present when the node i lies on the boundary.

The above discretization of the Green-Gauss gradient can reconstruct linear polynomials exactly only on triangles (in 2D) and tetrahedra (in 3D) that are located away from the boundary. An error is present on the boundary, which could be corrected by introducing in the above equation the contribution of the nearest-neighbors of the node i along the boundary [11]. The correction would make the Green-Gauss gradient equivalent to the P1 (linear triangles in 2D and linear tetrahedra in 3D) gradient used in the finite-element method. Still, an error would always be present for all other types of elements.

Numerical evidence shows that the error discussed above does not affect the accuracy of the reconstruction in an appreciable way. The Green-Gauss gradient can be computed by a loop on the edges of the mesh.

Least-squares gradient

The Least-squares gradient [9] reconstructs linear polynomials exactly on every type of element [10, 9]. By assuming a linear variation of the function v, one can write a number of equations for the node i equal to the number of its nearest neighbors N_i, i.e.,

$$\nabla v_i \cdot (\mathbf{x}_j - \mathbf{x}_i) = v_i - v_j \, , \qquad (j = 1, N_i) \, .$$

The above equation represents a problem of the type $[\mathbf{L}_1\ \mathbf{L}_2\ \mathbf{L}_3]\,\nabla v = \mathbf{F}$, which has solution

$$\nabla v = \begin{bmatrix} \mathbf{V}_1 \\ \mathbf{V}_2 \\ \mathbf{V}_3 \end{bmatrix} \mathbf{F}\,, \tag{2.33}$$

with the vectors $\mathbf{V}_1$, $\mathbf{V}_2$ and $\mathbf{V}_3$ that satisfy the relation

$$\begin{bmatrix} \mathbf{V}_1 \\ \mathbf{V}_2 \\ \mathbf{V}_3 \end{bmatrix} [\mathbf{L}_1\ \mathbf{L}_2\ \mathbf{L}_3] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The vectors are obtained via a Gramm-Schmidt orthonormalization process [9], which computes $\mathbf{V}_k = \mathbf{U}_k/(\mathbf{L}_k \cdot \mathbf{U}_k)$, where

$$\begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \\ \mathbf{U}_3 \end{bmatrix} = \begin{bmatrix} (l_{33}l_{22} - l_{23}l_{23}) & (l_{23}l_{13} - l_{33}l_{12}) & (l_{23}l_{12} - l_{22}l_{13}) \\ (l_{33}l_{11} - l_{13}l_{13}) & (l_{13}l_{23} - l_{33}l_{12}) & (l_{13}l_{12} - l_{11}l_{23}) \\ (l_{11}l_{22} - l_{12}l_{12}) & (l_{12}l_{23} - l_{22}l_{13}) & (l_{12}l_{13} - l_{11}l_{23}) \end{bmatrix} \begin{bmatrix} \mathbf{L}_1 \\ \mathbf{L}_2 \\ \mathbf{L}_3 \end{bmatrix}$$

and $l_{km} = \mathbf{L}_k \cdot \mathbf{L}_m$. The process simplifies considerably for the 2D case, for which $\mathbf{V}_3$ and $\mathbf{L}_3$ are equal to zero in the above equations and

$$\mathbf{V}_1 = \frac{l_{22}\mathbf{L}_1 - l_{12}\mathbf{L}_2}{l_{11}l_{22} - {l_{12}}^2}\,, \quad \mathbf{V}_2 = \frac{l_{11}\mathbf{L}_2 - l_{12}\mathbf{L}_1}{l_{11}l_{22} - {l_{12}}^2}\,.$$

The least-squares gradient is more expensive to compute than the Green-Gauss gradient. It is computed by looping on the edges. Nothing is stored and all the quantities defined above are computed on-the-fly at each gradient computation.

Slope limiter

The slope limiter is that of Venkatakrishnan [131], which is a modification of the multi-dimensional limiter introduced by Barth [10]. The latter limiter, although capable of enforcing monotonicity in a multi-dimensional fashion, is not differentiable. It is well known that the lack of differentiability causes a noisy convergence behavior that prevents the solution to converge completely. Specifically, after the solution is almost converged, the residual stalls to a certain value and keeps oscillating around that value. The same behavior is observed for the lift, the drag and for the pitching moment coefficients.

In the following, $\Delta v_{ij} = \nabla {v_i}^T \Delta \mathbf{x}_{ij}/2$ is the unlimited differential of v_i on the edge ij and ${v_i}^{\max}$ and ${v_i}^{\min}$ are the extrema of v on the stencil $\mathcal{N}_i$. The purpose of the limiting procedure is to ensure monotonicity, i.e.,

$${v_i}^{\min} \leq v_i \leq {v_i}^{\max}\,, \tag{2.34}$$

which means that the reconstructed solution on the node i never exceeds the extrema in its stencil. If $\Delta {v_i}^{\max} = {v_i}^{\max} - v_i$ and $\Delta {v_i}^{\min} = {v_i}^{\min} - v_i$, both limiters may be defined as

$$\sigma_i = \min_{j=1,N_i} \begin{cases} Lim(\Delta v_i{}^{\max}, \Delta v_{ij})\,, & \Delta v_{ij} > 0\,, \\ \\ Lim(\Delta v_i{}^{\min}, \Delta v_{ij})\,, & \Delta v_{ij} < 0\,. \end{cases} \tag{2.35}$$

The difference between the two limiters lies in the function Lim. The discontinuous limiting function used by Barth is

$$Lim(a,b) = \min\left(1, \frac{a}{b}\right)\,, \tag{2.36}$$

whereas the smooth function introduced by Venkatakrishnan is

$$Lim(a,b) = \frac{a^2 + 2ab + \varepsilon^2}{a^2 + ab + 2b^2 + \varepsilon^2}\,, \tag{2.37}$$

where ε is a threshold. The latter may be tuned in order to compromise between convergence and limiting. Usually, the price to pay for obtaining convergence is the presence of small over/under shoots in the solution, i.e., the monotonicity condition in Eq. (2.34) is not enforced strictly. An inappropriate too large value of the threshold may eliminate the oscillations, but also act in smooth regions, where the limiter is not required. It is recommended [131] to chose $\varepsilon = k\Delta\bar{x}$, where k is a constant, $k \approx 0.1 - 0.3$, and $\Delta\bar{x}$ is a characteristic length chosen globally. A local value for each vertex, Δx, may also be used.

The multi-dimensional limiter in Eq. (2.35) cannot be computed during the assembly of the residual, at least not if one uses only the edge-based data structure. In fact, loops over the edges are necessary to compute first $v_i{}^{\max}$ and $v_i{}^{\min}$, and then to compute the limiter in Eq. (2.35). Therefore, the limiter is precomputed, stored and used to assembly the residual in a later phase. Compared to one-dimensional limiters, which can be computed during the assembly of the residual, this limiter is clearly more expensive. A method to adapt one-dimensional limiters to the unstructured median-dual discretization is described in [40].

2.4 Implicit steady-state solution

A steady-state solution of the flow equations is obtained by an implicit pseudo-time stepping scheme. Due to the second-order spatial accuracy and to the mixed elliptic-hyperbolic type of the equations, the discretization generates a linear system that is poorly diagonally dominant. According to the defect-correction method, diagonal dominance is improved by replacing the exact Jacobian of the discretization with a low-order one. For large time steps the scheme becomes an inexact Newton method, which shows a linear convergence rate.

At each time step the system of linear equations that arises from the discretization is solved iteratively to the required level of accuracy. The iterative solution uses a symmetric Gauss-Seidel (SGS) preconditioner, which does not require any preparatory work. Moreover, the solution method features the possibility to choose between the storage of the preconditioning matrix and the completely matrix-free preconditioning.

2.4.1 Defect correction in pseudo-time

Equation (2.16) may be written in compact form for all the nodes in the mesh as

$$\mathbf{D}\frac{d\,\mathbf{U}}{d\,t}+\mathbf{R}=\mathbf{0}\,, \tag{2.38}$$

where $\mathbf{U}=[\mathbf{u}_1,\mathbf{u}_2,..,\mathbf{u}_N]^T$ is the conservative variables vector and $\mathbf{R}=[\mathbf{r}_1,\mathbf{r}_2,..,\mathbf{r}_N]^T$ is the residual vector. The diagonal matrix $\mathbf{D}$ contains the control volumes V_i.

Theoretically, a steady state solution should not require the time derivative in the above equation. It should be possible to solve $\mathbf{R}=\mathbf{0}$ directly, e.g., using a Newton scheme. However, since the problem is non-linear, a suitable starting solution should be available in order to converge to the proper solution. Methods exist to find such a solution [64]. In the present work, as it is standard practice, a simpler way to proceed has been adopted: the pseudo-time derivative is included in the equation and the flowfield is initialized to the free-stream. In fact, numerical evidence shows that marching in time from this initial solution yields convergence to the steady state solution.

An implicit pseudo-time stepping scheme for the above equation reads

$$\mathbf{D}_t(\mathbf{U}^{n+1}-\mathbf{U}^n)+\mathbf{R}(\mathbf{U}^{n+1})=\mathbf{0}\,, \tag{2.39}$$

where the time derivative is discretized using a forward approximation and $\mathbf{D}_t$ is a diagonal matrix that contains the cell volumes divided by the local time steps, $V_i/\Delta t_i$. The residual term in Eq. (2.39) can be expanded linearly,

$$\mathbf{R}(\mathbf{U}^{n+1})\approx\mathbf{R}(\mathbf{U}^n)+\frac{\partial\mathbf{R}}{\partial\mathbf{U}}^n(\mathbf{U}^{n+1}-\mathbf{U}^n)\,.$$

Substituting the expansion in Eq. (2.39), one obtains the backward Euler scheme, which can be written as

$$\left(\mathbf{D}_t+\frac{\partial\mathbf{R}}{\partial\mathbf{U}}\right)^n(\mathbf{U}^{n+1}-\mathbf{U}^n)=-\mathbf{R}^n\,. \tag{2.40}$$

For infinitely large time steps, the time derivative vanishes and Eq. (2.40) becomes Newton's method. As already mentioned, the Jacobian $\partial\mathbf{R}/\partial\mathbf{U}$ is poorly diagonally dominant because of the mixed elliptic-hyperbolic type of the equations and the second order spatial discretization (see the next chapter for more details). As a consequence, the iterative solution of the linear system of equations arising at each time step is hard to obtain.

In order to deal with a more diagonally dominant Jacobian, the defect-correction method [66] can be employed. The method works by replacing the residual $\mathbf{R}(\mathbf{U})$ in Eq. (2.39) with a low order residual $\widetilde{\mathbf{R}}(\mathbf{U})$ plus a defect $\Delta\mathbf{R}(\mathbf{U})$, where the latter is defined as

$$\Delta\mathbf{R}(\mathbf{U})=\mathbf{R}(\mathbf{U})-\widetilde{\mathbf{R}}(\mathbf{U})\,.$$

The defect-correction method treats the low-order residual implicitly and the defect explicitly. Thus, applying the defect-correction to Eq. (2.39), one has that

$$\mathbf{D}_t(\mathbf{U}^{n+1}-\mathbf{U}^n)+\widetilde{\mathbf{R}}(\mathbf{U}^{n+1})=-\Delta\mathbf{R}(\mathbf{U}^n)=\widetilde{\mathbf{R}}(\mathbf{U}^n)-\mathbf{R}(\mathbf{U}^n)\,. \tag{2.41}$$

Once the above problem is converged, the solution of the original problem is obtained. In fact, if $\mathbf{U}^{n+1} \approx \mathbf{U}^n$, the low order residual terms vanish and $\mathbf{R}(\mathbf{U}^n) = \mathbf{0}$. In practice, the method uses the low order residual to drive the defect to zero.

In the present work, $\widetilde{\mathbf{R}}$ is the residual of a first-order accurate and approximate discretization. Taking a first order expansion of the residual, after substitution in the above equation, one obtains

$$\left(\mathbf{D}_t + \frac{\partial \widetilde{\mathbf{R}}}{\partial \mathbf{U}}\right)^n (\mathbf{U}^{n+1} - \mathbf{U}^n) = -\mathbf{R}^n \, . \tag{2.42}$$

At first glance the above equation looks identical to Eq. (2.40). However, the Jacobian $\partial\widetilde{\mathbf{R}}/\partial\mathbf{U}$ that appears in the above equation is first-order accurate and thus diagonally dominant. Therefore, the solution of the linear system is relatively easy and robust.

The price to pay for using the first order Jacobian in Eq. (2.42) is the loss of quadratic convergence rate, which could be potentially achieved with the exact method of Eq. (2.40). The scheme given in Eq. (2.42) coincides with a backward Euler scheme (Newton for infinitely large time steps) that uses an approximate Jacobian [9, 131, 82].

Local time stepping

During the solution of the system in Eq. (2.42) to steady state, the local time steps Δt_i are increased at each time step according to an update of the Courant-Friedrichs-Lewy (CFL) number [26]. The local time steps are defined as

$$\Delta t_i = \frac{\text{CFL } L_i}{\sqrt{|\mathbf{w}_i|^2 + a_i^2}} \, , \tag{2.43}$$

where a is the speed of sound and L_i is a characteristic length associated with the node i. This length is defined as the length of the smallest edge connected to i. The CFL is updated as

$$\text{CFL}^n = \max\left[\beta \, \frac{L_2(\mathbf{R}^{n-2})}{L_2(\mathbf{R}^{n-1})}, 1\right] \text{CFL}^{n-1} \, , \tag{2.44}$$

where $L_2(\mathbf{R})$ is a discrete norm of the residual vector and β is a suitable parameter. The above formula may allow the CFL number to grow indefinitely. Since for certain types of flow a very large CFL number may cause the solution to break down, it may be best to limit the CFL number to a maximum value.

Initialization

An initial flow solution must be provided in order to initialize the iterative process. The whole field is initialized to the uniform free-stream flow. The latter is fully defined by the density ρ_∞, pressure p_∞, and velocity $\mathbf{w}_\infty$, the magnitude of which is U_∞. Recall that dimensionless variables are used. Therefore, indicating with $\mathbf{v}_\infty$ the dimensionless free-stream vector, one has that

$$\mathbf{v}_\infty = \left[1 \, , \hat{\mathbf{w}}_\infty \, , (\gamma M_\infty^2)^{-1}\right]^T \, . \tag{2.45}$$

where the second element is the velocity direction $\hat{\mathbf{w}}_\infty = \mathbf{w}_\infty / U_\infty$. For instance, in 2D the vector may be written as $\hat{\mathbf{w}}_\infty = [cos(\alpha), sin(\alpha)]^T$, where α is the angle of attack. The third element is obtained as follows:

$$\frac{p_\infty}{\rho_\infty U_\infty^2} = \frac{p_\infty}{\rho_\infty U_\infty^2}\frac{\gamma \rho_\infty}{\gamma \rho_\infty} = \frac{a_\infty^2}{\gamma U_\infty^2} = \frac{1}{\gamma M_\infty^2},$$

where M_∞ is the free-stream Mach number.

As can be seen from Eq. (2.45), the dimensionless nature of the equations, and the use of the simple ideal gas model, requires one to specify the flow conditions by two inputs only: the angle of attack α and the free-stream Mach number M_∞.

2.4.2 Iterative solution of the linear system of equations

At each time step, Eq. (2.42) implies the solution of a linear system $\mathbf{Mz} = \mathbf{b}$ where

$$\mathbf{M} = \left(\mathbf{D}_t + \frac{\partial \widetilde{\mathbf{R}}}{\partial \mathbf{U}}\right)^n, \qquad \mathbf{z} = \mathbf{U}^{n+1} - \mathbf{U}^n \qquad \text{and} \qquad \mathbf{b} = -\mathbf{R}^n. \tag{2.46}$$

A simple iterative procedure is employed that computes corrections of the type:

$$\mathbf{z}^{k+1} = \mathbf{z}^k + \Delta\mathbf{z}, \quad \text{where} \quad \Delta\mathbf{z} = \mathbf{P}_\mathrm{M}^{-1}\mathbf{R}_\mathrm{L}^k \quad \text{and} \quad \mathbf{R}_\mathrm{L}^k = \mathbf{b} - \mathbf{Mz}^k. \tag{2.47}$$

$\mathbf{R}_\mathrm{L}$ is the residual of the linear system and $\mathbf{P}_\mathrm{M}$ is the preconditioner computed from $\mathbf{M}$. Let the matrix $\mathbf{M}$ be written as

$$\mathbf{M} = \mathbf{L}_\mathrm{M} + \mathbf{D}_\mathrm{M} + \mathbf{U}_\mathrm{M},$$

where $\mathbf{L}_\mathrm{M}$, $\mathbf{U}_\mathrm{M}$ and $\mathbf{D}_\mathrm{M}$ are the strictly lower, the strictly upper and the diagonal parts of the matrix, respectively. $\mathbf{P}_\mathrm{M}$ is a symmetric Gauss-Seidel preconditioner. Therefore, using the above matrices, it can be written as

$$\mathbf{P}_\mathrm{M} = (\mathbf{D}_\mathrm{M} + \mathbf{L}_\mathrm{M})\,\mathbf{D}_\mathrm{M}^{-1}\,(\mathbf{D}_\mathrm{M} + \mathbf{U}_\mathrm{M}). \tag{2.48}$$

The procedure in Eq. (2.47) is iterated until the discrete norm of the linear residual is less than a fraction of that of the non-linear residual, i.e,

$$L_2(\mathbf{R}_\mathrm{L}^k) \leq \mathrm{c}\, L_2(\mathbf{R}^n). \tag{2.49}$$

Numerical experiments show that $\mathrm{c} = 0.1$ is a suitable value for obtaining convergence, although for some conditions a lower value may be needed. In general, for $\mathrm{c} = 0.1$, the average number of iterations that are needed at each pseudo-time step is $k \approx$ 7-8. Very low values of tolerance are not useful since no improvement in the convergence rate can be expected due to the approximations in the Jacobian.

The efficiency of the iterative procedure heavily relies on the effectiveness of the preconditioner. If the latter is not effective, the convergence can be poor and in some cases it can stall. The preconditioner should be a good approximation of the original matrix ($\mathbf{P}_\mathrm{M} \approx \mathbf{M}$) and moreover it should be relatively easy to invert. Application of the preconditioner takes extra computational cost of course. Usually the more effective the preconditioner, the more expensive its application is. The preconditioner used here is a very good compromise between effectiveness and cost (a comparison with another method is presented in Section 2.5.3).

An initial solution $\mathbf{z}^0$ must be provided before the iterations of Eq. (2.47) start. In the present work two different types of initializations are performed, depending on the convergence level of the non-linear problem. The first type is performed from the start of the iterations, until the non-linear problem has converged one order of magnitude, i.e., until $L_2(\mathbf{R}^n) \geq 0.1 L_2(\mathbf{R}^0)$. During that time the initial solution at iteration n is that obtained from the previous iteration: $\mathbf{z}^0 = \mathbf{U}^n - \mathbf{U}^{n-1}$. The second type is performed for the remaining iterations, for which $L_2(\mathbf{R}^n) < 0.1 L_2(\mathbf{R}^0)$. Since the solution should be relatively converged for those iterations, i.e., $\mathbf{U}^{n+1} \approx \mathbf{U}^n$, the initialization is $\mathbf{z}^0 = \mathbf{0}$.

In summary, in order to solve the linear system with Eq. (2.47) one has to do essentially two things: (i) compute the matrix-vector products $\mathbf{M}\mathbf{z}^k$, and (ii) apply the preconditioner to ${\mathbf{R}_L}^k$. Both tasks may be efficiently carried out by taking into account the topology of the approximate Jacobian.

The approximate Jacobian

The Jacobian $\partial\widetilde{\mathbf{R}}/\partial\mathbf{U}$ is first-order and approximate. It is first-order because the MUSCL reconstruction is neglected and approximate in terms of the numerical fluxes, which are not exactly differentiated. Compared to the exact Jacobian, the implementation and the assembly at run-time of this Jacobian are much simpler.

Neglecting the MUSCL reconstruction implies that the flux vector on the edge ij only depends on the average variables $\mathbf{u}_i$ and $\mathbf{u}_j$. Thus, each edge ij gives a contribution limited to the nodes i and j only. The latter means that a loop on the edges, similar to that used for the residual, is sufficient for gathering all the elements of the residual Jacobian. Such a loop can be derived by differentiating the residual assembly of Eq. (2.22) with respect to i and j, considering the residual to be first-order. Thus, neglecting the time-step contribution and the boundary terms, each edge ij gives four non-zero elements to the matrix $\mathbf{M}$, which read

$$\begin{aligned} &[ii] \quad \mathbf{D}_{\mathrm{M}i} = \mathbf{D}_{\mathrm{M}i} + \frac{\partial\tilde{\mathbf{\Phi}}_{ij}}{\partial\mathbf{u}_i}\,, && [ij] \quad \mathbf{L}_{\mathrm{M}ij} = \frac{\partial\tilde{\mathbf{\Phi}}_{ij}}{\partial\mathbf{u}_j}\,, \\ &[ji] \quad \mathbf{U}_{\mathrm{M}ji} = -\frac{\partial\tilde{\mathbf{\Phi}}_{ij}}{\partial\mathbf{u}_i}\,, && [jj] \quad \mathbf{D}_{\mathrm{M}j} = \mathbf{D}_{\mathrm{M}j} - \frac{\partial\tilde{\mathbf{\Phi}}_{ij}}{\partial\mathbf{u}_j}\,. \end{aligned} \tag{2.50}$$

As can be seen, the off-diagonal terms $\mathbf{L}_{\mathrm{M}ij}$ and $\mathbf{U}_{\mathrm{M}ij}$ do not need to be accumulated, i.e., a single visit on the edge ij is sufficient to evaluate them. Instead, the diagonal terms need to be accumulated. For instance, in order to evaluate $\mathbf{D}_{\mathrm{M}i}$, all the edges that share the node i must be visited. The assembly of the diagonal terms should be completed with the boundary flux Jacobians, which have been neglected in the above equation. In the Section 3.2, the more complicated topology of the second-order Jacobian is addressed.

The approximation of the numerical flux Jacobians, which are indicated with the tilde in Eq. (2.50), consists of the following. The numerical flux Jacobians for the Roe flux in Eq. (2.25) are obtained by approximating the Roe matrix $|\mathbf{A}|$ as a constant during the differentiation, which gives

$$\frac{\partial \widetilde{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_i} \approx \frac{1}{2}(\mathbf{A}_i + |\mathbf{A}_{ij}|)\,, \qquad \frac{\partial \widetilde{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_j} \approx \frac{1}{2}(\mathbf{A}_j - |\mathbf{A}_{ij}|)\,. \tag{2.51}$$

The numerical boundary flux in Eq. (2.27) is differentiated considering the splittings $\mathbf{A}^+$ and $\mathbf{A}^-$ fixed and reads

$$\frac{\partial \widetilde{\mathbf{\Phi}}^{\infty}_{ij}}{\partial \mathbf{u}_i} \approx \mathbf{A}_i\,. \tag{2.52}$$

Neglecting the differentiation of $|\mathbf{A}|$, $\mathbf{A}^+$ and $\mathbf{A}^-$ is an important simplification in terms of the implementation. In Section 3.2.3 the exact differentiation of these terms is addressed. The wall flux in Eq. (2.29) can be differentiated exactly since it involves only the derivatives of the pressure, $\partial p/\partial \mathbf{u}$, which can be easily obtained analytically.

As final remark, it is important to pay attention to the role of the numerical dissipation and of the pseudo-time step in the solution of the linear system. Consider the diagonal terms of $\mathbf{M}$ for internal nodes, the expression of which has been partially given in Eq. (2.50). The complete expression is obtained by adding the time step contribution. It reads

$$\mathbf{D}_{\mathrm{M}\,i} = \frac{V_i}{\Delta t_i}\mathbf{I} + \sum_{j=1}^{N_i} \frac{\partial \tilde{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_i}\,. \tag{2.53}$$

Taking into account that the Roe matrix is considered constant, the differentiation of the Roe flux reads

$$\frac{\partial \tilde{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_i} = \frac{\partial}{\partial \mathbf{u}_i}\left(\frac{\mathbf{F}_i + \mathbf{F}_j}{2}\right) \cdot \mathbf{n}_{ij} - \frac{1}{2}|\mathbf{A}_{ij}| \frac{\partial}{\partial \mathbf{u}_i}(\mathbf{u}_j - \mathbf{u}_i) = \frac{1}{2}\frac{\partial \mathbf{F}_i}{\partial \mathbf{u}_i} \cdot \mathbf{n}_{ij} + \frac{1}{2}|\mathbf{A}_{ij}|\,.$$

Summation over the N_i neighbors of the node i yields

$$\sum_{j=1}^{N_i} \frac{\tilde{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_i} = \frac{1}{2}\frac{\partial \mathbf{F}_i}{\partial \mathbf{u}_i} \cdot \sum_{j=1}^{N_i} \mathbf{n}_{ij} + \frac{1}{2}\sum_{j=1}^{N_i} |\mathbf{A}_{ij}|\,.$$

Since the control volume of the internal node i is closed, $\sum_{j=1}^{N_i} \mathbf{n}_{ij} = 0$, and only the summation on the right-hand side of the above equation is left. Therefore, the diagonal term in Eq. (2.53) can be re-written as

$$\mathbf{D}_{\mathrm{M}\,i} = \frac{V_i}{\Delta t_i}\mathbf{I} + \frac{1}{2}\sum_{j=1}^{N_i} |\mathbf{A}_{ij}|\,. \tag{2.54}$$

This equation clearly shows how both the time step and the numerical dissipation contribute to the diagonal dominance of the matrix. As can be seen, a relatively large time step or a low value of the dissipation may cause the diagonal dominance of the matrix to reduce at the point where the solution stalls. The former may be caused by a too large increase of the CFL number whereas the latter may be caused by a small mesh size. The trade-off between pseudo-time and mesh dissipation is observed when computations are performed on increasingly fine meshes. In fact, the more the mesh size is reduced, the more the CFL must be updated slowly.

Matrix-vector products

Matrix-vector products $\mathbf{Mz}$ are required for the solution of the linear system, as shown in Eq. (2.47). According to matrix algebra, the component $[\mathbf{Mz}]_i$ is given by:

$$[\mathbf{Mz}]_i = \frac{V_i}{\Delta t_i}\mathbf{z}_i + \sum_{k\in\mathcal{N}_i} \frac{\partial\tilde{\mathbf{\Phi}}_i}{\partial\mathbf{u}_k}\mathbf{z}_k\,,$$

where the summation only involves the non-zero elements, $k \in \mathcal{N}_i$, rather than the complete row, $k = 1, N$. The square brackets have been removed to simplify the notation.

An edge–based assembly can be derived to compute the matrix-vector product on-the-fly. Some algebra reveals that this summation can be easily assembled in the same way as the residual, see Eq. (2.22). Only the final result is given here, which is achieved by taking into account the topology of the approximate Jacobian. Neglecting the time-step contribution, it reads

$$\begin{aligned}
[\mathbf{Mz}]_i &= [\mathbf{Mz}]_i + \frac{\partial\tilde{\mathbf{\Phi}}_{ij}}{\partial\mathbf{u}_i}\mathbf{z}_i + \frac{\partial\tilde{\mathbf{\Phi}}_{ij}}{\partial\mathbf{u}_j}\mathbf{z}_j\,,\\
[\mathbf{Mz}]_j &= [\mathbf{Mz}]_j - \frac{\partial\tilde{\mathbf{\Phi}}_{ij}}{\partial\mathbf{u}_i}\mathbf{z}_i - \frac{\partial\tilde{\mathbf{\Phi}}_{ij}}{\partial\mathbf{u}_j}\mathbf{z}_j\,, \qquad (ij = 1, E)\,.
\end{aligned} \tag{2.55}$$

Another loop on the boundary nodes, similar to that of Eq. (2.23), should be performed in order to include the boundary contribution.

As will be mentioned below, the elements $\partial\tilde{\mathbf{\Phi}}_{ij}/\partial\mathbf{u}_i$ and $\partial\tilde{\mathbf{\Phi}}_{ij}/\partial\mathbf{u}_j$ may be stored in order to apply the preconditioner. In this case, it makes sense to use these elements in Eq. (2.55) in order to save the time required for their computation.

Preconditioning: storage & matrix–free approach

The preconditioner given in Eq. (2.48) can be inverted by means of a forward solve, followed by a backward solve. The distance-one topology defined in Eq. (2.24) is used to perform the sweeps on the nodes.

The forward solve, $(\mathbf{D}_\mathrm{M} + \mathbf{L}_\mathrm{M})\,\Delta\mathbf{z}^* = \mathbf{R}^k_\mathrm{L}$, is performed with the first sweep on the nodes

$$\Delta\mathbf{z}_i{}^* = \mathbf{D}_{\mathrm{M}_i}^{-1}\left(\mathbf{R}^k_{\mathrm{L}i} - \sum_{j\in\mathcal{L}_i}\mathbf{L}_{\mathrm{M}ij}\Delta\mathbf{z}_j{}^*\right), \qquad (i = 1, N)\,, \tag{2.56}$$

whereas the backward solve, $(\mathbf{I}+\mathbf{D}_{\mathrm{M}}^{-1}\mathbf{U}_{\mathrm{M}})\,\Delta\mathbf{z} = \Delta\mathbf{z}^{*}$, is performed with the second sweep on the nodes

$$\Delta\mathbf{z}_i = \Delta\mathbf{z}_i{}^{*} - \mathbf{D}_{\mathrm{M}}{}_i^{-1}\sum_{j\in\mathcal{U}_i}\mathbf{U}_{\mathrm{M}ij}\Delta\mathbf{z}_j, \qquad (i = N, 1)\,. \tag{2.57}$$

$\mathcal{L}_i$ and $\mathcal{U}_i$ are the subsets of the stencil $\mathcal{N}_i$, see Eq. (2.24), that contain the nodes of the lower and the upper part of the row.

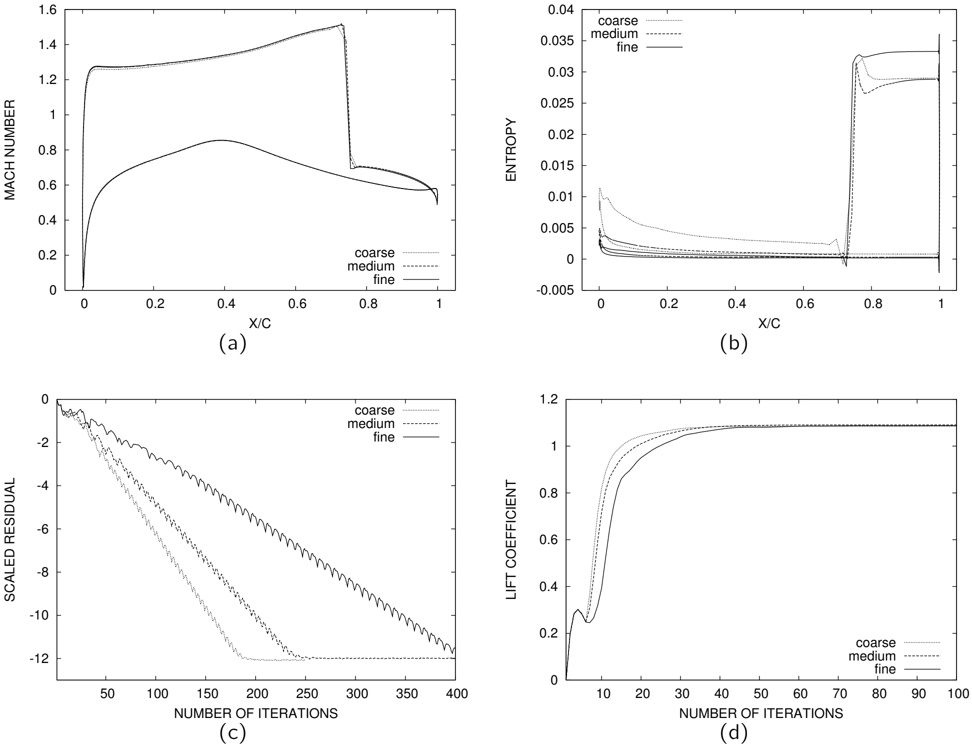
Storage of the matrix $\mathbf{M}$ and of its preconditioner $\mathbf{P}_{\mathrm{M}}$ is beneficial in terms of CPU time. However, depending on the size of the mesh, the amount of memory involved may be excessive. The preconditioning strategy implemented in this work has the feature of not requiring any preparatory work for the preconditioner. In fact, as shown in Eqs. (2.56) and (2.57), only the inversion of the diagonal sub-matrices is required. Consequently, there is no need to store both the matrix and the preconditioner. It is enough to store only $\mathbf{D}_{\mathrm{M}}$, $\mathbf{L}_{\mathrm{M}}$ and $\mathbf{U}_{\mathrm{M}}$, thus halving the storage. With these elements available one can: (*i*) perform the matrix vector product to compute the linear residual, see Eq. (2.55); and (*ii*) apply the preconditioner to the latter, see Eqs. (2.56) and (2.57). In practice, $\mathbf{D}_{\mathrm{M}}$, $\mathbf{L}_{\mathrm{M}}$ and $\mathbf{U}_{\mathrm{M}}$ are precomputed and stored at each non-linear iteration.

The approach just described can be referred to as the storage approach. In contrast, there is a matrix-free approach, which can be applied in this case because the preconditioner does not require preparatory work. Looking at the definition of the lower and the upper matrices, see Eq. (2.50), one understands that the sweeps of Eqs. (2.56) and (2.57) only require $\mathbf{D}_{\mathrm{M}}$ to be stored and allows for $\mathbf{L}_{\mathrm{M}}$ and $\mathbf{U}_{\mathrm{M}}$ to be computed on-the-fly. The matrix-free approach is very beneficial in terms of memory since it requires an amount of storage similar to that of an explicit scheme. However, recomputing $\mathbf{L}_{\mathrm{M}}$ and $\mathbf{U}_{\mathrm{M}}$ on-the-fly gives a penalty in terms of CPU time.

Similarities with the LU-SGS method

Apparently the solution method presented in this section is very similar to the LU-SGS method [41]. Nevertheless the method is different in several respects. For instance, the LU-SGS method uses increments to compute matrix-vector products of the flux Jacobian with vectors, e.g., $\mathbf{U}_{ij}\Delta\mathbf{z}_j = (\mathbf{F}(\mathbf{u}_j + \Delta\mathbf{z}_j)\cdot\mathbf{n}_{ij} - \mathbf{F}(\mathbf{u}_j)\cdot\mathbf{n}_{ij} + \rho_{Aij}\Delta\mathbf{z}_j)/2$. Moreover, rather than the Roe matrix $|\mathbf{A}(\bar{\mathbf{u}}_{ij},\mathbf{n}_{ij})|$, its spectral radius $\rho_{Aij} = |\mathbf{w}_{ij}\cdot\mathbf{n}_{ij}|$ is used [60]. Finally, the LU-SGS method does not perform linear iterations. I.e., at each non-linear iteration the LU-SGS method applies the preconditioner only once. Modifications aimed at improving the LU-SGS method have been suggested [25]. The modified method and the method presented here are similar.

Note that for the adjoint equations, which will be discussed in Chapter 3, the LU-SGS method is not suitable. In fact, increments cannot be employed because of the transposition that characterizes the Jacobians of the adjoint equations.



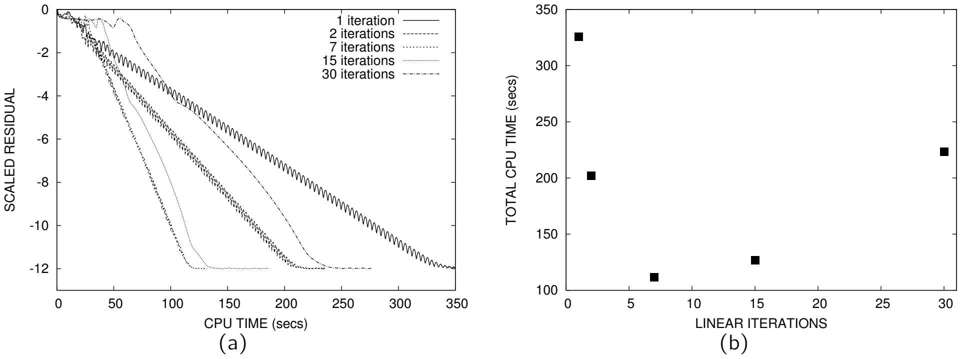
(a)

(b)

(c)

(d)

Figure 2.4: *RAE2822 at $M_\infty = 0.75$ and $\alpha = 3.0^\circ$. (a) The Mach number, (b) the entropy, (c) the residual and (d) the lift convergence are compared for three meshes of different sizes.*



(a)

(b)

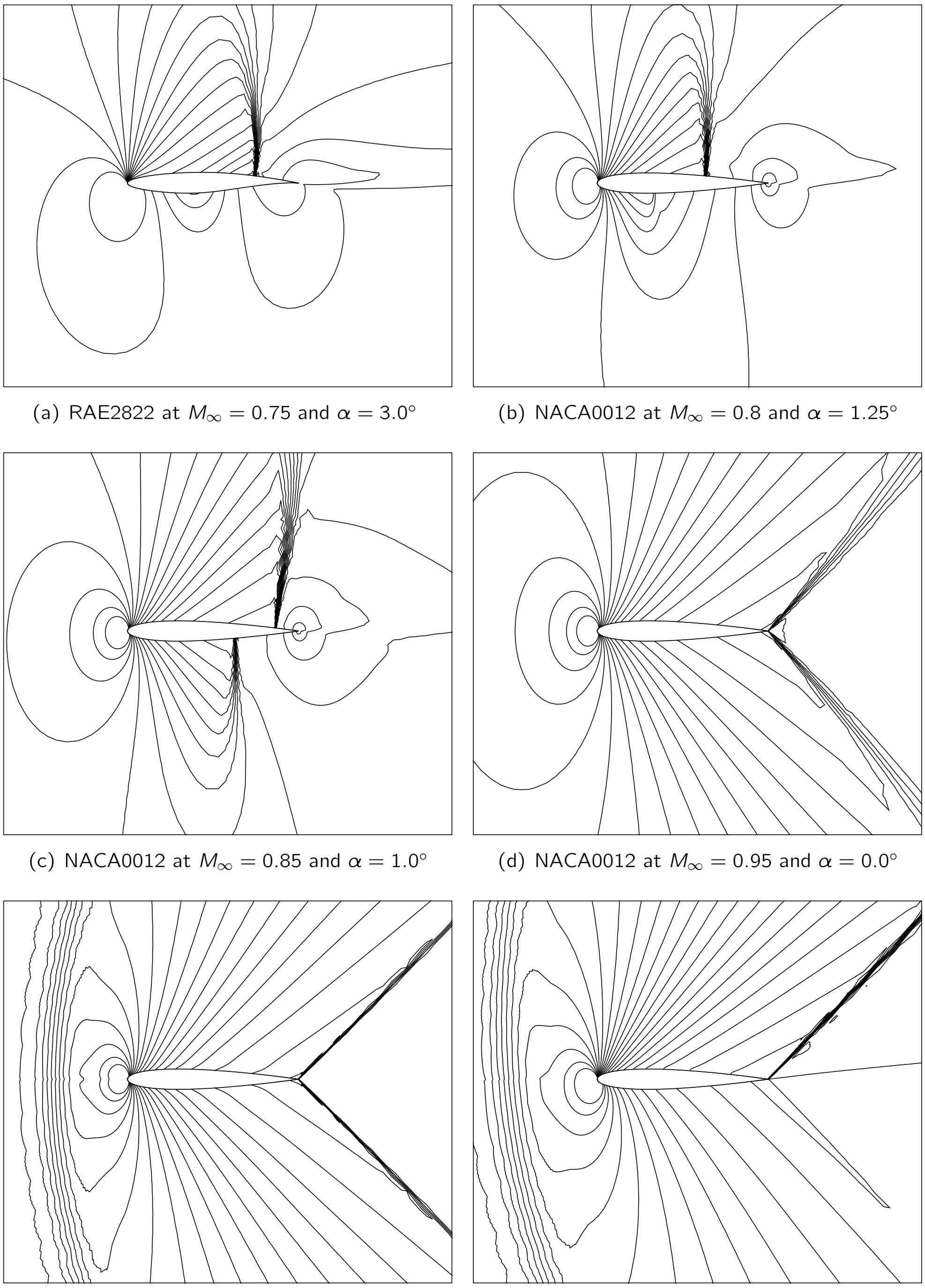
Figure 2.5: *RAE2822 at $M_\infty = 0.75$ and $\alpha = 3.0^\circ$. (a) Convergence history for different numbers of linear iterations. (b) Total time as function of the number of linear iterations.*

2.5 Results

Flow computations for 2D and 3D cases are presented. The flow conditions range from subsonic to supersonic. In order to provide a comparison with existing implementations, 2D [133] and 3D [109] test cases from the AGARD-AR-211 report (in the following AGARD report) for inviscid flow have been reproduced here. Moreover, for the 3D cases, a comparison with a solver developed by the FOI (Swedish Defense Research Agency) is presented.

2.5.1 2D cases

RAE2822 at $M_\infty = 0.75$ *and* $\alpha = 3.0°$ - Three meshes have been used for this case. The first one has 3477 nodes, the second one has 7240 nodes and the third one has 13261 nodes. In the following these meshes are referred to as coarse, medium and fine, respectively. The medium size mesh is that shown in Fig. 2.2a. As can be seen from the Mach number distribution shown in Fig. 2.4a, the prescribed flow conditions result in a transonic flow around the airfoil. The flow becomes supersonic on the top of the airfoil and shows a shock at 75% of the chord, which brings it back to subsonic flow conditions. Small differences between the results are visible at the shock and along the supersonic region, especially between the coarse and the medium mesh. The differences are more pronounced in terms of entropy distribution, which is depicted on Fig. 2.4b. The Mach number and the entropy distributions of the fine mesh agree very well with the distributions presented on the AGARD report. The convergence history of the scaled residual is shown in Fig. 2.4c. The convergence of the lift coefficient is shown on Fig. 2.4d. As can be seen 50 iterations are necessary for the lift to converge. The first line of Table 2.1 shows the agreement between the lift and the drag coefficient obtained here, with that taken from the AGARD report. The table shows the mean of the values obtained by different authors, c_l^* and c_d^*, and the scatter between the minimum and the maximum of the values, Δc_l^* and Δc_d^*. Figure 2.5a shows the convergence history of the medium size mesh, with the linear system solved for different numbers of iterations. The history is given in terms of CPU time. The plot shows that there is an optimum number of linear iterations that minimizes the time required to obtain the solution. Consider the total CPU-time required for the convergence of the residual up to the minimum point (i.e., the point where the curves in Fig. 2.5a become flat) against the number of linear iterations, as shown in Fig. 2.5b. As can be seen, there is an optimum number of linear iterations around 7. It is not efficient to solve the linear system very accurately or exactly. However, it is also not very convenient to specify a fixed number of linear iterations. In the next section, when discussing the 3D case, the strategy adopted here for the linear iterations will be addressed again. Note that for only 1 linear iteration the method presented in the previous section is equivalent to the LU-SGS method [60, 41]. Thus, as shown by Fig. 2.5b, the LU-SGS method can be dramatically improved by performing more iterations [25].



(a) RAE2822 at $M_\infty = 0.75$ and $\alpha = 3.0°$

(b) NACA0012 at $M_\infty = 0.8$ and $\alpha = 1.25°$

(c) NACA0012 at $M_\infty = 0.85$ and $\alpha = 1.0°$

(d) NACA0012 at $M_\infty = 0.95$ and $\alpha = 0.0°$

(e) NACA0012 at $M_\infty = 1.2$ and $\alpha = 0.0°$

(f) NACA0012 at $M_\infty = 1.2$ and $\alpha = 7.0°$

Figure 2.6: *Pressure contours for the AGARD test cases.*

NACA0012, transonic and supersonic flow - On the AGARD report [133], the flow around the NACA0012 airfoil has been considered for 5 different Mach numbers and angle of attacks, which are given in Table 2.1. Three of these conditions generate transonic flows whereas the other two generate supersonic flows. The transonic flows are computed on an unstructured mesh of triangles, with 12579 nodes, which is very similar to the one depicted in Fig. 2.3a. The pressure contours for these flows are shown in Figs. 2.6b, 2.6c and 2.6d. As can be seen from the pictures, away from the body the contour lines appear to be non-smooth and the discontinuities appear to be smeared. For instance, the latter behavior can be observed on the fish-tail shock of Fig. 2.6d. The behavior is caused by the increased size of the mesh elements away from the body. The two supersonic flows, which are depicted in Figs. 2.6e and 2.6f, have been computed on the mesh of Fig. 2.3a, which has 19520 nodes. The mesh allows a better capturing of the fish-tail shocks. In Fig. 2.6f, for which $M_\infty = 1.2$ and $\alpha = 7.0°$, a slip-line emanating from the trailing edge is visible. However, the distribution of the mesh elements ahead of the airfoil nose is not optimal for these supersonic flows. In fact, the mesh density is large in the vertical direction but small in the horizontal one. The result is that the two bow shocks of Figs. 2.6e and 2.6f appear to be smeared. The values of the lift and of the drag coefficients computed for the different NACA0012 cases are compared with the AGARD result in Table 2.1. As can be seen from the table, the values obtained here agree with those obtained by the other authors.

	M_∞	α	c_l	$c_l{}^*$	$\Delta c_l{}^*$	c_d	$c_d{}^*$	$\Delta c_d{}^*$
RAE	0.75	3.00°	1.0874	1.1058	.0322	0.0442	0.0467	.0056
NACA	0.80	1.25°	0.3487	0.3587	.0272	0.0223	0.0228	.0022
NACA	0.85	1.00°	0.3753	0.3532	.0588	0.0578	0.0545	.0126
NACA	0.95	0.00°				0.1088	0.1082	.0008
NACA	1.20	0.00°				0.0954	0.0953	.0014
NACA	1.20	7.00°	0.5235	0.5211	.0142	0.1540	0.1538	.0012
ONERA	0.84	3.06°	0.2808	0.2840	.011	0.0199	0.0168	.0014
ONERA	0.92	0.00°				0.0247	0.0218	.0068

Table 2.1: *c_l and c_d are the results of the present work. $c_l{}^* \pm \Delta c_l{}^*$ and $c_d{}^* \pm \Delta c_d{}^*$ are the results shown in the AGARD report.*

NACA0012, subsonic flow - The flow conditions in the AGARD test cases imply the presence of shock waves in the flow. Thus, the drag coefficients are different from zero. In the case of 2D inviscid flow with no shock waves it is known that the drag coefficient must be zero. Thus, inviscid flow with no shock waves are interesting to observe the dissipative behavior of the numerical method. Figure 2.7a shows the Mach-number distribution around the NACA0012 airfoil at $M_\infty = 0.63$ and $\alpha = 2.0°$ in the case of limited and non-limited reconstruction. The differences are very small and can be appreciated most on the upper side of the airfoil. Figure 2.7b shows the entropy distribution. The latter should be theoretically zero. Thus, the entropy that appears along the airfoil is to be attributed to the numerical scheme. It appears that especially at the top side of the airfoil, the limited reconstruction contributes more to the generation of entropy. The drag, which is expected to be zero, is $c_d = 3.3\,10^{-4}$ for the non-limited reconstruction and $c_d = 4.3\,10^{-4}$ for the limited reconstruction. Similar values of drag have been found by other authors [67].

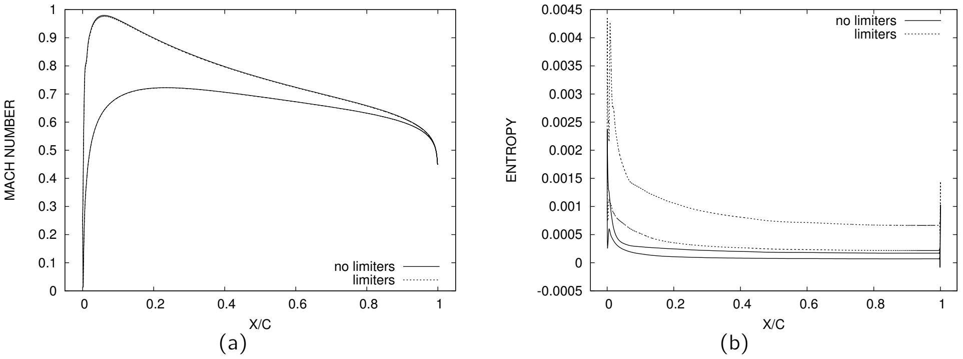


Figure 2.7: *NACA0012 at $M_\infty = 0.63$ and $\alpha = 2.0°$. (a) The Mach number and (b) the entropy distribution.*

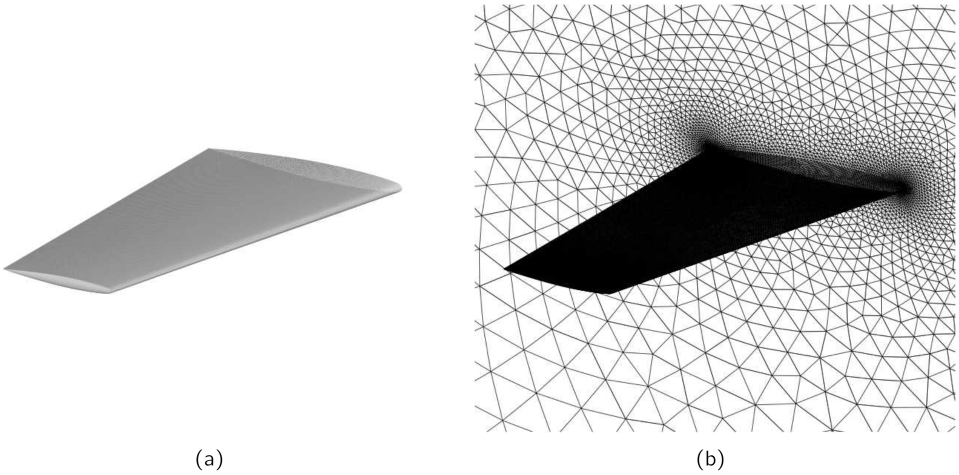


Figure 2.8: *(a) ONERA-M6 wing; (b) Unstructured mesh around the wing.*

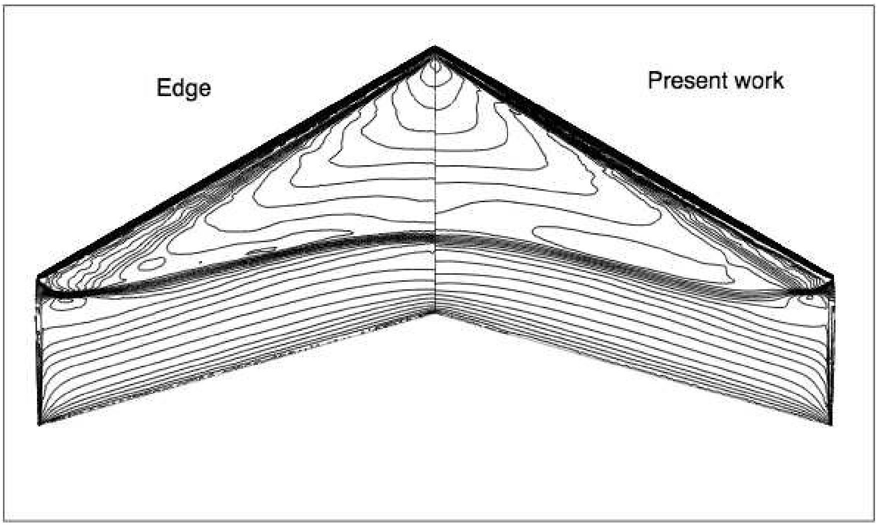


Figure 2.9: *ONERA-M6 wing at $M_\infty = 0.84$ and $\alpha = 3.06^\circ$. Contours of pressure coefficient obtained by the present solver (right) and by the Edge solver (left).*

2.5.2 3D cases

ONERA M6 wing - The geometry of this wing [112] is shown in Fig. 2.8a. The unstructured mesh of triangles generated around the wing is depicted in Fig. 2.8b. Results for this wing are available in the AGARD report [109] for two conditions: $M_\infty = 0.84$, $\alpha = 3.06^\circ$ and $M_\infty = 0.92$, $\alpha = 0$. For both conditions, the computed lift and drag coefficients are compared with the AGARD values given in Table 2.1. In order to compare the present formulation with one that uses a similar discretization, these two 3D cases have also been computed using the Edge solver [34] on the same mesh. The Edge solver, which has been developed by the FOI, is also based on the median-dual discretization. However, instead of the reconstruction-based scheme presented here, it uses a central scheme plus artificial dissipation. The two formulations behave differently in the presence of shocks, especially if these are weak. As can be seen from Fig. 2.9, the span-wise shock is resolved slightly differently by the two formulations. The pressure distributions are compared at different span-wise sections in Fig. 2.12. The differences that can observed on the span-wise distributions are of the same order as those observed by comparing the results of different authors [109]. When the shocks are very strong, as in the $M_\infty = 0.92$ and $\alpha = 0$ case depicted in Fig. 2.10, the agreement between the two formulations seems to be closer. The pressure distributions along the span are shown in Fig. 2.13. As can be seen, for both formulations, the top and the bottom pressures do not correspond exactly although the flow should be symmetric. The lack of symmetry in the flow is due to the lack of symmetry in the mesh. Figure 2.11a shows the convergence history of the solution in terms of CPU-time for the first case considered. Specifically it shows the logarithm of the scaled residual vector, i.e., Log($\mathbf{R}^n/\mathbf{R}^0$), which becomes flat after a drop of ten orders of magnitude. As can be seen, the matrix-free option is three times slower than the storage option. However, it requires almost three times less memory. Figure 2.11b shows linear iterations performed at each non-linear iteration. As already mentioned, the linear iterations are not fixed but they are established by the convergence criteria in Eq. (2.49), where usually $c = 0.1$. As can be seen from Figure 2.11, the strategy avoids unnecessary linear iterations. Also, it seems that the value of c used gives an average number of linear iterations almost equal to the optimal value found in Fig. 2.5b.

