One-Shot Methods for Aerodynamic Shape Optimization

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

M.Sc. Emre Özkaya

aus Bornova, Türkei

Berichter:
Univ.-Prof. Dr.rer.nat. Nicolas Ralph Gauger
Prof. Dr. Andreas Griewank

Tag der mündlichen Prüfung: 11.08.2014

Abstract

This thesis is concerned with the aerodynamic shape optimization based on the one-shot strategy, which aims at performing an optimization simultaneously with the simulation process. Combined with the consistent discrete adjoint method based on Automatic Differentiation (AD), one-shot methods enable performing a shape optimization at a small multiple of the time required for a single flow simulation. In the present work, we first investigate the preconditioning techniques and constructive conditions that are required to satisfy the contractivity of the coupled one-shot iterations. Further, an analysis concerning the quantification of the retardation rate, which is an indication of the computational cost of the overall optimization compared to a single flow simulation, is presented. We also discuss the implementation aspects concerning the aerodynamic design and optimization chains, and present a review of the shape parameterization techniques commonly used in aerodynamic shape optimization. An assessment of the most common sensitivity evaluation methods with respect to accuracy, computational cost and robustness is performed. Among all the methods, AD based consistent discrete adjoint method is the most robust and flexible method, which enables computing accurate sensitivity information at a fixed computation cost independent of the number of design parameters used in the optimization. The application of this strategy as well as the AD techniques to generate adjoint solvers from the in-house state equation solvers are discussed in detail. Furthermore, advanced techniques to improve the computational performance are introduced. In the last section, first validation results obtained from the developed adjoint Euler and Navier-Stokes solvers are presented. The run-time and memory requirements of the adjoint solvers using different grid levels are provided to demonstrate the efficiency of the chosen adjoint methodology. Then, optimization results that are obtained by applying the one-shot method to three different airfoil optimization scenarios are presented. Furthermore, a comparison between the one-shot method and the nested quasi-Newton method is performed using one of the test cases. The run-time measurements are presented to prove the efficiency of the one-shot method.

Zusammenfassung

Die vorliegende Dissertation befasst sich mit der aerodynamischen Formoptimierung anhand des One-Shot-Verfahrens, welches bei der Durchführung des Simulationsprozesses auf ein gleichzeitiges Erreichen der Optimalitätsbedingungen abzielt. Kombiniert mit der diskreten Adjungiertenmethode ermöglicht das One-Shot-Verfahren die Durchführung einer Formoptimierung mit einem Rechenaufwand, der nur ein kleines Vielfaches der Zeit für die einzelnen Strömungssimulation benötigt. In der vorliegenden Arbeit werden zunächst Präkonditionierungstechniken und Bedingungen untersucht, die die Kontraktivität der gekoppelten One-Shot Iteration sichern. Eine Analyse zur Quantifizierung der Verzögerungsrate, welche die Rechenaufwandsteigerung der Gesamtoptimierung im Hinblick auf einzelne Simulationen misst, wird dargestellt. Im Anschluss werden implementatorische Aspekte der aerodynamischen Entwurfs- und Optimierungskette und ein Überblick über allgemeine Parametrisierungstechniken geliefert. Verschiedene Methoden zur Berechnung der Sensitivitäten in Bezug auf Genauigkeit, Rechenaufwand und Robustheit werden untersucht und bewertet. Insbesondere die auf Automatischem Differenzieren (AD) basierende konsistente diskrete Adjungiertenmethode zeichnet sich durch Robustheit und Flexibilität aus und ermöglicht die exakte Berechnung der Sensitivitäten zu einem festen Rechenaufwand unabhängig von der Anzahl der Entwurfsparameter. Die Anwendung dieser Strategie sowie dafür eingesetzte AD-Techniken werden im Detail diskutiert. Darüber hinaus werden Strategien eingeführt und diskutiert, die die Laufzeit und den Speicherbedarf des adjungierten Lösers reduzieren. Im letzten Teil der Arbeit werden zunächst Validierungsergebnisse der entwickelten adjungierten Euler und Navier-Stokes Löser vorgestellt. Laufzeit- und Speicherbedarfmessungen für unterschiedliche Gitterstufen unterstreichen dabei die Effizienz der gewählten Methode. Anschließend werden Ergebnisse, die durch Anwendung des One-Shot-Verfahrens auf drei unterschiedliche Profiloptimierungsszenarien erzielt wurden, vorgestellt. Eine Vergleichsstudie zwischen der One-Shot Methode und einem klassischen Quasi-Newton-Verfahren wird durchgeführt, wobei Laufzeitmessungen die Effizienz des One-Shot-Verfahrens nachweisen.