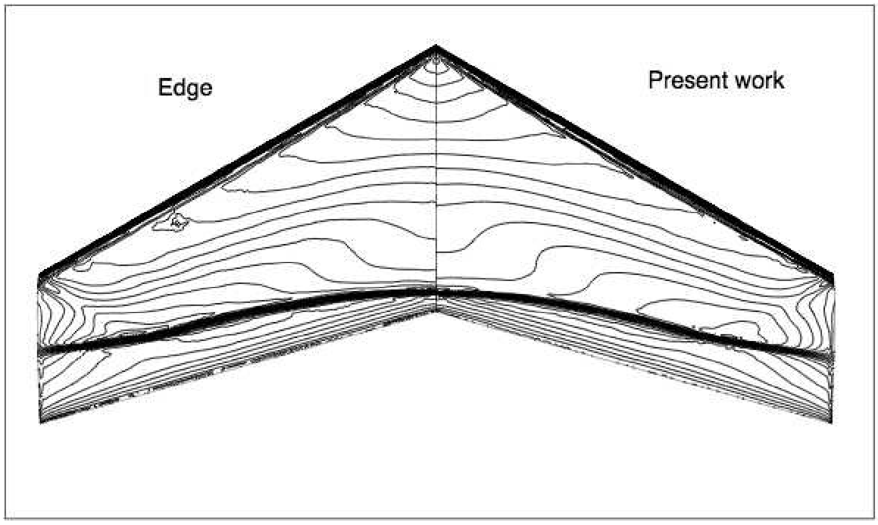
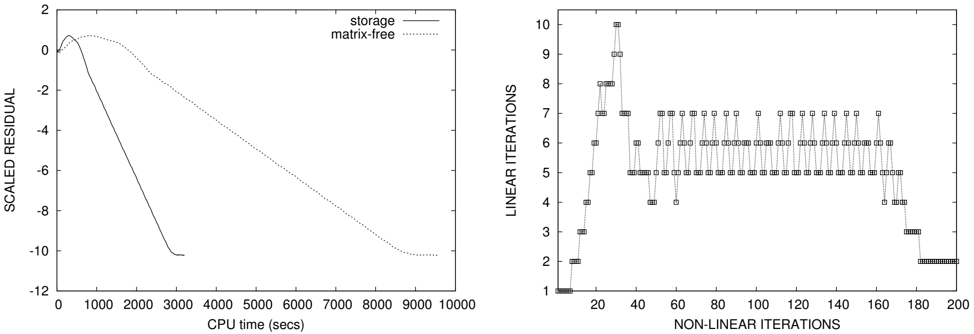
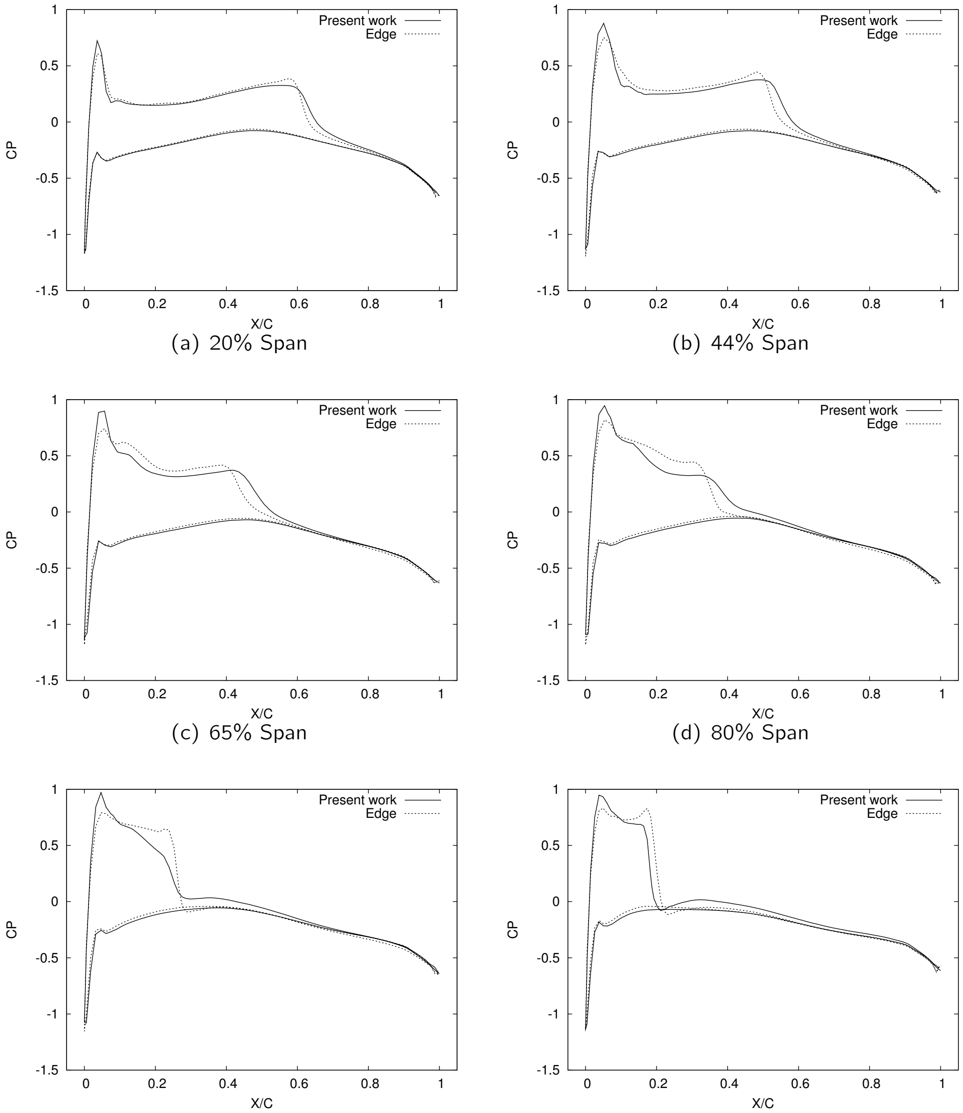


Figure 2.10: *ONERA-M6 wing at $M_\infty = 0.92$ and $\alpha = 0$. Contours of pressure coefficient obtained by the present solver (right) and by the Edge solver (left).*



(a) (b)

Figure 2.11: *Convergence history of the ONERA-M6 wing at $M_\infty = 0.84$ and $\alpha = 3.06°$. Logarithm of the scaled residual (left) and number of linear iterations (right).*



(a) 20% Span

(b) 44% Span

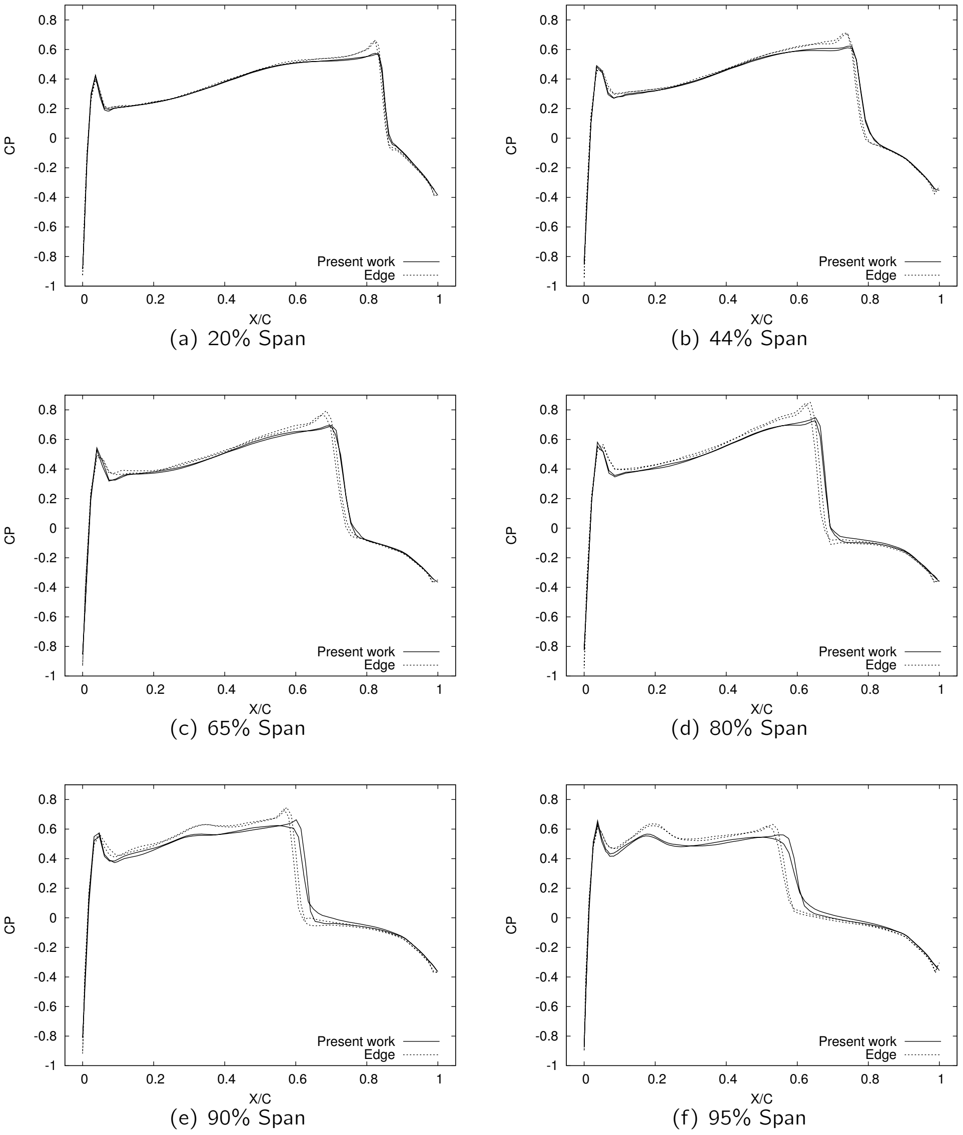
(c) 65% Span

(d) 80% Span

(e) 90% Span

(f) 95% Span

Figure 2.12: *Pressure coefficient distributions at different span-wise sections for the ONERA-M6 wing at $M_\infty = 0.84$ and $\alpha = 3.06^\circ$.*



(a) 20% Span

(b) 44% Span

(c) 65% Span

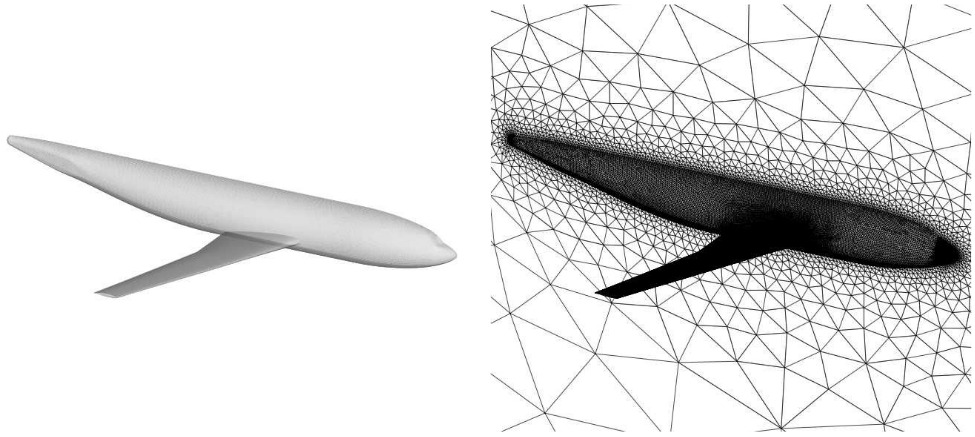
(d) 80% Span

(e) 90% Span

(f) 95% Span

Figure 2.13: *Pressure coefficient distributions at different span-wise sections for the ONERA-M6 wing at* $M_\infty = 0.92$ *and* $\alpha = 0.00°$.

DLR F6 wing-body - The geometry of this wing-body configuration [15] is depicted in Fig. 2.14a. The unstructured mesh of triangular elements is shown in Fig. 2.14b. As for the previous case, results obtained by the present formulation are compared with those obtained by the Edge solver. Figure 2.15 shows the pressure contours for $M_\infty = 0.75$ and $\alpha = 0.5°$. A magnification of the wing region is depicted in Fig. 2.16. The two pictures show that also in this case the shocks are captured slightly differently by the two formulations.



(a)

(b)

Figure 2.14: *Wing-body configuration: (a) DLR-F6 wing-body; (b) Unstructured mesh around the wing-body.*

Note that computing time differences between Edge and the solver implemented here have not been discussed. A comparison in terms of computing time would not be very appropriate. In fact, Edge uses a multigrid acceleration technique, which is not used in the present work. Moreover, Edge solutions have been obtained using the recommended solution parameters, which are suitable for obtaining a solution safely, but do not guarantee the best performances of the solver.

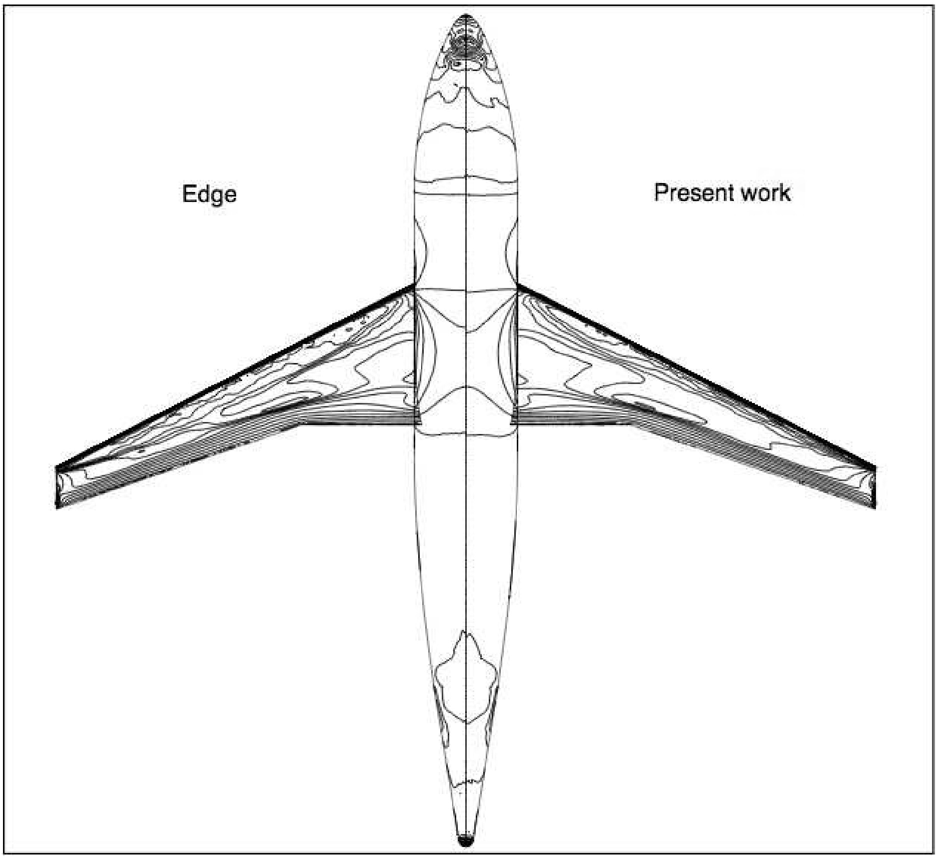


Figure 2.15: *DLR-F6 at* $M_\infty = 0.75$ *and* $\theta = 0.5^\circ$. *Contours of pressure coefficient obtained: by the present solver (right); by the Edge solver (left).*

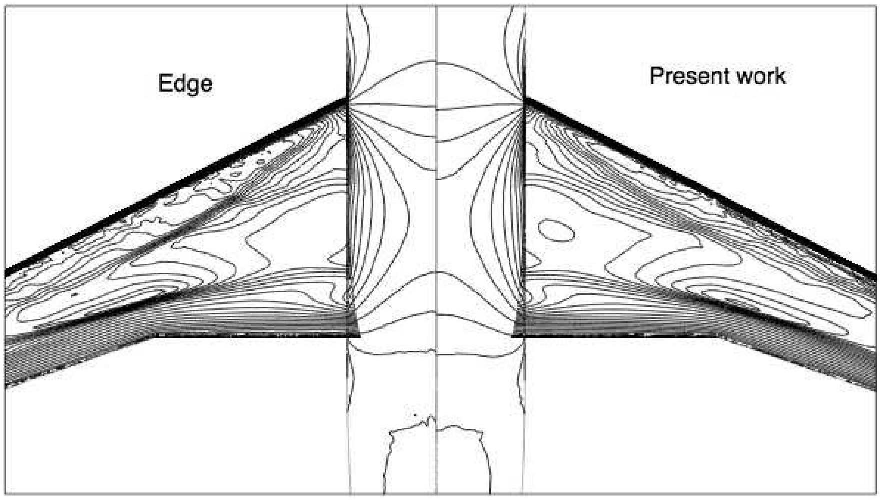


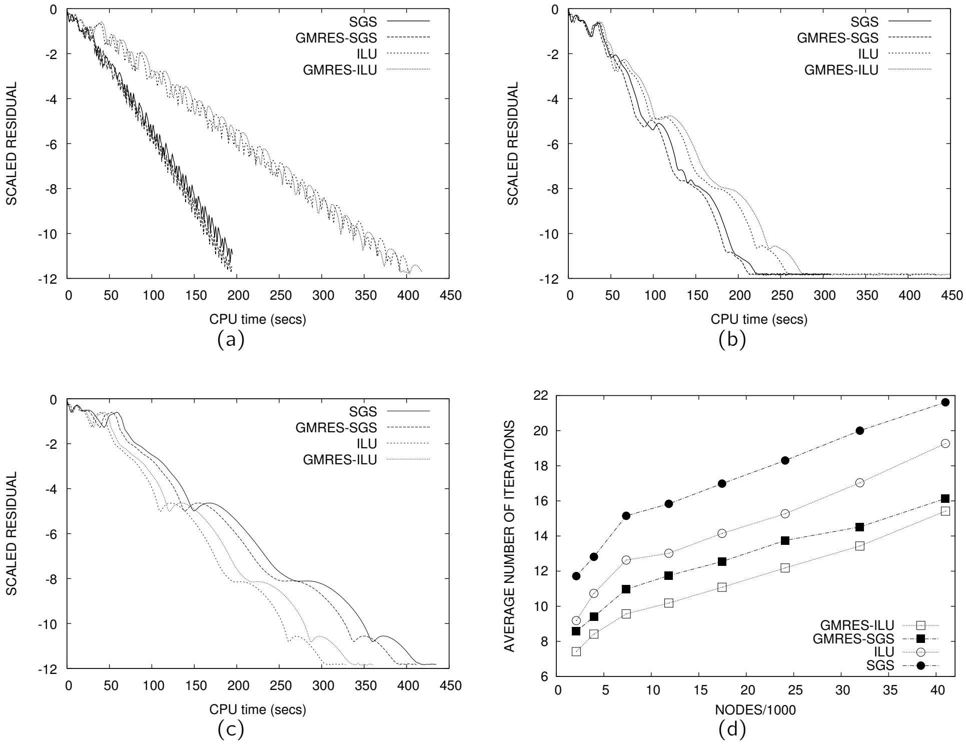
Figure 2.16: *DLR-F6 at $M_\infty = 0.75$ and $\theta = 0.5°$. Contours of pressure coefficient obtained: by the present solver (right); by the Edge solver (left).*

2.5.3 Alternative solution methods

The solution method adopted in the present work for the solution of the linear system of equations has been compared with other methods. An alternative preconditioner as well as an alternative iterative procedure have been considered. The alternative preconditioner is the Incomplete Lower Upper factorization with zero fill-in, better known as ILU(0). Incomplete with zero fill-in means that a LU factorization is performed, but only the elements that are located in the same positions as the non-zero elements of the original matrix are retained. The preparatory work that is required to perform the incomplete factorization is computationally expensive. The implementation of the ILU(0) preconditioner used for the comparison is taken from the SLATEC libraries [4].

The alternative iterative linear solver is the Generalized Minimum Residual Algorithm, better known as GMRES. It is a sophisticated iterative procedure, which is similar to the conjugate gradient method. Theoretically the procedure requires a number of vectors to be stored equal to that of the unknowns. In practice the cost of such storage may be unsustainable. Therefore restarted versions of the GMRES algorithm are used, which restart every predefined number of iterations with new vectors. The GMRES libraries described in [36] have been used, with 10 restart vectors. Such number of vectors is appropriate for the two dimensional case considered in this section, as the comparison between the methods should be made for similar levels of memory usage. Note that the node of an unstructured mesh of triangles is usually connected to 6-9 nodes, which means that 10 restart vectors take an amount of memory that is roughly equal to the size of the Jacobian matrix **M** of Eq. (2.46).

The following combinations are considered for the comparison: (i) SGS, the iterative method of Section 2.4.2, which uses the Symmetric Gauss-Seidel preconditioning (the



(a) (b) (c) (d)

Figure 2.17: *RAE2822 at $M_\infty = 0.73$ and $\alpha = 2.0°$. Convergence histories for different solution methods. The residual of the linear system is converged to a value that is (a) 1, (b) 2, and (c) 3 orders of magnitude smaller than the non-linear residual. (d) Average number of linear iterations for increasing mesh sizes.*

storage option is chosen); (ii) ILU, the iterative method of Section 2.4.2, with ILU preconditioning; (iii) GMRES-SGS, GMRES with Symmetric Gauss-Seidel preconditioning; (iv) GMRES-ILU, GMRES with ILU preconditioning.

In terms of memory usage SGS stores the diagonal, the lower and the upper part of the Jacobian matrix $\mathbf{M}$ (see Section 2.4.2). ILU stores the matrix $\mathbf{M}$ and the ILU(0) preconditioner created from $\mathbf{M}$. GMRES-SGS and GMRES-ILU requires the same storage of SGS and ILU, respectively, plus the 10 restart vectors. Therefore, SGS stores the equivalent of 1 Jacobian matrix, ILU and GMRES-SGS stores the equivalent of 2 Jacobian matrices and GMRES-ILU stores the equivalent of 3 Jacobian matrices.

The test case used for the comparison is the RAE2822 airfoil at $M_\infty = 0.73$ and $\alpha = 2.0°$ on a 12000 nodes mesh. The results are shown in Figs. 2.17a-c. Specifically, Fig. 2.17a shows the convergence history of the four methods for a relative tolerance of the linear system set to 1 order of magnitude, which means that the residual of the linear system is converged to a value that is 1 order of magnitude smaller than the non-linear residual (i.e., the constant c of Eq. (2.49) is set to 0.1). As can be seen SGS and GMRES-SGS converge at twice the speed of ILU and GMRES-ILU. Also, it appears that there is no difference between the simple iterative procedure of Section 2.4.2 and GMRES. In fact, the histories of SGS and GMRES-SGS almost overlap. The same is for GMRES-ILU and ILU. As for the results shown in Fig. 2.17b the relative tolerance is set to two orders of magnitude. As can be seen, the difference in convergence speed between SGS and ILU reduced appreciably. Still, SGS is faster. Also, it seems that GMRES is slightly faster than the simple iterative procedure. As for the results shown in Fig. 2.17c the relative tolerance is set to 3 orders of magnitude. As can be seen, ILU converges faster than SGS in this case. The reason is the preparatory work required by the ILU preconditioner at each non-linear iteration. The cost of such work is compensated by the effectiveness of ILU for the relative tolerance considered in this case. Instead, for the cases of Figs. 2.17a and 2.17b the relative tolerances are such that the cost is not compensated. As can be seen, the convergence speed of GMRES is again slightly higher than that of the simple iterative procedure. Finally, Fig. 2.17d shows the average number of linear iterations that are required for a relative tolerance of two orders of magnitude. Such average number is shown for increasing mesh sizes. As can be seen, GMRES-ILU is the most effective, followed in order by GMRES-SGS, ILU and SGS.

In summary, the results show that the ILU preconditioner is more effective than SGS. However, depending on the relative tolerance, the preparatory work required for the factorization affects negatively the overall performance of the method. For the cases considered in this thesis a relative tolerance between one and two orders of magnitude is satisfactory, there is no apparent benefit in achieving smaller tolerances. For those values of tolerances the results show that the SGS preconditioner is a better choice than ILU and that there is no appreciable benefit in using GMRES. Also from the point of view of memory requirements, SGS is a better choice compared to the other three methods.

2.6 Concluding remarks

The median-dual formulation appears to be very flexible. The additional cost of pre-processing the original mesh is fully repaid by the simplification introduced in the implementation of the flow solver. The degree of flexibility of the formulation has been demonstrated when the 2D flow solver has been extended to 3D problems. Only minor modifications were required for such extension.

As shown by the results, the reconstruction based scheme behaves properly in the subsonic, transonic and supersonic regime. The implicit solution scheme is very efficient and robust. Accurately converged solutions are always obtained. Fine tuning for the different types of flows can be obtained by acting on the parameters of the scheme. The matrix-free option makes it possible to run with very low memory requirements at the price of a lower solution speed.

Chapter 3

Sensitivity Analysis

3.1 Introduction to the problem

3.1.1 Primal and Dual problem

Sensitivity analysis may be defined [94] as an additional step of solution, which is performed after the flow solution, that aims at finding the sensitivity of some flow functionals with respect to certain parameters. It is performed by solving additional sensitivity equations, which are derived by manipulating the original equations.

In the present work two sensitivity analysis methods are considered. The first one solves the primal problem, which is also referred to as the linearized problem. The second one solves the dual problem, which is also referred to as the adjoint problem. As will be shown below, although the present work focuses mainly on the adjoint problem, the primal problem is a necessary intermediate step, which is required for the purpose of developing and testing the adjoint formulation.

The two methods are derived as follows. Consider a vector of N_α shape parameters $\bar{\alpha} = [\alpha_1, \alpha_2, ..., \alpha_{N_\alpha}]$. The parameters can also be referred to as controls or design variables and may be interpreted as the input of the system of steady Euler equations. The airfoil thickness, the twist angle or other geometric quantities that act on the boundary of the computational domain are examples of design parameters. Also, consider a vector of N_J functionals $\bar{J} = [J_1, J_2, ..., J_{N_J}]$, which represents the output of the system.

Elements of this vector may be the lift, the drag, the pitching moment and the like. The input parameters are linked to the output functionals by the relation $\bar{J} = \bar{J}(\mathbf{U}(\bar{\alpha}), \bar{\alpha})$, where $\mathbf{U}$ is the vector of conservative flow variables. The latter may be interpreted as a state variables vector. An implicit relation between the state variables $\mathbf{U}$ and the parameters $\bar{\alpha}$ is provided by the governing steady-state equation $\mathbf{R}(\mathbf{U}(\bar{\alpha}), \bar{\alpha}) = \mathbf{0}$, which can be referred to as the state equation.

Suppose that one has to compute the gradients $d\bar{J}/d\bar{\alpha}$, i.e., the gradient of all the functionals with respect to the shape parameters. The primal problem computes the gradient components by linearizing the functionals as

$$\frac{dJ_i}{d\alpha_j} = \frac{\partial J_i}{\partial \alpha_j} + \frac{\partial J_i}{\partial \mathbf{U}} \frac{\partial \mathbf{U}}{\partial \alpha_j}\,, \quad (i = 1, N_J\,;\ j = 1, N_\alpha)\,. \tag{3.1}$$

$\partial \mathbf{U}/\partial \alpha_j$ is the flow sensitivity, which can be computed by linearizing the state equation, i.e., by solving the equation

$$\frac{\partial \mathbf{R}}{\partial \mathbf{U}} \frac{\partial \mathbf{U}}{\partial \alpha_j} = -\frac{\partial \mathbf{R}}{\partial \alpha_j} \qquad (j = 1, N_\alpha)\,. \tag{3.2}$$

Thus, the primal problem computes the sensitivity of N_J functionals with respect to N_α parameters by solving N_α equations.

Alternatively, the gradient may be computed by solving the dual problem. For this purpose a new functional, which is referred to as the Lagrangian or augmented functional [126], has to be introduced. The functional is said to be augmented because the state equation, which is considered as a constraint, is added to the original functional. Specifically several augmented functionals have to be introduced, one for each functional J_i. The ith augmented functional reads

$$L_i = L(\mathbf{U}, \bar{\alpha}, \mathbf{\Lambda}_i) = J_i(\mathbf{U}, \bar{\alpha}) - \mathbf{\Lambda}_i^T \mathbf{R}(\mathbf{U}, \bar{\alpha})\,.$$

The augmented functional has the following interesting property:

$$\left[\frac{\partial L_i}{\partial \mathbf{U}}, \frac{\partial L_i}{\partial \mathbf{\Lambda}_i}\right] = \mathbf{0} \quad \Longrightarrow \quad \frac{dJ_i}{d\bar{\alpha}} \equiv \frac{\partial L_i}{\partial \bar{\alpha}}\,,$$

which means that if the stationary condition with respect to the state variables $\mathbf{U}$ and the multipliers $\mathbf{\Lambda}_i$ is found, then the sensitivity of the augmented functional coincides with that of the original functional. Equating $\partial L_i/\partial \mathbf{U}$ to zero gives an equation, which is referred to as the adjoint equation. It reads

$$\frac{\partial \mathbf{R}}{\partial \mathbf{U}}^T \mathbf{\Lambda}_i = \frac{\partial J_i}{\partial \mathbf{U}}^T\,, \qquad (i = 1, N_J)\,, \tag{3.3}$$

and must be solved for all the N_J functionals. Equating $\partial L_i/\partial \mathbf{\Lambda}_i$ to zero gives the state equation $\mathbf{R} = \mathbf{0}$, which is satisfied by the solution of the flow equations, for all the N_J functionals. Thus, assuming that the stationary condition is satisfied, the sensitivity of the functional J_i may be expressed as

$$\frac{dJ_i}{d\alpha_j} = \frac{\partial L_i}{\partial \alpha_j} = \frac{\partial J_i}{\partial \alpha_j} - \mathbf{\Lambda}_i^T \frac{\partial \mathbf{R}}{\partial \alpha_j}\,, \quad (i = 1, N_J\,;\ j = 1, N_\alpha)\,, \tag{3.4}$$

which is referred to as the dual sensitivity.

In summary, the primal problem is represented by Eqs. (3.1) and (3.2) whereas the dual problem is represented by Eqs. (3.3) and (3.4). Some algebra may be used to show that the dual sensitivity of Eq. (3.4) is equivalent to the primal sensitivity of Eq. (3.1). The two methods differ in the number of equations that must be solved in order to compute the sensitivity. The primal solves N_α equations whereas the dual solves N_J equations. Which of the two is the most convenient depends on the application.

Clearly, in the case of problems for which $N_\alpha \gg N_J$, the dual problem is appealing. Shape optimization problems often lie in this category. In fact, usually, there are a few functionals and a very large number of parameters. For instance, in the case of one functional and several shape parameters, the dual problem would require the solution of one equation only. The primal problem would require the solution of several equations.

As opposed to that, problems for which $N_\alpha \ll N_J$ are more efficiently solved by the primal problem. For instance, a problem with one parameter and several functionals would only require one solution of the primal problem but several of the dual problem. Structural analysis is one field in which problems exist with many more functionals than parameters [126].

3.1.2 Discrete approach to sensitivity analysis

The aforementioned primal and dual problems may be implemented by following the continuous or the discrete approach. The former derives the sensitivity equations by differentiating the flow equations and then by discretizing them. The latter derives the sensitivity equations by differentiating directly the discretized flow equations. In the present work, the discrete approach is preferred. The two approaches are different in terms of derivation as well as implementation. In the following, a brief account of these differences is given. More detailed accounts of these differences can be found elsewhere [92, 38].

Difficulties in the derivation of the continuous approach are mainly theoretical. For instance, boundary conditions must be provided for the adjoint variables, which are not directly observable quantities such as the primitive variables. Thus, a certain level of abstraction is required. However, once the problem has been defined, the implementation should be straightforward, resembling closely that of the flow solver. Still, the verification of the implementation may be ambiguous. In fact, due to the discretization of the adjoint equations, the gradient computed by the continuous approach is not equal to the discrete gradient. The latter mismatch between the gradient and the function can hamper the convergence in optimization applications. The continuous approach has been implemented and successfully applied to aerodynamic design problems by several authors [52, 116, 13].

As opposed to that, the discrete approach is relatively easy to derive. In fact, only basic linear algebra is required. The boundary conditions unfold naturally. However, it is relatively difficult to implement. Difficulties arise in the differentiation, which may become a huge undertaking in the case of non-linear problems. Moreover, due to the presence of large matrices that arise from the differentiation of the flow arrays, dealing with memory and CPU-time issues may not be a simple task. If those issues are not carefully considered, the resulting implementation can be extremely inefficient compared to that of the flow solver. An advantage of the discrete approach is that there is a systematic way to check the correctness of the implementation. If the adjoint is implemented exactly, which means no approximations are made in the differentiation, the computed gradient will match exactly with that of the original code. Moreover, also in the case of approximations being made, each part of the adjoint code can be verified against the linearized code. Implementations of the discrete approach may be found in [86, 35, 95, 93, 37, 6, 85, 84].

3.1.3 Implementation of the discrete sensitivity

Probably the most complicated task that must be accomplished in order to develop the adjoint method is the implementation of the matrix-vector product at the left-hand side of the adjoint equation, Eq. (3.3). The task is non-trivial because it involves the differentiation of complex non-linear formulations. Moreover, the resulting Jacobians must be multiplied by the vectors in a matrix-free fashion. The transposition complicates the task further because, as will be shown below, it inverts the operations in a counter-intuitive way. Clearly, storage of the Jacobians involved would make the implementation much easier, particularly from the point of view of the transposition. However, storage may require too much memory, especially for 3D cases. Thus, at the price of a more complex implementation, the matrix-free option is preferable.

The task of differentiating the code and assembling the matrix-vector products on-the-fly is addressed in Sections 3.2 and 3.3. In the following, in order to familiarize the reader with the challenges faced in the development of an adjoint formulation, a brief description of the issues faced when performing the transposition is given.

Implications of the transposition

Transposition implies that the operations in the code must be reversed compared to the linearized code. For instance, consider a code that produces a vector **F** as output, given the vector **Y** as input. The output vector is the result of the computations performed by a sequential code made of *n* routines and may be written in abstract form as

$$\mathbf{F}(\mathbf{Y}) = \mathbf{F}_n(\mathbf{F}_{n-1}(...(\mathbf{F}_2(\mathbf{F}_1(\mathbf{Y}))))).$$

The output of the first is the input to the second one and so on. In terms of implementation, one would not store *n* arrays but only 2. E.g., once $\mathbf{F}_1$ has been computed in the first subroutine, **Y** can be overwritten by $\mathbf{F}_2$, and so on. By means of the chain rule, the linearized matrix-vector product is obtained as

$$\frac{\partial \mathbf{F}}{\partial \mathbf{Y}}\delta \mathbf{Y} = \frac{\partial \mathbf{F}_\mathrm{n}}{\partial \mathbf{F}_{\mathrm{n}-1}}....\frac{\partial \mathbf{F}_2}{\partial \mathbf{F}_1}\frac{\partial \mathbf{F}_1}{\partial \mathbf{Y}}\delta \mathbf{Y}\,.$$

Each of the above Jacobians may be considered as a differentiated subroutine. Since the problem is non-linear, the differentiated residual routine $\partial \mathbf{F}_1/\partial \mathbf{Y}$ does not only need

$\delta\mathbf{Y}$, but also $\mathbf{Y}$. The same holds for $\partial\mathbf{F}_2/\partial\mathbf{F}_1$, which needs $\delta\mathbf{F}_1 = [\partial\mathbf{F}_1/\partial\mathbf{Y}]\delta\mathbf{Y}$ and $\mathbf{F}_1$, and so on. The differentiated routine, in case of the left product, may be run in parallel with the flow solver routines in order to use the flow solution vectors as soon as they are available.

The situation is different for the transposed matrix-vector product. According to matrix algebra the product is

$$\frac{\partial\mathbf{F}}{\partial\mathbf{Z}}^T\delta\mathbf{Y} = \frac{\partial\mathbf{F}_1}{\partial\mathbf{Z}}^T\frac{\partial\mathbf{F}_2}{\partial\mathbf{F}_1}^T\cdots\frac{\partial\mathbf{F}_\mathrm{n}}{\partial\mathbf{F}_{\mathrm{n}-1}}^T\delta\mathbf{Y}\,.$$

As shown by the above equation, the transposition implies that the differentiated routines must be applied in reverse order. The reversal of the operations is not only confusing but generates higher memory requirements too. The first routine to be executed is the one that computes $\partial\mathbf{F}_\mathrm{n}/\partial\mathbf{F}_{\mathrm{n}-1}$, which needs $\delta\mathbf{Y}$ but also $\mathbf{F}_{n-1}$. However, $\mathbf{F}_{n-1}$ is not available unless the routine $n-1$ has been run. Clearly, for the latter routine to run one needs also the vector $\mathbf{F}_{n-2}$, which on its turn needs the routine $n-2$ to be run and so on. Thus, the transposed product needs the original solver to be executed and all the vectors to be stored prior to its execution. Compared to the linearized product, which needs $2+2$ arrays for storage, the transposed product would need $n+2$ arrays. One understands that if n is very large the storage may become prohibitive and therefore actions must be taken in order to overcome this problem.

Hand-coding versus Automatic Differentiation

In the present work the sensitivity analysis with respect to the flow variables is hand-coded. An advantage of the hand-coding is that the developer may use his knowledge about the implementation to boost performance and efficiency by introducing simplifications or approximations in the development. A disadvantage of hand-coding is that the issues described above about the reverse ordering of the differentiation must be taken care of. Thus, a complex strategy must be devised. Also the differentiation may become a very tedious and error prone operation. Moreover, as soon as new features in the modeling of the flow equations are added, they have to be differentiated as well, which may turn out to be very time consuming especially if the modeling must be first assimilated by the developer. For dedicated applications the latter may not be a real problem. On the contrary, in the case of commercial applications, the developer of the original code and the one of the sensitivity code may very well be two different persons, which are not aware of the details of each other's implementation.

Automatic Differentiation (AD) is the automatic generation of ready-to-run sensitivity codes. AD tools manipulate the original source code and produce sensitivity code by applying basic linearization rules. For the transposed implementation, they use complicated methods to manage the memory issues caused by the reverse ordering of the differentiation. For instance, to limit memory usage, they may write arrays to disk when running the code from top to bottom and retrieve these arrays one by one when they run the code in reverse order. Obviously, an advantage of AD is that the developer is relieved from the burden of differentiating the code. The original code can be looked at as a black-box and the sensitivity code can be generated in a systematic way. A disadvantage of AD is that, when using the reverse mode, efficiency in terms of memory and CPU-time may be hard to obtain. For instance, writing to disk solves the memory problem caused by the reverse ordering but reduces the performance in terms of CPU-time.

AD in the present work has been used only in the so called forward-mode for the generation of the geometric sensitivities that are required to compute the gradients once the adjoint variables are available. Moreover, it has been used for validation purposes. The forward-mode of the Tapenade AD tool [3] has been found very easy to use and very efficient. Literature on AD and a comprehensive list of available AD tools can be found at the Community Portal for Automatic Differentiation [1].

3.2 Exact sensitivity: one-pass construction

A one-pass construction of the two matrix-vector products that appear in the linearized equation, Eq. (3.2), and in the adjoint equation, Eq. (3.3), may be derived. One-pass means that the complete assembly may be realized with one loop over the edges, where the control-volume interfaces are located. As will be shown later, this approach is not very efficient. In fact, the presence of the MUSCL reconstruction requires additional nested loops. Specifically, two loops must be performed for each edge: one over the stencil of the left node, and another one over the stencil of the right node. Nevertheless, the one-pass construction is easy to implement. Moreover, it is useful to derive a general result that will be used later.

In order to derive the assembly, following a strategy introduced in previous work [11], three new vectors of length E are introduced (as already mentioned in Chapter 2, E is equal to the number of edges in the mesh). The first vector $\mathbf{H} = [\hat{\boldsymbol{\Phi}}_1, \hat{\boldsymbol{\Phi}}_2, ..., \hat{\boldsymbol{\Phi}}_E]^T$ contains the second-order fluxes for each edge. The second and third vectors, $\mathbf{U}_L = [\hat{\mathbf{u}}_{L1}, \hat{\mathbf{u}}_{L2}, ..., \hat{\mathbf{u}}_{LE}]^T$ and $\mathbf{U}_R = [\hat{\mathbf{u}}_{R1}, \hat{\mathbf{u}}_{R2}, ..., \hat{\mathbf{u}}_{RE}]^T$, contain the reconstructed left and right states for each edge, respectively. The dependency of the residual Jacobian on the conservative variables can be expressed, using these vectors, as

$$\mathbf{R} = \mathbf{R}(\mathbf{H}(\mathbf{U}_L, \mathbf{U}_R)) . \tag{3.5}$$

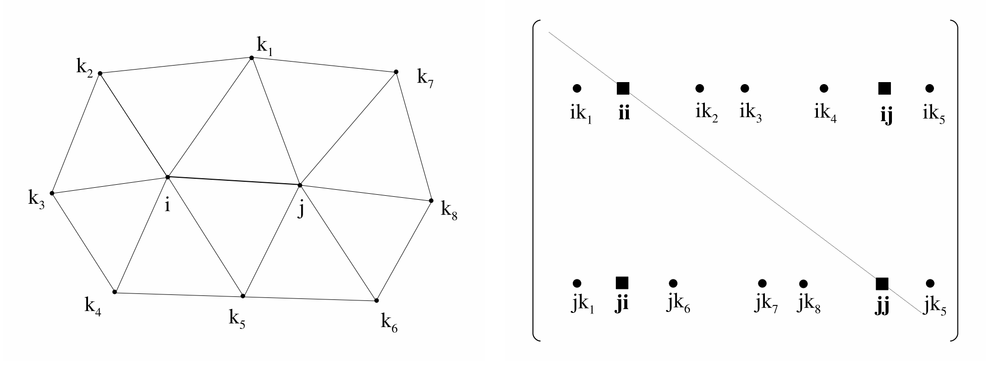
The left and the right states depend on the conservative variables, i.e,

$$\mathbf{U}_L = \mathbf{U}_L(\mathbf{U}) , \quad \mathbf{U}_R = \mathbf{U}_R(\mathbf{U}) . \tag{3.6}$$

The dependence of the above states on the gradients as well as on the limiters is considered implicitly. By means of the chain rule, the two matrix-vector products, which will be indicated in the following with $\mathbf{Z}$ and $\bar{\mathbf{Z}}$, are obtained as

$$\mathbf{Z} = \frac{\partial \mathbf{R}}{\partial \mathbf{U}}\mathbf{P} = \frac{\partial \mathbf{R}}{\partial \mathbf{H}}\left(\frac{\partial \mathbf{H}}{\partial \mathbf{U}_L}\frac{\partial \mathbf{U}_L}{\partial \mathbf{U}} + \frac{\partial \mathbf{H}}{\partial \mathbf{U}_R}\frac{\partial \mathbf{U}_R}{\partial \mathbf{U}}\right)\mathbf{P} , \tag{3.7}$$

$$\bar{\mathbf{Z}} = \frac{\partial \mathbf{R}}{\partial \mathbf{U}}^T\mathbf{P} = \left(\frac{\partial \mathbf{U}_L}{\partial \mathbf{U}}^T\frac{\partial \mathbf{H}}{\partial \mathbf{U}_L}^T + \frac{\partial \mathbf{U}_R}{\partial \mathbf{U}}^T\frac{\partial \mathbf{H}}{\partial \mathbf{U}_R}^T\right)\frac{\partial \mathbf{R}}{\partial \mathbf{H}}^T\mathbf{P} , \tag{3.8}$$



(a) (b)

Figure 3.1: *Stencils of the node i and j (a) and contributions to the Jacobian (b).*

respectively. In the above products $\mathbf{P} = [\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_\mathrm{N}]^\mathrm{T}$ is a generic vector. The matrices that appear in the above equation are square and rectangular sparse matrices. Each element of these matrices is a square sub-matrix, i.e., a block, of size equal to the number of variables, for instance 4×4 for the 2D Euler equations. Specifically:

> $\partial\mathbf{R}/\partial\mathbf{H}$ is a rectangular dummy matrix of size $E \times N$. Each column numbered as an edge has only two non-zero entries on the rows corresponding to the left and the right nodes of the edge. Values for these two non-zero entries are -1 and $+1$, respectively (see Eq. (3.11)). Therefore, the matrix-vector product $[\partial\mathbf{R}/\partial\mathbf{H}]^T\mathbf{P}$ gives a vector of length E, which has the components that correspond to the edge ij equal to $\mathbf{p}_i - \mathbf{p}_j$.
>
> $\partial\mathbf{H}/\partial\mathbf{U}_L$ and $\partial\mathbf{H}/\partial\mathbf{U}_R$ are block diagonal matrices of size $E \times E$. They represent the differentiation of the numerical flux formulation. For the line corresponding to the edge ij, the numerical flux Jacobians $\partial\hat{\mathbf{\Phi}}_{ij}/\partial\hat{\mathbf{u}}_i$ are the elements of $\partial\mathbf{H}/\partial\mathbf{U}_L$ whereas $\partial\hat{\mathbf{\Phi}}_{ij}/\partial\hat{\mathbf{u}}_j$ are the elements of $\partial\mathbf{H}/\partial\mathbf{U}_R$. When the node i is lying on the boundary, the numerical boundary fluxes $\partial\mathbf{\Phi}^{bc}{}_i/\partial\mathbf{u}_i$ are also present. The same holds for node j. The hat on the numerical flux Jacobians indicates that they are evaluated using the reconstructed variables $\hat{\mathbf{u}}_i$ and $\hat{\mathbf{u}}_j$.
>
> $\partial\mathbf{U}_L/\partial\mathbf{U}$ and $\partial\mathbf{U}_R/\partial\mathbf{U}$ are block rectangular matrices of dimensions $N \times E$. They contain the differentiation of the reconstructed states with respect to the cell averages in their stencil, i.e., $\partial\hat{\mathbf{u}}_i/\partial\mathbf{u}_k$ for $k \in \mathcal{N}_i$ and $\partial\hat{\mathbf{u}}_j/\partial\mathbf{u}_k$ for $k \in \mathcal{N}_j$, respectively. On each row the non-zero entries of $\partial\mathbf{U}_L/\partial\mathbf{U}$ ($\partial\mathbf{U}_R/\partial\mathbf{U}$) will be positioned in the columns corresponding to the left (right) states and their distance-one neighbors. E.g., in the case of the edge ij of Fig. 3.1a, the top line depicted in Fig. 3.1b will contribute to $\partial\mathbf{U}_L/\partial\mathbf{U}$ and the bottom line to $\partial\mathbf{U}_R/\partial\mathbf{U}$.

In the following a way to assemble the matrix-vector products by looping over the edges of the mesh is described. Also, the sub-matrices are considered in more detail.

3.2.1 Derivation of the edge-based assembly

The edge-based assemblies of the matrix-vector products $\mathbf{Z}$ and $\bar{\mathbf{Z}}$ may be derived by taking as starting point the assembly of the residual vector $\mathbf{R}$, which was briefly discussed in Section 2.2.2. Note that in the following the assemblies are always written using pseudo-code.

Before starting the derivation, some definitions must be given. Recall that the residual at node i is defined as the sum of the numerical fluxes across the control volume interfaces ∂V_{ik}, i.e.,

$$\mathbf{r}_i = \sum_{k=1}^{N_i} \mathbf{\Phi}(\hat{\mathbf{u}}_i, \hat{\mathbf{u}}_k, \mathbf{n}_{ik}) = \sum_{k=1}^{N_i} \hat{\mathbf{\Phi}}_{ik} \,. \tag{3.9}$$

The hat on the conservative variables is used to indicate that these variables are the results of a reconstruction scheme. Also, the hat is used on the numerical flux to indicate that it is evaluated using reconstructed variables.

Moreover, recall that a distance-one stencil is used for the linear reconstruction scheme. As can be seen from Eq. (2.24), the stencil $\mathcal{N}_i$ of a node i includes the distance-one neighbors of the node as well as the node itself. For instance, the stencil $\mathcal{N}_i$ of Fig. 3.1a contains the nodes k_1, k_2, k_3, k_4, k_5, j and i. Therefore, the reconstructed variables at nodes i and j have a dependence on the cell averages in their stencils of the type

$$\hat{\mathbf{u}}_i = \hat{\mathbf{u}}_i(\mathbf{u}_k; k \in \mathcal{N}_i)\,, \quad \hat{\mathbf{u}}_j = \hat{\mathbf{u}}_j(\mathbf{u}_k; k \in \mathcal{N}_j)\,. \tag{3.10}$$

As already mentioned, the derivation is started from the edge-based assembly of the residual, which was already given in Eq. (2.22) in the form of pseudo-code and is repeated here for clarity. Using the new notation for the reconstruction, that assembly reads

$$\begin{aligned} \mathbf{r}_i &= \mathbf{r}_i + \hat{\mathbf{\Phi}}_{ij}\,, \\ \mathbf{r}_j &= \mathbf{r}_j - \hat{\mathbf{\Phi}}_{ij}\,, \qquad (ij = 1, E)\,. \end{aligned} \tag{3.11}$$

Matrix algebra shows that the ith component of $\mathbf{Z}$ is defined as

$$\mathbf{z}_i = \sum_{k=1}^{N} \frac{\partial \mathbf{r}_i}{\partial \mathbf{u}_k} \mathbf{p}_k \,. \tag{3.12}$$

Differentiating Eq. (3.11) with respect to $\mathbf{u}_k$, multiplying by $\mathbf{p}_k$ and adding an additional internal loop on the nodes gives:

$$\begin{aligned} \frac{\partial \mathbf{r}_i}{\partial \mathbf{u}_k}\mathbf{p}_k &= \frac{\partial \mathbf{r}_i}{\partial \mathbf{u}_k}\mathbf{p}_k + \frac{\partial \hat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k}\mathbf{p}_k\,, \\ \frac{\partial \mathbf{r}_j}{\partial \mathbf{u}_k}\mathbf{p}_k &= \frac{\partial \mathbf{r}_j}{\partial \mathbf{u}_k}\mathbf{p}_k - \frac{\partial \hat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k}\mathbf{p}_k\,, \qquad (ij = 1, E;\ k = 1, N)\,. \end{aligned} \tag{3.13}$$

According to Eq. (3.12) the quantities accumulated on the nodes i and j are the components $\mathbf{z}_i$ and $\mathbf{z}_j$ of $\mathbf{Z}$. This means that the nested loop given in Eq. (3.13) can be rewritten as:

$$\begin{aligned} \mathbf{z}_i &= \mathbf{z}_i + \frac{\partial \hat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k}\mathbf{p}_k\,, \\ \mathbf{z}_j &= \mathbf{z}_j - \frac{\partial \hat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k}\mathbf{p}_k\,, \qquad (ij = 1, E;\;\; k = 1, N)\,. \end{aligned} \tag{3.14}$$

As indicated in Eq. (3.9) the numerical flux is dependent on the left and right states i and j. Because of the reconstruction, such a dependency must be extended to the stencils of the two nodes according to Eq. (3.10). The latter means that the numerical flux Jacobian is non-zero only for the elements contained in the stencil of node i and node j, i.e.,

$$\frac{\partial \hat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k} \neq \mathbf{0}; \qquad k \in \mathcal{N}_i \cup \mathcal{N}_j\,. \tag{3.15}$$

As a consequence, in Eq. (3.14) the inner loop on the nodes can be limited to a summation on both stencil $\mathcal{N}_i$ and $\mathcal{N}_j$ to give:

$$\begin{aligned} \mathbf{z}_i &= \mathbf{z}_i + \sum_{k \in \mathcal{N}_i} \frac{\partial \hat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k}\mathbf{p}_k + \sum_{k \in \mathcal{N}_j} \frac{\partial \hat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k}\mathbf{p}_k\,, \\ \mathbf{z}_j &= \mathbf{z}_j - \sum_{k \in \mathcal{N}_i} \frac{\partial \hat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k}\mathbf{p}_k - \sum_{k \in \mathcal{N}_j} \frac{\partial \hat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k}\mathbf{p}_k\,, \qquad (ij = 1, E)\,. \end{aligned} \tag{3.16}$$

The numerical flux derivative is with respect to the cell average $\mathbf{u}_k$. However, the numerical flux is actually evaluated with the reconstructed variables. Thus, the chain rule can be applied to isolate the flux derivative from the reconstruction derivatives. In doing this the numerical flux derivative can be taken out of the stencil summation:

$$\begin{aligned} \mathbf{z}_i &= \mathbf{z}_i + \frac{\partial \hat{\mathbf{\Phi}}_{ij}}{\partial \hat{\mathbf{u}}_i} \sum_{k \in \mathcal{N}_i} \left.\frac{\partial \hat{\mathbf{u}}_i}{\partial \mathbf{u}_k}\right|_{ij} \mathbf{p}_k + \frac{\partial \hat{\mathbf{\Phi}}_{ij}}{\partial \hat{\mathbf{u}}_j} \sum_{k \in \mathcal{N}_j} \left.\frac{\partial \hat{\mathbf{u}}_j}{\partial \mathbf{u}_k}\right|_{ij} \mathbf{p}_k\,, \\ \mathbf{z}_j &= \mathbf{z}_j - \frac{\partial \hat{\mathbf{\Phi}}_{ij}}{\partial \hat{\mathbf{u}}_i} \sum_{k \in \mathcal{N}_i} \left.\frac{\partial \hat{\mathbf{u}}_i}{\partial \mathbf{u}_k}\right|_{ij} \mathbf{p}_k - \frac{\partial \hat{\mathbf{\Phi}}_{ij}}{\partial \hat{\mathbf{u}}_j} \sum_{k \in \mathcal{N}_j} \left.\frac{\partial \hat{\mathbf{u}}_j}{\partial \mathbf{u}_k}\right|_{ij} \mathbf{p}_k\,, \qquad (ij = 1, E)\,. \end{aligned} \tag{3.17}$$

The subscript ij is necessary to remind that $\partial \hat{\mathbf{u}}_i / \partial \mathbf{u}_k$ and $\partial \hat{\mathbf{u}}_j / \partial \mathbf{u}_k$ are evaluated on the edge ij. The above equation is the edge-based assembly of $\mathbf{Z}$, i.e., the assembly of the matrix form given in Eq. (3.7).

The edge-based assembly of the transposed Jacobian-vector product $\bar{\mathbf{Z}}$ can be derived from Eq. (3.17) by applying transposition. The numerical flux Jacobians $\partial \hat{\mathbf{\Phi}}_{ij} / \partial \hat{\mathbf{u}}_i$ and $\partial \hat{\mathbf{\Phi}}_{ij} / \partial \hat{\mathbf{u}}_j$ are easy to transpose since they lie on the diagonal of the matrix, at position ii and jj, and on the off-diagonals, at position ij and ji. Those positions are indicated with the squares in Fig. 3.1b. The summations are more complicated to transpose because they involve off-diagonal terms that are located as indicated by the dots in Fig. 3.1b. As can be seen, the summation is over the row elements. Since by transposition they have to turn into column elements, the summations become scatterings on the stencil nodes:

$$\begin{aligned}
\bar{\mathbf{z}}_p &= \bar{\mathbf{z}}_p + \left.\frac{\partial \hat{\mathbf{u}}_i}{\partial \mathbf{u}_p}\right|_{ij}^T \frac{\partial \hat{\mathbf{\Phi}}_{ij}}{\partial \hat{\mathbf{u}}_i}^T (\mathbf{p}_i - \mathbf{p}_j) , \qquad p \in \mathcal{N}_i , \\
\bar{\mathbf{z}}_q &= \bar{\mathbf{z}}_q + \left.\frac{\partial \hat{\mathbf{u}}_j}{\partial \mathbf{u}_q}\right|_{ij}^T \frac{\partial \hat{\mathbf{\Phi}}_{ij}}{\partial \hat{\mathbf{u}}_j}^T (\mathbf{p}_i - \mathbf{p}_j) , \qquad q \in \mathcal{N}_j , \qquad (ij = 1, E) . \qquad (3.18)
\end{aligned}$$

Finally, the above loop is the assembly of $\bar{\mathbf{Z}}$, i.e., of the matrix form given in Eq. (3.8).

The edge-based assembly of both loops in Eqs. (3.17) and (3.18) involves the distance-one neighbors of the two nodes that share the edge. Thus, in order to perform the assembly in one pass, a pointer linking each node with its distance-one neighbors must be made available. Such a pointer is the same as that used for the solution of the linear system and was defined in Eq. (2.24).

3.2.2 Differentiation of the reconstruction operator

The reconstruction contribution $\partial \hat{\mathbf{u}}_i / \partial \mathbf{u}_k$ that appears in Eqs. (3.17) and (3.18) may be broken into three parts, i.e.,

$$\frac{\partial \hat{\mathbf{u}}_i}{\partial \mathbf{u}_k} = \frac{\partial \hat{\mathbf{u}}_i}{\partial \hat{\mathbf{v}}_i} \frac{\partial \hat{\mathbf{v}}_i}{\partial \mathbf{v}_k} \frac{\partial \mathbf{v}_k}{\partial \mathbf{u}_k} . \qquad (3.19)$$

The first and the third matrices are transformation matrices between conservative and primitive variables, the expression of which is given in Eqs. (2.6) and (2.7), respectively. The middle matrix contains the differentiation of the reconstructed primitive variables and reads

$$\frac{\partial \hat{\mathbf{v}}_i}{\partial \mathbf{v}_k} = \begin{bmatrix} \partial \hat{\rho}_i / \partial \rho_k & 0 & 0 & 0 \\ 0 & \partial \hat{u}_i / \partial u_k & 0 & 0 \\ 0 & 0 & \partial \hat{v}_i / \partial v_k & 0 \\ 0 & 0 & 0 & \partial \hat{p}_i / \partial p_k \end{bmatrix} . \qquad (3.20)$$

The expression of the elements of the above matrix can be obtained by differentiating the reconstruction. For instance, considering the MUSCL-like reconstruction for the y-velocity, which reads

$$\hat{v}_i = v_i + \frac{\sigma_i}{2} \nabla v_i^T (\mathbf{x}_j - \mathbf{x}_i) , \qquad (3.21)$$

differentiation with respect to v_k gives

$$\frac{\partial \hat{v}_i}{\partial v_k} = \delta_{ik} + \frac{1}{2} \frac{\partial \sigma_i}{\partial v_k} \nabla v_i^T (\mathbf{x}_j - \mathbf{x}_i) + \frac{\sigma_i}{2} \frac{\partial \nabla v_i}{\partial v_k}^T (\mathbf{x}_j - \mathbf{x}_i) , \qquad (3.22)$$

where δ_{ik} is the Kronecker delta.

The differentiation of the gradient is straightforward since only metrics quantities are involved. In the case of the Green-Gauss gradient and similar for the weighted least-squares gradient, one has that

$$\nabla v_i = \frac{1}{2V_i}\sum_{k=1}^{N_i}(v_i + v_k)\,\mathbf{n}_{ik}\,, \qquad \frac{\partial \nabla v_i}{\partial v_k} = \begin{cases} 1/(2V_i)\mathbf{n}_{ik}\,, & i \neq k\,, \\ 1/(2V_i)\sum_{k=1}^{N_i}\mathbf{n}_{ik}\,, & i = k\,, \end{cases} \tag{3.23}$$

where V_i is the control volume i.

The differentiation of the limiter is more involved than that of the gradient. Also, the limiter has a dependency on all the nodes in the stencil and the differentiation does not involve metric quantities only. Differentiating the limiter expression given in Eq. (2.35) gives

$$\frac{\partial \sigma_i}{\partial v_k} = \frac{\partial Lim}{\partial \Delta {v_i}^*}\frac{\partial \Delta {v_i}^*}{\partial v_k} + \frac{\partial Lim}{\partial \Delta v_{im}}\frac{\partial \Delta v_{im}}{\partial v_k}\,, \quad \Delta {v_i}^* = \begin{cases} \Delta {v_i}^{\max}\,, & \Delta v_{ij} > 0\,, \\ \\ \Delta {v_i}^{\min}\,, & \Delta v_{ij} < 0\,. \end{cases} \tag{3.24}$$

where $m \in \mathcal{N}_i$ is the index for which the right-hand side of Eq. (2.35) is minimal. The above expression requires the differentiated gradient. In fact, according to the definition given in Section 2.3.2 for Δv_{ij}, one has that

$$\frac{\partial \Delta v_{im}}{\partial v_k} = \frac{1}{2}\frac{\partial \nabla v_i}{\partial v_k}^T(\mathbf{x}_m - \mathbf{x}_i)\,. \tag{3.25}$$

3.2.3 Differentiation of the numerical fluxes

The exact differentiation of Roe's approximate Riemann solver given in Eq. (2.25) may be written as

$$\frac{\partial \mathbf{\Phi}_{ij}}{\partial \mathbf{u}_i} = \frac{1}{2}\mathbf{A}(\mathbf{u}_i, \mathbf{n}_{ij}) + \frac{1}{2}\frac{\partial \mathbf{t}_{ij}}{\partial \mathbf{u}_i}\,, \qquad \frac{\partial \mathbf{\Phi}_{ij}}{\partial \mathbf{u}_j} = \frac{1}{2}\mathbf{A}(\mathbf{u}_j, \mathbf{n}_{ij}) - \frac{1}{2}\frac{\partial \mathbf{t}_{ij}}{\partial \mathbf{u}_j}\,, \tag{3.26}$$

where $\mathbf{t}_{ij}$ is the dissipative part of the flux, which is defined as

$$\mathbf{t}_{ij} = \mathbf{X}|\mathbf{\Lambda}|\mathbf{X}^{-1}\Delta\mathbf{u}_{ij}\,. \tag{3.27}$$

In the above equation, the vector $\Delta\mathbf{u}_{ij}$ is the difference between the left and the right states and $\mathbf{X}$ and $\mathbf{\Lambda}$ are the eigenvectors and the eigenvalues, which must be evaluated using the Roe averages $\bar{\mathbf{u}}_{ij}$ defined in Section 2.3.1. In order to differentiate $\mathbf{t}_{ij}$ it is best to follow a strategy outlined in a previous work [8]. For simplicity, the 2D case is considered. Moreover, in order to make the derivation as general as possible, the subscripts i and j are temporarily dropped.

The first step to take in the derivation is to introduce the new vector of variables $\mathbf{\Psi}$, which reads

$$\mathbf{\Psi} = [\rho, w_x, w_y, a^2]^T\,. \tag{3.28}$$

It is similar to the primitive variables vector, except for the last component. In order to simplify the differentiation, the latter component is conveniently taken to be equal to the square value of the speed of sound a^2. Also, the two vectors $\boldsymbol{\Lambda}_D$ and $\mathbf{f}_\lambda$ are introduced. The first vector contains the diagonal of the matrix $\boldsymbol{\Lambda}$, i.e., the eigenvalues, whereas the second vector contains the absolute values of the eigenvalues. They read

$$\boldsymbol{\Lambda}_D = [\lambda_1, .., \lambda_4]^T , \quad \mathbf{f}_\lambda = [|\lambda_1|, .., |\lambda_4|]^T , \tag{3.29}$$

respectively. The second vector depends on $\boldsymbol{\Lambda}_D$, which in turn depends on $\boldsymbol{\Psi}$ according to the eigenvalues definition given in Section 2.3.1. Therefore, $\mathbf{f}_\lambda = \mathbf{f}_\lambda(\boldsymbol{\Lambda}_D(\boldsymbol{\Psi}))$. The eigenvector $\mathbf{X}$ in Eq. (3.27) also depends on the vector $\boldsymbol{\Psi}$, i.e., $\mathbf{X} = \mathbf{X}(\boldsymbol{\Psi})$.

By using the above definitions, the dependence of $\mathbf{t}$ may be written as

$$\mathbf{t} = \mathbf{t}(\boldsymbol{\Psi}, \mathbf{f}_\lambda, \Delta\mathbf{u}) . \tag{3.30}$$

Thus, by means of the chain rule, the differentiation of $\mathbf{t}$ reads

$$\frac{\partial \mathbf{t}}{\partial \mathbf{u}} = \frac{\partial \mathbf{t}}{\partial \Delta\mathbf{u}}\frac{\partial \Delta\mathbf{u}}{\partial \mathbf{u}} + \left(\frac{\partial \mathbf{t}}{\partial \boldsymbol{\Psi}} + \frac{\partial \mathbf{t}}{\partial \mathbf{f}_\lambda}\frac{\partial \mathbf{f}_\lambda}{\partial \boldsymbol{\Lambda}_D}\frac{\partial \boldsymbol{\Lambda}_D}{\partial \boldsymbol{\Psi}}\right)\frac{\partial \boldsymbol{\Psi}}{\partial \mathbf{v}}\frac{\partial \mathbf{v}}{\partial \mathbf{u}} . \tag{3.31}$$

According to Eq. (3.27), one has that $\partial\mathbf{t}/\partial\Delta\mathbf{u} = \mathbf{X}|\boldsymbol{\Lambda}|\mathbf{X}^{-1}$. The term $\partial\mathbf{t}/\partial\boldsymbol{\Psi}$ has a complicated expression, which can be derived conveniently using symbolic differentiation. In practice, the product at the right-hand side of Eq. (3.27) is implemented explicitly and differentiated. The final result of the differentiation is very lengthy and is not given here. The interested reader may find it also in the appendix of the original reference [8]. The components of the matrices $\partial\mathbf{t}/\partial\mathbf{f}_\lambda$ and $\partial\mathbf{f}_\lambda/\partial\boldsymbol{\Lambda}_D$ read

$$\left[\frac{\partial \mathbf{t}}{\partial \mathbf{f}_\lambda}\right]_{lm} = \sum_{k=1}^{4}[\mathbf{X}]_{lm}[\mathbf{X}^{-1}]_{mk}[\Delta\mathbf{u}]_k , \qquad \left[\frac{\partial \mathbf{f}_\lambda}{\partial \boldsymbol{\Lambda}_D}\right]_{lm} = sign(\lambda_l)\delta_{lm} , \tag{3.32}$$

respectively. According to the eigenvalues definition, the matrix $\partial\boldsymbol{\Lambda}_D/\partial\boldsymbol{\Psi}$ may be written as

$$\frac{\partial \boldsymbol{\Lambda}_D}{\partial \boldsymbol{\Psi}} = \begin{bmatrix} 0 & n_x & n_y & 0 \\ 0 & n_x & n_y & 0 \\ 0 & n_x & n_y & |\mathbf{n}|/2a \\ 0 & n_x & n_y & \text{-}|\mathbf{n}|/2a \end{bmatrix} . \tag{3.33}$$

In order to give the expressions for the remaining terms, $\partial\Delta\mathbf{u}/\partial\mathbf{u}$ and $\partial\boldsymbol{\Psi}/\partial\mathbf{v}$, the index i and j must be used again in the equations. Since $\Delta\mathbf{u}_{ij} = \mathbf{u}_j - \mathbf{u}_i$, one has that $\partial\Delta\mathbf{u}_{ij}/\partial\mathbf{u}_i = -1$ and $\partial\Delta\mathbf{u}_{ij}/\partial\mathbf{u}_j = 1$. Moreover, as already mentioned, the dissipative part in Eq. (3.27) is evaluated using the Roe averages $\bar{\mathbf{u}}_{ij}$ defined in Section 2.3.1. Therefore, $\boldsymbol{\Psi}$ depends on the Roe averages, i.e., $\boldsymbol{\Psi} = \boldsymbol{\Psi}(\bar{\mathbf{u}}_{ij})$. The Jacobian $\partial\boldsymbol{\Psi}(\bar{\mathbf{u}}_{ij})/\partial\mathbf{v}_i$ has the following structure:

$$\frac{\partial \boldsymbol{\Psi}(\bar{\mathbf{u}}_{ij})}{\partial \mathbf{v}_i} = \begin{bmatrix} \partial\bar{\rho}_{ij}/\partial\rho_i & 0 & 0 & 0 \\ \partial\bar{u}_{ij}/\partial\rho_i & \partial\bar{u}_{ij}/\partial u_i & 0 & 0 \\ \partial\bar{v}_{ij}/\partial\rho_i & 0 & \partial\bar{v}_{ij}/\partial v_i & 0 \\ \partial\bar{a}^2_{ij}/\partial\rho_i & \partial\bar{a}^2_{ij}/\partial u_i & \partial\bar{a}^2_{ij}/\partial v_i & \partial\bar{a}^2_{ij}/\partial p_i \end{bmatrix} . \tag{3.34}$$

Similar is the structure of the Jacobian $\partial\boldsymbol{\Psi}(\bar{\mathbf{u}}_{ij})/\partial\mathbf{v}_j$. As can be seen, the non-zero terms in the above matrix must be obtained by differentiating the Roe averages with respect to the primitive variables of the left and of the right nodes. The terms are too lengthy and thus they are not given here. Their expressions may be found in the original reference [8]. Finally, the transformation matrix $\partial\mathbf{v}_i/\partial\mathbf{u}_i$ and $\partial\mathbf{v}_i/\partial\mathbf{u}_j$ are calculated using Eq. (2.7).

The above formulation can also be used to differentiate the flux-vector splitting of Eq. (2.27), which is used for the far-field boundary conditions. In that case, $\mathbf{t}_i = \mathbf{X}\boldsymbol{\Lambda}^{+}\mathbf{X}^{-1}\mathbf{u}_i$. The positive splitting of the eigenvalues requires the components of $\partial\mathbf{f}_\lambda/\partial\boldsymbol{\Lambda}_D$ to be defined as

$$\left[\frac{\partial\mathbf{f}_\lambda}{\partial\boldsymbol{\Lambda}_D}\right]_{ij} = \frac{1 + sign(\lambda_i)}{2}\delta_{ij}\,. \tag{3.35}$$

Moreover, there are no Roe averages and the Jacobian $\partial\boldsymbol{\Psi}/\partial\mathbf{v}_i$ has a very simple expression.

When the parabolic entropy fix [42] in the Roe flux is used, the matrix $\partial\mathbf{f}_\lambda/\partial\boldsymbol{\Lambda}_D$ contains a continuous function rather than the discontinuous *sign* function as in Eq. (3.32). In fact, apart from ensuring that the numerical dissipation is never zero, the entropy fix removes the lack of differentiability caused by the absolute value in the Roe flux. Except for the matrix $\partial\mathbf{f}_\lambda/\partial\boldsymbol{\Lambda}_D$, the above differentiation is the same in case of entropy fix.

Note that if the eigenvectors and the eigenvalues are considered to be constant, the differentiated flux of Eq. (3.31) reduces to the approximate ones of Eq. (2.51), in the case of the Roe flux, and to the approximate flux of Eq. (2.52) in the case of the far-field flux.

Exact differentiation and differentiability

Exact differentiation should not be confused with differentiability. A code can be differentiated exactly without necessarily being differentiable. Consider for instance a code that implements the non-differentiable absolute value function $|x|$, which is defined as x for $x \geq 0$ and $-x$ for $x < 0$. One possibility is that such code might contain two if statements, e.g., $\mathtt{if(x} \geq \mathtt{0)f} = \mathtt{x}$ and $\mathtt{if(x} < \mathtt{0)f} = -\mathtt{x}$. The exactly differentiated code can be obtained by differentiating the two statements, i.e., $\mathtt{if(x} \geq \mathtt{0)df} = \mathtt{1}$ and $\mathtt{if(x} < \mathtt{0)df} = -\mathtt{1}$. This way the differentiated code computes a derivative that is consistent with the implementation of the original code. In practice, a non-differentiable function is implemented using different code branches, and each branch can be differentiated.

In the following section methods to verify the correctness of the differentiated code are used and exactness is shown. Functions that are not continuously differentiable exist in the code. E.g., in the boundary flux, see Eq. (3.35), or in the Roe flux when no entropy fix is used, see Eq. (3.32). The lack of continuous differentiability did not appear to be an obstacle for obtaining exact derivatives, i.e., derivatives that are consistent with the original code.

3.2.4 General case of low-order Jacobians

The assemblies of Eqs. (3.17) and (3.18) deal with the case of an edge flux function that is dependent on the nodes that share the edge as well as on their stencils. The dependence is such because of the second-order reconstruction scheme.

For a low-order flux, which depends only on the two nodes that share the edge, i.e., $\mathbf{\Phi}_{ij} = \mathbf{\Phi}(\mathbf{u}_i, \mathbf{u}_j)$, where $\mathbf{u}_i$ and $\mathbf{u}_j$ are cell averages, the two assemblies simplify considerably. In fact, if the dependence is limited to the two nodes, the absence of the reconstruction implies that

$$\frac{\partial \hat{\mathbf{u}}_i}{\partial \mathbf{u}_k} = \mathbf{I}\delta_{ik} \, ,$$

where δ_{ik} is the Kronecker delta. Therefore, by making the above substitution into the linearized and the adjoint assemblies of Eqs. (3.17) and (3.18) and by using a different notation for the vectors in the adjoint case, the two assemblies reduce to the ones given in Table 3.1.

Table 3.1: *Pseudo-code to assemble the linearized and adjoint matrix-vector products. The Jacobian is first-order, i.e., $\mathbf{\Phi}_{ij} = \mathbf{\Phi}(\mathbf{u}_i, \mathbf{u}_i)$, with $\mathbf{u}_i$ and $\mathbf{u}_j$ cell averages.*

Linearized - $\mathbf{Z} = \dfrac{\partial \mathbf{R}}{\partial \mathbf{U}}\mathbf{P}$	**Adjoint - $\mathbf{P} = \dfrac{\partial \mathbf{R}}{\partial \mathbf{U}}^T \mathbf{Z}$**
Loop over the edges, $ij = 1, E$ $\mathbf{z}_i = \mathbf{z}_i + \dfrac{\partial \mathbf{\Phi}_{ij}}{\partial \mathbf{u}_i}\mathbf{p}_i + \dfrac{\partial \mathbf{\Phi}_{ij}}{\partial \mathbf{u}_j}\mathbf{p}_j$ $\mathbf{z}_j = \mathbf{z}_j - \dfrac{\partial \mathbf{\Phi}_{ij}}{\partial \mathbf{u}_i}\mathbf{p}_i - \dfrac{\partial \mathbf{\Phi}_{ij}}{\partial \mathbf{u}_j}\mathbf{p}_j$	Loop over the edges, $ij = 1, E$ $\mathbf{p}_i = \mathbf{p}_i + \dfrac{\partial \mathbf{\Phi}_{ij}}{\partial \mathbf{u}_i}^T (\mathbf{z}_i - \mathbf{z}_j)$ $\mathbf{p}_j = \mathbf{p}_j + \dfrac{\partial \mathbf{\Phi}_{ij}}{\partial \mathbf{u}_j}^T (\mathbf{z}_i - \mathbf{z}_j)$

The vectors $\mathbf{Z}$ and $\mathbf{P}$ are inverted in the adjoint case because the result given in the table is valid also in the case that $\mathbf{R}$ is not the residuals vector but a general vector, which is obtained by discretizing a conservation law and can be assembled on the edges. For instance, it could be used for the differentiation of the Green-Gauss gradient $\partial \nabla \mathbf{V} / \partial \mathbf{V}$, which is a rectangular matrix and requires the vector $\mathbf{P}$ to have one component but the vector $\mathbf{Z}$ to have a number of components equal to the number of space dimensions. The Green-Gauss gradient has the flux function $1/2(v_i + v_j)\,\mathbf{n}_{ij}$, which is linear with respect to the left and right states, v_i and v_j, respectively. In Table 3.2 edge-based assemblies that involve the Green-Gauss gradient, and which are obtained by using the result of Table 3.1, are described.

When discussing the matrix-vector products required to solve the linear system, the linearized assembly of Table 3.1 was already given in Eq. (2.55), however without addressing its derivation. Note that the above assemblies are much faster to complete than the second-order ones of Eqs. (3.17) and (3.18). In fact, since the reconstruction is absent, there is no need to loop over the stencils of the nodes. As a consequence, it can be realized by using only the edge-based data structure, without using the stencil information $\mathcal{N}_i$ and $\mathcal{N}_j$ for each edge ij.

In the next section, the linearized and transposed assemblies of Table 3.1 will be used to assemble the second order matrix-vector products in a two-pass construction.

3.3 Approximate sensitivity: two-pass construction

The one-pass construction described in the previous section is not very efficient because, as shown by Eqs. (3.17) and (3.18), the distance-one stencils of the nodes that share the edge must be visited during the loops. The reason for visiting the stencil nodes is that Eqs. (3.7) and (3.8) contain more than one matrix with a sparsity pattern equal to the mesh graph, i.e., they contain $\partial\mathbf{R}/\partial\mathbf{H}$ and the reconstruction operators $\partial\mathbf{U}_L/\partial\mathbf{U}$ and $\partial\mathbf{U}_R/\partial\mathbf{U}$. The latter operators have sparsity patterns equal to the mesh graph because of the presence of the gradients and of the limiters. In the case of first-order accuracy, i.e., no gradient and no limiters, the matrices $\partial\mathbf{U}_L/\partial\mathbf{U}$ and $\partial\mathbf{U}_R/\partial\mathbf{U}$ would be diagonal and it would be possible to use the edge-based assembly of Table 3.1.

An efficient assembly may be achieved by considering a multi-pass construction. The latter breaks the assembly in a number of passes equal to the quantities that are involved in the residual evaluation and that needs an edge-based assembly to be computed. Thus, for the reconstruction scheme considered in this work, three steps would be required because, in addition to the residuals, one has the gradients and the limiters. If the assembly is broken this way, at each pass only one non-diagonal sparse matrix would be present and an edge-based assembly similar to the ones given in Table 3.1 could be used.

Therefore, in order to apply the multi-pass idea, it remains to break the assembly in parts. For this purpose, one can consider the dependencies of the reconstructed states in explicit form and then apply the chain rule. I.e,

$$\mathbf{U}_L = \mathbf{U}_L(\mathbf{V}_L(\mathbf{V}, \nabla\mathbf{V}, \mathbf{\Sigma}))\,, \quad \mathbf{U}_R = \mathbf{U}_R(\mathbf{V}_R(\mathbf{V}, \nabla\mathbf{V}, \mathbf{\Sigma}))\,. \tag{3.36}$$

The vector $\mathbf{V} = \mathbf{V}(\mathbf{U})$ contains the primitive variables. The vectors $\mathbf{V}_L$ and $\mathbf{V}_R$ are the reconstructed left and right states, in the form of primitive variables, as computed by the MUSCL reconstruction. The vector $\nabla\mathbf{V} = [\nabla\mathbf{v}_1, \nabla\mathbf{v}_2, ..., \nabla\mathbf{v}_N]^T$ contains the primitive variables gradients whereas the vector $\mathbf{\Sigma} = [\sigma_1, \sigma_2, ..., \sigma_N]^T$ contains the slope limiters. The gradient $\nabla\mathbf{V}$ depends on the primitive variables whereas the limiters depend on both the primitive variables and the gradients.

The matrix-vector products obtained by taking into account the explicit dependency of Eq. (3.36) may be obtained by means of the chain rule. Since the limiters also depend on the gradient, the result of the derivation is slightly involved. However, as will be shown in the next chapter, there are good reasons for assuming that the limiters may be neglected. By treating them as constants, an appreciable simplification is achieved.

The linearized matrix-vector product obtained by applying the chain rule reads

$$\frac{\partial \mathbf{R}}{\partial \mathbf{U}}\mathbf{P} = \mathbf{Q}_1 \frac{\partial \mathbf{V}}{\partial \mathbf{U}}\mathbf{P} + \mathbf{Q}_2 \frac{\partial \nabla \mathbf{V}}{\partial \mathbf{V}}\frac{\partial \mathbf{V}}{\partial \mathbf{U}}\mathbf{P}\,, \tag{3.37}$$

where $\mathbf{Q}_1$ is a square matrix of size $N \times N$ and $\mathbf{Q}_2$ is a matrix of size $N \times N$ for each space dimension. They are defined as

$$\begin{aligned}
\mathbf{Q}_1 &= \frac{\partial \mathbf{R}}{\partial \mathbf{H}}\left(\frac{\partial \mathbf{H}}{\partial \mathbf{U}_L}\frac{\partial \mathbf{U}_L}{\partial \mathbf{V}_L}\frac{\partial \mathbf{V}_L}{\partial \mathbf{V}} + \frac{\partial \mathbf{H}}{\partial \mathbf{U}_R}\frac{\partial \mathbf{U}_R}{\partial \mathbf{V}_R}\frac{\partial \mathbf{V}_R}{\partial \mathbf{V}}\right)\,, \\
\mathbf{Q}_2 &= \frac{\partial \mathbf{R}}{\partial \mathbf{H}}\left(\frac{\partial \mathbf{H}}{\partial \mathbf{U}_L}\frac{\partial \mathbf{U}_L}{\partial \mathbf{V}_L}\frac{\partial \mathbf{V}_L}{\partial \nabla \mathbf{V}} + \frac{\partial \mathbf{H}}{\partial \mathbf{U}_R}\frac{\partial \mathbf{U}_R}{\partial \mathbf{V}_R}\frac{\partial \mathbf{V}_R}{\partial \nabla \mathbf{V}}\right)\,,
\end{aligned} \tag{3.38}$$

respectively. The transposition of the above matrices gives the transposed matrix-vector product, which reads

$$\frac{\partial \mathbf{R}}{\partial \mathbf{U}}^T\mathbf{P} = \frac{\partial \mathbf{V}}{\partial \mathbf{U}}^T\mathbf{Q}_1^T\,\mathbf{P} + \frac{\partial \mathbf{V}}{\partial \mathbf{U}}^T\frac{\partial \nabla \mathbf{V}}{\partial \mathbf{V}}^T\mathbf{Q}_2^T\,\mathbf{P}\,. \tag{3.39}$$

The matrices $\partial \mathbf{R}/\partial \mathbf{H}$, $\partial \mathbf{H}\ /\partial \mathbf{U}_L$ and $\partial \mathbf{H}\ /\partial \mathbf{U}_R$ are the same as defined in the previous section. The other matrices are defined as follows,

> $\partial \mathbf{V}_L/\partial \mathbf{V}$ and $\partial \mathbf{V}_R/\partial \mathbf{V}$ are block diagonal matrices of size $N \times E$, which are filled by the sub-matrices $\partial \hat{\mathbf{v}}_i/\partial \mathbf{v}_i$ and $\partial \hat{\mathbf{v}}_j/\partial \mathbf{v}_j$, respectively. As will be shown below, the latter sub-matrices are identities. $\partial \mathbf{V}_L/\partial \nabla \mathbf{V}$ and $\partial \mathbf{V}_R/\partial \nabla \mathbf{V}$ are matrices of size $N \times E$ for each space dimension. They are filled by the sub-matrices $\partial \hat{\mathbf{v}}_i/\partial \nabla \mathbf{v}_i$ and $\partial \hat{\mathbf{v}}_j/\partial \nabla \mathbf{v}_j$, respectively. The four matrices represent the differentiation of the reconstruction formulation according to the explicit definition given in Eq. (3.36) and to the assumption of constant limiters.
>
> $\partial \mathbf{U}_L/\partial \mathbf{V}_L$ and $\partial \mathbf{U}_R/\partial \mathbf{V}_R$ are block diagonal matrices of size $E \times E$ that are filled by the sub-matrices $\partial \hat{\mathbf{u}}_i/\partial \hat{\mathbf{v}}_i$ and $\partial \hat{\mathbf{u}}_j/\partial \hat{\mathbf{v}}_j$, respectively. The latter sub-matrices are transformation matrices between primitive and conservative variables and are evaluated using Eq. (2.6). $\partial \mathbf{V}/\partial \mathbf{U}$ is a diagonal matrix of size $N \times N$ that operates the transformation from conservative to primitive variables. Thus, it is filled by the sub-matrices $\partial \mathbf{v}_i/\partial \mathbf{u}_i$, which can be evaluated using Eq. (2.7).
>
> $\partial \nabla \mathbf{V}/\partial \mathbf{V}$ is the differentiation of the gradient formulation. It is a sparse block matrix of size $N \times N$ for each space dimension. In fact, recall that the gradient is a multi-component vector, with a number of components equal to the number of space dimensions. The ith sub-matrix of $\partial \nabla \mathbf{V}/\partial \mathbf{V}$ is of the type $\partial \nabla \mathbf{v}_i/\partial \mathbf{v}_k$, and it is non-zero only for $k \in \mathcal{N}_i$. Each edge of the mesh contributes to the topology of the operator as in Fig. 3.1.

The edge-based assemblies of the matrix-vector products in Eqs. (3.37) and (3.39) are described next.

3.3.1 Two-pass assembly for the linearized problem

Although the matrix-vector product in Eq. (3.37) seems to be very involved, it is actually relatively easy to perform. The differentiated gradient operator contains only metric terms, i.e., it only depends on the geometry of the mesh. The consequence is that the product of the differentiated gradient operator with a vector is equivalent to the gradient of the vector, i.e.,

$$\frac{\partial \nabla \mathbf{V}}{\partial \mathbf{V}} \frac{\partial \mathbf{V}}{\partial \mathbf{U}} \mathbf{P} = \nabla \left(\frac{\partial \mathbf{V}}{\partial \mathbf{U}} \mathbf{P} \right) .$$

Thus, the gradient routine may be used to compute the matrix-vector product. Another important simplification comes from the reconstruction operator. In fact, in the assumption of constant limiters, the matrix form of the reconstruction operator in Eq. (3.21) may be written as

$$\mathbf{V}_L = \frac{\partial \mathbf{V}_L}{\partial \mathbf{V}} \mathbf{V} + \frac{\partial \mathbf{V}_L}{\partial \nabla \mathbf{V}} \nabla \mathbf{V} .$$

Therefore, if one defines the scaled vector $\mathbf{P}^* = [\partial \mathbf{V} / \partial \mathbf{U}] \mathbf{P}$ and substitutes the two results obtained above into the matrix-vector product in Eq. (3.37), the latter product becomes

$$\frac{\partial \mathbf{R}}{\partial \mathbf{U}} \mathbf{P} = \frac{\partial \mathbf{R}}{\partial \mathbf{H}} \left(\frac{\partial \mathbf{H}}{\partial \mathbf{U}_L} \frac{\partial \mathbf{U}_L}{\partial \mathbf{V}_L} \mathbf{P}_L^* + \frac{\partial \mathbf{H}}{\partial \mathbf{U}_R} \frac{\partial \mathbf{U}_R}{\partial \mathbf{V}_R} \mathbf{P}_R^* \right) , \qquad (3.40)$$

where $\mathbf{P}_L^*$ and $\mathbf{P}_R^*$ are the reconstructed versions of the vector $\mathbf{P}^*$. In practice, the above matrix-vector product is carried out as follows:

Step 1 The gradient routine is run with the input vector $\mathbf{P}$ and computes the gradient $\nabla \mathbf{P}^*$. The edge-based loop is described in Table 3.2 for the Green-Gauss gradient. For simplicity, the loop does not include boundary terms.

Step 2 The differentiated residual routine is run, which performs the MUSCL reconstruction on the vector $\mathbf{P}^*$. Once the reconstructed vector $\mathbf{P}_L^*$ and $\mathbf{P}_R^*$ are computed, the matrix-vector product can be assembled by using the linearized assembly given in Table 3.1. However, because the vectors $[\partial \mathbf{U}_L / \partial \mathbf{V}_L] \mathbf{P}_L^*$ and $[\partial \mathbf{U}_R / \partial \mathbf{V}_R] \mathbf{P}_R^*$ are present in Eq. (3.40), the sub-vectors $\mathbf{p}_i$ and $\mathbf{p}_j$ that appear in the linearized loop of Table 3.1 must be replaced by the sub-vectors $[\partial \hat{\mathbf{u}}_i / \partial \hat{\mathbf{v}}_i] \hat{\mathbf{p}}_i$ and $[\partial \hat{\mathbf{u}}_j / \partial \hat{\mathbf{v}}_j] \hat{\mathbf{p}}_j$, where $\hat{\mathbf{p}}_i$ and $\hat{\mathbf{p}}_j$ are the components of $\mathbf{P}_L^*$ and $\mathbf{P}_R^*$, respectively.

3.3.2 Two-pass assembly for the adjoint problem

As can be seen from Eq. (3.39), the differentiated residual routine must be run before the differentiated gradient routine. Thus, as already mentioned, the transposition inverts the order with which operations are performed within the code. A further complication is that the transposed differentiated gradient routine takes as input a multi-component vector, with the number of components equal to the number of space dimensions, and gives as output a vector. The different sizes of the input and the output makes sense because $\partial \nabla \mathbf{V}/\partial \mathbf{V}$ is a rectangular matrix. The implementation may not be straightforward as in the case of a square matrix, e.g., the first-order residual Jacobian. In practice, the matrix-vector product in Eq. (3.39) is carried out in two steps:

Step 1 The differentiated residual routine is run, which computes the two vectors $\mathbf{G} = [\partial \mathbf{V}/\partial \mathbf{U}]^T \mathbf{Q}_1^T \mathbf{P}$ and $\bar{\mathbf{G}} = \mathbf{Q}_2^T \mathbf{P}$, given the adjoint vector $\mathbf{P}$ as input. Note that in the case of first order spatial accuracy only the vector $\mathbf{G}$ needs to be computed. The multi-component vector $\bar{\mathbf{G}}$ can be thought of as a kind of gradient that, due to the transposition, is an output, rather than an input, of the differentiated residual routine. The assembly of the two vectors may be performed by means of an edge-based loop similar to that of Table 3.1. In order to simplify the notation, define the two sub-vectors

$$\mathbf{f}_i = \frac{\partial \hat{\mathbf{u}}_i}{\partial \hat{\mathbf{v}}_i}^T \frac{\partial \mathbf{\Phi}_{ij}}{\partial \mathbf{u}_i}^T (\mathbf{p}_i - \mathbf{p}_j) \,, \qquad \mathbf{f}_j = \frac{\partial \hat{\mathbf{u}}_j}{\partial \hat{\mathbf{v}}_j}^T \frac{\partial \mathbf{\Phi}_{ij}}{\partial \mathbf{u}_j}^T (\mathbf{p}_i - \mathbf{p}_j) \,.$$

Using the above vectors, the components of $\mathbf{G}$ and $\bar{\mathbf{G}}$ are assembled on the edge as

$$\begin{aligned} \mathbf{g}_i &= \mathbf{g}_i + [\partial \mathbf{v}_i/\partial \mathbf{u}_i]^T \mathbf{f}_i \,, \\ \mathbf{g}_j &= \mathbf{g}_j + [\partial \mathbf{v}_j/\partial \mathbf{u}_j]^T \mathbf{f}_j \,, \\ \bar{\mathbf{g}}_i &= \bar{\mathbf{g}}_i + [\partial \hat{\mathbf{v}}_i/\partial \nabla \mathbf{v}_i]^T \mathbf{f}_i \,, \\ \bar{\mathbf{g}}_j &= \bar{\mathbf{g}}_j + [\partial \hat{\mathbf{v}}_j/\partial \nabla \mathbf{v}_j]^T \mathbf{f}_j \,, \quad (ij = 1, E) \,. \end{aligned} \tag{3.41}$$

The elements of the sub-matrices $\partial \hat{\mathbf{v}}_i/\partial \nabla \mathbf{v}_i$ and $\partial \hat{\mathbf{v}}_j/\partial \nabla \mathbf{v}_j$ are obtained by taking into account the definition of the reconstruction. E.g., for the reconstruction of Eq. (3.21) one has that $\partial \hat{v}_i/\partial \nabla v_i = \Delta \mathbf{x}_{ij}/2$. Note that according to the definition of the matrix-vector product in Eq. (3.39) and to that of the matrices given in Eq. (3.38), the definition of the sub-vectors $\mathbf{f}_i$ and $\mathbf{f}_j$ should also include the sub-matrices $[\partial \hat{\mathbf{v}}_i/\partial \mathbf{v}_i]^T$ and $[\partial \hat{\mathbf{v}}_j/\partial \mathbf{v}_j]^T$, respectively. However, by definition of the MUSCL reconstruction, those sub-matrices are identities. For instance, consider the reconstruction of Eq. (3.21): $\partial \hat{v}_i/\partial v_i = 1$.

Step 2 The differentiated gradient routine, given the multi-component vector $\bar{\mathbf{G}}$ as input, generates the vector $\bar{\mathbf{P}} = [\partial \mathbf{V}/\partial \mathbf{U}^T][\partial \nabla \mathbf{V}/\partial \mathbf{V}^T]\bar{\mathbf{G}}$ as output. The edge-based assembly of the vector is described in Table 3.2. The table compares the assemblies of the linearized and transposed operators. As can be seen, given a vector of component equal to the number of space dimensions, the loop computes a vector of one component. The duality between the two types of assemblies in the table is evident. At the end of the assembly, the components of the computed vector $\bar{\mathbf{P}}$ are added to that of the vector $\mathbf{G}$, thus completing the assembly of $[\partial \mathbf{R}/\partial \mathbf{U}]^T \mathbf{P}$.

Table 3.2: *Pseudo-code to assemble the linearized and adjoint matrix-vector products. The matrix is the differentiated Green-Gauss gradient operator.*

Linearized - $\nabla \mathbf{P}^* = \frac{\partial \nabla \mathbf{V}}{\partial \mathbf{V}} \frac{\partial \mathbf{V}}{\partial \mathbf{U}} \mathbf{P}$	**Adjoint** - $\bar{\mathbf{P}} = \frac{\partial \mathbf{V}}{\partial \mathbf{U}}^T \frac{\partial \nabla \mathbf{V}}{\partial \mathbf{V}}^T \bar{\mathbf{G}}$
Set $\mathbf{v}_i = [\partial \mathbf{v}_i / \partial \mathbf{u}_i] \mathbf{p}_i \quad \forall i$	Scale $\nabla \mathbf{v}_i = \bar{\mathbf{g}}_i / V_i \quad \forall i$
Loop over the Edges, $ij = 1, E$	*Loop over the edges,* $ij = 1, E$
$\nabla \mathbf{v}_i = \nabla \mathbf{v}_i + \frac{(\mathbf{v}_i + \mathbf{v}_j)}{2} \mathbf{n}_{ij}$	$\mathbf{v}_i = \mathbf{v}_i + \frac{\nabla \mathbf{v}_i - \nabla \mathbf{v}_j}{2} \mathbf{n}_{ij}$
$\nabla \mathbf{v}_j = \nabla \mathbf{v}_j - \frac{(\mathbf{v}_i + \mathbf{v}_j)}{2} \mathbf{n}_{ij}$	$\mathbf{v}_j = \mathbf{v}_j + \frac{\nabla \mathbf{v}_i - \nabla \mathbf{v}_j}{2} \mathbf{n}_{ij}$
Scale $\nabla \mathbf{v}_i = \nabla \mathbf{v}_i / V_i \quad \forall i$	*Set* $\bar{\mathbf{p}}_i = [\partial \mathbf{v}_i / \partial \mathbf{u}_i]^T \mathbf{v}_i \quad \forall i$
Output $\nabla \mathbf{V} \equiv \nabla \mathbf{P}^*$	Output $\bar{\mathbf{P}}$

3.4 Solution of the sensitivity equations

The linearized and adjoint problems of Eqs. (3.2) and (3.3) are both linear. Thus, theoretically, they may be solved by means of a linear solver, for instance a direct or an iterative solver. In practice, both types of solvers are unsuitable. The direct solver would not be the best choice because of the large amount of memory involved, especially for 3D cases. An iterative solver, similar to the one used for the solution of the linear system in the flow solver, would be likely to stall because of the poor diagonal dominance of the residual Jacobian. In fact, the residual Jacobian that appears in the equations is that of a second-order scheme, which is not diagonally dominant like the Jacobian of a first-order scheme.

The poor diagonal dominance of the Jacobian can be verified by means of numerical experiments. In the present work numerical experiments have been performed in which the linearized problem of Eq. (3.2) has been solved by means of Jacobi iterations. The flow conditions were transonic and the sensitivity was computed with respect to the angle of attack. It appeared that when the Jacobian in the matrix-vector product of Eq. (3.37) only included the first-order contribution, i.e., the term $\mathbf{Q}_1[\partial \mathbf{V}/\partial \mathbf{U}]$, the Jacobi iterations converged. Instead, when the Jacobian also included the second-order contribution, i.e., the term $\mathbf{Q}_2[\partial \nabla \mathbf{V}/\partial \mathbf{V}][\partial \mathbf{V}/\partial \mathbf{U}]$, the Jacobi iterations did not converge. The lack of convergence in the latter case shows that the second-order contribution reduces the diagonal dominance. The cause for the reduced diagonal dominance may be found in the gradient operator $[\partial\nabla\mathbf{V}/\partial\mathbf{V}]$, which is filled by elements of the type $\partial\nabla\mathbf{v}_i/\partial\mathbf{v}_k$. The latter elements, for instance in the case of the Green-Gauss gradient, become zero when they are located on the diagonal, i.e., for $i = k$. In fact, as shown by Eq. (3.23), the diagonal terms are expressed as summation of integrated normals around closed control volumes. As such, they have to be zero, as shown by Eq. (2.14).

3.4.1 Implicit pseudo-time stepping solution

An interesting strategy to solve the equations is that of using the same implicit scheme in pseudo-time as the one that is used for the flow equations [7, 37, 84]. In practice, the sensitivity equations may be treated as non-linear equations. The concept is best explained by defining two residuals, $\mathbf{R}_p$ for the primal problem of Eq. (3.2) and $\mathbf{R}_d$ for the dual problem of Eq. (3.3), reading

$$\mathbf{R}_p(\mathbf{U}_\alpha) = \frac{\partial\mathbf{R}}{\partial\mathbf{U}}\mathbf{U}_\alpha + \frac{\partial\mathbf{R}}{\partial\alpha}, \qquad \mathbf{R}_d(\boldsymbol{\Lambda}) = \frac{\partial\mathbf{R}}{\partial\mathbf{U}}^T\boldsymbol{\Lambda} - \frac{\partial J}{\partial\mathbf{U}}^T. \tag{3.42}$$

$\mathbf{U}_\alpha$ stands for the flow sensitivity $\partial\mathbf{U}/\partial\alpha$. Note that the subscript i used in Eq. (3.3) is temporarily dropped for convenience. Below it will be used again. Instead, the subscript j used in Eq. (3.2) may be dropped permanently because in the present work the primal problem is only solved for a single parameter.

A pseudo-time contribution may be added to the above equations in order to cast these in the same form as the flow problem of Eq. (2.38), i.e.,

$$\mathbf{D}\frac{d\,\mathbf{U}_\alpha}{d\,t} + \mathbf{R}_p = \mathbf{0}, \qquad \mathbf{D}\frac{d\,\boldsymbol{\Lambda}}{d\,t} + \mathbf{R}_d = \mathbf{0}. \tag{3.43}$$

The same solution strategy devised in Section 2.4 for the flow problem may be used here for the above equations. After discretization of the time contribution and application of the defect correction method, the primal equation becomes

$$\left(\mathbf{D}_t + \frac{\partial\widetilde{\mathbf{R}}}{\partial\mathbf{U}}\right)^n (\mathbf{U}_\alpha^{n+1} - \mathbf{U}_\alpha^n) = -\left(\frac{\partial\mathbf{R}}{\partial\mathbf{U}}\mathbf{U}_\alpha^n + \frac{\partial\mathbf{R}}{\partial\alpha}\right). \tag{3.44}$$

In practice, the same solver of the flow equations may be used, the only thing being different is the right-hand side term. For the adjoint equation the same procedure gives

$$\left(\mathbf{D}_t + \frac{\partial\widetilde{\mathbf{R}}}{\partial\mathbf{U}}^T\right)^n (\boldsymbol{\Lambda}_i^{n+1} - \boldsymbol{\Lambda}_i^n) = -\left(\frac{\partial\mathbf{R}}{\partial\mathbf{U}}^T\boldsymbol{\Lambda}_i^n - \frac{\partial J_i}{\partial\mathbf{U}}^T\right). \tag{3.45}$$

The above equation, apart from having a different right-hand side, contains transposed Jacobians. Thus, the linear solver must be modified appropriately. In practice, the modifications amount only to minor changes of the original routines.

For both Eqs. (3.44) and (3.45) the Jacobians involved on the left- and on the right-hand sides are constants with respect to the solution variables. In fact, the residual

Jacobians depend only on the conservative flow variables. Therefore, when the storage option is used, time saving is achieved by computing the Jacobian terms required for the matrix-vector products and for preconditioning only at the beginning of the pseudo-time iterations. If the matrix-free option is chosen, the Jacobians are recomputed at each pseudo-time step.

3.4.2 Simultaneous solution of the adjoint equations

Aerodynamic design often requires the solution of constrained problems. E.g., lift constrained drag minimization. Those problems are best solved when the gradients of all the functionals are available. However, solving more than one adjoint equation may become computationally expensive and may be seen as a limitation of the adjoint method [38]. For this reason, constraints are often included as penalty terms in the objective function and only one adjoint is solved. The penalty approach seems to be convenient but may lead to ill-conditioning of the optimization problem [126].

In the present work constrained optimization algorithms are preferred. In order to efficiently compute the gradients that are needed by those algorithms, a strategy that solves several adjoint equations simultaneously is devised. The efficiency of the method is due to the presence of the same Jacobian for several adjoint equations. The Jacobian terms are relatively expensive to compute. Thus, once the terms are available, it makes sense to perform all the multiplications required by the preconditioning and by the matrix-vector products in one shot.

The matrix-vector products for the solution of the linear system in the adjoint case are computed with a slightly modified version of the matrix-vector product given in Table 3.1. In fact, for each edge there are several sub-vectors $\mathbf{p}_{i,k}$ and $\mathbf{p}_{j,k}$, for $k = 1, N_J$, which must be multiplied by the flux-Jacobians $\partial \mathbf{\Phi}_{ij}/\partial \mathbf{u}_i$ and $\partial \mathbf{\Phi}_{ij}/\partial \mathbf{u}_j$. Thus, it is necessary to add a nested loop within the edge loop. Also the assembly of the right-hand side of Eq. (3.45) must be realized in a similar way by adding nested loops in the procedure described to assemble the matrix-vector product of Eq. (3.39)

Regarding the preconditioner, taking into account the transposition, it becomes

$$\mathbf{P}_{\mathrm{M}} = (\mathbf{D}_{\mathrm{M}}^{T} + \mathbf{U}_{\mathrm{M}}^{T})\, \mathbf{D}_{\mathrm{M}}^{-T}\, (\mathbf{D}_{\mathrm{M}}^{T} + \mathbf{L}_{\mathrm{M}}^{T})\,. \tag{3.46}$$

In order to apply the above preconditioner, the forward and the backward sweeps of Eqs. (2.56) and (2.57) must be modified. Apart from the transposition, a nested loop must be added because of the presence of multiple right-hand sides. Thus, the forward sweep becomes

$$\Delta \mathbf{z}_{i,k}{}^{*} = \mathbf{D}_{\mathrm{M}}{}_{i}^{-T} \left(\mathbf{R}_{\mathrm{L}\,i}^{k} - \sum_{j \in \mathcal{L}_i} \mathbf{U}_{\mathrm{M}}{}_{ij}^{T} \Delta \mathbf{z}_{j,k}{}^{*} \right), \qquad (i = 1, N;\, k = 1, N_J)\,, \tag{3.47}$$

whereas the backward sweep becomes

$$\Delta \mathbf{z}_{i,k} = \Delta \mathbf{z}_{i,k}{}^{*} - \mathbf{D}_{\mathrm{M}}{}_{i}^{-T} \sum_{j \in \mathcal{U}_i} \mathbf{L}_{\mathrm{M}}{}_{ij}^{T} \Delta \mathbf{z}_{j,k}\,, \qquad (i = N, 1;\; k = 1, N_J)\,. \tag{3.48}$$

The time saving that may be realized by implementing the simultaneous solution depends on whether the storage or matrix-free option is used. The saving is in fact more appreciable when the matrix-free option is used, i.e., when the expensive matrix terms are recomputed each time.