Contents

Acknowledgements

I started this Ph.D. thesis in the Department of Mathematics in Humboldt University Berlin and finalized it in MathCCES, RWTH Aachen University. First of all I would like to thank Prof. Dr. Nicolas Gauger who supervised me during the thesis and enabled me to enjoy the great research atmosphere in these two distinguished institutes. It has been for me always a pleasant time in his research group.

I am a lot indebted to Prof. Dr. Andreas Griewank for his invaluable contributions to my thesis work and for many other fruitful discussions. It was a great pleasure and honor for me to work with him.

Special thanks to Dr. Anil Nemili and Dr. Adel Hamdi for their numerous suggestions for the thesis work, and most importantly for their company during all the years, which I enjoyed a lot.

Also many thanks to Prof. Dr. Andrea Walther and Dr. Laurent Hascoët for their friendly support, especially for Automatic Differentiation tools. Their help to me was of great value.

I wish to thank Dr. Angelo Carnarius for his great support on the flow solver and interesting discussions about CFD. I really enjoyed the nice interdisciplinary collaboration with him.

In addition, I am thankful to Stefanie Günther and Torsten Bosse for their help in correcting the thesis. I also thank all my ex-colleagues at the Humboldt University as well as colleagues at the RWTH Aachen University for the great time.

Finally, I would like to thank my family for their support and patience during all the years. Without them, completing this thesis would not have been possible.

CHAPTER 1

Introduction and outline

In the last decades, Computational Fluid Dynamics (CFD) methods have become a standard tool in aerodynamic shape optimization. CFD tools, which had been initially used only for the performance evaluation of the final design, have now found a wide usage in the design process. In order to take advantage of the large amount of information provided by the high-fidelity physical models of fluid flow, it is desirable to give a high degree of freedom to the shape parameterization, which leads to a large number of design parameters. In fact, aerodynamic shape optimization has two clear trends: increasing number of design parameters and increasing complexity of the simulation tools. Therefore, one requires more sophisticated optimization and sensitivity evaluation methods to tackle with the large number of design parameters and high computational cost. Among the deterministic optimization methods, gradient-based search methods are widely used as they require much less number of flow simulations compared to other methods.

For large-scale aerodynamic shape optimization using gradient-based methods, efficient evaluation of the shape sensitivities is an important aspect. Among the various sensitivity evaluation methods, the adjoint method is the most efficient way of evaluating the gradient vector of a scalar objective function with respect to a large number of shape parameters. In contrast to finite difference or linearization methods, computational cost of the adjoint methods remains bounded irrespective of the number of shape parameters involved in the optimization.

As the adjoint methods became rapidly popular, they have been utilized for diverse applications of shape optimization and flow control by various research groups. In general, these works were mainly based on the nested gradient-based search algorithms for constraint problems such as sequential quadratic programming (SQP) or interior point methods, in which adjoint and state equations are solved with a good accuracy in each optimization cycle. As an alternative to the nested methods, the one-shot method was presented in a work by Ta'asan [**Ta'91**], in which he suggested updating design parameters in a multi-grid framework. His basic idea was to decouple the smooth low-frequency and local high-frequency shape deformations from each other, and apply these on the most suitable grid level. Kuruvila et al. [**KTS95**] applied this method successfully for the optimization of a NACA 0012 airfoil using potential flow equations. Later, Ta'asan [**Ta'95**] suggested an improved method, in which the state and adjoint equations are solved in a pseudo-time embedding with the design equation being treated as an additional boundary condition. In contrast to the previous method, the new method does not require the multi-grid capability of flow solver. Iollo et al. [**IKT96**] have applied this method to the inverse design problems, in which a tracking type of objective functions for pressure distribution are optimized. Hazra et al. [**HSBG05**] extended Ta'asan's approach further and used simultaneous pseudo time-stepping, which is

applied to three equations of the optimality system. The convergence of the method was achieved by using a preconditioner motivated by the reduced-SQP method. They applied the suggested method for airfoil optimization using Euler equations and achieved a significant improvement for the overall run-time compared to nested methods. Later, the same strategy has been applied to different test cases using Navier-Stokes equations in [**SG09, HJ12**]. Parallel to these works, in a more broad context of PDE optimization, Griewank and Faure [**GF02**] have proposed solving state and adjoint equations simultaneously using coupled fixed-point iterations in a "piggy-back" fashion, and promoted Automatic Differentiation to compute the Jacobian-vector products that are required in the adjoint fixed-point scheme. In the successive work [**Gri06**], piggy-back approach has been extended to the one-shot method that includes design updates. Gauger et al. [**GGR08**] applied this method to a shape optimization problem using compressible Euler equations. Hamdi et al. [**HG09**] refined the method further and introduced a new preconditioner for the design updates based on a augmented Lagrangian functional, which involves penalty terms for the primal and adjoint residuals. The convergence analysis of the extended method in function space setting has been performed by Kaland et al. [**KRG14**].