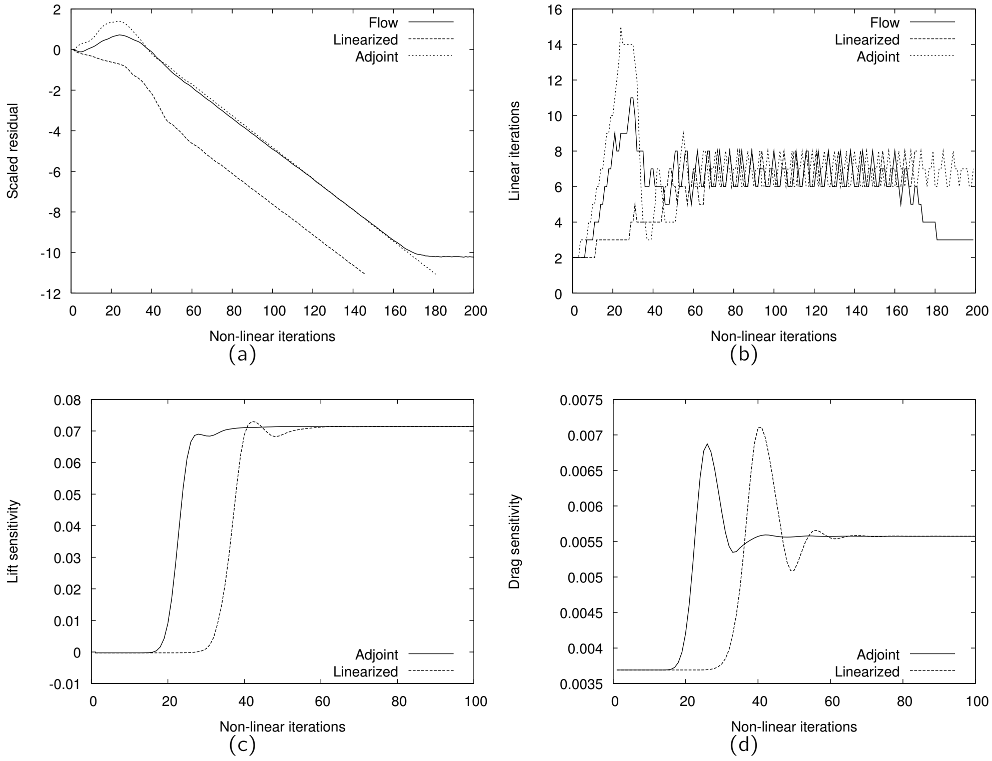
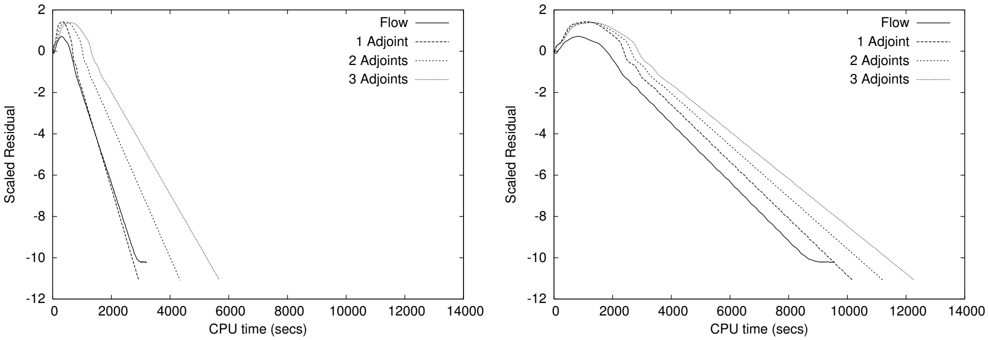


Figure 3.2: *ONERA-M6 wing. Convergence histories of flow, adjoint and linearized solver: (a) residual; (b) linear versus non-linear iterations; (c) lift sensitivity; (d) drag sensitivity.*

3.4.3 Numerical results of the solution scheme

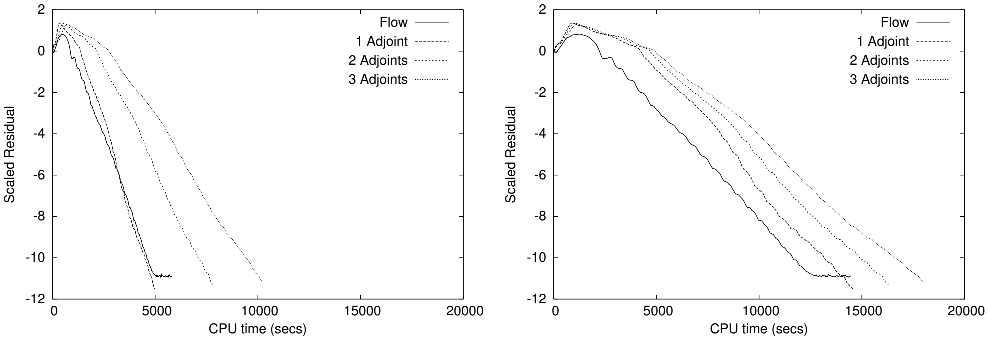
Numerical results of the solution strategy presented in this section are shown for two cases already discussed in Section 2.5.2: the ONERA-M6 wing at $M_\infty = 0.84$ and $\alpha = 3.06°$ and the DLR-F6 wing-body at $M_\infty = 0.75$ and $\alpha = 0.5°$.

Figure 3.2 shows the convergence histories of the flow, linearized and adjoint solvers for the ONERA-M6 wing case. These are presented in terms of non-linear iterations, i.e., in terms of the pseudo-time steps. As can be seen from Fig. 3.2a, the scaled residual of the flow solver, which is defined as $L_2(\mathbf{R}^n)/L_2(\mathbf{R}^0)$, is reduced by ten orders of magnitude. The linearized and adjoint solvers converge at the same rate as the flow solver. In terms of linear iterations, see Fig. 3.2b, an average of 6 of these are required for each non-linear iteration. A maximum of 11 linear iterations is required around the 30*th* non-linear iteration. In general, especially for 3D computations, the start-up phase of the implicit solution procedure appears to be a critical one. When the solution stabilizes itself, the average number of linear iterations required by the three solvers, see Fig. 3.2b, appears to be the same. Compared to the flow solver, the adjoint solver requires more iterations in the start-up phase whereas the linearized solver requires less.



(a) (b)

Figure 3.3: *ONERA-M6 wing. Flow and adjoint convergence histories: (a) storage of the preconditioner; (b) matrix-free preconditioning.*



(a) (b)

Figure 3.4: *DLR-F6 wing-body. Flow and adjoint convergence histories: (a) storage of the preconditioner; (b) matrix-free preconditioning.*

Convergence histories of the flow and the adjoint solvers for the two configurations are shown in Figs. 3.3 and 3.4, respectively. In order to compare the efficiency of the adjoint solver with that of the flow solver, these convergence histories are plotted in terms of CPU time. For the convergence histories of Figs. 3.3a and 3.4a, the elements $\mathbf{L}_\mathrm{M}$, $\mathbf{U}_\mathrm{M}$ and $\mathbf{D}_\mathrm{M}$ of the matrix $\mathbf{M}$ have been stored. It appears that the adjoint solver residual overlaps with the flow solver residual, i.e., one adjoint solution requires the same amount of time as one flow solution. This positive result can be explained as follows. The larger amount of CPU time required to assemble the right-hand side of Eq. (3.45), compared to Eq. (2.42), is cancelled out by the time saved on the left-hand side for the computation of the matrix elements. In fact, as already mentioned, these elements are computed only once for the adjoint. In the case of multiple adjoint solutions, looking at these pictures it appears that the simultaneous solution of two adjoints saves 25% of CPU time compared to two sequential solutions. In the case of three adjoint solutions, the CPU time saving rises to 33%.

The convergence histories of Figs. 3.3b and 3.4b have been produced using the matrix-free option, i.e., $\mathbf{L}_\mathrm{M}$, $\mathbf{U}_\mathrm{M}$ are always computed on-the-fly for both matrix-vector products and preconditioning. Clearly, there is a penalty in terms of time. As can be seen, the CPU time required to converge the flow is around three times more than in the storage case of Figs. 3.3a and 3.4a. A single adjoint solution requires in this case around 10% more CPU time than the flow solution. In fact, for the adjoint solution, on-the-fly computation means that the advantage of having the same $\mathbf{L}_\mathrm{M}$ and $\mathbf{U}_\mathrm{M}$ for all iterations is lost in this case. Nevertheless, on-the-fly computations allow maximal exploitation of the advantages given by simultaneous adjoint solutions. As can be seen from Figs. 3.3b and 3.4b, two simultaneous adjoint solutions give around 45% time saving compared to sequential solutions. In the case of three adjoint solutions, the time saving rises to 60%.

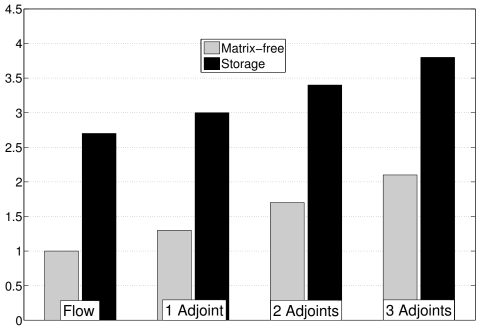


Figure 3.5: *Memory usage.*

In terms of memory requirements, the solution of the adjoint problem requires an amount of memory that is of the same order as that required by the flow solver. Figure 3.5 shows these requirements for storage and matrix-free computations. Since the picture is valid for both configurations, the memory is normalized by the memory required by the flow solver for matrix-free computations. As can be seen, the flow solver requires 2.7 times more memory when the matrix elements are stored than in the case of matrix-free computations. When the matrix elements are stored, the adjoint takes only 11% more memory than the flow solver. This memory increase rises to 40% in the case of three simultaneous adjoint solutions. In the case of matrix-free computations, the difference in memory use between the flow solver and the adjoint solver is larger. For instance, three adjoints simultaneously require slightly more than two times the memory required by the flow solver. The reason is that when the matrix-free option is active the memory is only used to store the arrays of flow, adjoint and working variables. And the memory required by each array is about one order of magnitude smaller than that required by the Jacobian. Hence, the ratio of the memory required by the adjoint solver to that required by the flow solver is larger.

3.5 Accuracy of the discrete sensitivity

The accuracy of the discrete sensitivity formulation must be verified in order to see if errors are present in the implementation. The sensitivity of the linearized code may be verified directly whereas that of the adjoint code must be verified indirectly, by using also the linearized code. Therefore, the verification of the adjoint code implies the verification of the linearized code first.

In the case of the differentiation being carried out exactly, the discrete sensitivities must coincide with that of the original flow solver. Thus, comparing the two sensitivities may be the best way to verify the correctness. However, obtaining the sensitivity of the original flow solver may not be easy.

In the case of an approximate sensitivity code, it is only possible to check the differences between the approximate sensitivity and the sensitivity of the original solver. Fortunately, there is still a way to check exactly if the two approximate codes, linearized and adjoint, are consistent with each other.

3.5.1 Verification of the exact linearized code

In the present work the correctness of the linearized code has been verified by using three different means: (*i*) the quadratic convergence rate property of Newton iterations; (*ii*) the comparison with a sensitivity code generated by an Automatic Differentiation tool; and (*iii*) the well known Fréchet derivative.

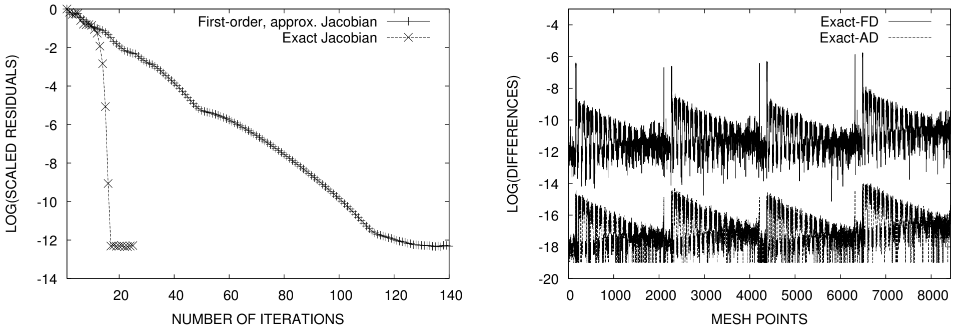
Quadratically convergent Newton iterations

The quadratic convergence rate property of Newton iterations has never been used in the context of sensitivity analysis. It checks the correctness of the Jacobian indirectly, employing the latter in an implicit pseudo-time stepping procedure (see Section 2.4), which becomes Newton's method for infinitely large time steps. If the Jacobian is exact, quadratic convergence is observed.

As can be seen from the residual history in Fig. 3.6a, the exact Jacobian attains quadratic convergence and in only 5 iterations the residual is reduced by 10 orders of magnitude. In the same plot also the convergence of the approximate Jacobian is shown, it converges linearly. Newton iterations are a very reliable method for checking the correctness of the code. Quadratic convergence is not obtained if, for instance, limiters are ignored (e.g., if they are considered to be constant in the differentiation, see Eq. (3.22)) or if a programming error is present.

Automatic Differentiation

One more check to verify the exactness of the Jacobian has been carried out by using Automatic Differentiation. The residual code has been differentiated in forward mode using the Automatic Differentiation (AD) tool Tapenade [3]. Differentiated code that is capable of computing exact matrix-vector products has been generated. Figure 3.6b shows the differences of the matrix-vector product implemented here compared with the AD code. As can be seen, the differences between the exact code and the AD code are smaller than 10^{-14}. For some nodes, the differences appeared to be exactly zero and a threshold of 10^{-19} has been used in the picture.



(a) Newton's method

(b) Matrix-vector product differences

Figure 3.6: *Exactness of the Jacobian. Results are obtained from the computation of the RAE2822 airfoil at* $M_\infty = 0.73$ *and* $\alpha = 2\,deg$ *on an unstructured* 2107 *nodes mesh.*

Fréchet derivative

In order to check the correctness of the linearized code, a very easy to implement method is the Fréchet derivative (FD), which may be written as

$$\frac{\partial \mathbf{R}}{\partial \mathbf{U}}\mathbf{P} \approx \frac{\mathbf{R}(\mathbf{U} + \epsilon\mathbf{P}) - \mathbf{R}(\mathbf{U})}{\epsilon}, \quad \epsilon \to 0\,. \tag{3.49}$$

The ϵ must be made small in order to improve the accuracy of the approximation. Unfortunately, the smaller it becomes the larger the influence of the truncation and the cancellation errors becomes. In practice, despite the simplicity of the method, it is impossible to obtain very accurate comparisons by using it. Figure 3.6b shows the differences between the Fréchet derivative and the exact code. A suitable value of $\epsilon = 10^{-7}$ has been used. As can be seen, the differences between the code are comprised in the range 10^{-14} to 10^{-6}, definitely above the level of precision revealed by using AD. Although not very accurate, the Fréchet derivative can still be used for preliminary checks of the differentiated code with little effort.

3.5.2 Approximations in the discrete sensitivity

Approximations may be introduced in the differentiation in order to simplify the development of code. Clearly, simplifications have a detrimental effect on the accuracy of the sensitivity. Moreover, one of the advantages of the discrete approach over the continuous one, i.e., that of having the discrete sensitivity to match the sensitivity of the original solver, is lost. The consequence is that the approximate sensitivity may only be compared with the exact one just for the purpose of quantifying the difference between the two.

In the following, three levels of approximations are presented. In practice, approximations imply that the linearized and adjoint problems of Eqs. (3.2) and (3.3) are solved with an approximate Jacobian rather than the exact Jacobian. The following list addresses approximations implemented and tested within the one-pass construction algorithm given in Section 3.2. The approximations are as follows:

$A1-$ The first one is obtained by neglecting the differentiation of the limiter in the reconstruction operator, which means that $\partial\sigma_i/\partial v_k$ is set to zero in Eq. (3.22). Thus, approximations are introduced in the matrices $\partial\mathbf{U}_L/\partial\mathbf{U}$ and $\partial\mathbf{U}_R/\partial\mathbf{U}$ of Eq. (3.8). As can be seen from Section 3.2.2, which shows how the limiter has to be differentiated and included in the assembly, the simplification introduced by neglecting them is appreciable.

$A2-$ The second one is obtained by neglecting the differentiation of the Jacobian matrix in the Roe flux, i.e., it uses Eq. (2.51) instead of Eq. (3.26). In this case approximations are also introduced in the matrices $\partial\mathbf{H}/\partial\mathbf{U}_L$ and $\partial\mathbf{H}/\partial\mathbf{U}_R$ of Eq. (3.8). This approximation saves a lot of human work because it avoids the differentiation presented in Section 3.2.3.

$A3-$ The third approximation is that obtained by ignoring the complete reconstruction operator, which makes the implementation of the adjoint trivial. In fact, the Jacobian is identical to the first-order approximate Jacobian already used for the implicit time stepping scheme described in Sections 2.4.1 and 3.4. In practice, the approximation implies that the Jacobian $[\partial\widetilde{\mathbf{R}}/\partial\mathbf{U}]$ is substituted in the primal and dual problems of Eqs. (3.2) and (3.3).

The two-pass construction assembly presented in Section 3.3 has been implemented according to approximation $A2$ and therefore has constant limiters and approximate numerical fluxes. In Chapter 5 optimization results are presented, which show that approximation A2 may be used safely for engineering purposes.

3.5.3 Effect of the approximations on the gradient

The price to be paid for introducing the approximations described above is a detrimental effect on the accuracy of the computed gradient. The latter aspect has been treated extensively in literature [7, 95, 63, 93]. Here, results are presented for two 2D cases that have already been addressed in Section 2.5.1. The first case is the transonic flow

Table 3.3: *Comparison of exact and approximate discrete adjoint with finite differences for transonic and supersonic NACA0012 cases.*

		Exact	*A*1	$\|\Delta\|$%	*A*2	$\|\Delta\|$%
Transonic	$c_{l\alpha}$	0.272864	0.263342	3.49	0.253712	7.02
	$c_{d\alpha}$	0.017790	0.017263	2.96	0.016785	5.65
	$c_{m\alpha}$	-0.086517	-0.082095	5.11	-0.07757	10.34
Supersonic	$c_{l\alpha}$	0.073866	0.073712	0.21	0.0740152	0.2
	$c_{d\alpha}$	0.016737	0.016685	0.31	0.016679	0.35
	$c_{m\alpha}$	-0.016285	-0.016157	0.78	-0.016152	0.82

around the NACA0012 at $M_\infty = 0.85$ and $\alpha = 1°$, the second case is the supersonic flow around the same airfoil at $M_\infty = 1.2$ and $\alpha = 7°$. The results presented in Table 3.3 show the sensitivities obtained with the exact adjoint code compared with the ones obtained by using the first and the second of the above mentioned approximations, *A*1 and *A*2, respectively. As can be seen, the error in the sensitivity is much larger for the transonic case. Probably, compared to supersonic flow, the mixed elliptic-hyperbolic nature of the equations in transonic flow results in higher non-linearities, which cause the approximations to be less accurate. Although it is not shown in the table, the error increases to a percentage of 10-30% when the third approximation *A*3 is used. The 3D sensitivity code is only implemented according to the approximation *A*2 and an exact version is not available. Thus, the gradient cannot be compared with the exact one. A comparison has been made with a Fréchet derivative, with a suitable increment of 10^{-6}, for the ONERA-M6 wing case also shown in Fig. 3.2. It turns out that the lift sensitivity differs 0.87%, the drag sensitivity differs 0.27% and the pitching moment sensitivity differs 2%.

The magnitude of the gradient error does not give an indication of the effectiveness of the code for optimization purposes. To establish whether an approximation in the sensitivity is acceptable, the error in the gradient is probably not the best indication. It seems more appropriate to consider the effect which the gradient has on the behavior of the optimization process. This aspect has received some attention only recently [32, 22]. The optimization results presented in Chapter 5 are obtained using the exact and the approximate adjoint codes. There it is possible to validate the effectiveness of these codes by directly looking at the solutions of the optimization problem.

3.5.4 Verification of the adjoint code

The adjoint code, for instance that of the residual Jacobian, can be verified by using the identity

$$\mathbf{P}_1{}^T \frac{\partial \mathbf{R}}{\partial \mathbf{U}} \mathbf{P}_2 = \mathbf{P}_2{}^T \frac{\partial \mathbf{R}}{\partial \mathbf{U}}^T \mathbf{P}_1 \,, \tag{3.50}$$

where $\mathbf{P}_1$ and $\mathbf{P}_2$ are two generic vectors. The above identity must be satisfied exactly, i.e., the difference between the left- and the right-hand side must be within machine precision. Rather than only for the residual Jacobian, the above identity may be used to check each part of the code, even a single subroutine. Therefore, the development of the code may be systematically broken into the development of several small blocks, which then are joined together and checked again. Note that in the case of the differentiated gradient operator $\partial\nabla\mathbf{V}/\partial\mathbf{V}$, the vector $\mathbf{P}_2$ must have a number of components equal to the number of space dimensions.

An important aspect of the above identity is that it can be used also to verify the approximate adjoint code. In fact, if approximations are introduced, as long as they are introduced consistently in both the linearized and the adjoint codes, the above identity must still be satisfied to machine accuracy. The same applies to the primal and to the dual sensitivities, which are given in Eqs. (3.1) and (3.4), respectively. The latter sensitivities must be exactly the same, for both the case of exact and approximate differentiation. For instance, convergence histories of the lift and the drag sensitivities of the ONERA-M6 wing case are shown in Figs. 3.2c and 3.2d, respectively. Only the first 100 non-linear iterations are shown. As can be seen, the primal and the linearized sensitivity converge to the same value.

3.6 Visualization of the sensitivity variables

The linearized variables $\mathbf{U}_\alpha$ and the adjoint variables $\mathbf{\Lambda}$ obtained by solving Eqs. (3.44) and (3.45) may be visualized in the computational domain in much the same way as the flow variables. Nevertheless, their interpretation is not as straightforward as for the flow variables is. This holds especially for the adjoint variables.

3.6.1 Linearized variables

By definition, the linearized flow variables $\mathbf{U}_\alpha$ are the sensitivity of the conservative variables with respect to a perturbation. Thus, for each node, $\mathbf{U}_\alpha$ contains the sub-vector $\mathbf{u}_\alpha = [\rho_\alpha\,,(\rho\mathbf{w})_\alpha\,,(\rho e_t)_\alpha]^T$. The contours of the density sensitivity ρ_α, where α is the angle of attack, may be visualized for the RAE airfoil case, which has been discussed in Section 2.5.1, and for which the chord-wise Mach number distribution and the pressure contours are shown in Fig. 2.4a and in Fig. 2.6a, respectively. The contours of the density ρ and of its sensitivity ρ_α are depicted in Figs. 3.7a and 3.7b, respectively.

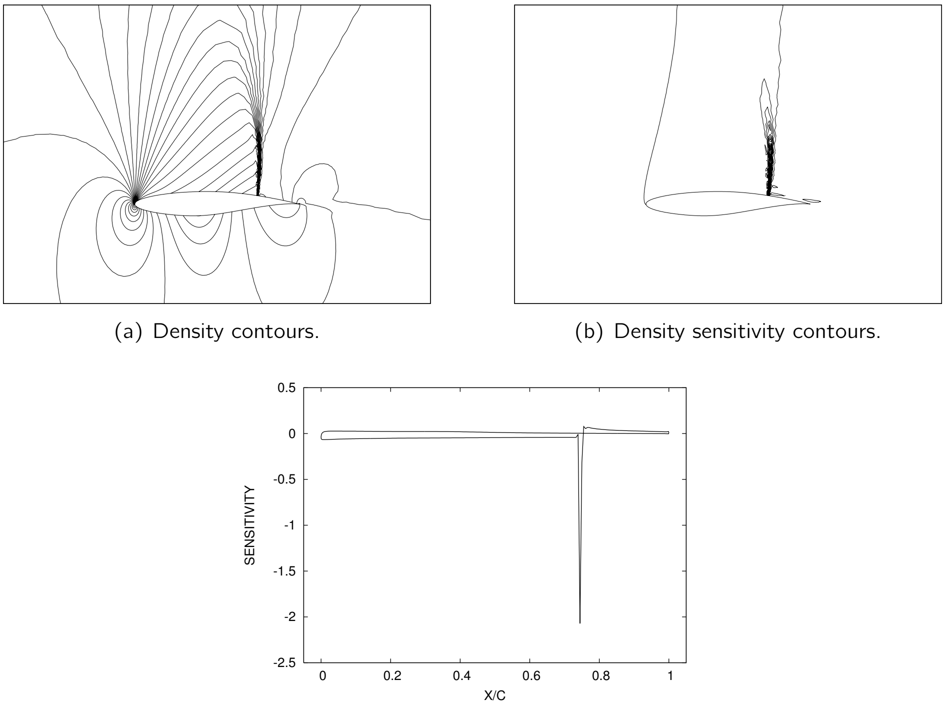
As can be seen, the contours of ρ_α appear to change in a discontinuous fashion around the shock. The chord-wise distribution of ρ_α depicted in Fig. 3.7c shows the discontinuous behavior near the shock. The discontinuity appears to be that of a Dirac type of function. It is of that type because, for the flow condition considered in the above case, a shock is present in the field. In fact, the density has a discontinuity like that of the Heaviside function, the derivative of which exists in the sense of distributions and is the Dirac function. As suggested in [101], the density $\rho = \rho(\mathbf{x})$ may be written as

$$\rho = \rho^{-} + [\rho]\,\mathcal{H}(\eta)\,, \qquad \mathcal{H}(\eta) = \begin{cases} 1\,,\ \eta \geq 0\,, \\ \\ 0\,,\ \eta < 0\,. \end{cases} \tag{3.51}$$

where $[\rho] = \rho^{+} - \rho^{-}$ is the density jump across the shock, $\eta = (\mathbf{x} - \mathbf{s}) \cdot \mathbf{n}$ is the normal distance from the shock line $\mathbf{s}$ and $\mathcal{H}$ is the Heaviside function, also known as step function. Differentiating the above equation with respect to α yields

$$\rho_\alpha = \rho_\alpha^{-} + [\rho_\alpha]\,\mathcal{H}(\eta) + [\rho]\,\delta_d(\eta)\,\eta_\alpha\,, \qquad \delta_d(\eta) = \frac{d\,\mathcal{H}}{d\,\eta} = \begin{cases} \infty\,,\ \eta = 0\,, \\ \\ 0\,,\ \eta \neq 0\,. \end{cases} \tag{3.52}$$

where δ_d is the Dirac function and η_α is the sensitivity of the shock position with respect to α. The latter sensitivity expresses how much the shock would displace along η for a given perturbation α. In the next chapter an application is presented that shows how the above relation may be used to obtain information about the shock displacement [101].

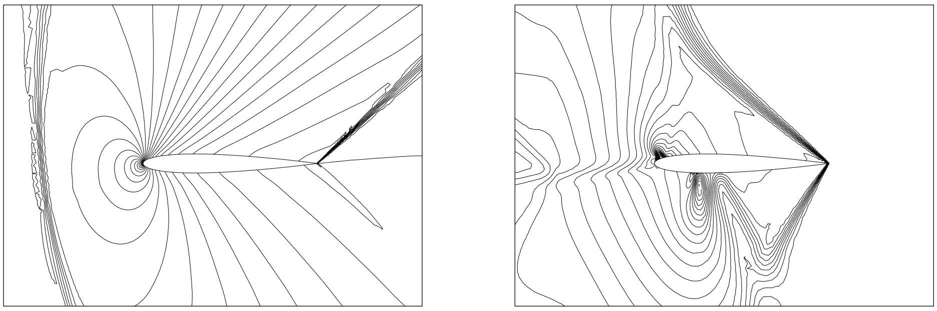


(a) Density contours.

(b) Density sensitivity contours.

(c) Density sensitivity.

Figure 3.7: *RAE airfoil at* $M_\infty = 0.75$ *and* $\alpha = 3$ *deg.*



(a) Mach number contours.

(b) Third adjoint variable for the lift coefficient.

Figure 3.8: *NACA0012 airfoil at* $M_\infty = 1.2$ *and* $\alpha = 7$ *deg.*

3.6.2 Adjoint variables

In engineering optimization [126] it is well known that the Lagrange multipliers represent the sensitivity of the objective function with respect to the active constraints. Thus, the adjoint variables $\mathbf{\Lambda}$ may be interpreted as the sensitivity of the functional with respect to the state of the system, which coincides with the residual $\mathbf{R}$. In fact, if the adjoint problem given in Eqs. (3.3) and (3.4) is manipulated, it can be shown that

$$\mathbf{\Lambda} = \frac{\partial J}{\partial \mathbf{R}} \, . \tag{3.53}$$

For instance, in the case of the lift coefficient of an airfoil, the vector $\mathbf{\Lambda}$ contains sub-vectors $[\partial c_l/\partial R_1 \, , \partial c_l/\partial R_2 \, , \partial c_l/\partial R_3 \, , \partial c_l/\partial R_4]^T$ for each node. According to the definition of residual given in Chapter 2, the variable $\lambda_3 = \partial c_l/\partial R_3$ is the sensitivity of the lift coefficient with respect to changes in the y-momentum. A contour plot of λ_3 is shown in Fig. 3.8b for the supersonic flow condition considered already in Section 2.5.1. The Mach contours for that condition are also given in Fig. 3.8a. As can be seen, λ_3 exhibits a reverse trend compared to the flow. The latter trend is due to the fact that the flow is supersonic and perturbations do not rise upstream. A line exists that divides the flow in two regions: (*i*) the upstream region in which perturbation in y-momentum affects the lift and in which λ_3 thus exhibits values different from zero; and (*ii*) the downstream region in which perturbation in y-momentum does not affect the lift and in which λ_3 thus vanishes. Instead of the latter physical explanation, one can look at the problem from a strictly mathematical point of view and realize that the reverse trend in the adjoint variables is due to the characteristics, which are reversed with the transposition. An explanation of the meaning of the adjoint variables may be found in [38].

Since the adjoint variables link the functional to the residual of the system, formulations have been suggested that use the adjoint variables to estimate the error and to produce indicators for eventually refining the computational mesh [127].

3.7 Concluding remarks

The discrete adjoint approach to sensitivity analysis is straightforward to understand. Its implementation is however very complex and demanding. Essentially, as shown in this chapter, the implementation of the discrete adjoint boils down to three tasks:

(i) Differentiating non-linear numerical functions – The task may be very tedious as well as complex if the differentiation has to be hand-coded. The exact differentiation of the numerical fluxes presented in this chapter is a clear example of such complexity. The hand-coding approach has the disadvantage that whenever new functions are introduced in the implementation, a new differentiation has to be performed. Tools such as Automatic Differentiation may provide a more systematic and less time-consuming approach to the differentiation. Easily, when new functions are introduced, their differentiated counterpart can be derived and added very quickly. Alternatively, the differentiation can be much faster if approximations are introduced. In this chapter some approximations have been presented. In addition to being faster to implement, those approximations are also faster to run. For instance, neglecting the limiter gives a faster code because several loops on the mesh are not performed. However, due to the approximations the adjoint code is not exact. In terms of computed sensitivity an error is introduced. In Chapter 5 the effect of such error on optimization test cases is discussed and acceptable approximations are identified;

(ii) Assembling the products that involve the transposed residual Jacobian – Since Automatic Differentiation produces the projection of the Jacobian onto a vector, this task could be performed by applying AD to the complete residual assembly routine. However, the resulting assembly may not be as fast and as memory efficient as the hand-coded assembly presented in this chapter. Results show that the latter assembly has a very limited memory usage and takes similar computational time as the residuals vector assembly. Moreover, it allows to construct efficiently multiple matrix-vector products, which are needed for the simultaneous solution of multiple adjoint equations. Clearly the hand-coded assembly requires more effort for the derivation. However, it must only be derived once. In fact, the architecture of the unstructured code is such that the edge-based assembly can be separated from the computation of the fluxes. The numerical flux routines, or similar functions, are in fact called on a edge-by-edge basis. It is therefore convenient to spend time and effort for the hand-coding of an efficient assembly. In the present work the use of Automatic Differentiation is only advocated for the differentiation of the numerical fluxes, or for other tasks for which performances and efficiency are not paramount;

(iii) Solving the adjoint equations iteratively – As shown in this chapter the inclusion of the reconstruction makes the residual Jacobian poorly diagonally dominant. Consequently the linear system of adjoint equations may prove difficult to solve using linear solvers. Instead, if the adjoint equations are treated as non-linear equations, the flow solution scheme can be used to solve the equations in a rather straightforward way. Such solution method is convenient because the solver is already available and it is also very robust. The adjoint solver presented in this chapter is characterized by computing times and storage requirements that matches almost exactly with that of the flow solver when the preconditioner is stored, and that are only slightly larger when the preconditioning is matrix-free. The solver has also the interesting feature of solving multiple adjoint equations simultaneously. The results show that the simultaneous solution is convenient. In fact, the increase in memory and computing time is acceptable.

Chapter 4

Shape Parameterization

4.1 Introduction

4.1.1 Definition and requirements

The parameterization of the shape plays a crucial role in shape optimization because it provides a representation of the design space. The shape parameters work as design variables and their variations are transformed by the parameterization in displacements of the shape. For the parameterization to be effective, the representation of the design space must be complete and a smooth behavior must be observed during the optimization process. The lack of completeness may preclude the possibility of finding optimal designs. For instance, a shape, which is optimal for certain conditions, may not be represented by the parameterization and thus it may not be found by the optimizer. The lack of smoothness may result in the appearance of non-smooth features on the shape, such as cusps or bumps. The latter features may cause the optimization process to stall in certain circumstances, for instance, when the mesh collapses because the mesh deformation algorithm cannot accommodate the displacements.

The completeness of the shape parameterization can be tested by the approximation of different shapes. In fact, it must be possible to approximate a generic shape to a predefined level of accuracy by including a certain number of parameters. The difference between the original coordinates and that of the approximation should vanish, ideally, as the number of parameters are increased to infinity. Instead, the smoothness of the parameterization cannot be tested quantitatively. Usually, any lack of smoothness is revealed immediately during the optimization process. If there is the risk that the process breaks down, additional smoothing procedures may be introduced between each update of the shape.

4.1.2 Existing methods to parameterize the shape

Several methods are available in the literature to parameterize the shape. In the following some of the existing methods are briefly mentioned. An exhaustive review can be found in [111].

Probably, in the context of numerical shape optimization the easiest parameterization method is that of directly using the nodes of the boundary mesh as parameters [86]. The method is very simple and can be applied to 2D and 3D problems. However, it has the disadvantage of implying a large number of parameters and of offering a poor representation of the shape. In fact, several nodes may be present on the boundary mesh and their displacements can easily give non-smooth features. A partial solution to the problem may be that of implementing a smoothing procedure for the boundary mesh.

Various methods have been investigated that try to represent a given 2D shape, or its displacements, with a reduced number of parameters. The Hicks and Henne functions [46], which are also called bumps, may be superimposed on a given shape at certain locations. The local nature of the functions may cause the displacements to be concentrated in very few locations, with the result that the final shape may appear to be bumpy. In certain cases, post-smoothing has to be applied to the optimized airfoils. The Bezier curves [117] and the Non Uniform Rational B-Splines [74], NURBS, are widely used. They both use control points, the location of which control the shape deformation. 2D applications show that the NURBS method requires the fewest control points and allows the shape to deform very smoothly [73]. NURBS have found applications to 3D problems by means of the free-form deformation method [110], which is very flexible and allows the parameterization of complex geometries. Application of the method with Bezier curves are found in [28]. The "Shape Function - Class Function" method [68] uses Bernstein Polynomials and class functions. The method has been used successfully for the parameterization of 2D and 3D shapes. In 3D not only wing-like shapes have been parameterized but also fuselage- and nacelle-like shapes.

4.1.3 Orthogonal representations

An interesting parameterization method is based upon the concept of orthogonal representation of the shape. According to the concept, the shape may be represented by a linear combination of orthogonal basis functions. The orthogonality property ensures completeness in the representation. The shape may be usually represented by a reduced number of basis functions. Moreover, due to the orthogonality, better convergence properties of the optimization process are expected.

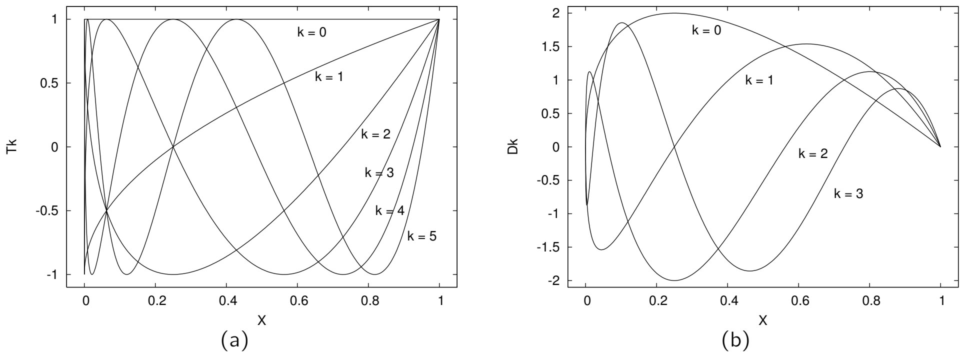


Figure 4.1: *(a) Chebyshev basis functions; (b) Design basis functions.*

In spite of their interesting features orthogonal representations are not widely used. Although some applications are available [132, 69, 23, 107], their number is still limited in comparison with other methods such as those using NURBS or Bezier curves. Probably, they are not widely used because their applications are limited to 2D cases.

The parameterization considered here is that based on Chebyshev polynomials [132]. In the following it is shown how the parameterization can approximate very accurately airfoil shapes. Moreover, an idea is presented to extend the applicability of the parameterization to wing-like shapes. Compared to the orthogonal representations used in other works [69, 107, 23], the representation used here has an important difference: the orthogonal shape functions are readily available in closed form, as shown below. Instead, other works derive the orthogonal shape functions by means of a Gram-Schmidt orthonormalization process, which requires numerical integration to compute the basis functions of a given shape.

4.2 Parameterization with Chebyshev polynomials

4.2.1 Parameterization of an airfoil

Airfoil curves can be represented by means of an orthogonal series:

$$f(x) = \sum_{k=0}^{N_D} \alpha_k D_k(x)\,, \qquad\qquad D_k = T_k - T_{k+2}\,. \tag{4.1}$$

α_k is the coefficient of the design basis function D_k. Suitable values of the coefficients α_k must be found in order to represent a given curve. T_k is the Chebyshev basis function, which is defined as

$$T_k(x) = \cos(k\gamma(x))\,, \qquad \gamma(x) = \cos^{-1}(2\sqrt{x} - 1)\,, \tag{4.2}$$

where $k \geq 0$ and $0 \leq x \leq 1$. The Chebyshev basis function T_k is not used directly because it cannot give closure at the leading and trailing edges, i.e., $T_k \neq 0$, for $x = 0$

and $x = 1$. Instead, the design basis function D_k defined in Eq. (4.1) yields closure at the two points. Figure 4.1 shows the two basis functions. As can be seen from Fig. 4.1b, the first design basis D_0 closely resembles an airfoil shape. The basis D_k crosses the abscissa k times.

In the present work the parameterization of Eq. (4.1) is used to independently approximate the upper and the lower curves of the airfoil. An alternative is to use the camber line method, which parameterizes the camber line of the airfoil and the thickness distribution independently. When using the camber line method, the lower/upper curve of the airfoil can be obtained by subtracting/adding half of the thickness to the camber line.

4.2.2 Evaluation of the shape coefficients

Suitable values of the α_k coefficients for a given curve can be found by minimizing the error between the expansion in Eq. (4.1) and the curve. The curve is given as a set of N_P points, $[x_n, y_n]$. The error may be represented by a quadratic error function, which depends on the basis function's coefficients $\bar{\alpha} = [\alpha_1, ..., \alpha_N]$ and reads

$$E(\bar{\alpha}) = \sum_{n=1}^{N_P} e_n^2 = \sum_{n=1}^{N_P} \left(f(x_n) - y_n\right)^2 . \tag{4.3}$$

e_n represents the local error, i.e., the difference in y-coordinate between the curve and the approximation at point n. According to Eq. (4.1), the above equation may be written as

$$E(\bar{\alpha}) = \sum_{n=1}^{N_P} \left(\sum_{k=0}^{N_D} \alpha_k D_k(x_n) - y_n \right)^2 . \tag{4.4}$$

Sensitivity information may be computed analytically and used to minimize efficiently the above error. Some algebra shows that the component i of the gradient ∇E may be written as

$$\frac{\partial E}{\partial \alpha_i} = 2 \sum_{n=1}^{N_P} \left(\sum_{k=0}^{N_D} \alpha_k D_k(x_n) - y_n \right) D_i(x_n) , \tag{4.5}$$

whereas the component ij of the Hessian $\mathbf{H}$ may be written as

$$\frac{\partial^2 E}{\partial \alpha_i \partial \alpha_j} = 2 \sum_{n=1}^{N_P} D_i(x_n) D_j(x_n) . \tag{4.6}$$

By means of the gradient and the Hessian, the error E may be expressed in the quadratic form

$$E(\bar{\alpha}) = E(\bar{\alpha}_0) + \nabla E^T (\bar{\alpha} - \bar{\alpha}_0) + \frac{1}{2} (\bar{\alpha} - \bar{\alpha}_0)^T \mathbf{H} (\bar{\alpha} - \bar{\alpha}_0) . \tag{4.7}$$

The latter form is also exact according to Eq. (4.4). The coefficients $\bar{\alpha}$ that minimize the error E are found by equating the first derivative of the above equation to zero. It gives

$$\bar{\alpha}_{\min} = \bar{\alpha}_0 + \mathbf{H}^{-1}\nabla E\,. \tag{4.8}$$

Given an airfoil curve, $\bar{\alpha}_{\min}$ is the vector that contains the N_D basis function coefficients that approximate the curve with the lowest error possible. $\bar{\alpha}_0$ is an initial value of the coefficients, which can be conveniently chosen to be zero. The number of basis functions N_D can be chosen appropriately according to the required level of accuracy that must be achieved by the representation. In order to define the level of accuracy, instead of using the global quadratic error E, it is best to use the maximum local error $e_{\max} = \max(e_n)$. The latter quantity is easier to interpret since it may be visually inspected. The level of accuracy of the approximation should increase with the number of basis functions N_D. In general, for a given level of accuracy, the number of basis functions N_D depends on the complexity of the airfoil shape.

Authors [122] investigated the appropriate value of $e_{\max}$ that should be obtained in order to avoid discrepancies between the flow solutions computed using the original and the approximate airfoil geometries. The investigation is relevant for flow solvers that discretize the boundary conditions with linear elements, like in the present work. The conclusion of the investigation is that for an airfoil of unit chord an error $e_{\max} = 8\times10^{-5}$ should be achieved.

4.2.3 Nose radius of curvature and trailing edge angle

Airfoil design often needs the nose radius and the trailing edge angle for the purpose of enforcing constraints on their values. The nose radius and the trailing edge angle of the parameterized airfoil may be evaluated using the derivatives of the curve. The parameterization given in Eq. (4.1) is differentiable and the derivatives are easily obtained analytically. The mth order derivative reads

$$f^m(x) = \sum_{k=0}^{N_D} \alpha_k (T_k^m(x) - T_{k+2}^m(x))\,, \tag{4.9}$$

which needs the Chebyshev basis function T_k to be differentiated. Tedious algebraic manipulations show that the first derivative $T_k'(x)$ can be written as

$$\frac{\partial T_k}{\partial x} = k\eta(x)\sin(k\gamma(x))\,, \tag{4.10}$$

whereas the second derivative $T_k''(x)$ can be written as

$$\frac{\partial^2 T_k}{\partial x^2} = -k^2\eta(x)^2 T_k(x) - \frac{\eta(x)^2}{2}(1-\omega(x)-2\omega(x)^2)\frac{\partial T_k}{\partial x}\,, \tag{4.11}$$

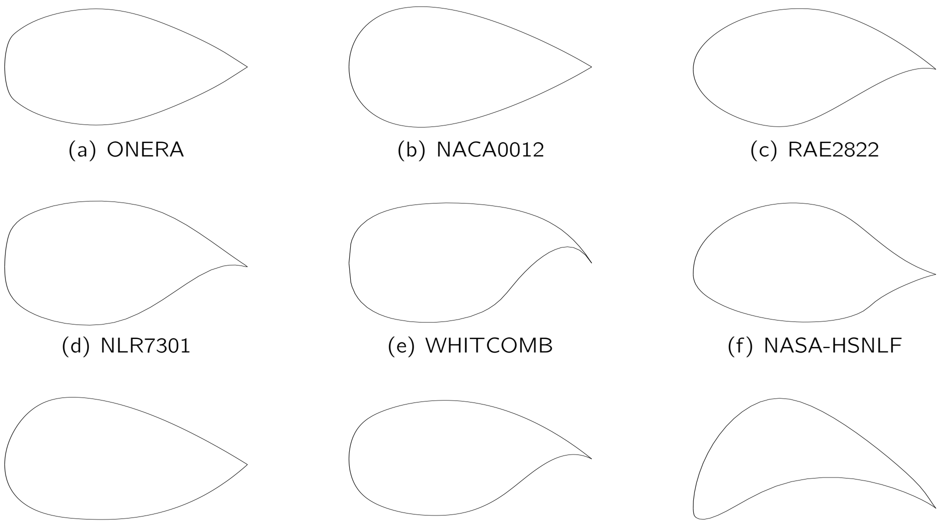
with the functions ω and η defined as

$$\omega(x) = 2\sqrt{x} - 1\,, \qquad \eta(x) = \frac{2}{\sqrt{(1-\omega(x))(1+\omega(x))^3}}\,. \tag{4.12}$$

The first and the second derivative of the polynomial, f' and f'', can be used to compute the trailing edge angle δ and the node radius ρ, which are defined respectively as

$$\delta = f' \,, \qquad \rho = \frac{\sqrt{1+f'^3}}{f''} \,. \tag{4.13}$$

ρ should be evaluated for $x = 0$ and δ for $x = 1$. For these values of x, the function η is not defined. This problem can be overcome by evaluating Eqs. (4.10) and (4.11) in the limit for $x \to 0$ and $x \to 1$. In practice, a small value ϵ is used to evaluate ρ and $1 - \epsilon$ is used to evaluate δ. The value $\epsilon = 1^{-16}$ was found to be satisfactory.



(a) ONERA (b) NACA0012 (c) RAE2822

(d) NLR7301 (e) WHITCOMB (f) NASA-HSNLF

(g) BWB-F (h) BWB-W (i) EPPLER-420

	N_D	N_P		N_D	N_P		N_D	N_P
ONERA	13	70	NACA0012	5	65	RAE2822	11	65
NLR7301	21	40	WHITCOMB	27	40	NASA-HSNLF	44	60
BWB-F	10	65	BWB-W	14	65	EPPLER-420	19	40

Figure 4.2: *Airfoils. The table shows the number of required basis functions N_D for which the maximum error* $e_{\max} < 8 \times 10^{-5}$.

4.2.4 Parameterization of 3D displacements

The simplest way of extending a 2D parameterization to 3D problems is probably that of parameterizing only some sections along the wing span and interpolating all the points that lie in between. However, although the method is very easy to implement, the quality of the parameterization is poor. Non-smooth areas or bumps may appear as the result of the local nature of the interpolation between the sections.

A smooth 3D parameterization of the wing may be achieved by using a Chebyshev polynomial in which the coefficients change along the span according to another polynomial. Instead of parameterizing the shape, as for the 2D case, it is probably more convenient to parameterize its displacements. In fact, problems may arise if the geometry is linked to a CAD model that must be updated when the geometry is changed. The parameterization of the wing displacements achieves a greater flexibility because the displacements can be conveniently sent to the CAD model only at the end of the optimization process. The idea is explained in the following. For simplicity the exposition is limited to a wing with a flat planform.

Consider a wing point x, y on the planform. The y coordinate is scaled by the span s. The x coordinate is scaled by the chord c, which varies along the span. I.e.,

$$y^* = \frac{y}{s}\,, \qquad x^* = \frac{x - x_{le}(y^*)}{c(y^*)}\,, \tag{4.14}$$

where x_{le} is the coordinate, along the chord-wise direction, of the leading edge of the section. In practice, the above scaling maps the planform into the unit square. The displacement of the point, which must be added on the existing shape, may be defined as

$$\Delta f(x, y) = c(y^*) \sum_{k=0}^{N_D} \alpha_k\left(y^*\right) D_k\left(x^*\right)\,, \tag{4.15}$$

where $\alpha_k(y^*)$ is the coefficient that depends on the dimensionless y coordinate. In the present work a polynomial is used, which reads

$$\alpha_k(y^*) = \sum_{m=1}^{M} \alpha_{km} {y^*}^{m-1}\,. \tag{4.16}$$

The coefficients α_{km} are used as design variables. If $M = 2$ the above relation prescribes a linear variation of the coefficient α_k along the span. If $M = 3$ the variation is quadratic. If necessary, constraints on the displacements may be enforced explicitly by assigning certain values to some of the above coefficients. For instance, zero displacement at the root is obtained by setting

$$\alpha_{k1} = 0 \quad \text{for} \quad k = 1, N_D\,, \tag{4.17}$$

which is equivalent to start the summation in the above equation from $m = 2$. Instead, zero displacement at the tip section, where $y^* = 1$, is obtained by setting

$$\alpha_{kM} = -\sum_{m=1}^{M-1} \alpha_{km} \quad \text{for} \quad k = 1, N_D\,. \tag{4.18}$$

Note that other kind of polynomials can be employed, e.g., an orthogonal expansion similar to the Chebyshev representation.

When the wing has a twist or a dihedral angle the planform is not flat. Nevertheless, the parameterization described above applies almost identically because the planform can be made flat by subtracting the twist and the dihedral components. In general, the latter components should also be present for optimization purposes. For instance, the twist has an important role in defining the lift distribution along the span. In the present work twist and dihedral angle are not considered.

4.2.5 Scaling of the coefficients for shape optimization

The α_k coefficients are not used directly as design variables. They are scaled first by their initial values, i.e., by the values that define the initial shape. Consequently, an identical variation of the design variables imply different variations in the parameters, each of which varies proportionally to its initial value. For instance, if α'_k is the scaled parameter/design variable and α^0_k is the initial value of the parameter, the variation of the parameter becomes $\Delta\alpha_k = \alpha^0_k \Delta\alpha'_k$.

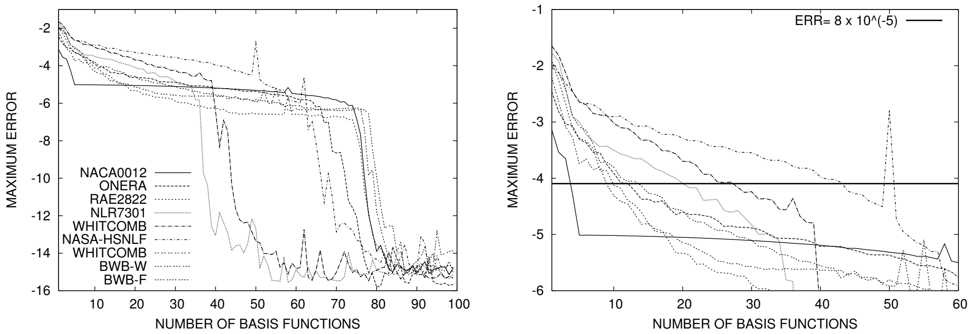
In practice, scaling is not optional but essential. As will be shown later, high-frequency basis (large values of k) tend to have coefficients of several orders of magnitude smaller than the low-frequency ones. It could not be otherwise because, as can be seen by Fig. 4.1b, the sum of basis functions with similar values of the coefficients may not result in an airfoil-like shape. Without the aforementioned scaling it is highly likely that an equal variation of the parameters could give a new set of parameters of comparable order of magnitude and thus a shape different from that of an airfoil. In general, also for convergence reasons, scaling of the design variables is a recommended practice in optimization [126].

For 2D problems how to scale is clear because the scaling coefficients are that of the initial shape. Instead, for 3D problems the scaling coefficients may not be clearly identified. In fact, each section of the wing may be different and therefore may have different coefficients. A possibility is to define a set of coefficients that are averaged along the span.

4.3 Examples

4.3.1 Approximation of airfoils

The accuracy of the parameterization is demonstrated with the approximation of some airfoils, which are depicted in Fig. 4.2. They represent different designs, which are suitable for conditions that go from low subsonic to transonic. The geometries of the airfoils are expressed by different numbers of points N_P, which are shown in the table at the bottom of the figure. The number of points N_P is usually not a variable, unless the airfoil may be expressed analytically. For instance, the NACA0012 airfoil is defined by an analytic formula, which allows to specify the number of points and to define their distribution along the x-axis. Instead, the RAE2822 airfoil is defined by a table of points, which do not leave the same freedom. Both the formula and the table of points can be found in [133].



(a) (b)

Figure 4.3: *Maximum error*

The coefficients are found by solving Eq. (4.8). The values of the maximum error for the different airfoils are shown in Fig. 4.3, which has for abscissa the number of basis functions N_D. Since each airfoil has two curves, the plotted value of the error is the maximum of the errors evaluated on the two curves. Figure 4.3a shows that the maximum error is reduced up to machine precision when a number of basis $N_D \approx 60-90$ is used. However, in order to approximate the airfoils to the recommended level of accuracy [122] $e_{\max} = 8 \times 10^{-5}$, a smaller number of basis functions can be used. Figure 4.3b shows a magnification of Figure 4.3a. The range of the maximum error is limited to a value of 10^{-6} whereas that of the number of basis functions is limited to a value of 60. The horizontal line $e_{\max} = 8 \times 10^{-5}$ is superimposed on the figure. The exact number N_D which is required for achieving the recommended level of accuracy, i.e., the number at which the curves in Figure 4.3b cross the horizontal line, is given in the table of Fig. 4.2. The maximum number of basis, 44, is required by the NASA-HSNLF airfoil, whereas the lowest, 5, is required by the NACA0012.

Figure 4.4 shows the values of the first 10 coefficients for the NACA0012, RAE2822 and NASA-HSNLF airfoils. The lower curves of the latter two airfoils are the ones that start with negative values. As can be seen, although the airfoils require different number of basis functions, the values of their coefficients have similar order of magnitude. For each airfoil, as anticipated in the previous section, the magnitude of the coefficients decrease rapidly and become relatively small after the fifth or sixth basis function. When scaling is applied, also the gradient tends to have the same decreasing trend. From an optimization point of view, the components for which the gradient is relatively small are not very influential. In practice, the optimization may be carried out with a lower number of basis functions than that required for approximating the shape.

As a final remark, it is important to realize that the number of basis functions N_D and the number of airfoil points N_P may not be chosen arbitrarily. If N_D is too large compared to N_P, it is possible that certain "high-frequency" variations are filtered out. In practice, if the airfoil data is too coarse, high-frequency basis may not be captured at the approximation stage. However, when the parameterization is later used to represent

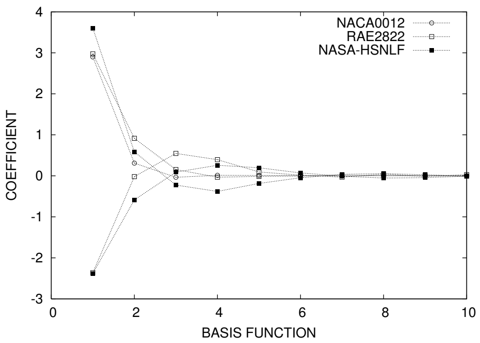


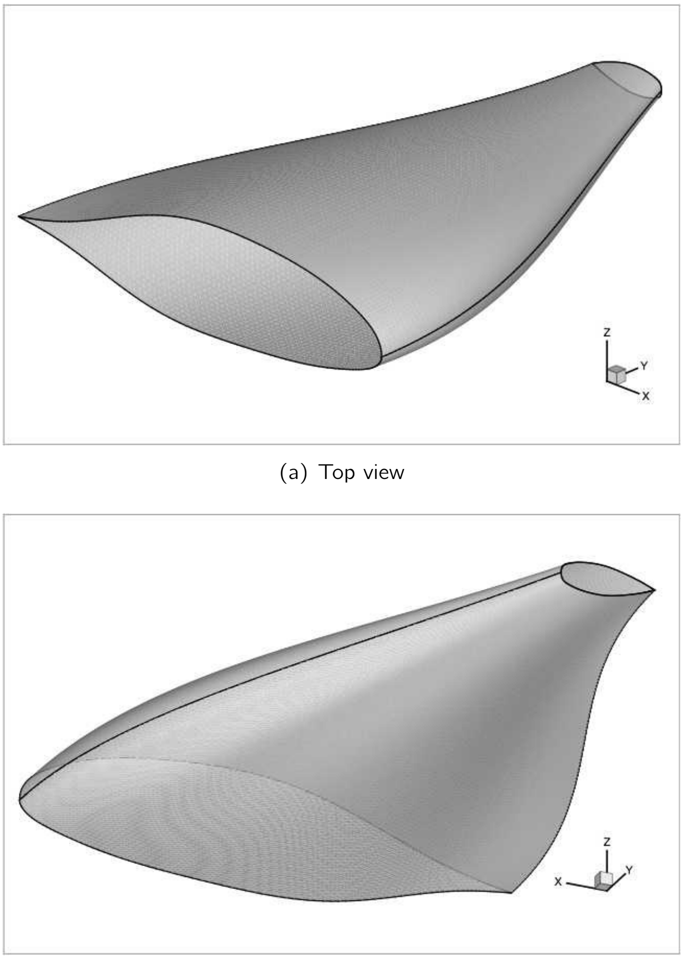
Figure 4.4: *Basis functions coefficients*

the airfoil on a fine distribution of points, the high-frequency basis may appear and may cause unwanted oscillations. Hence, in order to parameterize airfoils that are given with very few points, the method to extract the parameters should include additional constraints on the smoothness.

4.3.2 Generation of a wing-like shape

In order to show the qualities of the 3D displacements given by Eq. (4.15), a wing-like shape is generated. The starting point is a flat planform, which has non-straights leading and trailing edge lines. Therefore, it has a chord that varies non-linearly along the span. The planform configuration is not from an existing design, it has been defined only for the purpose of creating a complex geometry. Three airfoils are used: The NASA-HSNLF airfoil of Fig. 4.2f is placed at the root section, $y^* = 0$; the ONERA airfoil of Fig. 4.2a at the tip section, $y^* = 1$; and the BWB-W airfoil of Fig. 4.2h is placed at mid-span, $y^* = 0.5$. The coefficients in Eq. (4.16) are defined explicitly to be equal to that of the three airfoils at their corresponding locations. In practice, $M = 3$ is used in Eq. (4.16).

The wing planform is mapped into the unit square and is meshed. The displacements of the lower and upper surfaces are computed on a mesh with 100 points along both chord- and span-wise directions. The wing is shown in Fig. 4.5. The figure has been realized by a post-processing tool. As can be seen both sides of the wing appear to be very smooth. The BWB-W airfoil located at the middle of the wing-span is not visible because it has been perfectly blended within the wing surfaces. The lower side of the wing, see Fig. 4.5b, has a complex geometry. On the lower side, in proximity of the trailing edge line, the chord-wise first derivative changes signs according to the shapes of the three airfoils.



(a) Top view

(b) Bottom view

Figure 4.5: *Wing-like shape generated by the 3D parameterization*

4.4 Concluding remarks

The Chebyshev polynomials are capable of approximating airfoil shapes accurately, using a limited number of parameters. A least-squares procedure may be used to find the parameters for a given airfoil. In the three-dimensional space the extension of the Chebyshev parameterization is not obvious. In this chapter a concept suitable for wing-like shapes has been presented. It assumes that the parameters change along the span-wise direction according to another polynomial. The concept is not intended for the parameterization of the wing, but for the parameterization of its displacements. Hence, the computation of the wing parameters is avoided and the flexibility of the method is increased. Results show that the three-dimensional displacements appear to be very smooth, also in the case of wings with non-linear planform, for which the surfaces do not have a simple curvature.

Chapter 5

Shape Optimization

5.1 Optimization problems and algorithms

5.1.1 Inverse design

Given a certain flow field, inverse design aims at recovering the shape capable of generating that field. An engineering description of the method, which cast the inverse design problem in the form of a least-squares minimization, is as follows.

Consider a shape $S(\bar{\alpha})$, which can deform according to the parameters $\bar{\alpha}$ and is characterized by the pressure $p(\bar{\alpha})$. Suppose that the target pressure p^* has been defined, but the parameters α^* that define the shape $S^* = S(\bar{\alpha}^*)$ to which the target pressure corresponds are unknown. The latter parameters may be found by solving an unbounded minimization problem, which reads

$$\min \; f(\bar{\alpha}) \,, \qquad f(\bar{\alpha}) = \oint_{S(\bar{\alpha})} (p(\bar{\alpha}) - p^*)^2 dS(\bar{\alpha}) \,. \tag{5.1}$$

Note that the target pressure is fully achieved when the functional f vanishes. However, there is no guarantee of the existence and uniqueness of a shape to which the target pressure corresponds. If the shape does not exist, even if a feasible shape S^* that minimizes the above functional can be found, the functional does not vanish.

The inverse design problem defined above may be solved by an unconstrained optimization algorithm. A possible one is the steepest descent, which is the simplest amongst the available algorithms. It moves in the direction opposite to that of the functional gradient, i.e.,

$$\bar{\alpha}^{n+1} = \bar{\alpha}^{n} - \lambda \nabla f^{n} \,, \tag{5.2}$$

where λ is a step-size parameter. A possibility for the determination of λ is a line-search. For instance, the function may be approximated by a parabola along the gradient direction and λ computed accordingly. Because of the lack of second order information the algorithm may converge slowly towards the optimum solution. More sophisticated Quasi-Newton algorithms are available [126], which approximate the Hessian of the functional in order to make second order information available. When the starting point is in the neighborhood of the optimum solution, those algorithms are expected to converge faster.

Note that from an engineering point of view the practice of inverse design may be acceptable when the designer knows how the best pressure should look like. For instance, that was the case with transonic airfoils, with which designers gained considerable experience by means of both experiments and theoretical analysis. Instead, the practice may be questionable when the designer is operating in a vacuum. The most obvious example of such a situation is the case of an innovative configuration for which the best pressure may be unknown. In the latter case, a method that allows the designer to find the optimal shape for a given set of design requirements should be employed. Such a method may be implemented using constrained design. In the following a description is given.

5.1.2 Constrained design

Constrained design aims at finding the shape that minimizes an objective function and satisfies some design constraints. The objective function is a figure of merit, which is representative of the design performances, whereas the constraints are design requirements. For instance, a typical constrained design problem may be that of minimizing the drag coefficient while enforcing constraints on the lift, on the pitching moment and on some geometric quantities such as wing span or wing volume.

The general form in which constrained optimization problems are written is

$$\begin{aligned} &\min \; f(\bar{\alpha}) \\ &\quad g_j(\bar{\alpha}) \leq 0 \,, \qquad (j = 1, n_g) \,, \\ &\quad h_k(\bar{\alpha}) = 0 \,, \qquad (k = 1, n_h) \,, \\ &\bar{\alpha}_L \leq \bar{\alpha} \leq \bar{\alpha}_U \,. \end{aligned} \tag{5.3}$$

where f is the objective function, g_j are the inequality constraints, h_k are the equality constraints and $\bar{\alpha}_L$ and $\bar{\alpha}_U$ are the lower and upper bounds for the design variables, respectively. In the case of the aforementioned constrained drag minimization, a possible definition for the objective and for the constraints on the functionals is as follows,

$$f = \frac{c_d}{c_{dR}}\,, \quad h_l = \frac{c_l}{c_{lR}} - 1\,, \quad g_m = \frac{c_m}{c_{mR}} - 1\,. \tag{5.4}$$

Instead, a possible definition of the geometric constraints on the thickness, g_t, on the leading-edge radius of curvature, g_ρ, on the trailing edge angle, g_θ, and on the volume, g_V, is as follows,

$$g_t = 1 - \frac{tc_{\max}}{tc_{\max R}}\,, \quad g_\rho = 1 - \frac{\rho_c}{\rho_{cR}}\,, \quad g_\theta = 1 - \frac{\theta}{\theta_R}\,, \quad g_V = 1 - \frac{V}{V_R}\,. \tag{5.5}$$

The volume constraint is only present for 3D optimization problems. In 2D it can be replaced by a constraint on the area of the section.

The subscript R in the two equations above indicates reference values. The latter value may be chosen to be equal to the initial value, so that at the beginning of the optimization the objective has value 1 and the constraints have values 0. However, in the case of the constraints, other reference values could be chosen to specify the design requirements. For instance, if the pitch moment coefficient is allowed to halve, its reference value should be chosen as $c_{mR} = 0.5c_{m0}$, so that during the optimization $c_m \leq 0.5c_{m0}$ always.

The constrained design problem in Eq. (5.3) may be cast in an unconstrained form if the constraints are included as penalty terms in the objective [126]. The penalty approach allows the use of an unconstrained optimization algorithm, which means that only the gradient of the objective has to be computed, saving the time required for computing the adjoint of the constraint functionals. However, problems may arise due to ill-conditioning. Moreover, the approach is not very accurate in enforcing the constraints.

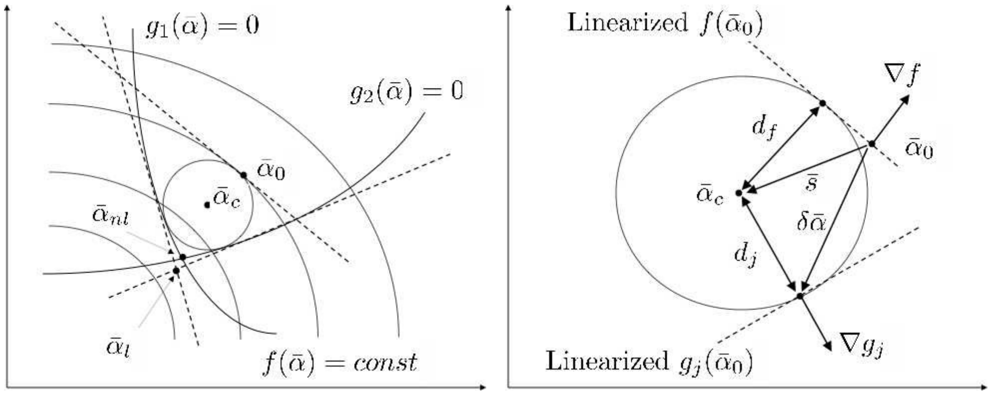
A better choice is to use constrained optimization algorithms, which are specifically devised to solve problems of the type given in Eq. (5.3). For instance, Sequential Quadratic Programming (SQP) algorithms have been used in the present work for some problems. They use second order information and are known to be very efficient and accurate. Their implementation is complex and a description is not given here because they have been used as black-boxes.

In the following a very simple Sequential Linear Programming (SLP) algorithm is briefly described. The description serves the purpose of introducing the reader to the problem of constrained optimization at a basic level. The concept behind the algorithm is very simple and can be grasped with basic calculus. In practice, the algorithm shows to be effective for shape optimization problems. Nevertheless, it only uses first-order information and thus it is not very efficient.

5.1.3 Sequential Linear Programming: the method of centers

The simplest possible Sequential Linear Programming algorithm linearizes directly the constrained problem in Eq. (5.3). The result is a linear problem that can be easily solved by a Linear Programming algorithm, e.g., the Simplex method. Unfortunately, as can be shown by Fig. 5.1a, the optimum of the linearized problem may lie outside of the feasible design space. Therefore, the optimum may be approached from the infeasible region of the design space and accuracy issues may arise.

The method of centers tries to find the largest hypersphere that fits into the linearized design space, which is delimited by the gradients of the objective function f and of the constraints g_j and h_k. Once the latter has been found, a move in the center of the hypersphere will bring the design in a position which is closer to the optimum. A new linearization may be performed, a new hypersphere may be computed and again the design may be moved towards its center. The process may be repeated until convergence to the non-linear optimum.



(a) $\bar{\alpha}_0$ = linearization point, $\bar{\alpha}_c$ = center of the circle,$\bar{\alpha}_l$ = optimum of the linearized problem, $\bar{\alpha}_{nl}$ = optimum of the non–linear problem

(b) $\bar{s}$ = move toward the center of the circle, $\delta\bar{\alpha}$ = move toward the constraint boundary

Figure 5.1: *Geometric interpretation of the method of centers*

The interesting feature is that in the latter case the optimum is always approached from the feasible region. Moreover, unfeasible designs can be brought into the feasible region if the linearized objective function is ignored when the hypersphere is computed. As can be seen from Fig. 5.1b, the largest hypersphere is computed by means of some geometric arguments [126]. In fact, the hypersphere radius r should be the result of a maximization problem under the constraints that $d_f - r \geq 0$ and $d_j - r \geq 0$ for $j = 1, n_g$ with

$$d_f = -\frac{\nabla f^T \bar{s}}{|\nabla f|}\,, \qquad d_j = \frac{\nabla {g_j}^T(\delta\bar{\alpha} - \bar{s})}{|\nabla g_j|} = -\frac{g(\bar{\alpha}_0) + \nabla {g_j}^T \bar{s}}{|\nabla g_j|}\,, \tag{5.6}$$

where d_j is obtained considering that, being $\delta\bar{\alpha}$ the move to the constraint boundary, one has $g_j(\bar{\alpha}_0 + \delta\bar{\alpha}) = g_j(\bar{\alpha}_0) + \nabla {g_j}^T \delta\bar{\alpha} = 0$. If equality constraints have to be included, the move $\bar{\alpha}$ towards the center should not violate these, which means $h_k(\bar{\alpha}_0 + \bar{\alpha}) = h_k(\bar{\alpha}_0) + \nabla {h_k}^T \delta\bar{\alpha} = 0$ for $k = 1, n_h$. After some algebra, the maximization problem for the radius r of the hypersphere may be stated as

$$
\begin{array}{rcll}
\max\ r\,, & & & \\
\nabla f^T \Delta\bar{\alpha} + |\nabla f| r & \leq & 0\,, & \\
\nabla {g_j}^T \Delta\bar{\alpha} + |\nabla g_j| r & \leq & -g_j(\bar{\alpha}_0) \qquad (j = 1, n_g)\,, & \qquad (5.7)\\
\nabla {h_k}^T \Delta\bar{\alpha} & = & -h_k(\bar{\alpha}_0) \qquad (k = 1, n_h)\,, & \\
\min(\bar{s}, \bar{\alpha}_L - \bar{\alpha}_0) \leq \Delta\bar{\alpha} & \leq & \min(\bar{s}, \bar{\alpha}_U - \bar{\alpha}_0)\,, & \\
r & \geq & 0\,. &
\end{array}
$$

Equation (5.7) is a Linear Programming problem for the vector $[\bar{s}, r]^T$. After the solution of the problem, the new design located in the center of the hypersphere can be obtained as $\bar{\alpha}_{new} = \bar{\alpha}_0 + \bar{s}$. However, as can be seen from Fig. 5.1a, one can also take an additional move in the direction of the steepest descent in order to get closer to the optimum. In this case $\bar{\alpha}_{new} = \bar{\alpha}_0 + \bar{s} - \beta r \nabla f / |\nabla f|$ where $0 \leq \beta \leq 1$. Due to such additional move, also the line corresponding to the equality constraints in Eq. (4) should be slightly modified in order to have $h_k(\bar{\alpha}_0 + \bar{s} - \beta r \nabla f / |\nabla f|) = 0$ for $k = 1, n_h$. Apart from the positive effect of accelerating the convergence to the optimum, the additional move can increase the risk of violating the constraints.

A negative aspect of this algorithm is the requirement of move limits [126] for those problems that are under-constrained. In fact, in such a case the linearization can lead to problems that are unbounded. Unfortunately, this is the case for the problems considered here. Therefore, move limits have to be defined together with a reduction factor. The latter has to reduce the move limits when the optimum is approached. One can think the move limits as a box in which the search is limited, an artificial bound for a problem that would be otherwise unbounded.

In practice, when move limits $\bar{b}$ are introduced, the fifth line of Eq. (5.7) is replaced by $-\min(\bar{b}, \bar{s}, \bar{\alpha}_L - \bar{\alpha}_0) \leq \Delta\bar{\alpha} \leq \min(\bar{b}, \bar{s}, \bar{\alpha}_U - \bar{\alpha}_0)$. In the present work the move limits are equal for each variable, therefore they can be written as $\bar{b} = b\bar{u}$, where $\bar{u}$ is the unit vector. The value of b is determined by numerical experiments. Choosing relatively large values for the purpose of speeding up convergence may cause oscillations. Instead, smaller values yield a smooth convergence at the price of increased computing time.

It is possible for certain applications that the initial design is unfeasible. However, before starting the optimization it is desirable to achieve a feasible design. As already mentioned, with minor modifications the method of centers may be used for that purpose. In the geometric construction presented above the linearized objective function does not have to be included. It means that the second line in Eq. (5.7) may be simply neglected and one has to calculate the largest hypersphere that only fits the space delimited by the linearized constraint. In Section 5.5.2 an application is presented where the design is initially unfeasible.

5.2 Mesh deformation and geometric sensitivities

The description of the shape optimization framework is not complete without a description of the mesh deformation strategy. Recall that the latter component kicks in when new boundary mesh displacements, $\Delta\mathbf{X}_B = \Delta\mathbf{X}_B(\bar{\alpha})$, become available from the shape parameterization. The displacements must be imposed on the computational domain and therefore the mesh coordinates $\mathbf{X}$ must be updated accordingly.

Several mesh deformation strategies are available. Probably the simplest and the most widely used one is the spring analogy, which is also used in the present work. The strategy works fine for the Euler type of meshes considered here. It does not work fine on Navier-Stokes meshes, which are characterized by the presence of slender cells in the boundary layer. As discussed below, those cells have the tendency to collapse when the spring analogy is used. Apart from the effectiveness of the mesh deformation strategy, the solution method adopted plays an important role in the efficiency of the whole procedure. In fact, the linear system arising from the mesh deformation equations may be time-consuming to solve, especially for very large meshes, such as for 3D cases.

The sensitivity of the mesh deformation with respect to the displacements must be included in the computation of the final sensitivity. Since a large linear system may be involved, the computation of the sensitivity may be a limiting factor in achieving global efficiency of the framework. In fact, similarly to the flow solver, solving the equations for a number of times equal to the number of the shape parameters may be time consuming.

5.2.1 Spring analogy

The deformation method used here works as follows. The displacement of each node is initialized to zero, except the displacement of some of the boundary points, contained in the set $\mathcal{B}$, which are initialized to $\Delta\mathbf{X}_B$. Those boundary points are only the points that must be displaced (as described below, other boundary points must remain fixed). The displacement of each internal point $\Delta\mathbf{X}_i$ is then iteratively solved with Jacobi iterations:

$$\Delta\mathbf{X}_i{}^{m+1} = \frac{\sum_{j=1}^{N_i} k_{ij}\Delta\mathbf{X}_j{}^m}{\sum_{j=1}^{N_i} k_{ij}}\,, \qquad (i = 1, N\,; i \notin \mathcal{B})\,, \tag{5.8}$$

where N_i is the number of nearest neighbors of the node i and k_{ij} is the stiffness associated with the edge ij. The latter has a numerical value equal to the inverse of the edge length. By means of Eq. (5.8), the boundary displacement $\Delta\mathbf{X}_B$ is iteratively propagated in the whole domain. When the iterations are stopped, the mesh coordinates are updated as $\mathbf{X}^{new} = \mathbf{X} + \Delta\mathbf{X}$.

Note that although the method is referred to as spring analogy, it resembles such an analogy only for the one-dimensional case. In fact, Eq. (5.8) is decoupled and gives a number of independent equations equal to that of the space dimensions. The practical consequence is that a displacement in a certain coordinate direction is only propagated in that direction, e.g., a z boundary displacement only generates z mesh displacements. Intuitively, one can see that, for a deformation method to work properly for large deformations, the displacements should also be propagated in other directions. If not, it is difficult to preserve the shape of the mesh cells, with the result that the mesh may collapse. In general, the algorithm works fine for Euler-flow types of meshes.

The set $\mathcal{B}$ contains a sub-set of the complete set of boundary points. In fact, certain boundary points, such as the farfield or symmetry plane points are not displaced. Still the complete set of points may be used, provided that the displacement of fixed points is set to zero. For certain regions particular care is required. For instance, the tip region of the wing must follow the deformation of the tip airfoil. In the present work, the boundary points at the tip are allowed to displace only along directions that are parallel to the tip section. It means that those points, along the displacement directions, are treated as unknown and solved together with the internal points according to Eq. (5.8). The method is easy to implement since it only implies the use of an additional flag, with no major modifications of the algorithm. However, the computational time appears to increase appreciably as the problem converges at a lower speed. An alternative could be that of smoothing the tip points prior to the application of the mesh deformation algorithm.

As shown by Eq. (5.8), Jacobi iterations make the solution of the spring analogy very easy to implement. Nevertheless, convergence speed is low. Whereas for 2D cases the time required for the solution appears to be negligible, for 3D cases it increases appreciably. Other solution methods, such as the symmetric Gauss-Seidel scheme used for the flow solver, may improve the performance dramatically.

5.2.2 Geometric sensitivities

The term geometric sensitivities is used to indicate the derivatives $\partial \mathbf{R}/\partial \alpha_j$ and $\partial J_i/\partial \alpha_j$ that appear in Eqs. (3.2) and (3.3), which are taken with respect to geometric quantities. It is possible to identify and to distinguish between the contributions of the different operations to the final value of the geometric sensitivities. For instance, consider the residual derivative. The situation is similar for the functional derivatives. In abstract form, the residual vector has a dependency $\mathbf{R} = \mathbf{R}(\mathbf{X}, \mathbf{N}(\mathbf{X}), \nabla \mathbf{V}(\mathbf{X}), \mathbf{\Sigma}(\mathbf{X}))$ on the mesh points $\mathbf{X}$. The latter points are related to the parameters α_j by the relation $\mathbf{X} = \mathbf{X}(\Delta \mathbf{X}_B(\alpha_j))$, where $\Delta \mathbf{X}_B$ are the boundary displacements imposed on the mesh. When a displacement is imposed, the mesh points $\mathbf{X}$ are displaced using the spring analogy described in the previous section. The vector $\mathbf{N}$ contains the metrics of the dual mesh, i.e., the integrated normals and the control volumes. Applying the chain rule gives

$$\frac{\partial \mathbf{R}}{\partial \alpha_j} = \left(\frac{\partial \mathbf{R}}{\partial \mathbf{X}} + \frac{\partial \mathbf{R}}{\partial \mathbf{N}} \frac{\partial \mathbf{N}}{\partial \mathbf{X}} + \frac{\partial \mathbf{R}}{\partial \nabla \mathbf{V}} \frac{\partial \nabla \mathbf{V}}{\partial \mathbf{X}} + \frac{\partial \mathbf{R}}{\partial \mathbf{\Sigma}} \frac{\partial \mathbf{\Sigma}}{\partial \mathbf{X}} \right) \frac{\partial \mathbf{X}}{\partial \Delta \mathbf{X}_B} \frac{\partial \Delta \mathbf{X}_B}{\partial \alpha_k} \,. \tag{5.9}$$

The term $\partial \Delta \mathbf{X}_B/\partial \alpha_j$ represents the differentiation of the shape parameterization whereas the term $\partial \mathbf{X}/\partial \Delta \mathbf{X}_B$ represents the differentiation of the mesh deformation algorithm. The terms $\partial \mathbf{R}/\partial \mathbf{X}$, $\partial \mathbf{R}/\partial \mathbf{N}$, $\partial \mathbf{R}/\partial \nabla \mathbf{V}$ and $\partial \mathbf{R}/\partial \mathbf{\Sigma}$ represent the differentiated residual routine with respect to the different inputs. The terms $\partial \nabla \mathbf{V}/\partial \mathbf{X}$ and $\partial \mathbf{\Sigma}/\partial \mathbf{X}$ represent the differentiation of the gradient and limiter formulation. The term $\partial \mathbf{N}/\partial \mathbf{X}$ represents the differentiation of the median-dual mesh metrics with respect to the mesh points.

In the present work, at least in 2D, the source code to assemble the above terms has been generated by the forward vectorial mode of Tapenade [3]. In 3D the code has not been differentiated yet and the geometric sensitivities have been evaluated by means of a finite difference step, i.e.,

$$\frac{\partial \mathbf{R}}{\partial \alpha_j} = \frac{\mathbf{R}(\alpha_j + \epsilon) - \mathbf{R}(\alpha_j)}{\epsilon}, \qquad \epsilon \to 0\,. \tag{5.10}$$

The finite difference approach is not the best in terms of accuracy. However, it is very easy and straightforward to implement since it just requires an additional residual evaluation for each variable.

Note that the two approaches are not computationally efficient because of the presence of the mesh sensitivity $\partial \mathbf{X}/\partial \Delta \mathbf{X}_B$. In fact, the mesh deformation algorithm is iterative and the above method of computing the geometric sensitivities requires the differentiated mesh deformation algorithm to be run as many times as the number of design variables. Therefore, for very large meshes the computation of the mesh sensitivity may become a bottleneck.

Forward and reverse mesh sensitivity

Equation (5.8) shows that the mesh deformation is linear because the left-hand side terms are obtained by multiplying the right-hand side terms by a constant. Therefore, the mesh sensitivity $\partial \mathbf{X}^{new}/\partial \alpha_j \equiv \partial \Delta \mathbf{X}/\partial \alpha_j$ can be computed with the same algorithm, providing suitable inputs for the boundary displacements. If the boundary displacements in Eq. (5.8) are initialized with $\partial \Delta \mathbf{X}_B/\partial \alpha_j$, which is the sensitivity of the boundary displacements with respect to the shape parameter j, Eq. (5.8) will produce the mesh sensitivity $\partial \Delta \mathbf{X}_i/\partial \alpha_j$, which is the sensitivity of each node displacement with respect to the jth parameter. Note that the sensitivity is consistent only if the number of iterations m used for its computation is the same as that used for the deformation.

Since the sensitivity must be computed for the total number of parameters, the algorithm must be run several times. If there are several parameters and the time required to solve the deformation is large, for instance for 3D cases with deformations solved by Jacobi iterations, the mesh sensitivity becomes very expensive to compute. In the present work, the computational time required to compute 2D mesh sensitivities was a fraction of that required by the flow solver. Instead, for 3D cases the computational times become comparable. For a larger number of shape parameters, say more than 100, more time will be spent on computing the mesh sensitivity than on solving the flow equations.

In order to reduce the computational time, similarly to the flow equations, adjoint equations may be derived also for the mesh deformation equations [96, 84, 5]. Since the deformation operator is symmetric, there are no matrix transposition issues. The same algorithm as that used for the mesh deformation and for the forward sensitivity may be used, providing suitable initial values for the displacements. Note that since the problem is reversed, the displacements are initialized on the internal nodes, and propagated on the boundary. The mesh adjoints have not been implemented in the present work and therefore the equations are not provided here.

Hand-coding of the geometric sensitivities

An explicit implementation of the geometric sensitivities is presented in [5]. In that work the geometric sensitivities are hand-coded and the gradient of the functional is first assembled with respect to the median-dual mesh, and then is propagated to the boundary mesh. The adjoint of the mesh deformation equations is used.

In the present work an explicit implementation of the geometric sensitivities could be performed following the approach presented in Chapter 3. However, instead of using Eq. (5.10), one could express the residual vector only in terms of median-dual quantities. An additional array $\mathbf{X}_E$ of size equal to the number of edges could be introduced. The components of the latter array would be the edge vectors. I.e., the component of $\mathbf{X}_E$ corresponding to the edge ij would be $\mathbf{x}_j - \mathbf{x}_i$. It can be shown that the equations for the metrics in Section 2.2.2 can be rearranged in a way that the integrated normals $\mathbf{n}_{ij}$ and $\mathbf{n}_{Bi}$ are expressed using terms of the type $\mathbf{x}_j - \mathbf{x}_i$, which means that $\mathbf{N} = \mathbf{N}(\mathbf{X}_E)$. The least-squares gradient depends explicitly on $\mathbf{X}_E$, whereas the Green-Gauss gradient depends on $\mathbf{X}_E$ through $\mathbf{N}$. By neglecting the limiters, as in the two-pass assembly of Section 3.3, the residual can be written as

$$\mathbf{R} = \mathbf{R}(\mathbf{H}(\mathbf{U}_\mathrm{L}, \mathbf{U}_\mathrm{R}, \mathbf{N}))\,,$$

whereas the left and right states can be written as:

$$\mathbf{U}_\mathrm{L} = \mathbf{U}_\mathrm{L}(\mathbf{V}_\mathrm{L}(\mathbf{X}_E, \nabla\mathbf{V}))\,, \qquad \mathbf{U}_\mathrm{R} = \mathbf{U}_\mathrm{R}(\mathbf{V}_\mathrm{R}(\mathbf{X}_E, \nabla\mathbf{V}))\,.$$

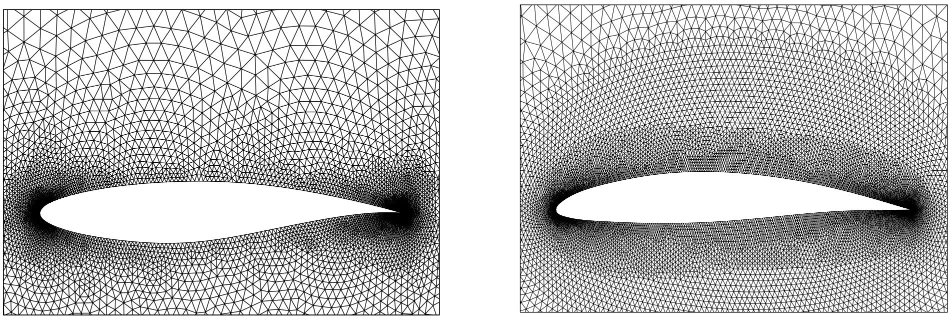
In fact, the numerical fluxes in $\mathbf{H}$ depend on the reconstructed left and right states as well as on the integrated normals $\mathbf{N}$, see Eq. (2.25). The dependency of the reconstructed primitive variables $\mathbf{V}_\mathrm{L}$ and $\mathbf{V}_\mathrm{R}$ on $\mathbf{X}_E$ is due to the MUSCL reconstruction, see Eq. (2.31). The chain rule can be applied to the above equations in order to obtain $\partial\mathbf{R}/\partial\mathbf{X}_E$. The Jacobian with respect to the mesh points can be obtained by multiplying the latter term with the matrix $\partial\mathbf{X}_E/\partial\mathbf{X}$, which has a very simple structure.

Unfortunately, because of the MUSCL reconstruction, the implementation of the aforementioned approach is as complex as that presented in Chapter 3 for the adjoint solver. The expression for $\partial\mathbf{R}/\partial\mathbf{X}_E$, which is not given here, is very lengthy. Several matrix-vector products must be evaluated and a conspicuous amount of differentiation is involved. However, the contribution of the implementation to the overall efficiency of the sensitivity analysis is much smaller than that of the adjoint solver. In fact, the term $\partial\mathbf{R}/\partial\mathbf{X}_E$ is not involved in an iterative process. It must be evaluated as many times as the number of parameters. When the adjoint of the mesh deformation equations is available, only a single evaluation is required.

5.3 2D Applications

The 2D optimization test cases presented below are performed on the mesh of Fig. 5.2a. According to the shape of the initial airfoil, which is given by the shape parameterization, the latter mesh is deformed using the spring analogy described in the previous section.

The mesh has 8238 nodes, that is, 1000 more nodes than the medium mesh used to compute the transonic flow results of Fig. 2.4. Those results involved a coarse mesh of 3477 nodes, a medium mesh of 7240 nodes and a fine mesh of 13261 nodes. As shown by Fig. 2.4 there is no appreciable difference between the solution computed on the medium mesh and the one computed on the fine mesh. Hence, the mesh of Fig. 5.2a, which is finer than the medium mesh, is deemed to be adequate for producing accurate results.



(a) 8238 Nodes, 16096 Cells

(b) 12161 Nodes, 23782 Cells

Figure 5.2: *Unstructured meshes*

The mesh in Fig. 5.2b is used to investigate the effect of the adjoint approximations on the optimization. The number of elements and their density are more than what would be required to compute the flow accurately. In fact, the mesh has been made very fine in proximity of the wall in order to reduce the discretization error as much as possible. Recall that one of the adjoint approximations presented in Chapter 3 neglects the MUSCL reconstruction, which means that the sensitivity of a second-order accurate formulation is computed inconsistently by assuming first-order spatial accuracy. As shown in Section 5.4, the inconsistency of the approximation is such that the gradient is not effective for transonic design cases. By using a very fine mesh one can exclude the possibility that the effectiveness of the approximation can be improved by further reducing the size of the mesh and hence the discretization error.

5.3.1 Inverse design in transonic flow

An inverse design is performed that starts from the RAE2822 airfoil and recovers the NACA0012 airfoil. The free-stream conditions are $M_\infty = 0.75$ and $\alpha = 2^\circ$ angle of attack. The pressure distributions of the two airfoils are shown in Fig. 5.3a, where the dotted line is used for the NACA0012 and the solid line is used for the RAE2822. As can be seen, both airfoils exhibit a strong shock on the upper surface.

The quasi-Newton optimization algorithm converged in slightly more than 180 iterations and the objective function is depicted in Fig. 5.3b. The target NACA0012 pressure distribution is recovered accurately as the pressure of the final airfoil, which is depicted by the black circles in Fig. 5.3a, exactly lies on top of the target pressure. The maximum difference in y-coordinate between the final and the target airfoils is only 2×10^{-4}, a very small value considering that the airfoil is of unit chord. It is remarkable to see that the final airfoil matches the target very accurately also in proximity of the shock.

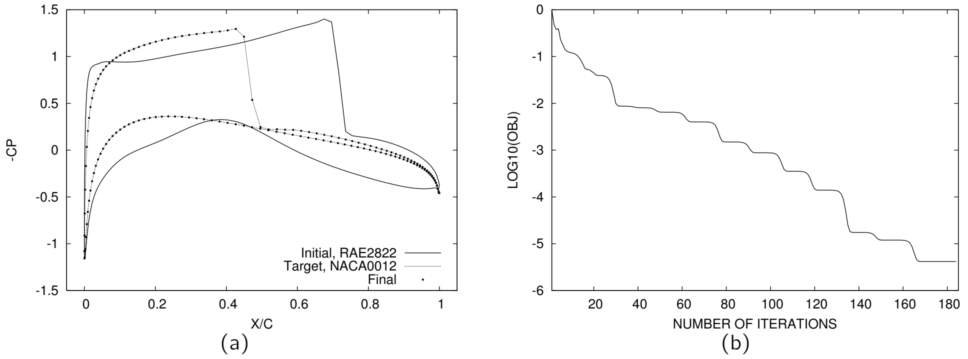


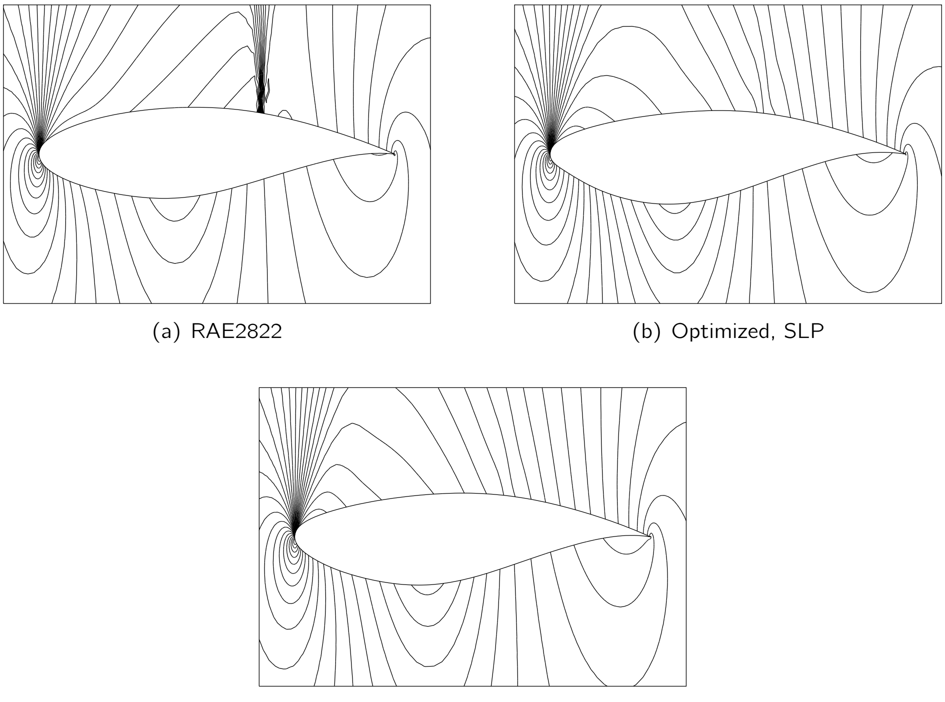
Figure 5.3: *Inverse design. (a) Pressure distributions; and (b) objective function*

5.3.2 RAE2822 airfoil

At $M_\infty = 0.73$ and $\alpha = 2°$ the RAE2822 airfoil has a lift coefficient $c_l = 0.83$ and a drag coefficient $c_d = 8.43\,10^{-2}$. Figure 5.4a, which shows the pressure contours around the airfoil, reveals that a shock is present on the upper side of the airfoil at around 65% of chord. The optimization aims at minimizing the drag while maintaining constant lift and satisfying some geometric constraints. The latter constraints are such that the relative maximum thickness is not allowed to decrease whereas the nose radii and the trailing edge angle cannot decrease more than 30% and 10% of their initial values, respectively. The SLP and SQP algorithms are used to perform the optimization. The contours of the airfoil obtained by the SLP algorithm, see Fig. 5.4b, are similar to that obtained by the SQP algorithm, see Fig. 5.4c. The two results are both shock-free. The shapes are very similar, however small differences exist. Figure 5.5b shows a comparison of the Mach number distributions. It appears that differences are mainly concentrated in the lower side, along the first half of the chord. However different, both designs satisfy the constraints. The lift equality is very accurate. For the SLP algorithm, percentage differences between the final and the initial lift coefficient are of the order of 0.005%.

The objective function is depicted in Fig. 5.5b. For the SLP algorithm two values of the move limits are used: the value 0.05, for which the objective converges smoothly; and the value 0.1, for which the objective initially converges faster, and then starts to oscillate. The two behaviors were already anticipated in the previous section. It appears that the SQP algorithm converges much faster than the SLP algorithm. The greater convergence speed is due to the second order information used by the SQP algorithm.

At the fourth iteration the SQP algorithm experiences a jump in the objective, which is of similar magnitude to the oscillations experienced by the SLP algorithm (with the



(a) RAE2822

(b) Optimized, SLP

(c) Optimized, SQP

Figure 5.4: *RAE2822 optimization. Pressure contours of initial and optimized airfoils.*

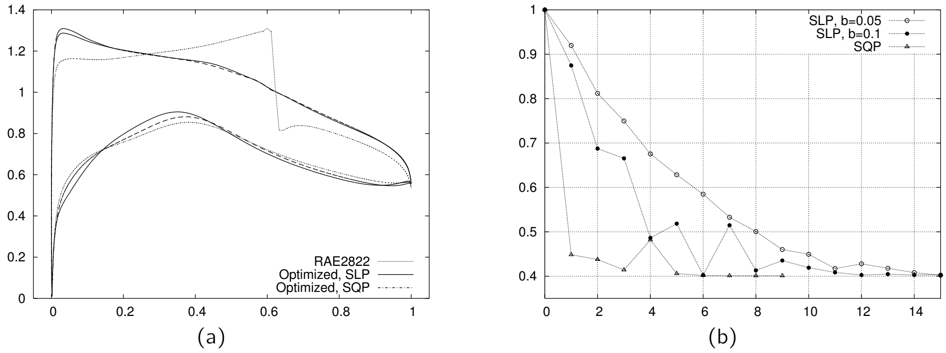
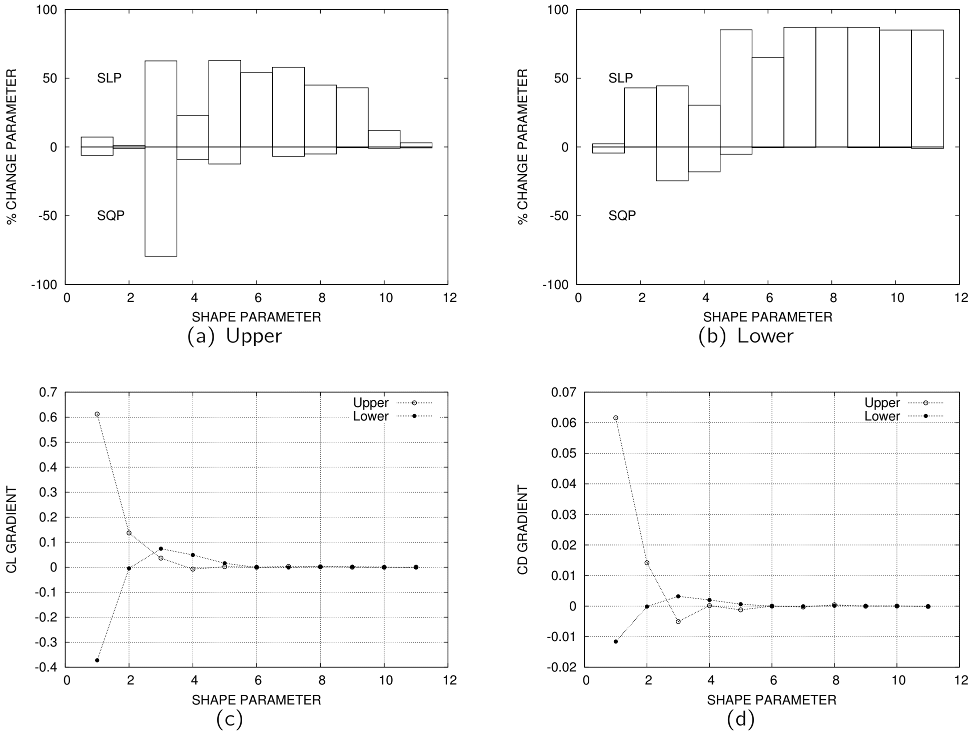


Figure 5.5: *RAE2822 optimization: (a) Mach number distributions; (b) Objective function.*

larger move limit). In general jumps or oscillations are observed when the step sizes chosen by the optimization algorithm are relatively too large. As already mentioned it is desirable to have smooth convergence. The reason is that jumps or oscillations may be the reflection of excessive changes in shape, which may cause the optimization process to break down in certain circumstances. In fact, it is possible that the mesh collapses during the deformation. Moreover, since at each design iteration the flow is restarted from the previous solution in order to save time, it is possible that the solution process diverges because the initial solution is too far from the converged solution.

In case of the SLP algorithm smooth convergence is ensured by choosing small values of the move limits. Instead, in the case of the SQP algorithm the parameters that control the step-size may not be modified because the algorithm is from the Matlab library. For some of the cases solved with the SQP algorithm break-down problems have been experienced.



(a) Upper

(b) Lower

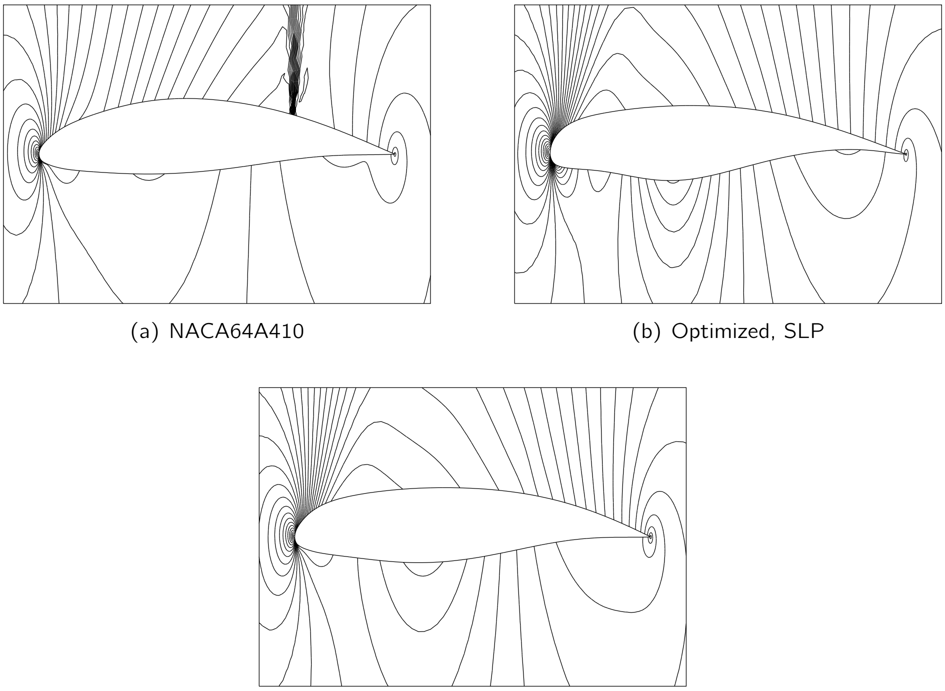
(c)

(d)

Figure 5.6: *RAE2822 optimization: (a-b) Differences in the optimized parameters between the SLP and the SQP algorithm; (c-d) Initial value of the gradients for each parameter.*

It is interesting to look at the differences of the final shape parameters for the two algorithms. Figure 5.6a shows the percentage changes for the upper side of the airfoil. Except for the first three parameters, it appears that differences between the two algorithms are large. Differences become even larger for the lower side parameters, which are shown in Fig. 5.6b. In spite of the large differences, the shapes appear to be similar because high-frequency basis functions have relatively smaller influence on the final shape. In the case of the SLP algorithm, the larger changes in the parameters of the high-frequency basis are the result of using a unique move limit for all the parameters.

Figure 5.6c shows the value of the c_l gradient whereas Fig. 5.6d shows that of the c_d gradient. As can be seen, the magnitude of the gradient components decreases rapidly and starting from the sixth basis becomes very small. The latter behavior of the gradient makes it useless to use a large number of shape parameters. The 11 basis functions used here for each side appear to be satisfactory.



(a) NACA64A410

(b) Optimized, SLP

(c) Optimized, SQP

Figure 5.7: *NACA64A410 optimization. Pressure contours of initial and optimized airfoils.*

5.3.3 NACA64A410 airfoil

The NACA64A410 airfoil at $M_\infty = 0.75$ and $\alpha = 0°$ has $c_d = 1.48\,10^{-1}$ and $c_l = 0.64$. Figure 5.7a shows a strong shock on the upper side of the airfoil. The optimization settings are the same as that of the previous case, except for the constraints on the nose radii and on the trailing edge angle. In this case the latter quantities cannot decrease more than 10% of their initial values.

The pressure contours of the optimized airfoils are shown in Fig. 5.7b, for the airfoil optimized by the SLP algorithm, and in Fig. 5.7c, for the airfoil optimized by the SQP algorithm. Both designs are shock-free and satisfy the constraints accurately, which means that they are equivalent in terms of design specifications. However, they appear to be very different. Specifically, the front and the rear region of the airfoil in Fig. 5.7b are more cambered than that of Fig. 5.7c. A comparison between the initial and the optimized Mach number distributions is shown in Fig. 5.8a. The Mach number distribution on the lower side of the SLP airfoil shows the effect of its large camber. The SQP airfoil, which is less cambered, shows a more flat distribution.

The objective function is depicted in Fig. 5.8b. It shows a reduction of 85% for both optimization algorithms. Again, the SQP algorithm converges faster. Although the designs are clearly different, the final values of the objective functions are the same. Note that in this case, as well as in the previous and next cases, the final objective value, c_d/c_{d_R} , should be zero because the Euler equations are used and the optimized airfoils are shock-free. In practice, the drag does not vanish completely because of the artificial dissipation created by the numerical scheme.

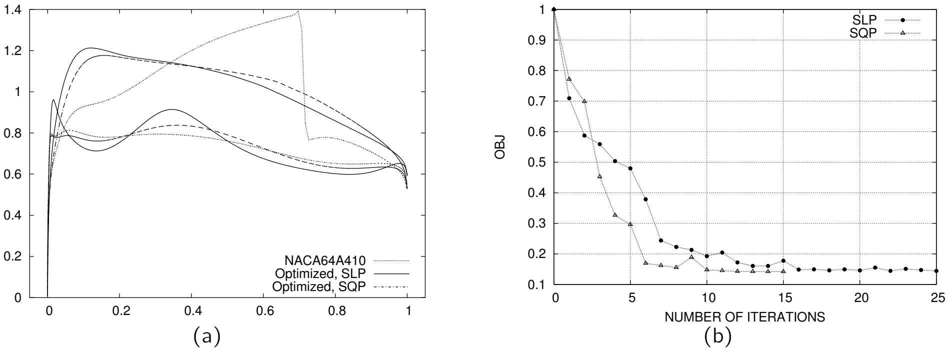
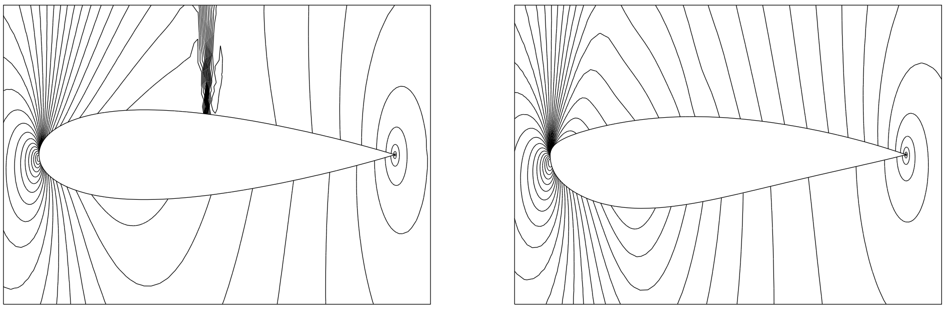


Figure 5.8: *NACA64A410 optimization: (a) Mach number distributions; (b) Objective function.*

The differences between the solutions of the two algorithms are probably due to the fact that, unlike the drag and the lift, the pitching moment is not included in the optimization problem. Imposing a constraint on the pitching moment could have the effect of reducing the feasible design space, hence making the differences between the solutions disappear. In the following section a case is presented, see Fig. 5.13, in which the pitching moment constraint has that effect.

5.3.4 NACA0012 airfoil

The drag of the NACA0012 airfoil is minimized at $M_\infty = 0.75$ and $\alpha = 2°$, for which the airfoil has $c_d = 1.25\,10^{-2}$ and $c_l = 0.42$. Figure 5.9a shows a strong shock on the upper side of the airfoil, almost at half chord. The optimization settings are the same as those of the previous case and only the SLP algorithm has been used. Also in this case the optimized airfoil is shock-free, as shown by the pressure contours of Fig. 5.9b. All the constraints are satisfied. A comparison of the Mach number distribution is shown

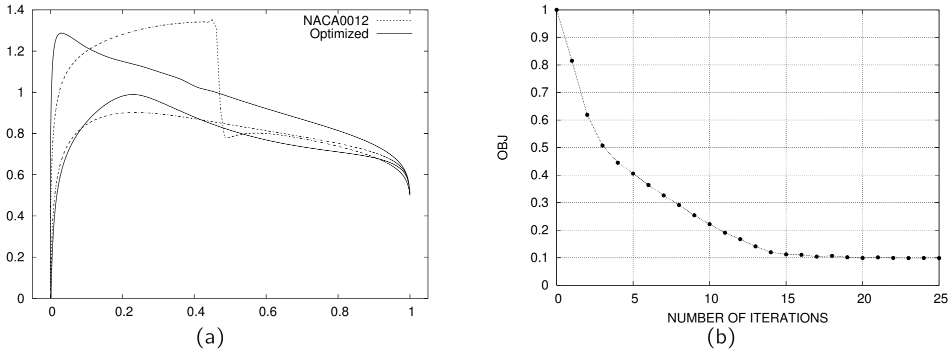


(a) NACA0012

(b) Optimized, SLP

Figure 5.9: *NACA0012 optimization. Pressure contours of initial and optimized airfoils..*

in Fig. 5.10a. The optimized airfoil has a less pronounced camber compared to the previous cases because the lift coefficient is also smaller. It has half the lift of the RAE2822 airfoil. In general, for increasing lift coefficients, shock-free transonic airfoils need an increased camber in the rear of the airfoil in order to increase the high-pressure at the bottom side. Figure 5.10a shows the objective function, which smoothly reduced by 90%.



(a)

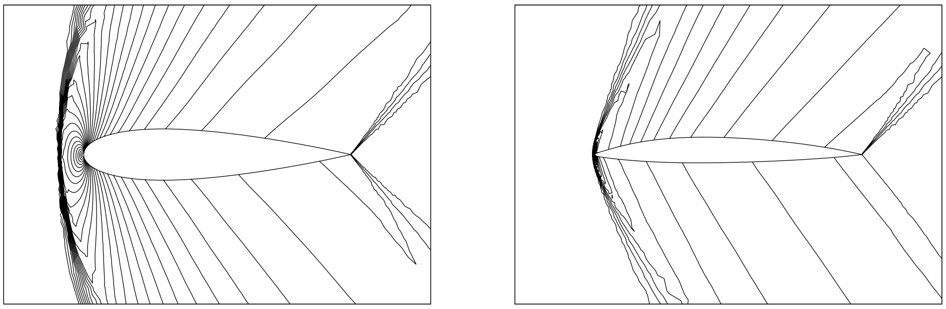
(b)

Figure 5.10: *NACA0012 optimization. (a) Mach number distributions; (b) Objective function.*

5.3.5 NACA0012 airfoil at supersonic flow conditions

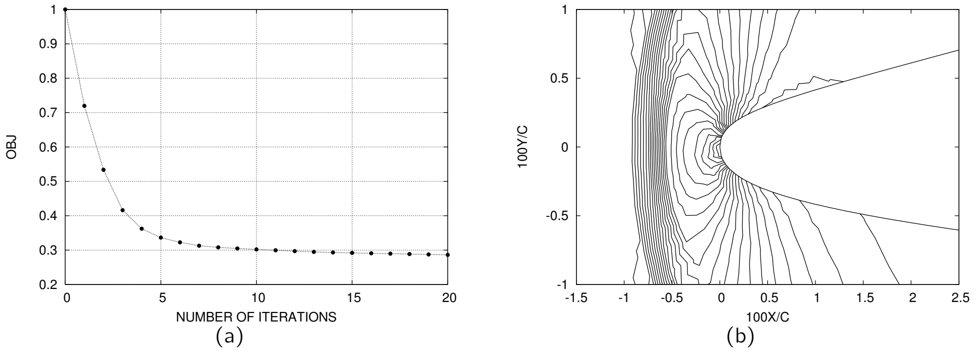
The NACA0012 airfoil has also been optimized at $M_\infty = 1.5$ and $\alpha = 2°$, a supersonic flow condition for which a strong detached bow shock is visible, as shown in Fig. 5.11a. A supersonic optimization is a good test case for the shape parameterization and for the geometric constraints. Both play a crucial role in supersonic optimization because, in order to reduce the drag, the airfoil nose is expected to become sharper, causing the bow shock to be almost attached to it and oblique almost everywhere. Shock-wave drag increases with increasing velocity normal to the shock wave. Consequently, oblique shocks feature smaller drag than normal shock waves for the same upstream Mach number. However, the situation of a sharpening nose must be handled carefully. The danger exists that the upper and the lower curves cross each other. Moreover, real supersonic airfoils do not have perfectly sharp noses but noses with small radii of curvature. Thus, geometric constraints must be enforced properly that allow small radii to exist without generating unfeasible designs.

To facilitate the movement of the shock in the direction of the nose, the radii of curvature are allowed to reduce up to 90% of their initial values. In terms of relative thickness, the 12% of the NACA0012 airfoil is clearly unsuitable for supersonic flow conditions. A more suitable value could be less than or equal to 6%. Therefore, the relative maximum thickness is allowed to halve. Also in this case, c_l must be kept constant. Only the SLP algorithm has been used.



(a) (b)

Figure 5.11: *NACA0012 optimization. Pressure contours: (a) NACA0012 airfoil; (b) Optimized airfoil.*



(a) (b)

Figure 5.12: *NACA0012 optimization: (a) Objective function; (b) Magnification of the nose of the optimized airfoil.*

The pressure contours of the optimized airfoil are shown in Fig. 5.11b. As can be seen, the shock is almost attached to the nose. The repositioning of the shock caused the drag to reduce by 70%, as shown by the objective function in Fig. 5.12a, which converged smoothly in 20 design iterations. A magnification of the nose region is shown in Fig. 5.12b, which reveals that the shock is located at a distance from the nose that is less than 1% of the chord. The normal region of the shock has a very small extent compared to the initial airfoil and the nose appears to be still rounded and smooth. Although not shown, both the upper and lower nose curvature constraints are critical. It means that they approached zero and therefore the curvature equals the reference curvature value. As already mentioned, the reference curvature is set to 10% of the initial value. The thickness constraint is also critical.

It is remarkable that also in this case, in spite of the challenging flow conditions and the large deformations involved, the whole framework proved to be effective and robust. Also, it is interesting to see that the final shape resembles that of lenticular airfoils, typically used in the supersonic regime.

5.4 Effect of approximations

The approximations on the adjoint code have been discussed in Chapter 3 and are summarized here for convenience:

– Approximation 1 is obtained by neglecting the differentiation of the limiter in the MUSCL reconstruction operator;

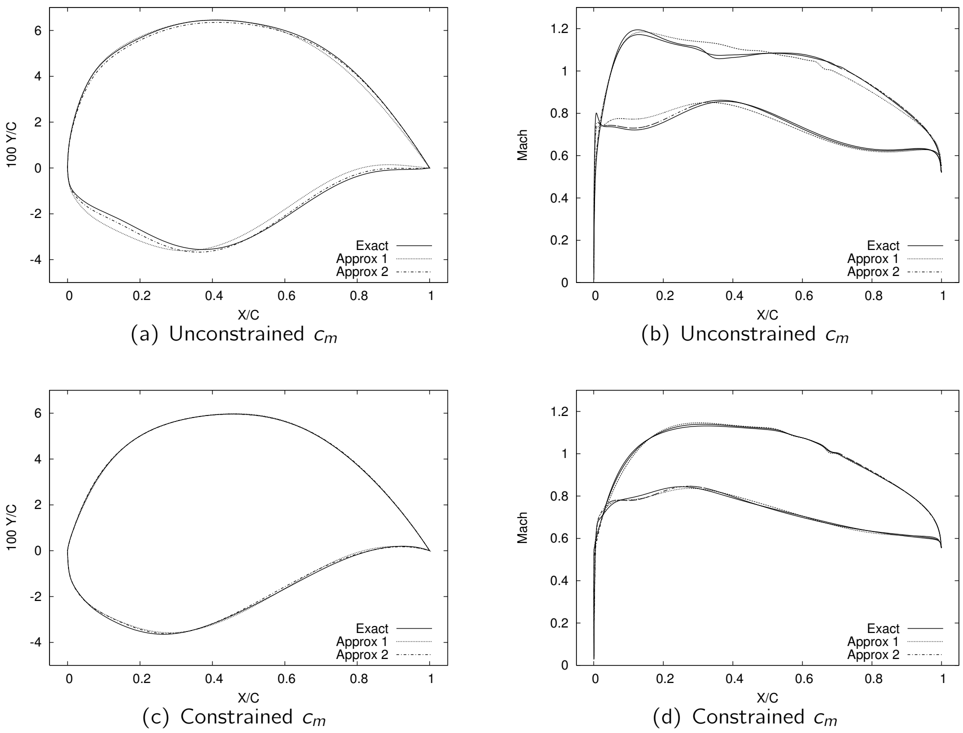
– Approximation 2 is obtained by neglecting the differentiation of the Jacobian matrix in the Roe flux;

– Approximation 3 is obtained by neglecting the complete MUSCL reconstruction operator.

The above approximations simplify the adjoint code considerably. However, as already mentioned, simplifications come at the price of producing inexact gradients, which may affect the optimization negatively. Thus, when introducing approximations, it is important to verify whether the approximate code is effective or not. The error in the approximate gradient, which has been discussed in Section 3.5.3, may not be a good indication of effectiveness. It is best to test the effect of the approximations directly on the optimization test cases. In the following some of the optimizations presented in the previous section are repeated with the exact and the approximate sensitivities.

5.4.1 SQP algorithm

The NACA64A410 optimization test case of the previous section is performed with the SQP algorithm. The shapes obtained by using the exact formulation and the first two approximations are shown in Fig. 5.13a. Differences are visible, especially at the lower side. The corresponding shock-free Mach number distributions are shown in Fig. 5.13b whereas the pressure contours are shown in Fig. 5.15 a-c, respectively. When the third approximation is used, the optimization stalls. The result is a shape that has a rather strange pressure distribution, as shown by Fig. 5.15d, with three shocks on the upper side of the airfoil.



(a) Unconstrained c_m

(b) Unconstrained c_m

(c) Constrained c_m

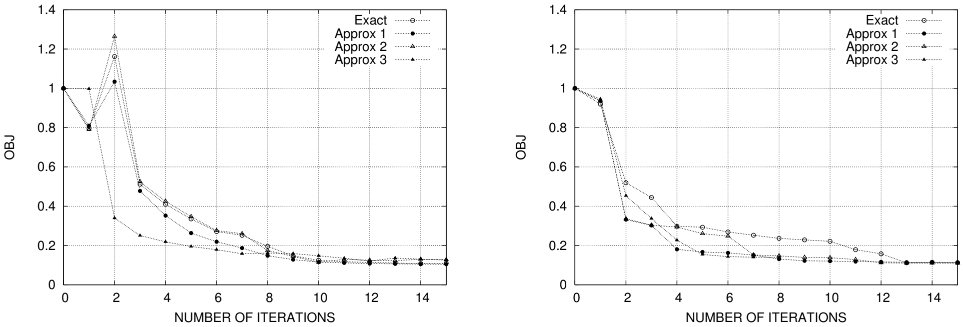
(d) Constrained c_m

Figure 5.13: *NACA64A410 optimization. Shapes and Mach distributions for (a-b) unconstrained; and (c-d) constrained c_m cases. Different adjoint approximations used.*

Figure 5.14a shows the objective functions whereas the table at the bottom of the figure shows the number of function calls and gradients evaluations for each approximation. As can be seen, the objective function corresponding to the third approximation achieves a final value that is very close to that of the other approximations. The good agreement of the values seems strange because the third approximation gives airfoils that are not shock-free, for instance, see Fig. 5.15d and Fig. 5.15h. However, the shocks are very weak, with upstream Mach numbers ≈ 1.1. Therefore, the pressure drag rise, which is a quadratic function of the upstream Mach number, remains very small.

The function calls for the third approximation, see the table of Fig. 5.14, exceeded the maximum allowed number of calls, which was set to 2000. 97% of the latter calls have been performed at the 17th iteration, when the algorithm stalled and could not find any direction of improvement.

Differences in shapes between the different approximations are less pronounced if the

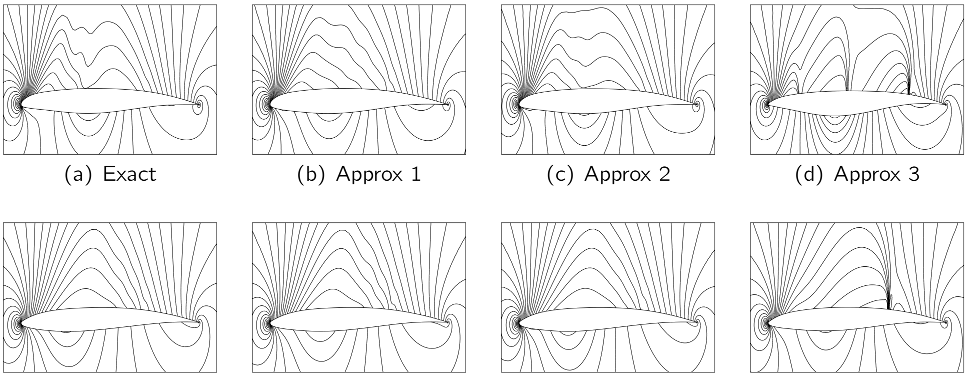


(a) Unconstrained c_m

(b) Constrained c_m

Function calls	Gradients		Function calls	Gradients
49	19	Exact	51	24
44	17	Approx 1	63	22
86	34	Approx 2	71	19
∞	17	Approx 3	∞	11

Figure 5.14: *NACA64A410 optimization, with and without c_m constraint.*

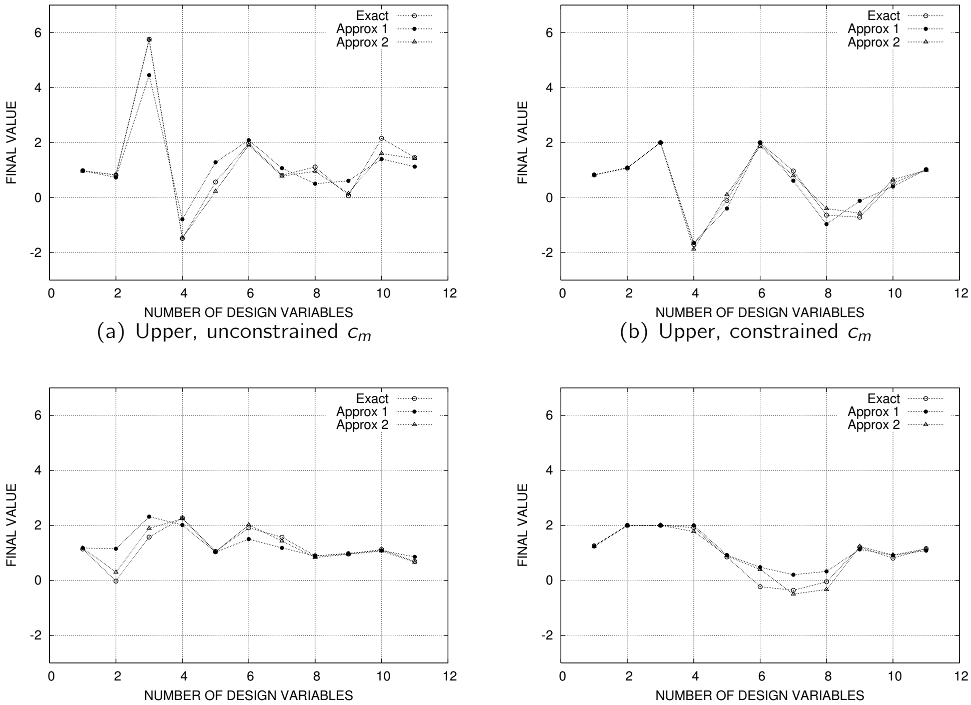


(a) Exact (b) Approx 1 (c) Approx 2 (d) Approx 3

(e) Exact (f) Approx 1 (g) Approx 2 (h) Approx 3

Figure 5.15: *NACA64A410 optimization. Pressure contours of optimized airfoils with unconstrained c_m, top row, and constrained c_m, bottom row.*

optimization test case is repeated with an inequality constraint on the pitching moment. Shapes obtained when the latter constraint is imposed are shown in Fig. 5.13c. The corresponding shock-free Mach number distributions and pressure contours are shown in Fig. 5.13d and Fig. 5.15 e-g, respectively. The final shapes are probably similar because of the restriction imposed on the feasible design space by the additional constraint. The shape parameters for the constrained and unconstrained cases are shown in Fig. 5.16. The spread between the different approximations is evident in the unconstrained case, see Fig. 5.16 a-c, especially for the low-frequency parameters, which highly influence the final shape. Instead, for the constrained case, see Fig. 5.16 b-d, a small spread only appears for the high-frequency parameters, which have a small influence on the final shape.



(a) Upper, unconstrained c_m

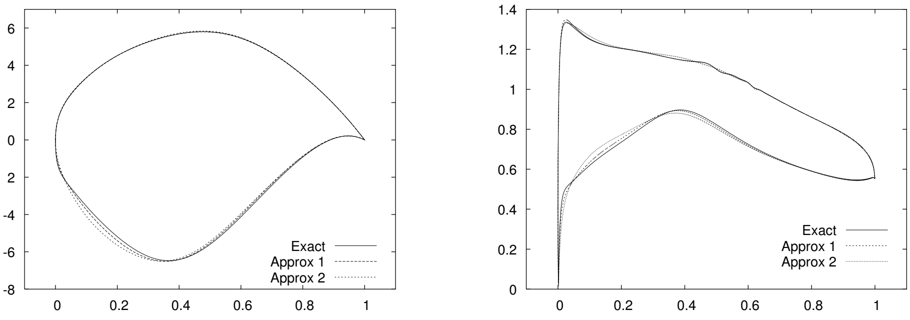
(b) Upper, constrained c_m

(c) Lower, unconstrained c_m

(d) Lower, constrained c_m

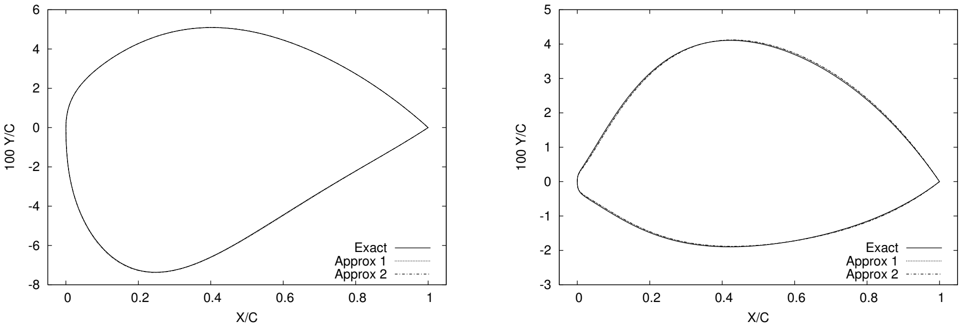
Figure 5.16: *NACA64A410 optimization with different adjoint approximations. Final value of the shape parameters for constrained and unconstrained pitching moment.*

Also in the constrained case, when the third approximation is used, the optimization stalls. The result is an airfoil that still has a shock on the upper surface, as shown by Fig. 5.15h. The table of Fig. 5.14 shows that for the constrained case the approximations require less gradient evaluations but more function calls. The ratio of function calls to gradient evaluations appears to be increasing from 2.12, for the exact code, to 3.74, for the second approximation. Instead, the unconstrained case features ratios around the value 2.6. The above values of the ratio are what is usually expected from an SQP type of algorithm [126].



(a) (b)

Figure 5.17: *RAE2822 optimization. (a) Airfoil shapes; and (b) Mach number distributions, obtained by using different adjoint approximations.*



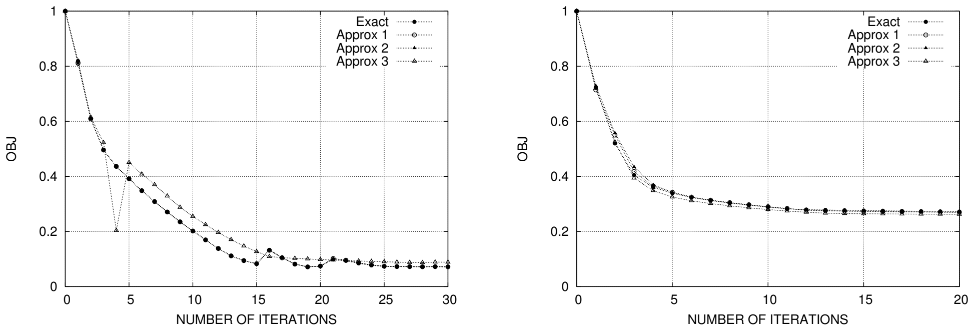
(a) Subsonic case (b) Supersonic case

Figure 5.18: *NACA0012 optimization. Shapes obtained with different adjoint approximations.*

Another case for which the different approximations have been employed is the optimization of the RAE airfoil. Similarly to the previous case (the one with unconstrained c_m), Fig. 5.17a shows that the optimized airfoils are slightly different. Differences are concentrated in the front part of the lower side. The airfoils are shock-free, see Fig. 5.17b, except for the one obtained by the third approximation, where the shock is not completely removed.

5.4.2 SLP algorithm

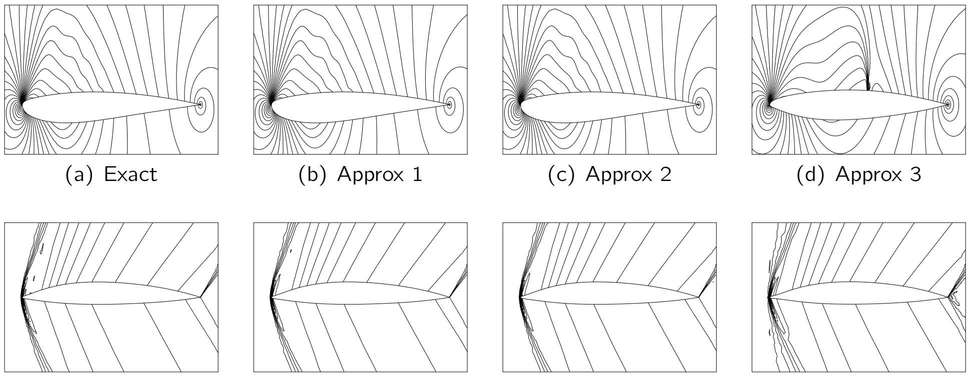
The NACA0012 transonic and supersonic cases have been repeated with the SLP algorithm. Figure 5.18 shows the final shapes, which are almost identical. Figure 5.19 shows the objective functions. In the transonic case the objectives overlap, except for



(a) Subsonic case (b) Supersonic case

Figure 5.19: *NACA0012 optimization. Objective function for different adjoint approximations.*

the third approximation, for which the optimization stalls. The pressure contours are shown in the top row of Fig. 5.20. As can be seen from Fig. 5.20d, the airfoil obtained with the third approximation is not shock-free.



(a) Exact (b) Approx 1 (c) Approx 2 (d) Approx 3

(e) Exact (f) Approx 1 (g) Approx 2 (h) Approx 3

Figure 5.20: *NACA0012 optimization. Pressure contours of optimized airfoils for the subsonic, top row, and the supersonic cases, bottom row.*

The bottom row of Fig. 5.20 shows the pressure contours for the supersonic case. The airfoil obtained with the third approximation, see Fig. 5.20h, is different, especially in the nose region. Differently from the transonic cases, the airfoil obtained by the third approximation is slightly better. In fact, the objective labeled 'Approx 3' in Fig. 5.19b is 3.5% smaller than the other objectives. Probably, in terms of gradient accuracy, the supersonic design considered here is less demanding than the shock-free and the inverse design cases. I.e., there are several shapes available that can improve the drag. Instead, one can intuitively see that the shock-free condition is a rather isolated condition in the design space. In fact, it is known that there is only one design point for which the design is shock-free [88]. Moreover, this design point may be ill-conditioned: a small change in a flow condition or design parameter may cause a strong deterioration of the design. The same is for the inverse design (see below), for which there is only a single shape that can match a certain pressure. Probably, the reconstruction plays a major role in those cases and the third approximation is not effective because it neglects its contribution.

5.4.3 BFGS algorithm

The inverse design case presented in the previous section has been performed with a quasi-newton optimization algorithm known as BFGS. The effect of the different approximations on the convergence of the objective function is shown in Fig. 5.21a. It appears that the first and second approximations behave very similarly and they almost overlap. As can be seen, the two approximations reduce the objective of fourth order of magnitudes, which is not as much as that obtained by the exact formulation. Nevertheless, there is no appreciable difference in coordinates between the airfoils obtained by the exact and the approximate formulations. Figure 5.21b shows a magnification of the lower region of the airfoil, from the nose to the first quarter of the chord. As can be seen, the points corresponding to the exact and to the first two approximate formulations are indistinguishable. Instead, the third approximation is not effective and the points appear to be far from the target. In fact, as shown by Fig. 5.21a, the optimization stalled and the objective reduced by slightly more than two orders of magnitude.

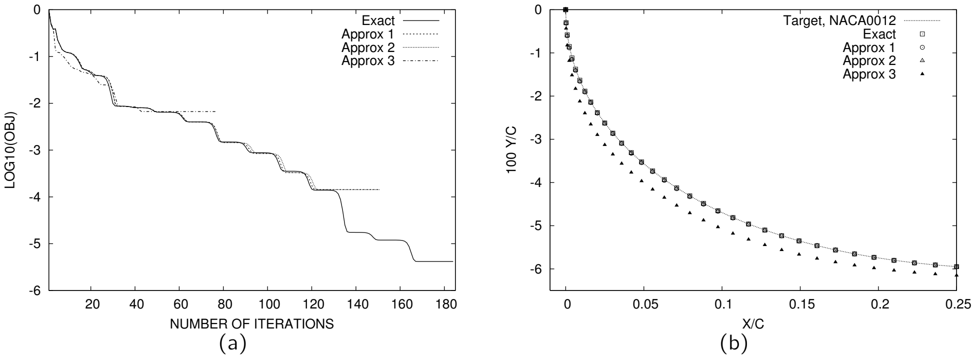


Figure 5.21: *Inverse design with different adjoint approximations: (a) Objectives; and (b) magnified airfoil shapes.*

5.5 3D Applications

In the following the shape of the ONERA-M6 wing is optimized by means of the SLP algorithm for three different cases, all at $M_\infty = 0.84$ and $\alpha = 3.06°$. The same unstructured mesh is used for all the cases. The mesh is very similar to that of Fig. 2.8, but with a different number of elements (125389 nodes, 591713 cells). Only the shape functions described in the previous chapter are used to parameterize the shape. The sweep angle of the wing, the twist angles and the chord lengths of the wing sections are frozen. Therefore, the planform of the wing is always flat and its projection on the horizontal plane does not change during the optimization. Clearly, a more realistic optimization case would have to include the possibility to deform the planform. For the shape parameterization a quadratic representation is chosen, as described in Section 4.2.4, which uses 30 parameters for each side of the wing (i.e., a total of 60 parameters for the complete wing).

Geometric constraints could be imposed on the relative maximum thickness, on the nose radii and on the trailing edge angle of each section. In order to reduce their number, the constraints are imposed on the maximum values of the latter quantities along the span. E.g., once the relative maximum thicknesses are evaluated along the span for each section, only the maximum of those values is taken for imposing the thickness constraint. An inequality constraint on the wing volume may also be imposed, which avoids the volume to decrease below a certain value. The constraint may be useful in the case that a minimum volume is required inside the wing, for instance to accommodate a fuel-tank.

An additional constraint has been imposed on the maximum camber of the tip airfoil. The constraint must be considered as a quick fix to the problems caused by the adoption of a frozen wing planform. In fact, the lack of twist deformation forces the optimizer to change the camber, which is the only available degree of freedom for redistributing the lift along the span. Unfortunately, in certain circumstances, large variations in camber and thickness may result in unfeasible shapes. As shown below, constraining the camber is a quick fix in the sense that it is not a final and satisfactory solution to the problem.

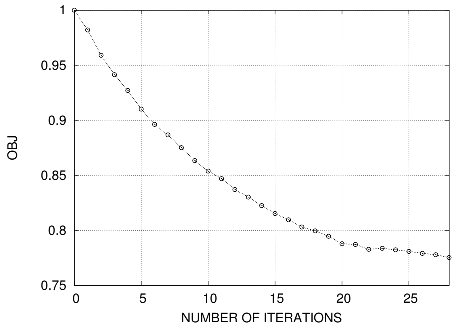


Figure 5.22: *ONERA-M6 optimization. Objective function.*

5.5.1 Drag minimization

The objective function of a drag minimization at constant lift is shown in Fig. 5.22. It takes 28 iterations for the drag to reduce 23%. All the constraints are satisfied, although they are not shown. As can be seen, the objective is not completely flat and probably there is still room for some improvements. Unlike 2D cases, which are converged accurately because they are inexpensive to run, 3D cases are time consuming and they are not fully converged. Parallelization of the code will make 3D computations much less time consuming.

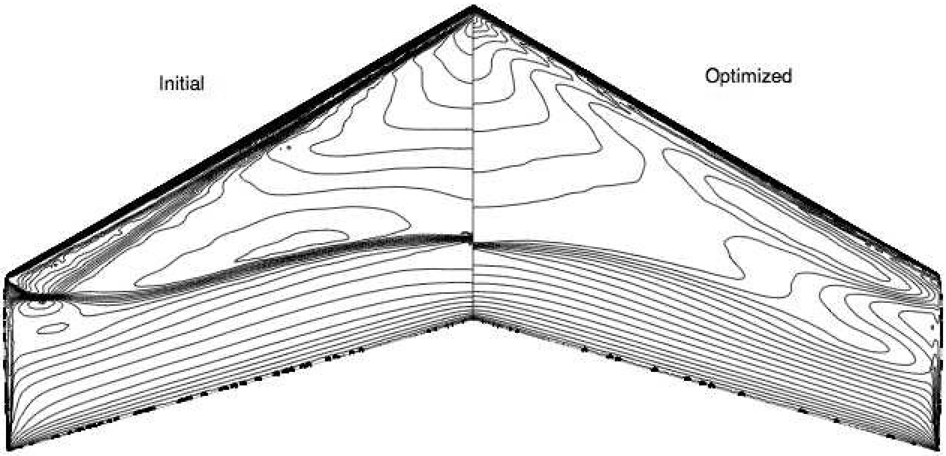
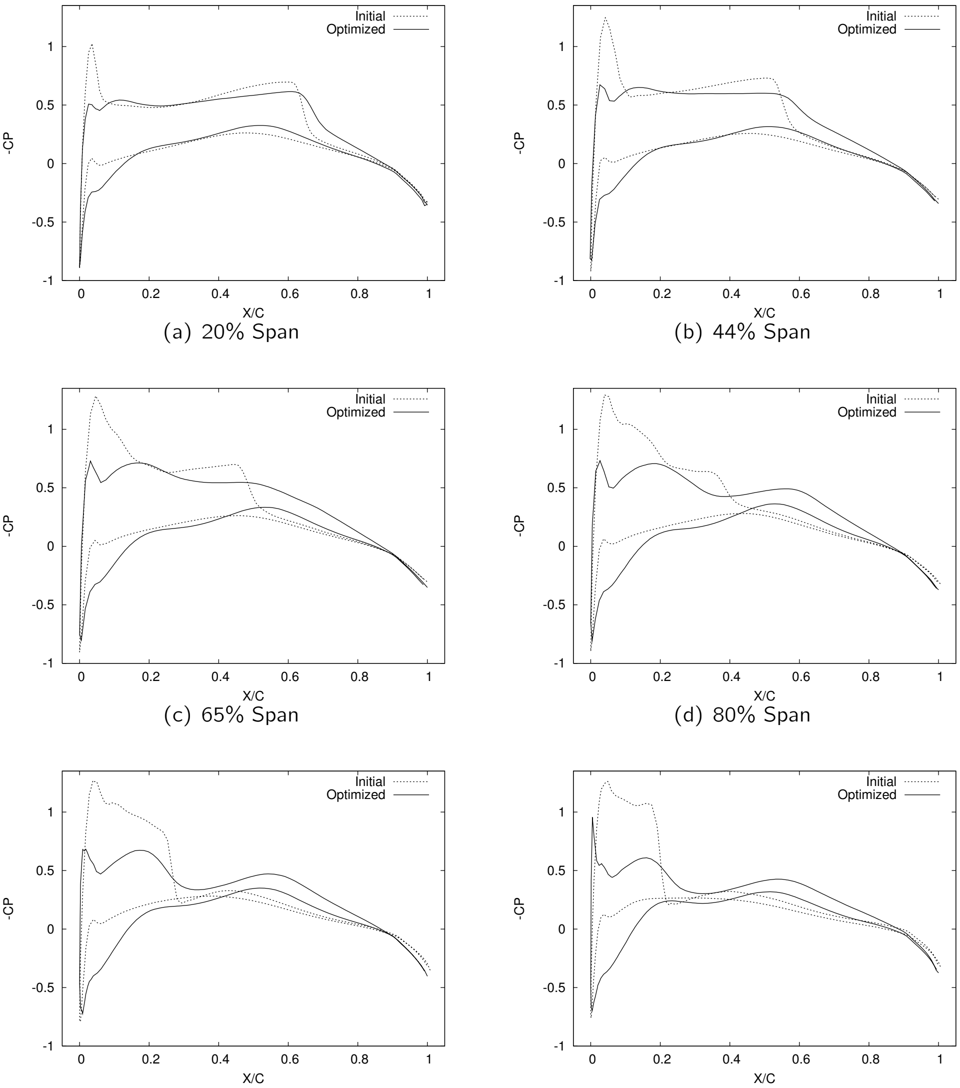


Figure 5.23: *ONERA-M6 optimization. Pressure contours: ONERA-M6 wing (left); Optimized wing (right).*

Figure 5.24 shows the comparisons of the initial and the optimized pressure distributions at different sections along the span. Figure 5.23 shows a comparison of the pressure contours on the upper surface of the wing. The strength of the shocks appears to be reduced appreciably. It also appears that the velocity spikes along the leading edge of the original wing, which result in a strong shock immediately after the leading edge, are reduced considerably in the optimized wing. For instance, in Fig. 5.24a and Fig. 5.24b the negative pressure coefficient exceeds the value 1 for the initial wing, but stays around 0.5 on the optimized wing.

The comparison between the initial and the optimized airfoils at different sections along the span is shown in Fig. 5.25. As can be seen, the optimized shapes of the 90% and 95% sections, which are very close to the tip, do not appear to be very smooth in proximity of the leading edge. The lack of smoothness in the region is also reflected in the pressure distribution, see Fig. 5.24f, which shows a very sharp spike. The undesired behavior is caused by the very bad initial condition. In fact, the left side of Fig. 5.23 shows that the wing region between 90% of the span and the tip is affected by a strong shock, which appears to be the result of the union of the two inboard shocks.

The only pressure difference in the region, (i.e., the difference between the pressure at the upper and the one at the lower side) as shown by Fig. 5.24e and Fig. 5.24f, is present upstream of the shock itself (first 30% of the chord). The latter pressure difference must be redistributed along the section for the purpose of reducing the shock strength. However, in this redistribution there is a lift constraint and the planform is fixed. As for the other sections, the pressure difference has been shifted down, reducing



(a) 20% Span

(b) 44% Span

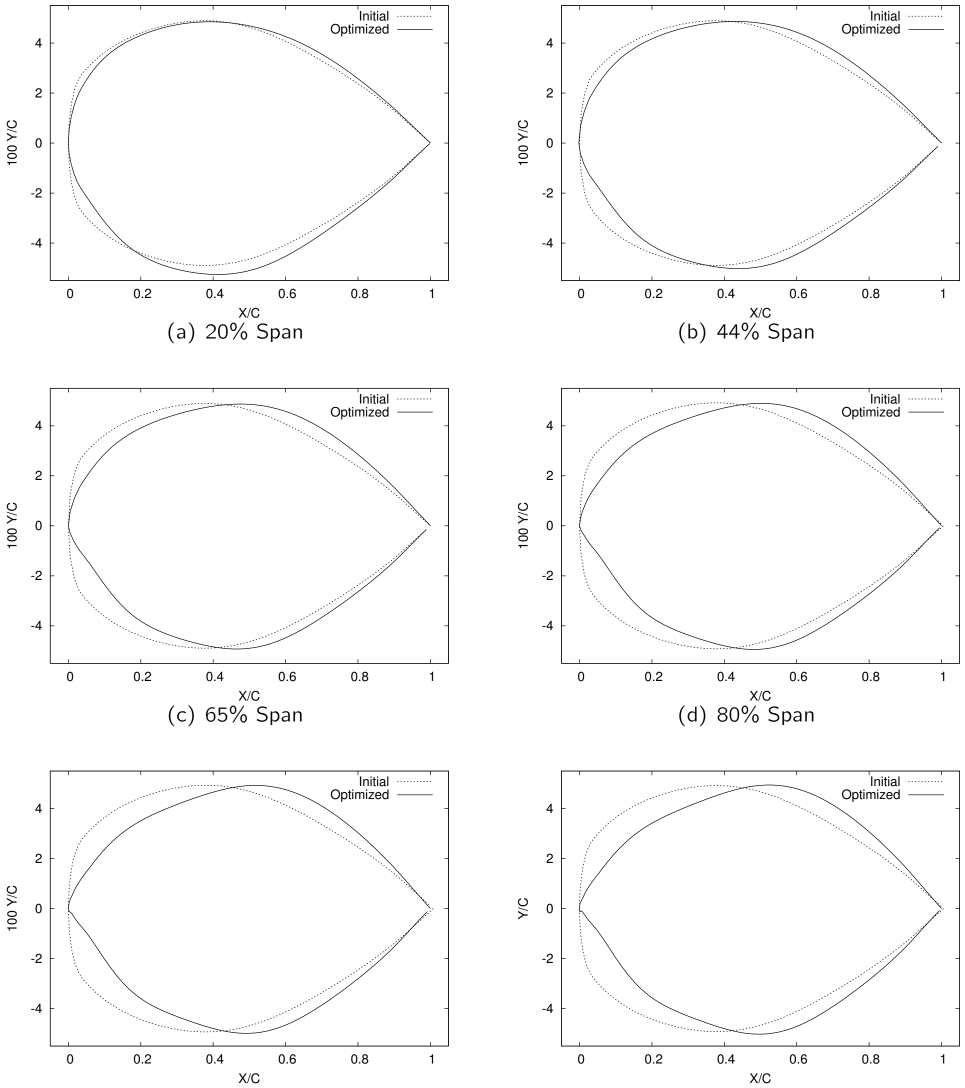
(c) 65% Span

(d) 80% Span

(e) 90% Span

(f) 95% Span

Figure 5.24: *ONERA-M6 optimization. Comparison of initial and optimized pressure distributions at different sections along the span.*



(a) 20% Span

(b) 44% Span

(c) 65% Span

(d) 80% Span

(e) 90% Span

(f) 95% Span

Figure 5.25: *ONERA-M6 optimization. Comparison of the initial and optimized airfoil shapes at two different wing sections.*

the low-pressure on the upper side and increasing the high-pressure on the lower side. Unfortunately, the pressure difference is very large compared to that of the other sections and requires very large deformations, which cannot be accommodated solely by the shape of the airfoil.

Without a constraint on the camber of the tip airfoil, the shape of the airfoils in the tip region tends to be even worse than that in Fig. 5.25f. However, as already mentioned the constraint on the camber is only a quick fix and other solutions should be adopted. It is likely that the situation could be improved in two ways. First, the implementation of twist deformation would be very beneficial for redistributing the pressure, giving more degrees of freedom to the optimizer. Second, the implementation of thickness constraints at different points along the chord, instead of a single one on the maximum value only, would be very beneficial for avoiding extreme thickness reduction at the nose. Those improvements are left as future work.

5.5.2 Doubling the lift and minimizing the drag

In the last section the cause for obtaining non-smooth airfoil shapes was attributed to the combination of the severe initial condition and the lack of additional degrees of freedom such as the planform twist. In order to provide evidence that the shape parameterization is not responsible, another optimization case is presented, which has a different initial condition. The test case gives the possibility to show another feature of the SLP algorithm, which is the already mentioned capability to bring unfeasible designs into the feasible design space. The test case is divided in two parts. In the first part a wing that has two times the lift of the original ONERA-M6 wing is obtained. In the second part the drag of the new design is minimized, similarly to the previous case.

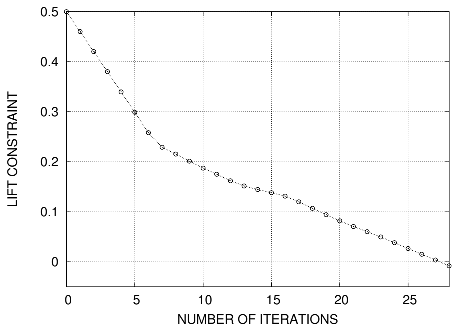


Figure 5.26: *Design to double the lift of the ONERA-M6 wing. Lift constraint.*

Part 1: Doubling the lift

The problem of doubling the lift may be addressed as an optimization problem and solved by the SLP algorithm, with the modifications described in Section 5.1.3. It should not

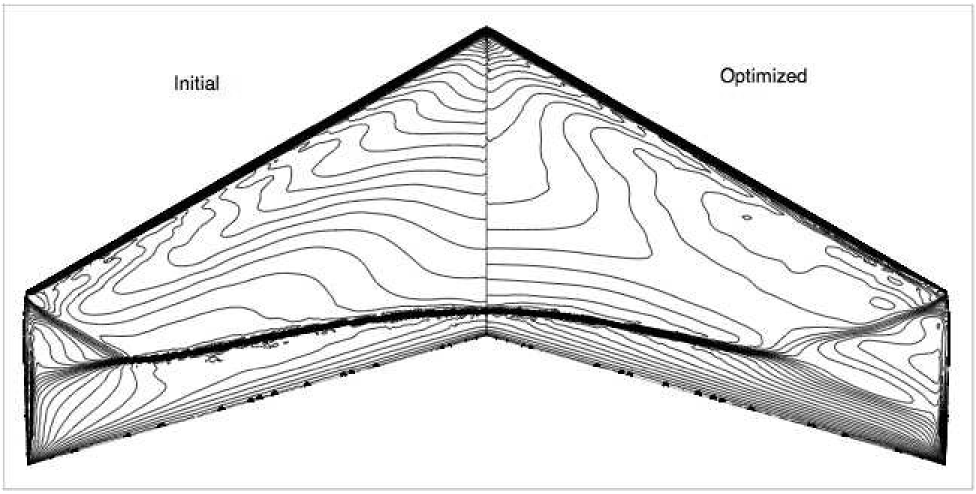


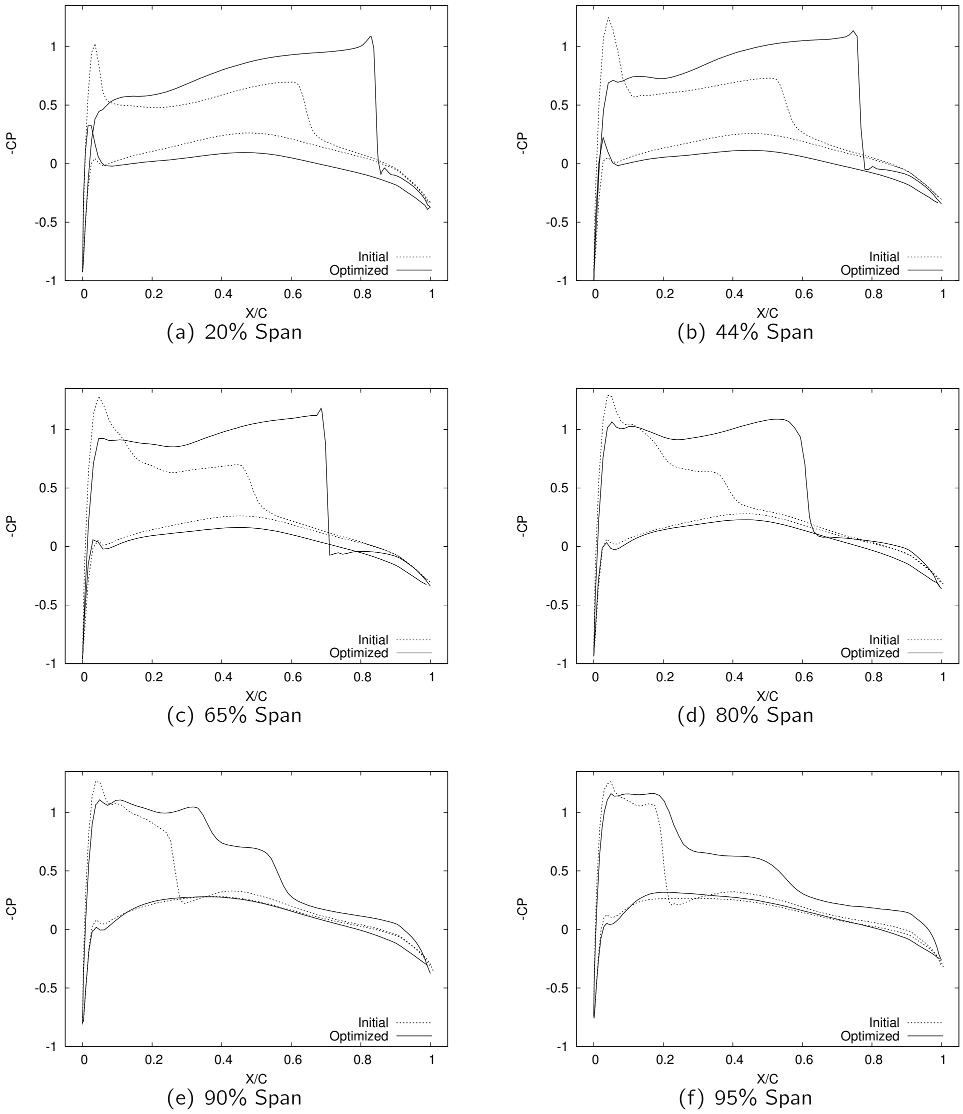
Figure 5.27: *Pressure contours of the wing obtained by doubling the lift of the ONERA-M6 wing (left); and by subsequently minimizing the drag (right).*

be confused with a lift maximization, as the aim is just to bring the design in the feasible space. The latter space is defined by the constraints. Since the lift must be doubled, the lift constraint must be written as $g = 1 - c_l/2c_{l0}$. The constraint is satisfied when $c_l > 2c_{l0}$ and by definition has an initial value of 0.5 because the initial lift is c_{l0}. All the other constraints enforced in the previous case are left in place, with no modifications.

Figure 5.26 shows the value of the lift constraint during the iterations. As can be seen, after 28 iterations the constraint is satisfied, $g < 0$. The increased lift is clearly visible when the initial and the final pressure distributions are compared, see Fig. 5.28. The new pressure distribution shows a strong shock, which is distributed along the whole span, and which causes a considerable increase in total drag. Figure 5.29 shows a comparison between the airfoils. The final airfoils appear to be deformed appreciably compared to the initial ones. Nevertheless, their shapes appear to be very smooth, also close to the tip. Especially for the airfoils that are close to the root, there has been a large increase in camber. For instance, the root section has a maximum value of the camber of 3%.

Part 2: Drag minimization

The design to double the lift of the ONERA wing has produced a new wing with a drag coefficient that is 3.1 times larger than that of the ONERA wing. In order to reduce the drag of the new wing, a drag optimization similar to that of Section 5.5.1 has been performed. The optimization has reduced the large drag value with almost 21%. As can be seen from Fig. 5.27, which shows a comparison of pressure distributions on the upper surface of the wing, part of the shock has disappeared around the tip region. However, as the shock on the whole wing has not disappeared, but only reduced in strength, the



(a) 20% Span

(b) 44% Span

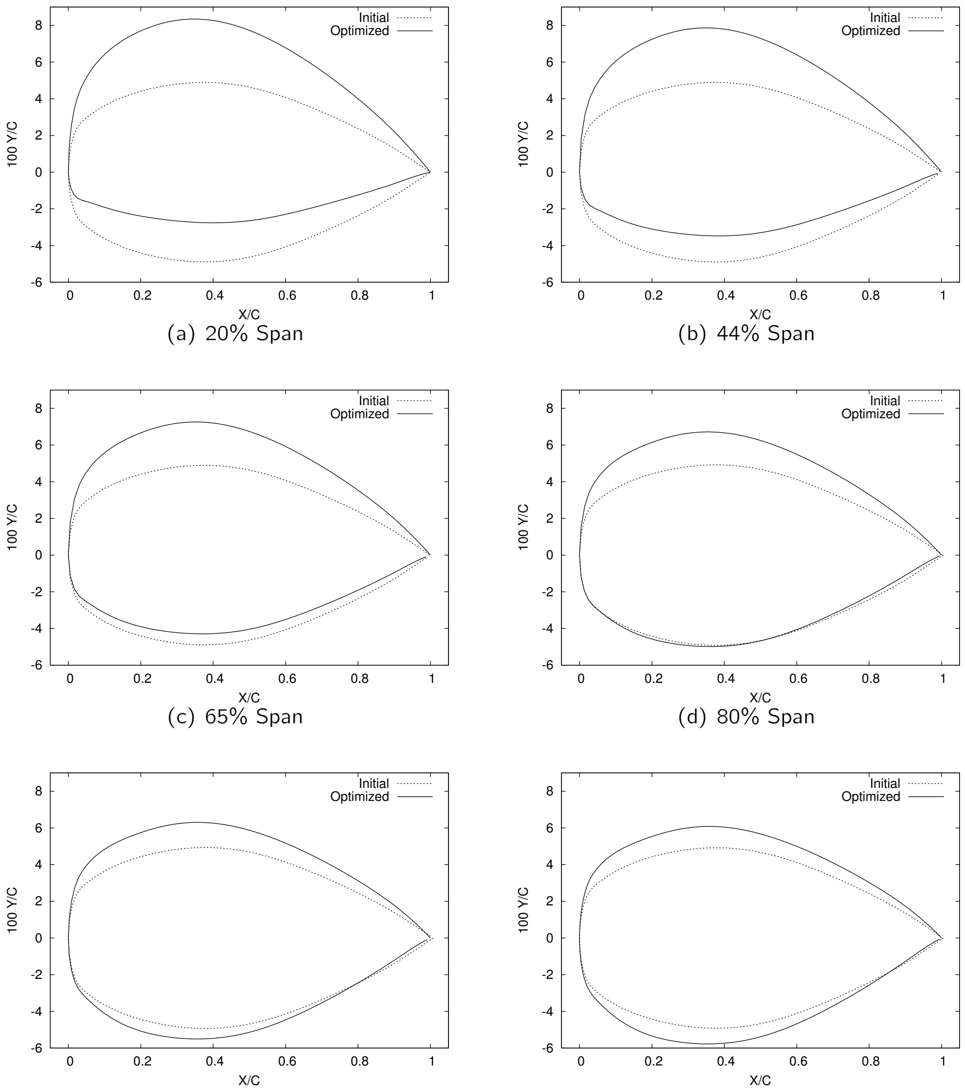
(c) 65% Span

(d) 80% Span

(e) 90% Span

(f) 95% Span

Figure 5.28: *Design to double the lift of the ONERA-M6 wing. Comparison of initial (ONERA-M6) and final pressure distributions at different sections along the span.*



(a) 20% Span

(b) 44% Span

(c) 65% Span

(d) 80% Span

(e) 90% Span

(f) 95% Span

Figure 5.29: *Design to double the lift of the ONERA-M6 wing. Comparison of the initial and final airfoil shapes at different sections along the span.*

distributions in Fig. 5.27 do not provide the best view of the improvement in pressure. Instead, Fig. 5.30 better shows the shock strength reduction. For instance, Fig. 5.30c shows a considerable reduction for the section at 65% of the span.

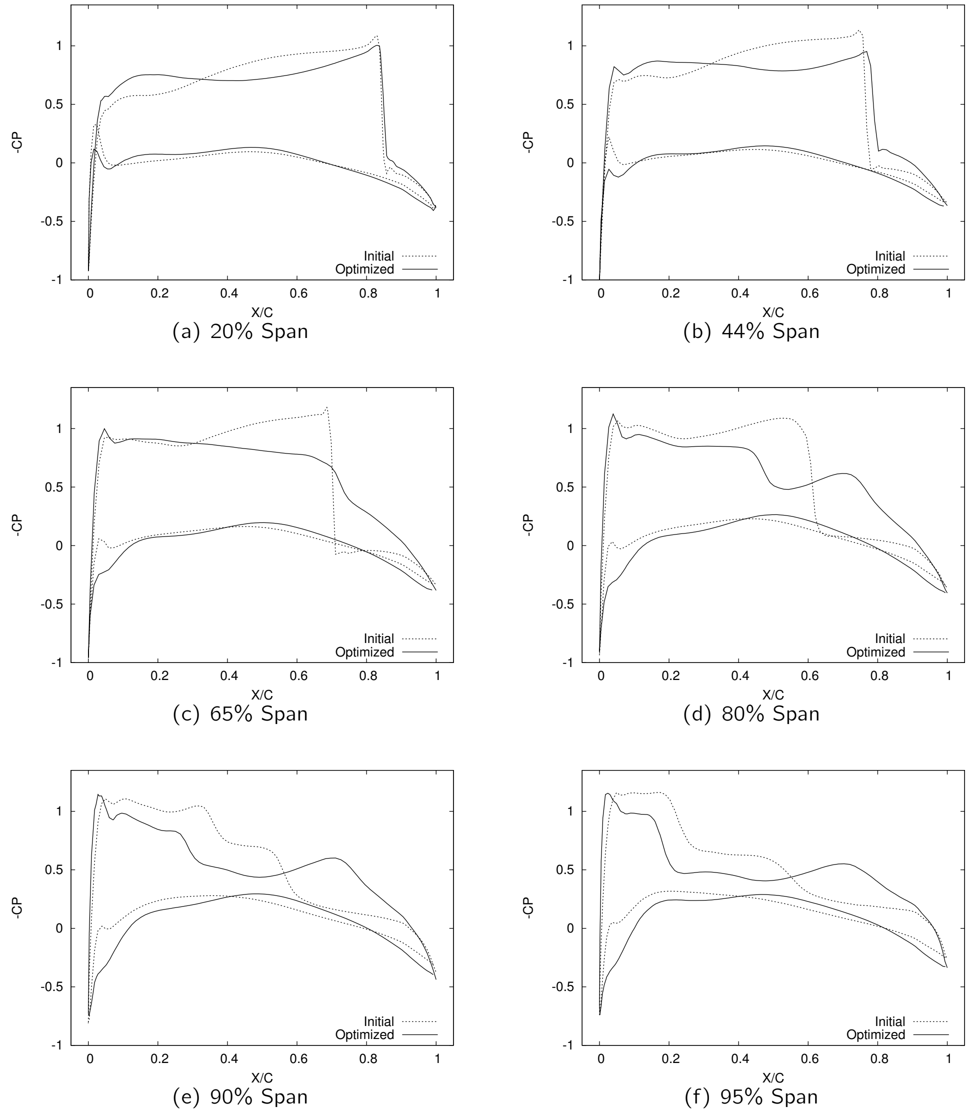
The airfoils obtained by this second optimization also appear to be smooth. Figure 5.31 shows the airfoils compared to that of the wing obtained by the first optimization (i.e., the wing with two times the lift of the ONERA wing). It seems that the nose of the airfoils , especially those depicted in Figs. 5.31 c-f, has changed considerably. The changes help in having higher pressure at the lower part of the airfoil nose. Moreover, it seems that the thickness of these airfoils has been redistributed towards the rear part, helping to weaken the shock strength.

Comparison of the wing shapes

Figure 5.32 shows a comparison of the three wing shapes: the ONERA-M6 wing and the two wings obtained by the present test case. In the left column the views are from the symmetry plane towards the wing tip. In the right column they are from the back of the wing (e.g., standing behind the trailing edge line) towards the leading edge. The two different types of view help to better visualize the deformations of the two wings compared to the ONERA-M6 wing.

It is interesting to see how the camber distribution of the wing has changed during the lift design, as shown by Fig. 5.32d. As can be seen, the wing surface is not linear and a slight curvature may be seen, especially on the upper side. The latter curvature increases in magnitude and changes sign when the drag is optimized, as shown by Fig. 5.32f. The redistribution of the thickness towards the rear part of the wing, when the drag is optimized, is visible by comparing Fig. 5.32e with Fig. 5.32c.

The present case shows that the 3D shape parameterization behaves smoothly also in the presence of large deformations. It is interesting to see that such large deformations have only been obtained with 30 parameters for each side. The number appears small if one realizes that it is of the same order of magnitude as that used to parameterize an airfoil.



(a) 20% Span

(b) 44% Span

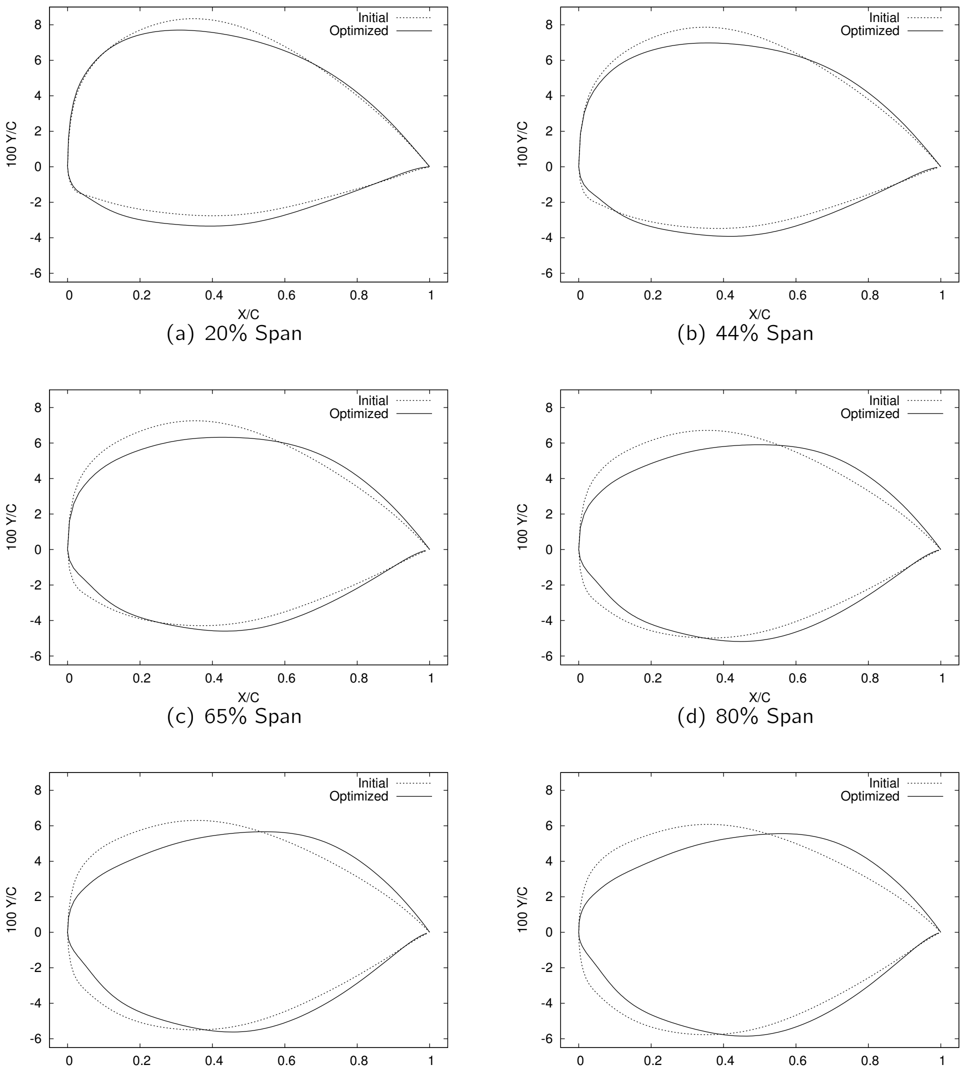
(c) 65% Span

(d) 80% Span

(e) 90% Span

(f) 95% Span

Figure 5.30: *Drag optimization of the wing obtained by doubling the lift of the ONERA-M6 wing. Comparison of initial and optimized pressure distributions at different sections along the span.*



(a) 20% Span

(b) 44% Span

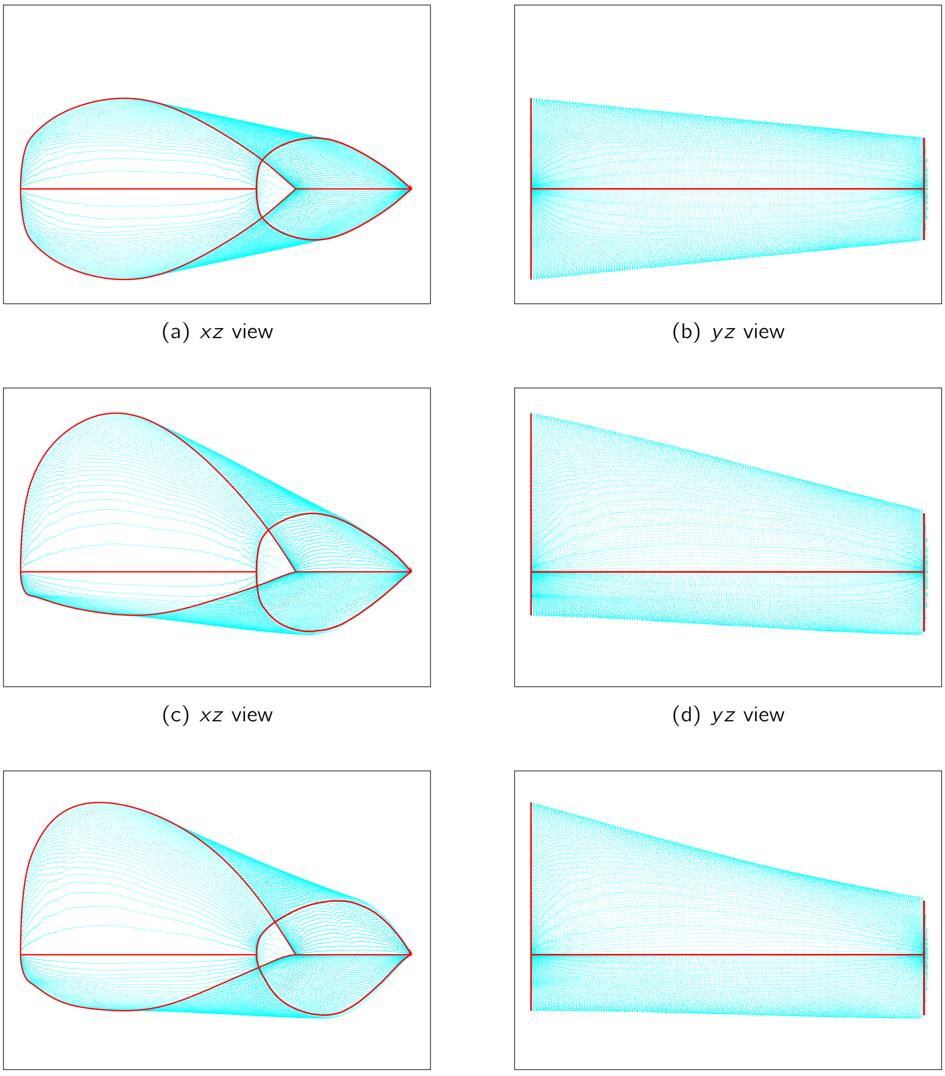
(c) 65% Span

(d) 80% Span

(e) 90% Span

(f) 95% Span

Figure 5.31: *Drag optimization of the wing obtained by doubling the lift of the ONERA-M6 wing. Comparison of the initial and optimized airfoil shapes at two different wing sections.*


(a) xz view

(b) yz view

(c) xz view

(d) yz view

(e) xz view

(f) yz view

Figure 5.32: *Views of: (a-b) the ONERA-M6 wing; (c-d) the with obtained by doubling the lift of the ONERA-M6 wing; (e-f) and by subsequently minimizing its drag.*

5.6 A future application: shock position control

Shock-free airfoils may be designed by means of shape optimization. Instead, in the case of wings, shocks are rarely removed completely. Of course, shape optimization may be used to reduce the strength of the shocks, sometimes eliminating them in some areas of the wing surface, as shown by the previous applications. Operating a wing configuration shock-free is unrealistic because operating conditions change during flight. According to the theorem of Morawetz [88], shock-free flows may only be attained at a single design point. Away from the latter point, shocks start to appear. In practice, the design is carried out with the aim of ensuring satisfactory performances over the complete flight envelope, making sure that the shocks, away from the design point, remain weak.

By means of traditional shape optimization the designer does not have a direct control on shocks. In fact, by minimizing the drag the position of the shock is not controlled. It is interesting for the designer to have a more direct control on the position of the shock for design purposes. For instance, considerations about the interaction of airflow and structure may make certain positions more suitable than others. Clearly, in order to optimize the position of the shock, the designer needs to compute the sensitivity of the shock displacement with respect to the shape parameters. Pironneau had an idea of how to compute such sensitivity [101]. Recently, at a conference in Erice, he shared his idea with the author, and suggested its implementation in the present framework. Below, the concept is explained and some examples are provided. Unfortunately, due to time constraints, the concept could not be further expanded into an application similar to those presented in this chapter.

5.6.1 Sensitivity of the shock position

Consider Eq. (3.52), which gives the expression for the sensitivity of the density across a shock, and which is repeated below for clarity:

$$\rho_\alpha = \rho_\alpha^- + [\rho_\alpha]\,\mathcal{H}(\eta) + [\rho]\,\delta_d(\eta)\,\eta_\alpha\,. \tag{5.11}$$

As already mentioned in Section 3.6.1, $\mathcal{H}$ and δ_d are the step and the Dirac functions, whereas η is the shock position. The above equation may be integrated in order to compute the shock sensitivity η_α. Integration along a stream line l yields

$$\int_l \rho_\alpha dl = \int_{l^- + l^+} \rho_\alpha^- + [\rho_\alpha]\,\mathcal{H}(\eta) dl + \int_{Shock} [\rho]\,\delta_d(\eta)\,\eta_\alpha dl\,. \tag{5.12}$$

where l is divided in three pieces: a point that contains the shock and l^- and l^+, upstream and downstream of the shock, respectively . Bringing the first right-hand side term to the left, and recalling the sampling property of the Dirac function[1], allows the above equation to be written as

$$\int_{Shock} \rho_\alpha dl = [\rho]\,\eta_\alpha\,, \tag{5.13}$$

[1] $\int f(x)\delta(x - x_0)dx = f(x_0)$

which shows how the shock sensitivity may be computed as the ratio of two quantities: the integrated density sensitivity across the shock, and the density jump across the shock.

In theory the density jump may be easily computed, although it is not always easy to unambiguously identify the shock, as the latter may be spread over several grid points. Moreover, over- and under-shoots may be present, which slightly alter the solution near the jump. Caution is needed for the integral of the density sensitivity. The latter integral is in fact the integral of a discontinuous quantity (the density sensitivity appears to be a Dirac function at the shock), which should be evaluated in a single point (the domain of integration is the shock). In fact, as already mentioned the shock is spread over more than one grid point. Consequently, also the Dirac function is spread. Therefore, at the discrete level, a domain of integration with more than one point exists. In the following numerical examples are provided, which show that the integral makes sense.

Assuming that the sensitivity of the shock displacement can be calculated, verification of its value is not straightforward. In fact, a comparison value must be available. For instance, computing the sensitivity by finite differences may be possible only for relatively large values of the finite-difference step. Relatively small values cannot be used as the shock does not displace continuously, but may only jump from one grid point to the other. However, large values reduce the accuracy of the sensitivity, as high order terms should also be included. Fortunately, numerical evidence shows that the shock displacement is not a highly non-linear function. In fact, as shown below, the ratio of the shock displacement to the finite-difference step appears to remain constant for large values of the step.

Nodes	$\int \rho_\alpha dl$	$[\rho]$	η_α	$\Delta\eta/\Delta\alpha$
2100	0.01608	0.36932	0.04323	0.03887
8000	0.01640	0.38306	0.04250	0.03870
23000	0.01561	0.39786	0.03896	0.03367
30000	0.01432	0.40299	0.03529	0.03209

Table 5.1: *RAE2822 at $M_\infty = 0.73$ and $\alpha = 2$. Integral, jump and shock displacement sensitivity for different mesh sizes.*

5.6.2 Numerical examples

The integral has been evaluated for different mesh sizes in the case of the RAE2822 airfoil at $M_\infty = 0.73$ and $\alpha = 2°$. Integration is only performed along the x-coordinate by means of the trapezoidal rule. Variations along the y-direction have been neglected because they are very small. The sensitivity of the shock position η_α has been computed with respect to the angle of attack α. Results are shown in Table 5.1. As can be seen, the integral yields close values for increasing mesh sizes. Figure 5.33 shows the Dirac functions of the density sensitivities, which have been used to evaluate the integral. It appears that when the mesh size increases, although the Dirac function seems to become more sharp, its area does not change much. The last two columns of Table 5.1 compare the sensitivity of the shock displacement η_α with that obtained by using finite differences, $\Delta\eta/\Delta\alpha$, with a large step, $\Delta\alpha = 0.5°$. As can be seen, although there is not a very accurate match, the values are very similar.

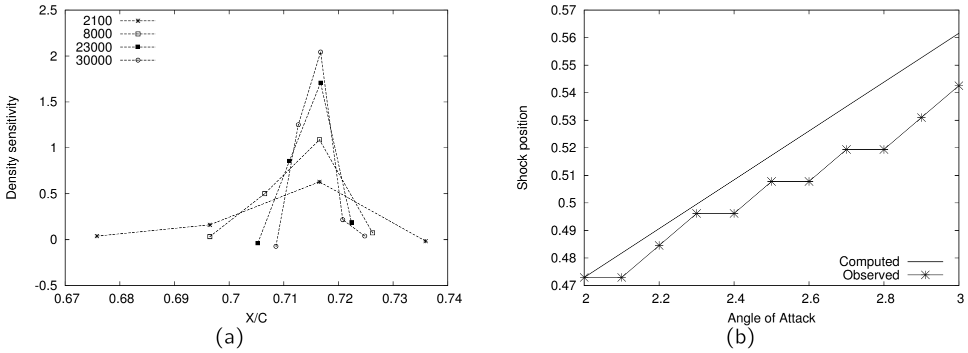


Figure 5.33: *(a) RAE2822 at* $M_\infty = 0.73$ *and* $\alpha = 2$. *Density sensitivity for different meshes. (b) NACA0012 at* $M_\infty = 0.75$ *and* $\alpha = 2$. *Shock displacement for different angles of attack.*

	η_α	$\Delta\eta/\Delta\alpha$
RAE2822, $\alpha = 3°$	0.1040	0.0861
NACA0012, $\alpha = 2°$	0.0765	0.08164
NACA64A410, $\alpha = 2°$	0.0343	0.0382

Table 5.2: *Shock displacement sensitivity for airfoils at* $M_\infty = 0.7$.

Smaller steps, for instance steps of the order 10^{-3}, cannot be used because the shock may not change its position. However, for large steps the accuracy of the derivative deteriorates. Nevertheless, it seems that the shock displaces almost linearly with the angle of attack. Figure 5.33 shows the shock displacement as a function of the angle of attack for the NACA0012 at $M_\infty = 0.75$, starting from $\alpha = 2°$. As can be seen, when $\alpha = 2.1, 2.4, 2.6$ and 2.8, the shock does not move. As already mentioned, this is caused by the fact that the shock does not move continuously but discretely on the mesh points. In general, the continuous line, which represents the displacement of the shock implied by the sensitivity η_α, agrees reasonably with the observed displacements. When $\alpha = 3$, which means for 1° variation in the angle of attack, the predicted displacement is only 4% larger than the observed displacement. Three other results, which have been performed at $M_\infty = 0.7$, are shown in Table 5.2. The RAE2822 airfoil shows a 20% difference between the sensitivities, much larger than the differences of Table 5.1, which have been obtained for a different free-stream condition. Differences are smaller for the other two cases, 6.3% and 10.2%, respectively.

The fact that the computed sensitivity may not be checked reliably poses some issues on the validation of the method. Future work should try to use the computed sensitivity in order to perform a design in which the shock must be placed at a specific location. As already discussed for the approximate adjoint, it is possible that although the sensitivity is not very accurate, it is still effective when used in shape optimization.

5.7 Concluding remarks

In this chapter the framework components presented in the thesis have been combined and put into practice. The results show that the framework is a very effective tool for the design of transonic airfoils. Shock-free designs are easily obtained starting from subsonic designs. Inverse design between two airfoils – both characterized by the presence of shocks – is performed accurately. Supersonic designs are obtained starting from subsonic ones. The framework also performs well in case of wings. Strong shocks on the wing surface are decreased considerably in strength. Designs that have twice the lift of the initial ones can be obtained.

Such effectiveness is the result of the combined action of the different components of the optimization framework. Nevertheless, a crucial component for the success of the optimization is the Chebyshev parameterization. This representation synthesizes the most important information of the design space topology into a handful of parameters. Without such complete representation of the design space the framework would not deliver results such as those presented in this chapter, no matter the effectiveness of the other components. The supersonic design is an extreme case in which the qualities of the Chebyshev parameterization are evident. In that case the shape is deformed considerably, a very sharp nose is generated, which, however, is still rounded and smooth. Another demanding case is the wing design in which the lift of the wing is doubled. In that case the 3D extension of the parameterization dealt with very large deformations, which completely changed the shape of the wing.

The approximations to the discrete adjoint described in Chapter 3 introduce an error in the computed sensitivity. The results presented in this chapter show clearly that when the limiters are neglected, and the numerical fluxes are approximated, the sensitivity is still as effective as in the exact case. When the accuracy is important, e.g., in the shock-free design or in the inverse design, the approximation works very well. However, when the reconstruction formulation is not differentiated, the computed sensitivity is not effective. In fact, when that sensitivity is used for shock-free design cases or for inverse design cases, the optimization stalls.

In the final part of this chapter a method to compute the sensitivity of the shock position, with respect to the shape parameters, has been presented. The method uses the linearized sensitivity, which must be integrated across the shock. The evaluation of the integral is not very obvious and, in practice, is open to interpretations. A possible interpretation has been presented, and it seems reasonable. Also, there is not a well defined way of establishing if the results are correct. In fact, the shock displaces slightly, sometimes only one or two grid points away from its original position. Hence, the finite difference derivative may be undefined. However, taking into account the above limitations, the results presented show how the computed sensitivity is in agreement with the observed displacement of the shock.

Chapter 6

Final remarks

The thesis presents an adjoint-based aerodynamic shape optimization framework for two- and three-dimensional problems in which the flow is governed by the Euler equations. The important aspects of the framework, i.e., the solution of the flow equations, the solution of the adjoint equations, the parameterization of the shape and the optimization strategy have been discussed in detail and results have been presented.

The flow solver, which is based on an unstructured median-dual formulation, appears to be very flexible and robust. It has been demonstrated on several problems, e.g., subsonic, transonic and supersonic steady flows. As the work in the appendix shows, it has also been extended to efficiently cope with the computation of unsteady flows around moving and deforming bodies.

Further improving the flow solver is possible with respect to several aspects. For instance, better representation of the physics can be achieved by also modeling diffusive phenomena and turbulence. The extension is not trivial but is worthwhile as it would provide the user with the total drag. From the point of view of the solution process, parallelization of the solver could contribute dramatically to reduce the computing time, giving the user the possibility to evaluate the flow around complex configurations in a reasonable time. These extensions would certainly add considerable value to the solver and would make it more appealing also to an industrial audience.

The adjoint solver is based on the discrete formulation of the adjoint method. This formulation requires the solution of a linear system of equations in which the matrix is the transposed residual Jacobian and the right-hand side is the sensitivity of the flow functional. In the present work the matrix-vector product that appears on the left-hand side has been derived in closed form, which involved differentiating non-linear functions present in the flow solver and assembling complex loops on the mesh. Exact and approximate derivations have been presented and optimization results have been used to establish the suitability of certain approximations. It appears that some approximations are as effective as the exact code and are considerably easier to implement. The solution strategy adopted for the adjoint equations is that of using the flow solver scheme, which means that the adjoint equations are treated as non-linear. Such a solution strategy is straightforward because the solver is already in place and appears to be very robust. In addition, a modification of the scheme has been presented, which allows the simultaneous solution of multiple adjoints. The capability is particularly interesting for constrained shape design, in which the objective function and also the constraints depend on flow functionals. Overall, the adjoint solver is very efficient. It features performance comparable to that of the flow solver in terms of memory and CPU usage.

The adjoint implementation can be extended in order to solve more complex design problems, for instance, shape optimization in turbulent flows, provided the flow solver is modified to deal with such flows. The extension requires the addition of the adjoint of the viscous terms and the adjoint of the turbulence model. Another interesting application would be the solution of the adjoint equations for the unsteady formulation presented in the appendix. Such extension would be very interesting for the optimization of shapes for which the design requirements are time-dependent, for instance in the case of shape optimization subject to aeroelastic constraints. Note that the implementation of the unsteady adjoint requires to store the flow field at each time step, which is needed in order to solve the adjoint equations backward in time.

The method to compute the sensitivity of the shock with respect to the shape parameters, for which preliminary results have been presented, still requires investigation. In fact, several issues have been identified, whose solution is not clear yet. However, the results look encouraging and it would be very interesting to see a design application involving the positioning of the shock.

For the parameterization of airfoil shapes, Chebyshev polynomials are employed. A least-squares procedure may be used to find the parameters for a given airfoil. As shown by the results, accurate parameterization is obtained using a small number of parameters only. An extension of the method to the parameterization of wing-like shapes has also been presented. Such an extension assumes that the parameters change along the span-wise direction according to another polynomial and that the displacements of the wing, rather than the wing itself, are parameterized. The method is flexible and results show that the three-dimensional displacements appear to be very smooth, also in the case of wings with non-linear planform, for which the surface does not have a simple curvature.

More test cases are needed to further test the method in three dimensions. For the parameterization along the span-wise direction, other types of functions may be tested to get a better picture. It would also be interesting to compare the shape-parameterization method with other more widely used methods, such as the NURBS parameterization. From the point of view of the applicability of the method, an interface with a CAD package should be implemented. In practice, CAD packages are used to create the shape. The optimization process adds displacements to the mesh created around the shape. The displacements need to be fed back into the CAD package. Thus, the package must have the parameterization implemented, with which it can generate the displacements internally according to a set of parameters. This way, the CAD model can be updated in order to take the optimized shape.

Another aspect that needs to be improved for the applicability of the method is the calculation of the sensitivity of the mesh-movement strategy with respect to the shape parameters. At the moment only the forward sensitivity is implemented, which means that the time required to compute the sensitivities is proportional to the number of shape parameters. Three-dimensional sensitivities are therefore very time-consuming to compute. A solution is to solve the adjoint equations for the mesh displacements. The implementation should not pose any problem because the mesh-displacement operator is symmetric and thus the transposition is not an issue.

As a whole, the design method appears to be very effective and efficient. Optimization results have been shown for two-and three-dimensional problems: shock-free airfoil designs, inverse airfoil design in transonic flow, airfoil drag reduction in supersonic flow, and finally wing designs in transonic flow. The proper representation of the design space provided by the Chebyshev parameterization is a key component for successful applications. Also, the use of constrained optimization algorithms made it possible to satisfy accurately the complex design requirements enforced in all the optimization cases. The results show that even a very simple Sequential Linear Programming algorithm can successfully drive the optimization process.

Fine-tuning of the design method is still required for three-dimensional cases, which are very time consuming and to which only limited time could be dedicated in the present work. In three-dimensions, the complexity of the design increases considerably and also the definition of the design problem in terms of constraints becomes a challenging task. Provided some of the aforementioned improvements being implemented, such as the parallelization and the adjoint of the mesh, computational times would reduce dramatically and therefore the fine-tuning could be performed quickly by trials and errors. Like with everything in life, practice makes perfect.

Appendix A

Unsteady Flows

A.1 Unsteady flow solver for deforming meshes

The unsteady Euler equations are solved by a modified version of the numerical method presented in Chapter 2. For computations on fixed meshes modifications are made to the solution process, which must be time accurate. For computations on moving meshes, also the evaluation of the fluxes must be modified to reflect the presence of moving and deforming control volumes, through which the fluxes are calculated. In the following the equations and the solution scheme are discussed for the deforming mesh case. The standard case is obtained by dropping the speed of the mesh out of the equations.

A.1.1 Discretization of the unsteady Euler equations on moving meshes

The unsteady Euler equations on a deforming control volume may be written as

$$\frac{\partial}{\partial t}\int_{V(t)} \mathbf{u}\, d\Omega + \oint_{S(t)} \mathbf{F}(\mathbf{u}, \dot{\mathbf{x}}) \cdot \mathbf{n}\, d\Gamma = \mathbf{0}\,, \tag{A.1}$$

where $\mathbf{u}$ is the vector of conservative variables, $\dot{\mathbf{x}}$ is the speed of the boundary domain and $\mathbf{F}$ is the projection of the flux vector along the outward unit normal on the domain boundary. The conservative variables are defined in the same way as in Chapter 2. The flux vector may be obtained by replacing $\mathbf{w}^T$ in the steady flux vector of Eq. (2.3) by $(\mathbf{w}-\dot{\mathbf{x}})^T$ (only $\mathbf{w}^T$ and not $\mathbf{w}$). Rearrangement of the flux vector gives

$$\mathbf{F}(\mathbf{u},\dot{\mathbf{x}}) = \mathbf{F}(\mathbf{u}) - \mathbf{u}\dot{\mathbf{x}}^T \,. \tag{A.2}$$

Equation (A.1) is discretized using the unstructured finite-volume formulation on the median-dual mesh. For the control volume i, the discretization may be written as

$$\frac{\partial(V_i\mathbf{u}_i)}{\partial t} + \sum_{j=1,N_i} \mathbf{\Phi}_{ij}(\hat{\mathbf{u}}_i,\hat{\mathbf{u}}_j,\dot{\mathbf{x}}_{ij},\mathbf{n}_{ij}) + \mathbf{\Phi}_i^{bc}(\mathbf{u}_i,\dot{\mathbf{x}}_i,\mathbf{n}_{Bi}) = \mathbf{0}\,. \tag{A.3}$$

The numerical flux $\mathbf{\Phi}_{ij}$ is evaluated using the Roe flux, which in the case of mesh deformation may be written as

$$\mathbf{\Phi}_{ij} = \frac{\mathbf{F}(\mathbf{u}_i,\dot{\mathbf{x}}_i)+\mathbf{F}(\mathbf{u}_j,\dot{\mathbf{x}}_j)}{2}\cdot\mathbf{n}_{ij} - \frac{1}{2}|\mathbf{A}(\tilde{\mathbf{u}}_{ij},\dot{\mathbf{x}}_{ij},\mathbf{n}_{ij})|(\mathbf{u}_j-\mathbf{u}_i)\,. \tag{A.4}$$

The differences with the case with no deformations are minor as the eigenvectors $\mathbf{P}$ of the Roe matrix $\mathbf{A}$ are the same for both cases. In fact, it is possible to write the matrix as

$$\mathbf{A}(\mathbf{u},\dot{\mathbf{x}},\mathbf{n}) = \frac{\partial}{\partial\mathbf{u}}\left(\mathbf{F}(\mathbf{u},\dot{\mathbf{x}})\cdot\mathbf{n}\right) = \mathbf{A}(\mathbf{u},\mathbf{n}) - (\dot{\mathbf{x}}\cdot\mathbf{n})\mathbf{I}\,. \tag{A.5}$$

The above equation shows that the only difference with the original Roe matrix is on the diagonal. Therefore only the eigenvalues change. In practice, in terms of implementation, the projection of the mesh speed along the normal must be subtracted from each eigenvalue and the same eigenvectors as in the case with no deformations are used.

A.1.2 Second-order accurate GCL compliant scheme

Equation (A.3) may be written in compact form as

$$\frac{d(\mathbf{V}_{cv}\mathbf{U})}{dt} + \mathbf{R}(\mathbf{U}) = \mathbf{0}, \tag{A.6}$$

where $\mathbf{U}$ is the vector of conservative variables, $\mathbf{V}_{cv}$ is the vector of control volumes and $\mathbf{R}$ is the residuals vector. The time discretization of the above equation must be at least second-order accurate for practical use. First order accuracy would require too many time steps for the solution to converge in time. Moreover, the scheme must be GCL compliant, which means that the Discrete Geometric Conservation law [121] must be satisfied.

The GCL states that the above equation must be satisfied in a uniformly constant flow, for instance the free-stream flow, when the mesh is moving and deforming. In fact, the free-stream variables are solution of the equation. In practice, if a flow-field is initialized to the free-stream solution and an arbitrary mesh motion is prescribed, the change in residual must be exactly compensated by the change in the control volume derivative. A free parameter for the achievement of a GCL compliant scheme is the speed of the mesh, which may be computed appropriately. It can be shown that in 2D the scheme is GCL compliant if the mesh speed is computed by the same formula used to discretize the time derivative [99]. In 3D the situation is more complicated and a lengthy derivation is necessary to show the results. In the following an existing result has been used [65], for which the satisfaction of the GCL is achieved using an averaging technique for the mesh coordinates and for their speeds. Using this result the second-order GCL compliant scheme for Eq. (A.6) may be written as

$$\frac{\beta_1 \mathbf{V}_{cv}^{t+\Delta t}\mathbf{U}^{t+\Delta t} + \beta_2 \mathbf{V}_{cv}^{t}\mathbf{U}^{t} + \beta_3 \mathbf{V}_{cv}^{t-\Delta t}\mathbf{U}^{t-\Delta t}}{\Delta t} = -\mathbf{R}(\mathbf{U}^{t+\Delta t}, \overline{\mathbf{X}}, \dot{\overline{\mathbf{X}}}) \,, \tag{A.7}$$

where Δt is the physical time step and the coefficients β_1, β_2 and β_3 are taken equal to 1.5, -2 and 0.5, respectively. The mesh coordinates $\overline{\mathbf{X}}$ and their speeds $\dot{\overline{\mathbf{X}}}$ that appear in the above formula are not taken at time $t + \Delta t$. They are obtained by averaging at specific time instants between $t - \Delta t$ and $t + \Delta t$, and are defined as

$$\dot{\overline{\mathbf{X}}} = \sum_{k=1}^{k_f} \alpha_k \dot{\mathbf{X}}^k \quad \text{and} \quad \overline{\mathbf{X}} = \sum_{k=1}^{k_f} \alpha_k \mathbf{X}^k \,, \tag{A.8}$$

where α_k is the weighting coefficient for the time instant k. In order to satisfy the GCL, for the problem considered here, i.e., the three-point scheme in 3D, there must be $k_f = 4$ of such points. The coefficients α_k are such that $\sum_{k=1}^{k_f} \alpha_k = 1$. Their values can be found in [65]. In the following, in order to simplify the notation, the residual of Eq. (A.7) will be denoted by $\overline{\mathbf{R}}$ and the averaged quantities will be omitted.

A.1.3 Implicit solution scheme

The conservative variables $\mathbf{U}^{t+\Delta t}$ in Eq. (A.7) have to be computed given the values $\mathbf{U}^t$ and $\mathbf{U}^{t-\Delta t}$ at the previous time steps. For this purpose, after naming $\mathbf{W} = \mathbf{U}^{t+\Delta t}$ and after shifting the t and $t - \Delta t$ terms of Eq. (A.7) to the right-hand side, the following "unsteady" residual can be defined:

$$\mathbf{R}^*(\mathbf{W}) = -\overline{\mathbf{R}}(\mathbf{W}) - \frac{\beta_1}{\Delta t}\mathbf{V}^{t+\Delta t}\mathbf{W} + \mathbf{S}(\mathbf{V}^t\mathbf{U}^t, \mathbf{V}^{t-\Delta t}\mathbf{U}^{t-\Delta t}) \,. \tag{A.9}$$

$\mathbf{S}$ is a source term, which is defined as

$$\mathbf{S} = -\frac{\beta_2}{\Delta t}\mathbf{V}^t\mathbf{U}^t - \frac{\beta_3}{\Delta t}\mathbf{V}^{t-\Delta t}\mathbf{U}^{t-\Delta t}. \tag{A.10}$$

The unsteady residual must be driven to zero within each time step. Newton iterations can be used, which may be written as

$$\mathbf{R}^*(\mathbf{W}^{n+1}) = \mathbf{0} \,. \tag{A.11}$$

Note that also a pseudo-time contribution could be added, similarly to the steady problem solved in Chapter 2. However, whereas the steady solver starts from a uniform field, which is very far from the steady solution, the unsteady solver starts from the solution obtained at the previous time step, which is relatively closer. Thus, the pseudo-time step is not really necessary for unsteady computations, at least according to numerical experiments performed in the present work.

Similarly to the steady problem, the above problem may also be solved by a defect correction approach as

$$\widetilde{\mathbf{R}}^*(\mathbf{W}^{n+1}) + \mathbf{R}^*(\mathbf{W}^n) - \widetilde{\mathbf{R}}^*(\mathbf{W}^n) = \mathbf{0}\,, \tag{A.12}$$

where $\widetilde{\mathbf{R}}^*$ is a low-order discretization. The low-order residual $\widetilde{\mathbf{R}}^*(\mathbf{W}^{n+1})$ in the above equation can be linearized by taking into account the definition of the unsteady residual in Eq. (A.9). Further rearrangement of the terms yields

$$\left[\frac{\beta_1}{\Delta t}\mathbf{V}^{t+\Delta t} + \frac{\partial \widetilde{\mathbf{R}}}{\partial \mathbf{W}}\right]^n \Delta\mathbf{W} = -\mathbf{R}^*(\mathbf{W}^n)\,. \tag{A.13}$$

In practice, the above problem can be solved by the same method used for the solution of the steady-state problem. The only difference is that the physical time step is used instead of the local pseudo-time step and that the unsteady residual is the input of the Newton procedure and not the steady one. Also the procedure used to solve the linear system in Eq. (A.13) is the same. From the point of view of implementation it means that the software already implemented for the steady case may be used with very minor modifications.

A.1.4 Oscillating airfoils and wings

Results from the computations of pitching airfoils are shown in Fig. A.1. The pitching motion is sinusoidal, i.e., it is governed by the law

$$\alpha(t) = \alpha_0 + \alpha_M sin(2\pi f t)\,, \tag{A.14}$$

where α_0 is the initial angle of attack, α_M is the amplitude of the pitching oscillation and f is its frequency. The c_l history in Fig. A.1a is from a NACA0012 airfoil, for which $\alpha_0 = 0.016°$, $\alpha_M = 2.51°$ and $f = 6.6551$Hz. The result compares well with that obtained by other authors [130]. Note that the history shows one cycle for which the transient phase may be considered to be terminated. In fact, the first cycles usually have different shapes, which eventually converge to a unique shape. The c_l history in Fig. A.1b is from a NACA64A010 airfoil, for which $\alpha_0 = -0.21°$, $\alpha_M = 1.01°$ and $f = 17.412$Hz. In the same figure, experimental results obtained in the wind tunnel are shown [30]. As can be seen, the numerical results compare well with the experimental results.

Figure A.2 shows some results for the computation of unsteady flow around the ONERA-M6 wing. The wing is perturbed by an impulse in pitch (the type of impulse is described in the following section). Figure A.2a shows the number of non-linear iterations (continuous line) that are required to solve Eq. (A.13) at each time step. The number of iterations increase, and reach a peak in correspondence with the pitch impulse. Also,

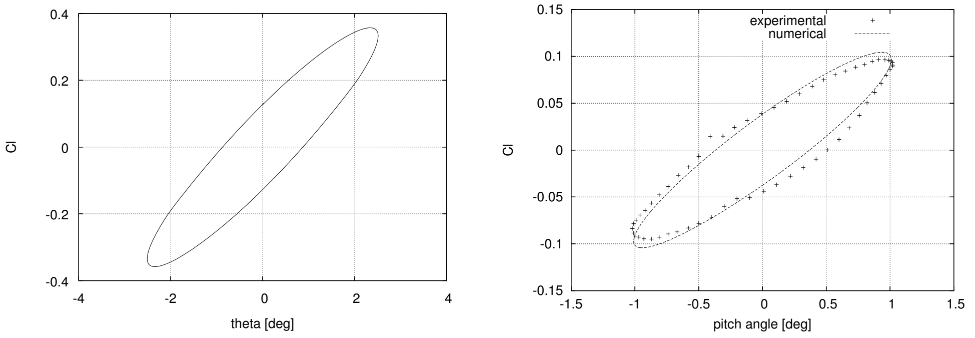


Figure A.1: *c_l history for oscillating airfoils: (a) NACA0012; and (b) NACA64A410, with experimental data.*

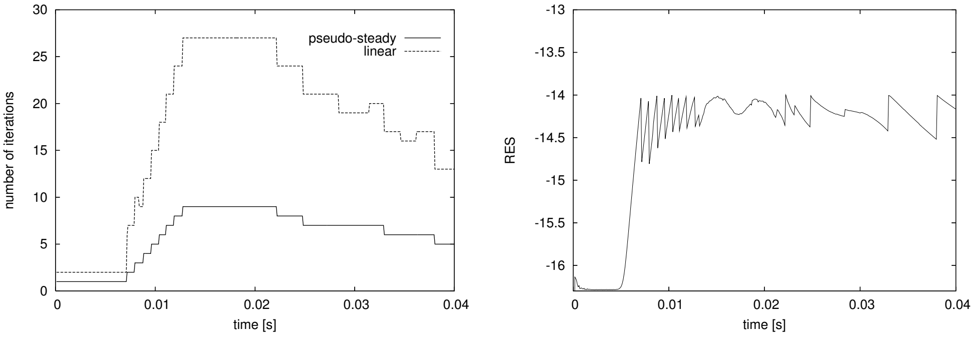
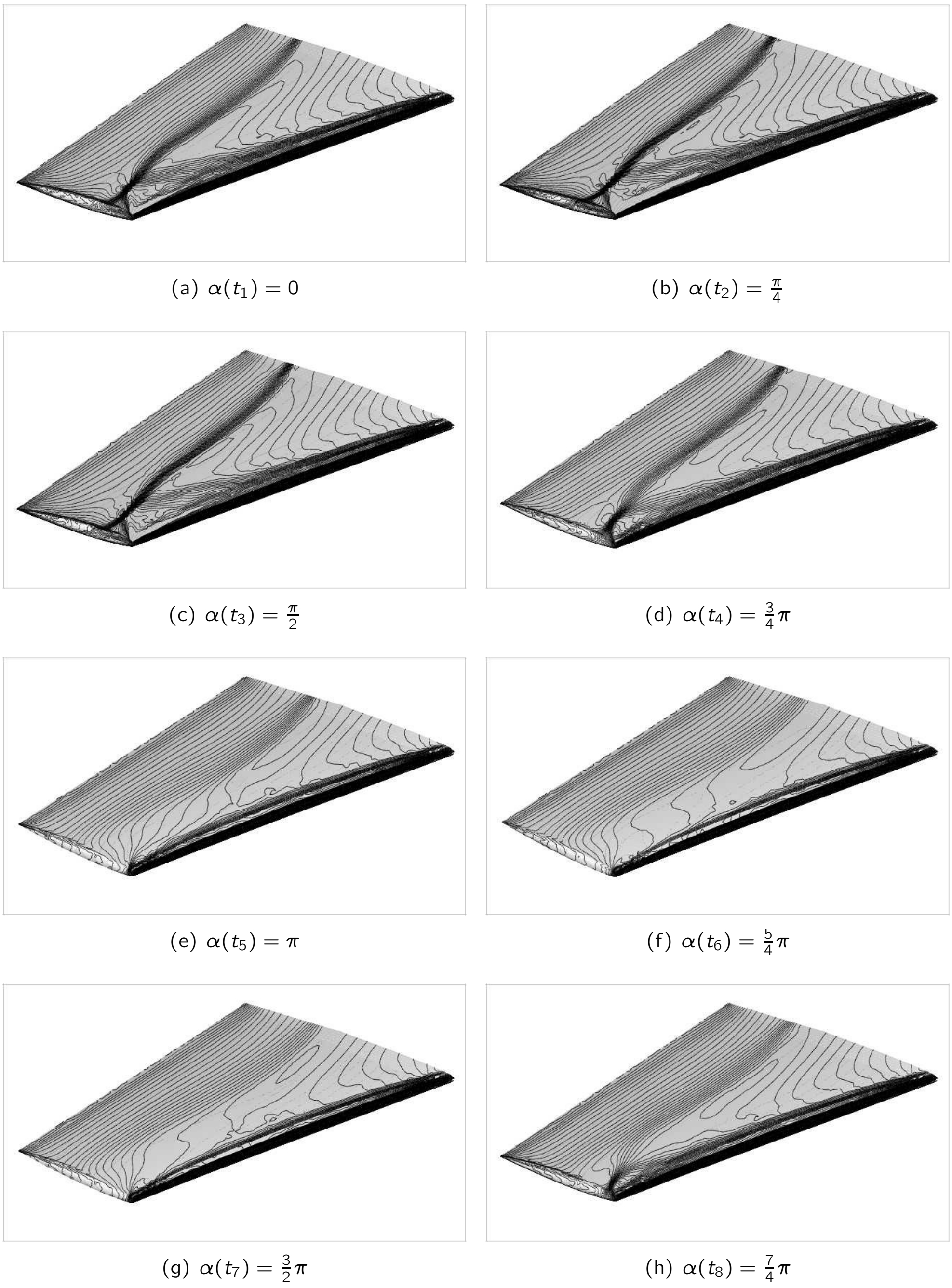


Figure A.2: *The solution process of the impulse response of the ONERA-M6 wing: (a) the number of pseudo-steady and linear iterations and (b) the residual development*

it shows the total number of linear iterations, with the dashed line, which are required at each time step. Figure A.2b shows the residual, which has a value around 10^{-16} at the beginning of the unsteady solution process (it has the value that results from the solution of the steady equations). As can be seen, the computation is very accurate since the residual never becomes larger than 10^{-14} during the unsteady iterations.

Figure A.3 shows the ONERA-M6 wing undergoing sinusoidal pitch oscillations of the type described by Eq. (A.14). The wing is rotating rigidly, with no deformations. The pressure contours are shown for a complete period and are equally spaced in the pitch angle. After the last one, the wing returns in the position with zero pitch, which corresponds to the first contour plot. The unsteady nature of the flow is clearly visible as the strong shock on the wing weakens, it has almost disappeared in the sixth and seventh contour plot, and starts to strengthen again in the last one.



(a) $\alpha(t_1) = 0$

(b) $\alpha(t_2) = \frac{\pi}{4}$

(c) $\alpha(t_3) = \frac{\pi}{2}$

(d) $\alpha(t_4) = \frac{3}{4}\pi$

(e) $\alpha(t_5) = \pi$

(f) $\alpha(t_6) = \frac{5}{4}\pi$

(g) $\alpha(t_7) = \frac{3}{2}\pi$

(h) $\alpha(t_8) = \frac{7}{4}\pi$

Figure A.3: *Pressure contours on the ONERA-M6 wing in sinusoidal pitching motion at* $M_\infty = 0.8$ *and* $\alpha = 3.6°$.

A.2 Application to aeroelasticity

Aeroelasticity investigates the interaction between the airflow and the structure. For aircraft, which are lightweight structures, in certain circumstances the interaction may prove to be devastating for the integrity of the structure. The structure may enter into oscillations, the amplitude of which is more than it can sustain. The phenomena is commonly known as flutter. An exhaustive description of the phenomena can be found in [31]. In the design phase, aeroelastic analysis is usually employed to determine the boundary within which the system is stable.

The equations of the coupled system are as follows. Assume that a finite-element method is used to discretize the structure, with M degrees of freedom, which are contained in the vector $\mathbf{x}$. The mass matrix $\mathbf{M}$ and the stiffness matrix $\mathbf{K}$ are available from the method. If $\mathbf{f}(t)$ is a vector of aerodynamic forces that act on the structure, the equations may be written, without damping term, as

$$\mathbf{M}\ddot{\mathbf{x}} + \mathbf{K}\mathbf{x} = \mathbf{f}(t)\,. \tag{A.15}$$

Usually a structure may have a huge number of degrees of freedom, such that the solution of the above equations, which are coupled with the flow equations through $\mathbf{f}(t)$, becomes very expensive. With an unsteady flow solver like the one developed in this work, a direct coupling could be realized by simultaneously time-marching the structural equations and the flow equations, the so-called monolithic approach. A loose coupling would solve the structure first, pass the deformation to the aerodynamic mesh and next solve the flow equations, then pass the pressure/forces to the structure and restart, until the time steps are completed, the so-called partitioned approach. The latter approach may be easier to apply as the two solvers may already be available and have to exchange only deformations and pressures at the boundary. Instead, the monolithic approach may be extremely complicated as the two equations must be solved by a single method, which may have to be developed from scratch.

If the structure is assumed to behave linearly, the equations simplify dramatically because modal analysis can be used. The eigenvectors $\mathbf{Z} = [\mathbf{z}_1, ..., \mathbf{z}_N]$ can be found such that

$$\mathbf{Z}^T\mathbf{M}\mathbf{Z} = \mathbf{I}\,, \qquad \text{and} \qquad \mathbf{Z}^T\mathbf{K}\mathbf{Z} = \boldsymbol{\Omega}\,. \tag{A.16}$$

The degrees of freedom may be expressed as a linear expansion

$$\mathbf{x}(t) = \mathbf{Z}\bar{\xi}(t) = \sum_{i=1}^{N} \mathbf{z}_i \xi_i(t)\,, \tag{A.17}$$

where $\bar{\xi} = [\xi_1, ..., \xi_N]^T$ is the vector that contains the modal amplitudes ξ_i, which are often referred to as generalized coordinates. If the latter expression for the degrees of freedom is substituted in Eq. (A.15), and the equation is then multiplied on the left by $\mathbf{Z}^T$, the following decoupled equations are obtained:

$$\ddot{\xi}_i + \omega_i \xi_i = q_i(t)\,, \qquad (i = 1, N)\,, \tag{A.18}$$

where q_i is the ith component of the vector $\mathbf{q} = \mathbf{Z}^T\mathbf{f}$ and is referred to as generalized force.

The advantage of solving the latter problem instead of the one given in Eq. (A.15) is evident as the number of modes is usually orders of magnitudes smaller than the number of degrees of freedom, i.e., $N \ll M$. Moreover, the latter problem is given as a set of decoupled equations, which may be easily solved once the generalized forces are available. For instance, realizing a direct coupling by solving Eq. (A.18), even a strong one, is much simpler than in the case of Eq. (A.15). In practice, given the generalized forces, the generalized coordinates may be computed algebraically, and the degrees of freedom can then be computed using Eq. (A.17). Note that all the operations that are necessary to compute the modes $\mathbf{Z}$ and the eigenvalues $\mathbf{\Omega}$ can be performed at once, prior to the aeroelastic analysis.

Nevertheless, also the coupled solution of Eq. (A.18) may still be too complex for design purposes. Moreover, it may be time consuming as different conditions on the flight envelope need to be considered. A less expensive and widely used approach is to evaluate the generalized forces off-line, with a sampling technique, and then to investigate the stability of Eq. (A.18) in the frequency domain for different design conditions.

A.2.1 Impulse response and linear behavior

The response $u(t)$ of a linear system to a perturbation $g(t)$ may be computed by convolution if its transfer function $f(t)$ is known. The convolution integral is defined as

$$u(t) = \int_0^t f(\tau)g(t-\tau)d\tau\,. \tag{A.19}$$

The above relation may be conveniently written in the frequency domain, where it appears to be much simpler:

$$\tilde{u}(\omega) = \tilde{f}(\omega)\tilde{g}(\omega)\,. \tag{A.20}$$

Initially the transfer function is not known. The impulse function, i.e., the Dirac function $\delta(t)$ may be used for perturbing the system in order to evaluate the transfer function. The Dirac function has the interesting property of exciting all the frequencies equally. Its transform is $\tilde{\delta}(\omega) = 1, \forall\omega$. Therefore, perturbing the system with the impulse, gives the response $\tilde{u}(\omega) = \tilde{f}(\omega)$, which is the transfer function.

An example of the impulse response is the following. Consider a pitching and plunging airfoil. If the airfoil can only move rigidly, only two degrees of freedom exist: the pitching and the plunging motion. The lift and pitching moment responses may be computed with respect to impulses in pitch and in plunge independently. One can simply give an impulse and solve the unsteady equations until the response becomes flat. The unsteady lift and pitch, which have been recorded during the computation, are the transfer functions. They may be used in Eq. (A.18) and the stability of the system may be studied in the frequency domain.

Assuming linear behavior

Theoretically, if the flow equations are linear, the impulse response may be used to sample the generalized forces for whatever perturbation amplitude. In fact, the response of a linear system to a perturbation is always scalable by the amplitude of the perturbation, at all frequencies. For instance, in the case of subsonic flow modeled by the potential flow equations, the response is linear. However, in the case of transonic flow modeled by the Euler equations, the response may not be linear because of the presence of the shock.

The non-linear behavior of transonic flow has motivated several researchers to investigate methods to identify the non-linear operator in the time domain by a series of convolution integrals, the number of which depends on the degree of non-linearity. The method may be expensive, depending on the number of convolutions [102].

In the present work a different route has been taken. The author has participated in research in which it was assumed that the transonic flow around airfoils and wings could still be modeled linearly for the purpose of aeroelastic analysis of most design cases. Computations have been made that show how several airfoils behave linearly and results appeared in the literature [79]. Unfortunately, at the time of the investigation, the unsteady solver implemented in the present work was not yet available and the scope of the investigation was limited to 2D airfoils. In the present work, with the unsteady solver available, calculations have been performed also for 3D wings. Numerical results given below show that the assumption of linear behavior is reasonable. Before showing the results, the type of impulse used is briefly described.

The enlarged Gaussian impulse

Performing the perturbation of a flow solver with a digital impulse is not straightforward. For instance, consider again a perturbation in pitch, which starts at time t_{imp}. At that time the body must be instantaneously rotated of an angle equal to the amplitude of the impulse. The mesh has to be deformed using a mesh deformation algorithm. Since the equations are solved iteratively, a sudden change, rather than a smooth continuous change, may require several iterations in order to converge. Also the flow solver may be put under stress, as the solution may be very far from that of the previous time step. If the solvers are not very robust, the consequence may be that the solution process breaks down.

An alternative is to use a smoothed impulse. Lisandrin proposed the use of a Gaussian type of impulse, which has smooth derivatives. The time law for the Gaussian impulse reads

$$f(t) = \delta_A \, exp\left[\frac{T_{imp} - 2t}{2K\,T_{imp}}\right]^2 , \tag{A.21}$$

where δ_A is the amplitude of the impulse, T_{imp} is the time for which the impulse is non-zero and K is a smoothness parameter. The parameters are chosen to be $K = 0.08$ and $T_{imp} = 0.027s$ in order to have the bandwidth of the signal equal to a reduced frequency of $k_r = 1$. In fact, it is usually in the interval delimited by the latter frequency that aeroelastic phenomena of aeronautical interest make their appearance.

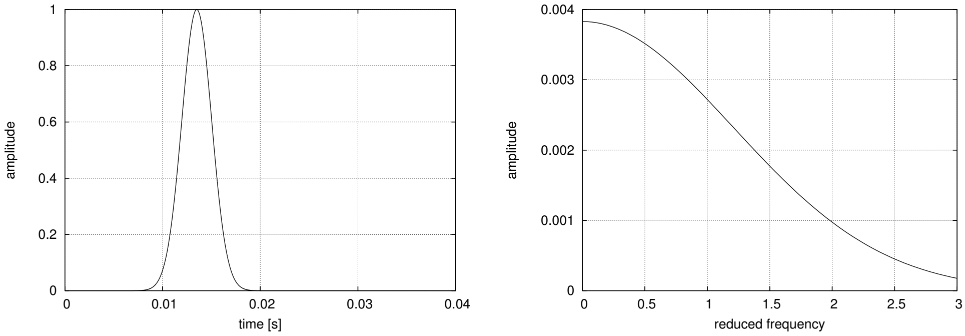


Figure A.4: *The Gaussian impulse: (a) in the time domain and (b) in the frequency domain.*

Figure A.4 shows the impulse function in time and frequency domains for $\delta_A = 1$. In the frequency domain it can be observed that the amplitude of the impulse has decreased to 70% of its initial value at $k_r = 1$.

A.2.2 Impulse response of wings

ONERA-M6 wing

The impulse response of the ONERA-M6 wing has been investigated. A structural model of the wing was not available. Thus, in order to study the response for a condition that could reflect a realistic situation, two deformation shapes have been assumed: bending and twist. The latter shapes are typically obtained for the first two fundamental modes of vibrations. Here the bending deformation is generated using a quadratic function of the wing span, which gives no deformation at the root. The twist deformation is generated applying a twist to the wing sections, around the quarter of chord line, the amount of which grows linearly with the wing span.

Three different amplitudes are considered for the impulse: 0.025, 0.5 and 1 for the twist and 0.05, 1 and 2 for the bending. The amplitudes of the impulses are chosen in a way that the maximum tip deformation due to the bending has the same value as the maximum tip deformation due to the twist. As there are only two deformations, the responses in terms of the lift and pitching moment coefficient are considered. The generalized forces are not computed.

Figure A.5a shows the scaled responses. The lift and the pitch responses to bending are shown on the first row. The second row shows the responses to the twist. In each figure the three responses have been scaled by the amplitude of the corresponding perturbation. As can be seen, the individual responses are hardly distinguishable.

The same responses have been transformed into the frequency domain and are shown in Fig. A.6. As the impulse is not the theoretical one (Dirac), the transform of the responses have been divided by the transform of the perturbation, frequency by frequency, so that the transfer function is obtained. The real part represents the magnitude of the signal whereas the imaginary part represents the phase of the signal. Very small differ-

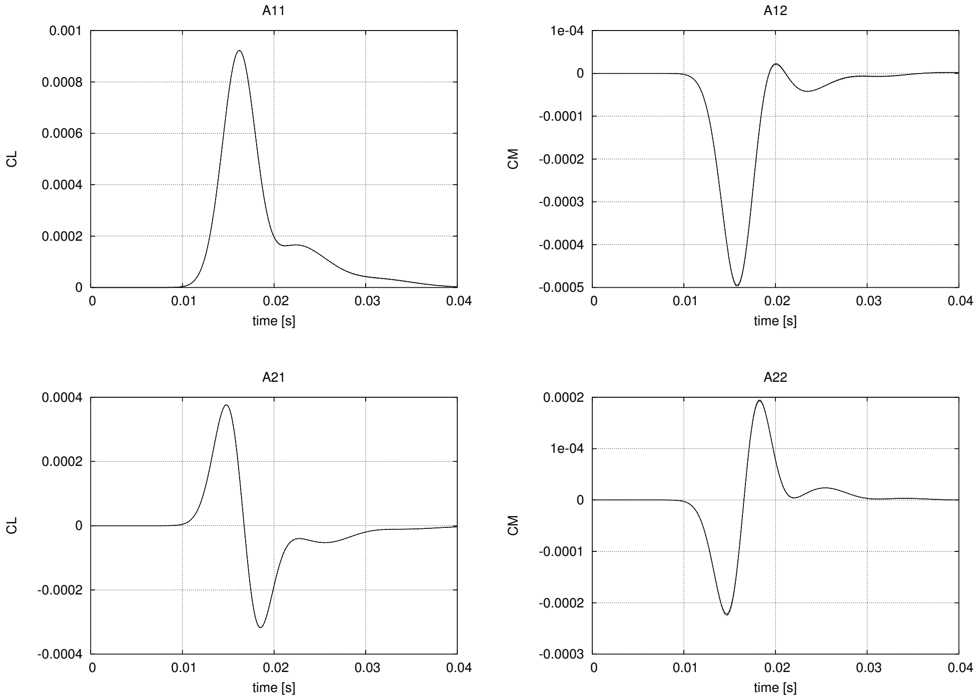


Figure A.5: *Time history of the C_L and C_M coefficients for three different perturbations around an angle of attack of 3.06 degrees.*

ences are present in the real part of the lift response to the twist and in the imaginary part of the pitching moment response to the twist. The system may be considered to behave linearly. Flutter computations in 2D [79], with the same amount of differences in the transfer function, have given very similar results.

AGARD 445.6 wing

The first AGARD standard aeroelastic configuration for dynamic response, which is known as AGARD 445.6 wing, has been tested in the 16-foot Transonic Dynamics Tunnel (TDT) at the NASA Langley Research Center [33]. Geometric data and results of the testing are given in the AGARD report 765. The wing has a root of 0.556m, a semi-span of 0.762m, a quarter-chord sweep of 45 degrees, a panel aspect ratio of 1.65, a taper ratio of 0.66 and a NACA 65004 airfoil section. The modes of vibration and the eigen-frequencies have been measured during the testing, and are also available in AGARD report 765.

Four of the available modes have been used, they are depicted in Fig. A.7. The modes are given on a rectangular grid. In order to link them to the aerodynamic mesh, a linear interpolation procedure is used. The Generalized Aerodynamic Forces (GAF) matrix, which gives the response i with respect to a perturbation j, is computed by

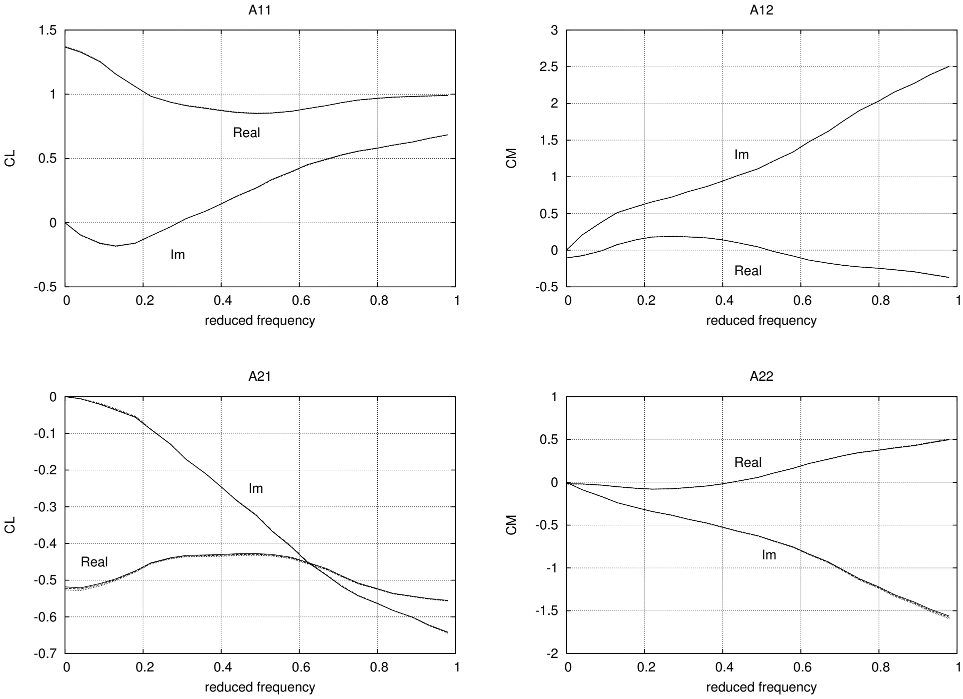
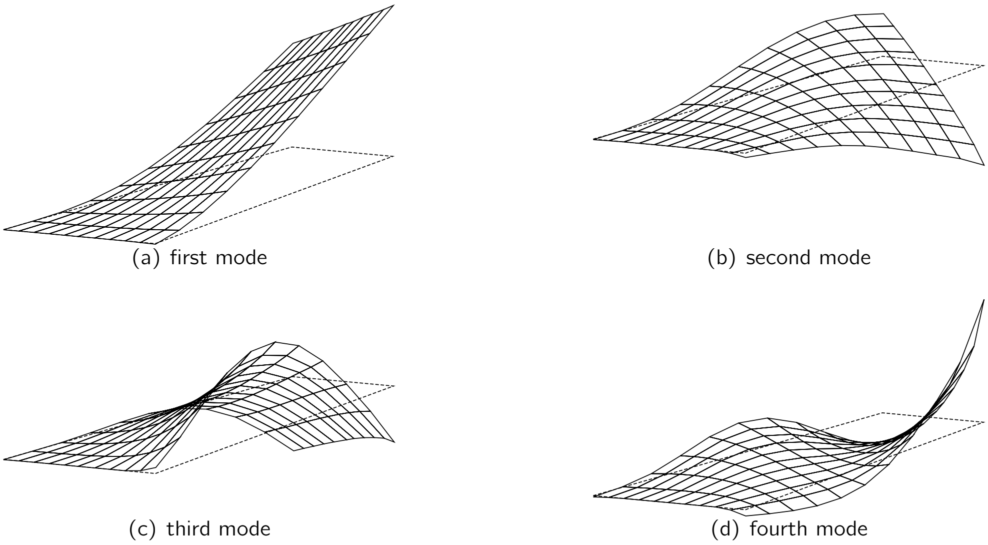


Figure A.6: *Real and imaginary parts of the scaled C_L and C_M response in the frequency domain for different perturbations around an angle of attack of 3.06 degrees.*


(a) first mode

(b) second mode

(c) third mode

(d) fourth mode

Figure A.7: *Fundamental modes of vibration of the AGARD 445.6 wing.*

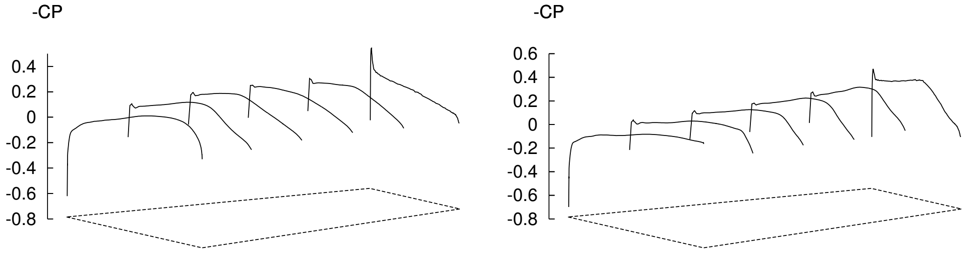


Figure A.8: *The steady pressure distribution of the AGARD 445.6 wing at several sections in spanwise direction: (a) at $M_\infty = 0.96$ and (b) at $M_\infty = 1.141$.*

integrating the modes and the unsteady pressure as

$$G_{ij} = q_\infty \frac{1}{S} \int_S \mathbf{Z}_i \cdot \mathbf{n} C_{p_j} dS \,, \tag{A.22}$$

where S is the wing planform area and q_∞ the farfield dynamic pressure. The dynamic pressure is the parameter which is varied in order to study the stability of the aeroelastic system. Note that the solver computes G_{ij}/q_∞ because the variables in the flow equations are dimensionless. As only four modes are used, the GAF matrix is a 4 × 4 matrix.

For each mode, three amplitudes are considered, which are given in Table A.2.2. The amplitudes are chosen to give the same maximum displacement at the tip of the wing. Two flow conditions are considered: $M_\infty = 0.96$ and $M_\infty = 1.141$, both at zero angle of attack. The latter conditions have also been considered in the original investigation [33]. The pressure distributions along the span at those two conditions are given in Fig. A.8.

	mode 1	mode 2	mode 3	mode 4
δ_1	3.6×10^{-2}	2.0×10^{-2}	5.0×10^{-2}	1.0×10^{-2}
δ_2	3.6×10^{-3}	2.0×10^{-3}	5.0×10^{-3}	1.0×10^{-3}
δ_3	3.6×10^{-4}	2.0×10^{-4}	5.0×10^{-4}	1.0×10^{-4}

Table A.1: *Magnitudes of the modal displacements of the first four eigenmodes*

The GAF matrix is only given in the frequency domain. For the subsonic condition the GAF matrix is given in Fig. A.9 whereas for the supersonic condition it is given in Fig. A.10. For both conditions the matrices compare well with those obtained by other authors [115, 72]. As can be seen the responses for different amplitudes of the perturbations seem to overlap. Only for some elements small differences appear. For instance, Fig. A.11 shows the element for which the differences are the largest. As can be seen the differences are indeed small. In practice, also in this case the behavior of the system may be considered to be linear.

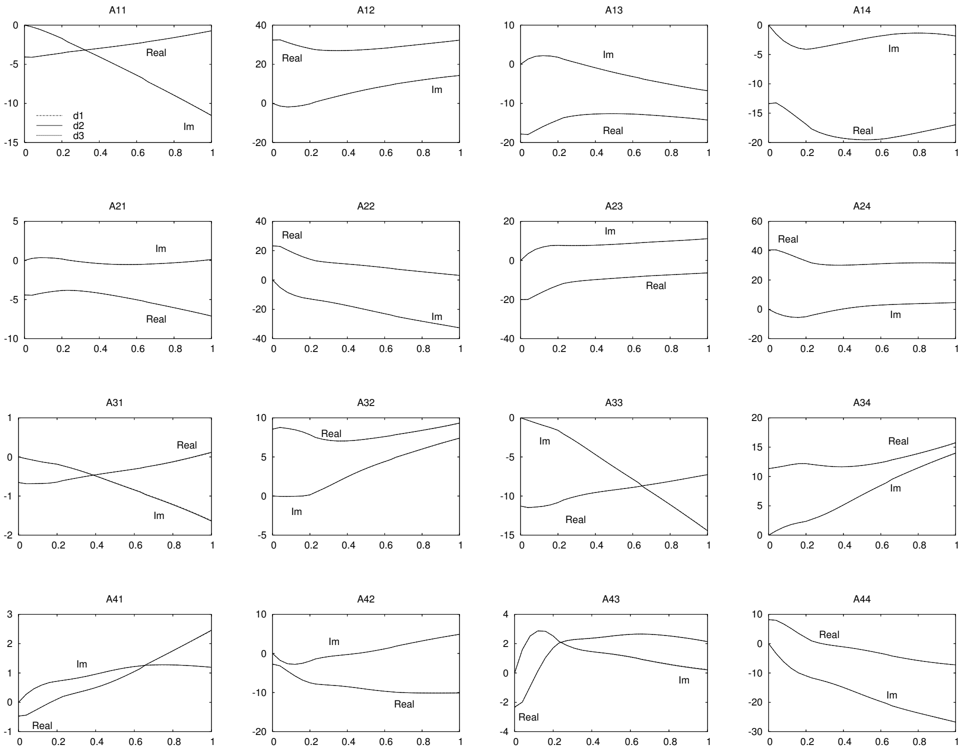


Figure A.9: *GAF matrix. AGARD 445.6 wing at* $M_\infty = 0.96$ *and* $\alpha = 0.0^\circ$.

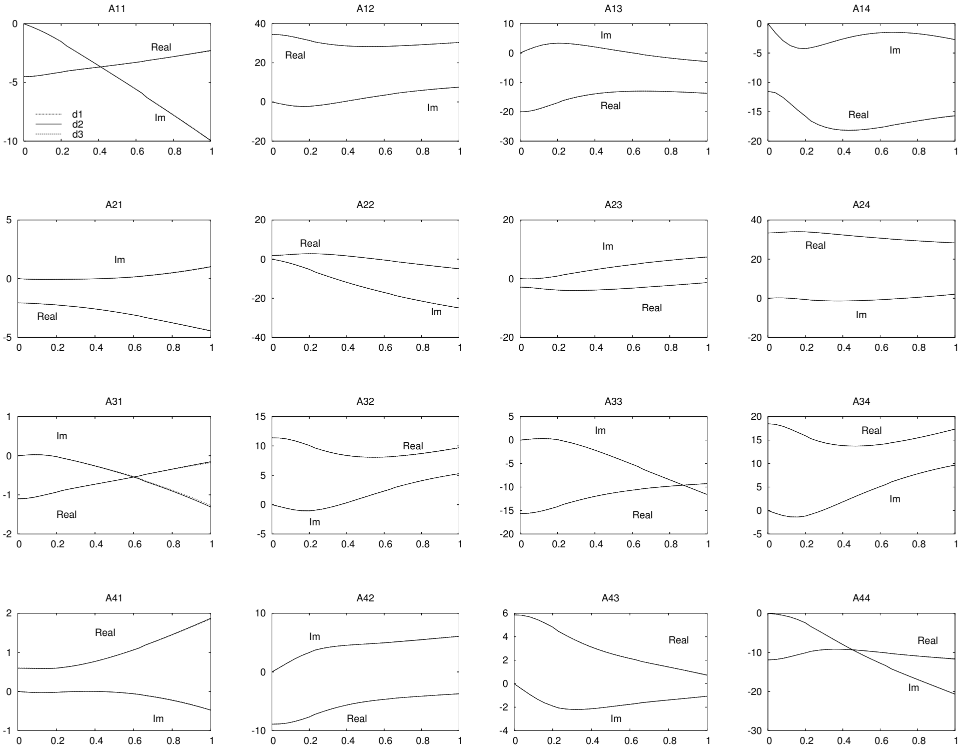


Figure A.10: *GAF matrix. AGARD 445.6 wing at* $M_\infty = 1.141$ *and* $\alpha = 0.0^\circ$.

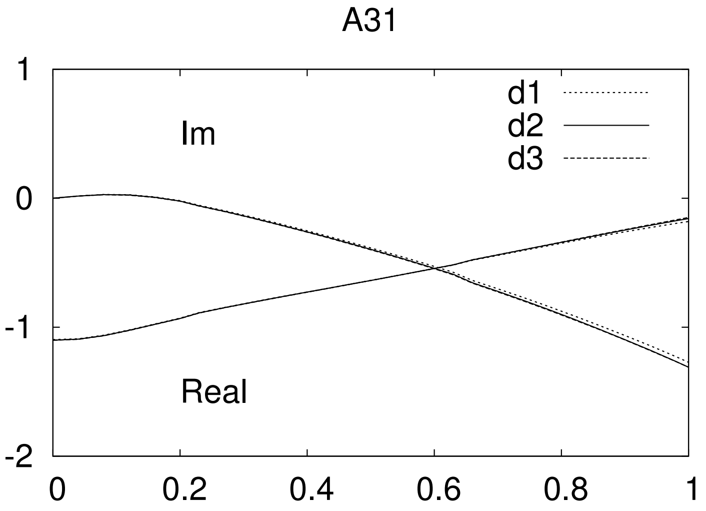


Figure A.11: *A31 element of the GAF matrix. AGARD 445.6 wing at* $M_\infty = 1.141$ *and* $\alpha = 0.0°$.

A.3 Concluding remarks

The unsteady solver implemented in the present work is suitable for the computation of the inviscid flow around a deforming body. Numerical results show that the solver is accurate and efficient, and may be used for cases of aeronautical interest. The computation of the transonic flow around oscillating airfoils and wings, which have been presented, is an example of one of such cases.

The solver has also been used for investigating the impulsive response of wings in transonic flows. The purpose of the investigation was to show that wings may be assumed to behave linearly in transonic flows and, therefore, the aeroelastic stability of the system may be studied very easily using frequency domain analysis of the impulsive response. Results for two wings appeared to confirm the linear behavior that was previously observed for airfoils. In fact, the responses obtained with different impulse amplitudes appeared to overlap when scaled properly.

In the present work there has been no time to use the responses for the stability analysis and therefore for the computation of the flutter speed. It is recommended that future work should also consider whether the flutter speed obtained by the different scaled responses is the same. Moreover, it would be interesting to compare the speed obtained by the stability analysis with that obtained by time-marching the structure and the airflow simultaneously. The solver seems to be suitable for such applications and would require minor modifications only.

Bibliography

[1] Community Portal for Automatic Differentiation. Website on Automatic Differentiation available at http://www.autodiff.org.

[2] Edge User Guide. Edge software documentation available at http://www.foi.se/edge.

[3] Tapenade ©INRIA 2002, version 2.0. Software and documentation available at http://tapenade.inria.fr:8080/tapenade/index.jsp.

[4] SLATEC Common Mathematical Library, Version 4.1, 1993. Public domain software available at http://www.netlib.org/slatec.

[5] O. Amoignon. *Numerical Methods for Aerodynamic Shape Optimization*. PhD thesis. Voume 135 of *Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology*, Uppsala: Acta Universitatis Upsaliensis, 2005.

[6] O. Amoignon and M. Berggren. Adjoint of a median-dual finite-volume scheme: Application to transonic aerodynamic shape optimization. Technical Report 2006-013, Department of Information Technology, Uppsala University, Uppsala, Sweden, 2006.

[7] W. Anderson and D. Bonhaus. Airfoil Design on Unstructured Grids for Turbulent Flows. *AIAA Journal*, 37:185–191, 1999.

[8] T. J. Barth. Analysis of Implicit Local Linearization Techniques for Upwind and TVD Algorithms. AIAA Paper 87-0595, Reno, NV, Jan. 1987.

[9] T. J. Barth. Aspects of Unstructured Grids and Finite-Volume Solvers for the Euler and Navier-Stokes Equations. In *Unstructured Grid Methods for Advection Dominated Flows*. AGARD-R-787, May 1992.

[10] T. J. Barth and D. C. Jespersen. The Design and Application of Upwind Schemes on Unstructured Meshes. AIAA Paper 89-0366, Reno, NV, Jan. 1989.

[11] T. J. Barth and S. W. Linton. An Unstructured Mesh Newton Solver for Compressible Fluid Flow and its Parallel Implementation. AIAA Paper 95-0221, Reno, NV, Jan. 1995.

[12] F. Bauer, P. Garabedian, and D. Korn. *A Theory of Supercritical Wing Sections, with Computer Programs and Examples*. Springer Verlag, 1972.

[13] O. Baysal and K. Ghayour. Continuous Adjoint Sensitivities for Optimization with General Cost Functionals on Unstructured Meshes. *AIAA Journal*, 39(1):48–55, 2001.

[14] M. O. Bristeau, R. Glowinski, J. Périaux, P. Perrier, O. Pironneau, and G. Poirier. On the Numerical Solution of Nonlinear Problems in Fluid Dynamics by Least Squares and Finite Element Methods II. Application to Transonic Flow Simulations. *Computer Methods in Applied Mechanics and Engineering*, 51:363–394, 1985.

[15] O. Brodersen and A. Stürmer. Drag Prediction of Engine-Airframe Interference Effects Using Unstructured Navier-Stokes Calculations. AIAA Paper 2001-2414, June 2001.

[16] G. Carpentieri, B. Koren, and M. J. L. van Tooren. Adjoint-Based Aerodynamic Shape Optimization on Unstructured Meshes. *Journal of Computational Physics*, 224(1):267–287, 2007.

[17] G. Carpentieri, B. Koren, and M. J. L. van Tooren. Development of the Discrete Adjoint for a 3D Unstructured Euler Solver. *Journal of Aircraft*, 45(1):237–245, 2008. Also AIAA Paper 2007-3954. Presented at the 18th AIAA Computational Fluid Dynamics Conference, 25-28 June 2007, Miami, FL.

[18] G. Carpentieri and M. J. L. van Tooren. A Framework for Aerodynamic Shape Optimization. In G. Buttazzo and A. Frediani, editors, *Variational Analysis and Aerospace Engineering*, volume 33 of *Springer Optimization and Its Applications*, 2009. Proceedings of the 47th Workshop on Variational Analysis and Aerospace Engineering held in Erice, Italy in September 2007.

[19] G. Carpentieri, M. J. L. van Tooren, M. Kelly, and R. Cooper. Airfoil Optimization Using an Analytical Shape Parameterization. CEIAT Paper 05-0073, Belfast, UK, Aug. 2005.

[20] G. Carpentieri, M. J. L. van Tooren, and B. Koren. Improving the Efficiency of Aerodynamic Shape Optimization on Unstructured Meshes. AIAA Paper 2006-298. Presented at the 44th AIAA Aerospace Sciences Meeting and Exhibit, 9-12 January 2006, Reno, Nevada.

[21] G. Carpentieri, M. J. L. van Tooren, and B. Koren. Aerodynamic Shape Optimization by means of Sequential Linear Programming Techniques. ECCOMAS CFD 2006 conference paper, Egmond aan Zee, The Netherlands, 2006.

[22] G. Carpentieri, M. J. L. van Tooren, and B. Koren. Comparison of Exact and Approximate Discrete Adjoint for Aerodynamic Shape Optimization. In H. Deconinck and E. Dick, editors, *Computational Fluid Dynamics 2006*. Springer, 2009. Proceedings of the Fourth International Conference on Computational Fluid Dynamics, ICCFD, Ghent, Belgium, 10-14 July 2006.

[23] L. Catalano, A. Dadone, V. Daloiso, and G. Mele. Progressive Optimization Using Orthogonal Shape Functions and Efficient Finite-Difference Sensitivities. AIAA Paper 2003-3961, June 2003.

[24] T. Cebeci. *Analysis of Turbulent Flows*. Elsevier Science, second edition, 2004.

[25] R. F. Chen and Z. Wang. An Improved LU-SGS Scheme with Faster Convergence for Unstructured Grids of Arbitrary Topology. AIAA Paper 99-16759, Jan. 1999.

[26] R. Courant, K. Friedrichs, and H. Lewy. On the Partial Difference Equations of Mathematical Physics. *IBM Journal*, 11(2):215–234, 1967.

[27] N. Cumpsty. *Jet Propulsion: A Simple Guide to the Aerodynamic and Thermodynamic Design and Performance of Jet Engines*. Cambridge University Press, Cambridge, United Kingdom, 2nd edition, 2003.

[28] J. Désidéri, B. E. Maid, and A. Janca. Nested and Self-Adaptive Bézier Parameterization for Shape Optimization. *Journal of Computational Physics*, 224(1):117–131, 2007.

[29] J. A. Désidéri and A. Dervieux. Compressible Flow Solvers using Unstructured Grids. Lecture Series 1988-05, Von Karman Institute for Fluid Dynamics, 1988.

[30] S. Davis. NACA 64A010 (NASA Ames Model) Oscillatory Pitching. AGARD Report 702, Jan. 1982.

[31] E. Dowell, E. Crawley, H. J. Curtiss, D. Peters, R. Scanlan, and F. Sisto. *A Modern Course in Aeroelasticity*. Kluwer Academic Publisher, Dordrecht, The Netherlands, 3rd revised and enlarged edition, 1995.

[32] R. P. Dwight and J. Brezillon. Effect of Approximations of the Discrete Adjoint on Gradient-Based Optimization. *AIAA Journal*, 44(12):3022–3031, 2006.

[33] J. E. C. Yates. AGARD Standard Aeroelastic Configurations for Dynamic Responses. Candidate Configuration I- Wing 445.6. AGARD Report-765, 1988.

[34] P. Eliasson. Edge, a Navier-Stokes Solver, for Unstructured Grids. Technical Report FOI-R-0298-SE, Swedish Defence Research Agency, Stockholm, Sweden, 2001.

[35] J. Elliott and J. Peraire. Practical 3D Aerodynamic Design and Optimization Using Unstructured Meshes. *AIAA Journal*, 35(9):1479–1485, 1997.

[36] V. Frayssé, L. Giraud, S. Gratton, and J. Langou. A Set of GMRES Routines for Real and Complex Arithmetics on High Performance Computers. CERFACS Tech. Rep. TR/PA/03/3, 2003. public domain software available on www.cerfacs/algor/Softs.

[37] M. B. Giles, M. C. Duta, J.-D. Müller, and N. A. Pierce. Algorithm Developments for Discrete Adjoint Methods. *AIAA Journal*, 41(2):198–205, 2003.

[38] M. B. Giles and N. A. Pierce. An Introduction to the Adjoint Approach to Design. *Flow, Turbulence and Combustion*, 65(3-4):393–415, 2000.

[39] S. Godunov. A Difference Scheme for Numerical Solution of Discontinuous Solution of Hydrodynamic Equations. *Math. Sbornik*, 47:271–306, 1969.

[40] A. Guardone and L. Quartapelle. Spatially Factorized Galerkin and Taylor-Galerkin Schemes for Multidimensional Conservation Laws. Scientific Report DIA-SR 00-18, Politecnico di Milano, 2000.

[41] R. L. H. Luo, J.D. Baum. A Fast, Matrix-Free Implicit Method for Compressible Flows on Unstructured Grids. *Journal of Computational Physics*, 146:664–690, 1998.

[42] A. Harten. High Resolution Schemes for Hyperbolic Conservation Laws. *Journal of Computational Physics*, 49:357–393, 1983.

[43] A. Haselbacher, J. McGuirk, and G. Page. Finite Volume Discretization Aspects for Viscous Flows on Mixed Unstructured Grids. *AIAA Journal*, 37(2):177–184, 1999.

[44] P. Hemker and S. Spekreijse. Multiple Grid and Osher's Scheme for the Efficient Solution of the Steady Euler Equations. *Applied Numerical Mathematics*, 2(6):475–494, 1986.

[45] J. Hess and A. Smith. Calculation of Potential Flow About Arbitrary Bodies. *Progress in Aeronautical Sciences*, 8:1–138, 1967.

[46] R. Hicks and P. Henne. Wing Design by Numerical Optimization. *Journal of Aircraft*, 15(7):407–412, 1978.

[47] R. Hicks and G. Vanderplaats. Application of Numerical Optimization to the Design of Supercritical Airfoils Without Drag-Creep. SAE Paper 770440, 1977.

[48] R. Hicks, G. Vanderplaats, E. Murman, and R. King. Airfoil Section Drag Reduction at Transonic Speeds by Numerical Optimization. SAE Paper 760477, 1976.

[49] C. Hirsch. *Numerical Computation of Internal and External Flows. Volume 1: Fundamentals of Numerical Discretization*. John Wiley & Sons, 1988.

[50] C. Hirsch. *Numerical Computation of Internal and External Flows. Volume 2: Computational Methods for Inviscid and Viscous Flows*. John Wiley & Sons, 1988.

[51] A. Jameson. Iterative Solution of Transonic Flows over Airfoils and Wings, Including Flow at Mach 1. *Communications in Pure and Applied Mathemathics*, 27:238–309, 1974.

[52] A. Jameson. Aerodynamic Design via Control Theory. *Journal of Scientific Computing*, 3:233–260, 1988.

[53] A. Jameson. Optimum Aerodynamic Design using CFD and Control Theory. AIAA Paper 95-1729-CP, 1995.

[54] A. Jameson. Re-engineering the Desing Process Through Computation. *Journal of Aircraft*, 36(1):36–50, 1999.

[55] A. Jameson. CFD for Aerodynamic Design and Optimization: Its Evolution over the Last Three Decades. AIAA Paper 2003-3438, 2003.

[56] A. Jameson, T. Baker, and N. Weatherill. Calculation of Inviscid Transonic Flow Over a Complete Aircraft. AIAA Paper 86-0103, 1986.

[57] A. Jameson, N. Pierce, and L. Martinelli. Optimum Aerodynamic Design using the Navier-Stokes Equations. AIAA Paper 97-0101, 1997.

[58] A. Jameson, W. Schmidt, and E. Turkel. Numerical Solution of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time-Stepping Schemes. AIAA Paper 81-1259, 1981.

[59] A. Jameson and J. Vassberg. Computational Fluid Dynamics for Aerodynamic Design: Its Current and Future Impact. AIAA Paper 2001-0538, 2001.

[60] A. Jameson and S. Yoon. Lower-Upper Implicit Schemes with Multiple Grids for the Euler Equations. *AIAA Journal*, 25:929–935, 1987.

[61] F. Johnson, E. Tinoco, and N. Yu. Thirty Years of Development and Application of CFD at Boeing Commercial Airplanes, Seattle. *Computers & Fluids*, 34(10):1115–1151, 2005.

[62] J. Katz and J. Plotkin. *Low Speed Aerodynamics*. Cambridge University Press, second edition, 2001.

[63] C. Kim, C. Kim, and O. Rho. Sensitivity Analysis for the Navier-Stokes Equations with Two-Equation Turbulence Models. *AIAA Journal*, 39:838–845, 2001.

[64] D. A. Knoll and D. E. Keyes. Jacobian-Free Newton-Krylov Methods: a Survey of Approaches and Applications. *Journal of Computational Physics*, 193:357–397, 2004.

[65] B. Koobus and C. Farhat. Second-Order Time-Accurate and Geometrically Conservative Implicit Schemes for Flow Computations on Unstructured Dynamic Meshes. *Computed Methods in Applied Mechanics and Engineering*, 170:103–130, 1999.

[66] B. Koren. Defect Correction and Multigrid for an Efficient and Accurate Computation of Airfoil Flows. *Journal of Computational Physics*, 77(1):183–206, 1988.

[67] B. Koren. *Multigrid and Defect Correction for Steady Navier-Stokes Equations: Application to Aerodynamics* . PhD thesis, Delft University of Technology, Delft, The Netherlands, May 1989.

[68] B. M. Kulfan. A Universal Parametric Geometry Representation Method-"CST". AIAA Paper 07-0062, Reno, NV, Jan. 2007.

[69] G. Kuruvila, S. Ta'asan, and M. Salas. Airfoil Design and Optimization by the One-Shot Method. AIAA Paper 95-0478, 1995.

[70] C. B. Laney. *Computational Gasdynamics*. Cambridge University Press, 1998.

[71] R. J. Le Veque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2002.

[72] E. M. Lee-Rausch and J. T. Batina. Calculation of AGARD Wing 445.6 Flutter Using Navier-Stokes Aerodynamics. AIAA Paper 93-3476, 11th Applied Aerodynamics Conference, Monterey, California, 1993.

[73] J. Lépine, F. Guibault, J. Trépanier, and F. Pépin. Optimized Non Uniform Rational B-Spline Geometrical Representation for Aerodynamic Design of Wings. *AIAA Journal*, 39(11):2033–2041, 2001.

[74] J. Lépine, J. Trépanier, and F. Pépin. Wing Aerodynamic Design Using an Optimized NURBS Geometrical Representation. AIAA Paper 2000-0669, Jan. 2000.

[75] M. Lighthill. A New Method of Two Dimensional Aerodynamic Design. Aeronautical Research Council's Reports and Memoranda 2112, 1945.

[76] M. Lighthill. The Hodograph Transformation in Trans-sonic Flow. I-III. *Proceedings of the Royal Society*, 191:323–369, 1947.

[77] M. Lighthill. On Displacement Thickness. *Journal of Fluid Mechanics*, 4:383–392, 1958.

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

[79] P. Lisandrin, G. Carpentieri, and M. J. L. van Tooren. An Investigation over CFD-Based Models for the Identification of Non-Linear Unsteady Aerodynamics Responses. *AIAA Journal*, 44(9):2043–2050, 2006.

[80] D. J. Mavriplis. Multigrid Techniques for Unstructured Meshes. Lecture Series VKI-LS 1995-02, Von Karman Institute for Fluid Dynamics, 1995.

[81] D. J. Mavriplis. Unstructured Grid Techniques. *Annual Review of Fluid Mechanics*, 29:473–514, 1997.

[82] D. J. Mavriplis. On Convergence Acceleration Techniques for Unstructured Meshes. AIAA Paper 98-2966, June 1998.

[83] D. J. Mavriplis. Formulation and Multigrid Solution of the Discrete Adjoint for Optimization Problems on Unstructured Meshes. AIAA Paper 05-0319, Reno, NV, Jan. 2005.

[84] D. J. Mavriplis. Discrete Adjoint-Based Approach for Optimization Problems on Three-Dimensional Unstructured Meshes. *AIAA Journal*, 45(4):740–750, 2007.

[85] J.-D. Müller and P. Cusdin. On the Performance of Discrete Adjoint CFD Codes Using Automatic Differentiation. *International Journal for Numerical Methods in Fluids*, 47(8-9):939–945, 2003.

[86] B. Mohammadi. A New Optimal Shape Design Procedure for Inviscid and Viscous Turbulent Flows. *International Journal for Numerical Methods in Fluids*, 25(2):183–203, 1997.

[87] B. Mohammadi and O. Pironneau. Shape Optimization in Fluid Mechanics. *Annual Review of Fluid Mechanics*, 36:255–279, Jan. 2004.

[88] C. Morawetz. On the Non-Existence of Continuous Transonic Flows Past Profiles, Part 1 . *Communications on Pure and Applied Mathematics*, 9:45–68, 1956.

[89] L. Morino and C. Kuo. Subsonic Potential Aerodynamics for Complex Configurations: a General Theory. *AIAA Journal*, 12(2):191–197, 1974.

[90] W. A. Mulder. Multigrid Relaxation for the Euler Equations. *Journal of Computational Physics*, 60(2):235–252, 1985.

[91] E. Murman and J. Cole. Calculation of Plane Steady Transonic Flows. *AIAA Journal*, 9:114–121, 1971.

[92] S. K. Nadarajah and A. Jameson. A Comparison of the Continuous and Discrete Adjoint Approach to Automatic Aerodynamic Optimization. AIAA Paper 00-16527, Jan. 2000.

[93] N. Nemec and D. Zingg. Newton-Krylov Algorithm for Aerodynamic Design Using the Navier Stokes Equations. *AIAA Journal*, 40(6):1146–1154, 2002.

[94] J. C. Newman III, A. C. Taylor III, R. Barnwell, P. Newman, and G.-W. Hou. Overview of Sensitivity Analysis and Shape Optimization for Complex Aerodynamic Configurations. *Journal of Aircraft*, 36(2):87–96, 1999.

[95] E. Nielsen and W. Anderson. Aerodynamic Design Optimization on Unstructured Meshes Using the Navier-Stokes Equations. *AIAA Journal*, 37:1411–1419, 1999.

[96] E. Nielsen and M. Park. Using an Adjoint Approach to Eliminate Mesh Sensitivities in Computational Design. *AIAA Journal*, 44(5):948–953, 2006.

[97] E. J. Nielsen, J. Lu, M. A. Park, and D. L. Darmofal. An Implicit, Exact Dual Adjoint Solution Method for Turbulent Flows on Unstructured Grids. *Computers & Fluids*, 33(9):1131–1155, 2004.

[98] G. Nieuwland. Transonic Potential Flow Around a Family of Quasi-Elliptical Airfoil Sections. NLR Report TRT 172, 1967.

[99] B. Nkonga and H. Guillard. Godunov-Type Method on Non-Structured Meshes for Three-Dimensional Moving Boundary Problems. *Computational Methods in Applied Mechanics and Engineering*, 113:183–204, 1994.

[100] O. Pironneau. On Optimum Design in Fluid Mechanics. *Journal of Fluid Mechanics*, 64:97–110, 1974.

[101] O. Pironneau. Control of Transonic Shock Position. *ESAIM: Control, Optimisation and Calculus of Variations*, 8(2):907–914, 2002.

[102] R. J. Prazenica, P. H. Reisenthel, A. J. Kurdila, and M. J. Brenner. Volterra Kernel Extrapolation for Modeling Nonlinear Aeroelastic Systems at Novel Flight Conditions. *Journal of Aircraft*, 44(1):149–162, 2007.

[103] J. Reuther and A. Jameson. Control Based Airfoil Design Using the Euler Equations. AIAA Paper 94-4272-CP, 1994.

[104] J. Reuther, A. Jameson, J. Alonso, M. Remlinger, and D. Saunders. Constrained Multipoint Aerodynamic Shape Optimization Using an Adjoint Formulation and Parallel Computers, Part 1. *Journal of Aircraft*, 36(1):51–60, 1999.

[105] J. Reuther, A. Jameson, J. Alonso, M. Remlinger, and D. Saunders. Constrained Multipoint Aerodynamic Shape Optimization Using an Adjoint Formulation and Parallel Computers, Part 2. *Journal of Aircraft*, 36(1):51–60, 1999.

[106] J. Reuther, A. Jameson, A. Farmer, J. Martinelli, and D. Saunders. Aerodynamic Shape Optimization of Complex Aircraft Configurations via an Adjoint Formulation. AIAA Paper 96-0094, 1996.

[107] G. Robinson and A. Keane. Concise Orthogonal Representation of Supercritical Airfoils. *Journal of Aircraft*, 38(3):580–583, 2001.

[108] P. Roe. Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes. *Journal of Computational Physics*, 43(2):357–372, 1981.

[109] P. Sacher. Numerical Solutions of Two-Dimensional Reference Test Cases. In *Test Cases for Inviscid Flow Field Methods*. AGARD-AR-211, May 1985.

[110] J. Samareh. Novel Multidisciplinary Shape Parameterization Approach. *Journal of Aircraft*, 38(6):1015–1024, 2001.

[111] J. A. Samareh. Survey of Shape Parameterization Techniques for High-Fidelity Multidisciplinary Shape Optimization. *AIAA Journal*, 39(5):877–884, 2001.

[112] V. Schmitt and F. Charpin. Pressure Distributions on the ONERA-M6 Wing at Transonic Mach Numbers. In *Experimental Data Base for Computer Program Assessment*. AGARD-AR-138, May 1979.

[113] V. Selmin and L. Formaggia. Unified Construction of Finite Element and Finite Volume Discretizations for Compressible Flows. *International Journal for Numerical Methods in Engineering*, 39(1):1–32, 1996.

[114] J. Shim, K. D. Lee, and A. Verhoff. An Efficient Aerodynamic Design Method Using Asymptotic Solution of Euler Equations. AIAA Paper 02-3141, St. Louis, Missouri, June 2002.

[115] W. A. Silva and R. E. Bartels. Development of Reduced-Order Models for Aeroelastic Analysis and Flutter Prediction using the CFL3Dv6.0 Code. *Journal of Fluids and Structures*, 42:581–606, 2003.

[116] B. Soemarwoto and T. Labrujere. Airfoil Design and Optimization Methods: Recent Progress at NLR. *International Journal for Numerical Methods in Fluids*, 30(2):217–228, 1999.

[117] M. Sohn and K. Lee. Bézier Curve Application in the Shape Optimization of Transonic Airfoils. AIAA Paper 2000-4523, Aug. 2000.

[118] S. Spekreijse. Multigrid Solution of Monotone Second-Order Discretizations of Hyperbolic Conservation Laws. *Mathematics of Computation*, 49(179):135–155, 1987.

[119] J. Steger and R. Warming. Flux Vector Splitting of the Inviscid Gasdynamic Equations with Application to Finite-Difference Methods. *Journal of Computational Physics*, 40(2):263–293, 1981.

[120] T. Theodorsen and I. Garrick. General Potential Theory of Arbitrary Wing Sections. NACA Tech. Rept. 452, 1933.

[121] P. D. Thomas and C. K. Lombard. Geometric Conservation Law and its Application to Flow Computations on Moving Grids. *AIAA Journal*, 17:1030–1037, 1979.

[122] J. I. Trépanier, J. Lépine, and F. Pépin. An Optimized Geometric Representation for Wing Profile using NURBS. *CASI Journal*, 46(1):12–19, 2000.

[123] G. van Albada, B. van Leer, and W. W. Roberts. A Comparative Study of Computational Methods in Cosmic Gas Dynamics. *Astronomy and Astrophysics*, 108(1):76–84, 1982.

[124] B. van Leer. Towards the Ultimate Conservative Difference Scheme. V. A second-Order Sequel to Godunov's Method. *Journal of Computational Physics*, 32:101–136, 1979.

[125] G. Vanderplaats, R. Hicks, and E. Murman. Application of Numerical Optimization Techniques to Airfoil Design. NASA Report sp-347, 1975.

[126] G. N. Vanderplaats. *Numerical Optimization Techniques for Engineering Design*. Vanderplaats Reasearch & Development, Inc, third edition, 2001.

[127] D. Venditti and D. Darmofal. Grid Adaptation for Functional Outputs: Application to Two-Dimensional Inviscid Flows. *Journal of Computational Physics*, 176(1):40–69, 2002.

[128] V. Venkatakrishnan. Convergence to Steady State Solutions of the Euler Equations on Unstructured Grids with Limiters. *Journal of Computational Physics*, 118(1):120–130, 1995.

[129] V. Venkatakrishnan. Perspective on Unstructured Grid Flow Solvers. *AIAA Journal*, 34(3):533–547, 1996.

[130] V. Venkatakrishnan and D. Mavriplis. Implicit Method for the Computation of Unsteady Flows on Unstructured Meshes. AIAA Paper 95-1705-cp, 1995.

[131] V. Venkatakrishnan and D. J. Mavriplis. Implicit Solvers for Unstructured Meshes. *Journal of Computational Physics*, 105:83–91, 1993.

[132] A. Verhoff, D. Stooksberry, and A. B. Cain. An Efficient Approach to Optimal Aerodynamic Design Part 1: Analytic Geometry and Aerodynamic Sensitivities. AIAA Paper 93-0099, Jan. 1993.

[133] H. Viviand. Numerical Solutions of Two-Dimensional Reference Test Cases. In *Test Cases for Inviscid Flow Field Methods*. AGARD-AR-211, May 1985.

[134] P. Wesseling. *Principles of Computational Fluid Dynamics*. Springer, 2001.

[135] D. C. Wilcox. *Turbulence Modeling for CFD*. D. C. W. Industries, second edition, 2002.

[136] O. Zienkiewicz, R. Taylor, and P. Nithiarasu. *The Finite Element Method For Fluid Dynamics*. Butterworth-Heinemann, sixth edition, 2005.

Acknowledgements

The present thesis could have never been realized without Prof. Michel van Tooren. We met in his department when I arrived in Delft to work on my master thesis. He noticed my interest for aerodynamics and engineering optimization and suggested working on the multidisciplinary design of the Blended Wing Body aircraft. I did not know anything about aircraft design but I was fascinated by the assignment. He introduced me to the basics of aircraft design and pointed my attention to several interesting optimization problems. I gained valuable knowledge from that assignment and realized that I wanted to continue. At the end of the master thesis work, when he offered me a PhD position, I accepted it immediately. I really appreciated that I was given the freedom to chose the direction of my research. However, the first year I felt the pressure of having to decide which way to go and complained with him about it. I remember him answering that I was given the very rare opportunity of doing what I liked, for four years, with even being paid for it. He was right and since then I have just tried to make the most of that opportunity. It was not difficult to do so because Prof. van Tooren was very supportive. He was always there to show his appreciation at every little achievement and to provide encouragement to do more.

A person who played a major role in my research is Prof. Barry Koren. I remember the day I passed by his office for the very first time, with the intention of asking suggestions for my work. He manifested a genuine interest for what I was doing and immediately offered his help. In the following three years we met constantly to discuss future research directions and ways to improve the work. He drew my attention to interesting conferences and seminars, within which I presented my work and obtained valuable feedback. Also, he helped me very much in putting my work on paper. His attention to details was important and decisive for the writing. I really feel that Prof. Barry Koren helped me in constructing this thesis, one piece at a time, in a very structured way.

During my PhD time it was really pleasant to spend time at the faculty with my colleague and friend Paolo Lisandrin. At lunch times or coffee breaks we always had discussions on several non-work related issues. Those discussions continue until today. I was also a co-worker of Paolo, and it was very interesting to work with him on unsteady flow problems. It is a pity that we had no chance to continue with that work.

A student of the department, Wouter Terra, helped me in extending the work carried out with Paolo on airfoil sections, to wings. In fact, Wouter implemented almost all the work contained in the appendix. He carried out his master thesis under my supervision and later became my co-worker. I do not really like teaching/supervising but I have to admit that it gives a fine feeling to see a student achieve so much as Wouter did.

In order to implement my code I spent the second and third year of my PhD project in an office room for my own. I shared an office room in the first and fourth year. Ton van der Laan was my roommate in the first year. He was also my "Belasting Adviseur" (tax consultant). His help was crucial in understanding how to buy a house in the Netherlands and how to make the most of mortgage interest deductions. Apart from taxes, it is thanks to him that I discovered a lot about Dutch culture and habits. Gandert van Raemdonck was my roommate in the fourth year. We shared the same taste for music and, unfortunately, also the same kind of humour. It was very nice to see him making the first steps in the creation of his own company. Luckily, there is someone who is transferring knowledge to the industry.

I have been very happy to be surrounded by nice people at the faculty. I remember the original members of the department: Gianfranco La Rocca, Lin Pijpaert and Lars Kracker. They had to cope with me since the very beginning. When I was tired of my work, every occasion was good to make a trip to their desks and engage in endless conversations. It has been very nice to meet other students and employees in the department. Martijn, Mary, Haroon, Michiel, Jochem, Joost, Wim, Teodor, Meng and Benedetto have contributed to make my stay at the faculty very pleasant.

All the friends that my girlfriend Tamara and I met in Delft have been special for us. They are: Ali, Barbara, Alexis, Marco, Lara, Simon, Christian, Elvira, Pascal, Greg, Alet, Adriano, Verena, Emile, Eleonora, Aurelie, Marcos, Carla, Lena, Nicholas, Marco, Regan, Amer, Valeria and Mariska. Together we shared beautiful moments and without them living in Delft would have been much less enjoyable.

My girlfriend Tamara was the person who supported me the most during my PhD project. She moved with me to The Netherlands when I started on the project; she had to give up family and friends. In the beginning there were plenty of difficulties for her. However, she was very strong and never stepped back. Her optimism kept me going and when I felt really tired and unmotivated she was always there to encourage me and to tell me that everything would be fine. We went through a lot together but it was really worth it because the time I spent with her has been the happiest of my life so far. I am very confident that we will make it, also this time.

Finally, I could reach the point of starting my PhD research because of the support that I received from my family since I was a child. My sister, my mother and my father were always there for me and always provided encouragement in all the challenges that I faced. My mother and my father worked very hard for my sister and me to make sure that we always got the best of everything. And I know that it was not easy for them, not at all. Words such as holidays or weekends were not in their dictionary. By looking at them I understood that things do not get done from 9 to 5, especially if you want to change your luck.

About the author

Giampietro Carpentieri was born on the 16th of October 1978 in Rome, Italy. From September 1992 to July 1997 he attended high school at the "Instituto Tecnico Agrario Statale G. Garibaldi," in Rome. In October 1997 he enrolled at the Faculty of Mechanical Engineering of the "Università degli Studi Roma Tre." He carried out most of the work for his graduation thesis at the Faculty of Aerospace Engineering of the Delft University of Technology, in The Netherlands. He obtained his cum laude university degree, "Laurea di Dottore in Ingegneria Meccanica," in October 2003. In December of the same year he accepted a PhD student position at the Faculty of Aerospace Engineering in Delft. During the following four years he conducted research on numerical methods for aerodynamic shape optimization and obtained the results presented in this thesis. In March 2008 he started working as a quantitative analyst for a Dutch bank in London. In March 2009, following the worldwide financial crisis, he was relocated to The Netherlands, where he is currently working.