One important issue while developing one-shot methods is the type of adjoint method employed in the optimization process. In general, robustness and numerical efficiency of the optimization process rely to a great extent on the chosen adjoint method. In contrast to the nested methods, the one-shot method requires derivative information at a design point, in which convergence of the state and adjoint solvers is not achieved. Therefore, an adjoint method, which is supposed to be suitable for the one-shot method, must be able to compute required sensitivity information at any residual level in an accurate and robust way. In general, the adjoint approaches can be divided into two classes: continuos and discrete adjoint methods.

The continuous adjoint method in fluid mechanics context was first suggested by Pironneau [**Pir74**]. It became popular after Jameson [**Jam88**] used it for the shape optimization of airfoils using adjoint Euler equations. Jameson et al. [**JMP98**] extended the continuous adjoint approach to the Navier-Stokes equations and employed it further for the optimization of wing and wing-body configurations. Later, several drawbacks associated with the continuous adjoint method such as inconsistency errors due to grid resolution, convergence problems due to numerical stiffness, difficulties in deriving adjoint turbulence models, etc., motivated some researchers to derive the adjoint equation based on the discrete formulation of the optimization problem. This approach, which is called as the discrete adjoint method, consequently became more popular than the continuous adjoint method and has been applied to a wide variety of aerodynamic shape optimization problems [**EP97, NA99, GDMP03**].

In order to ease the development of discrete adjoint codes, various researchers have suggested using Automatic Differentiation (AD) [**Moh97, GGD08, CKH03**] to differentiate the spatial discretization scheme used in the flow solver. In this way, the error-prone hand linearization is avoided as this task can be performed automatically by the AD tool. Further, the adjoint system of linear equations can be assembled on-the-fly, without explicitly storing the entries of the flux Jacobian matrix. Since this matrix is sparse, reverse or forward modes of AD can be applied to the routines, which realize flux discretization schemes. Although being

computationally efficient, this approach still does not guarantee that the adjoint system is consistent with the underlying nonlinear solver, and therefore may lead to robustness problems as pointed out by Nielsen and Anderson [**NA02**]. Later following Giles' exact dual approach for linear systems [**GDM01, Gil02**], Nielsen et al. [**NLPD04**] presented a more refined solution scheme for the adjoint equation. They suggested using a duality preserving pseudo time-stepping scheme for the adjoint equations. Yet in a more recent approach, it has been suggested by various authors to differentiate the complete pseudo time-stepping solution scheme written as a fixed-point iteration using the reverse mode of AD [**SWGH08, GWMW07**]. In this way, the adjoint solution scheme itself is automatically implemented by the AD tool, and is fully consistent with the solution scheme that is used to solve the nonlinear state equation. Furthermore, the adjoint solver inherits the convergence behavior of the underlying flow solver. Today, using AD techniques it is possible to generate a fully consistent unsteady adjoint RANS (Reynolds-averaged Navier-Stokes) solver for highly complex configurations [**ONG12, NOG+11**].

The German Research Foundation (DFG) funded project "Automated Extension of Fixed Point PDE Solvers for Optimal Design with Bounded Retardation", which has been carried out in the Priority Program 1253 "Optimization with Partial Differential Equations", aimed at developing an efficient one-shot framework for various applications such as aerodynamic shape optimization and inverse problems in climate research. This thesis was initiated and elaborated in this project. The main objectives of the thesis are formulated as:

- To extend the one-shot method to meet the specific requirements of the aerodynamic shape optimization problems.
- To review the existing sensitivity evaluation methods and develop a framework, which enables the efficient evaluation of shape sensitivities for Euler, Navier-Stokes and RANS equations.
- To implement a consistent adjoint approach for the in-house compressible and incompressible flow solvers and build a shape optimization chain for the one-shot method.
- To quantify the retardation factor of the one-shot method, which is a measure of the slowdown of the overall optimization process compared to a single simulation, in terms of problem parameters.
- To assess the convergence behavior and optimization performance for different variants of the one-shot method using standard test cases of aerodynamic shape optimization.
- To compare the performance of the one-shot method with an established nested optimization algorithm.

An overview of the structure of the thesis is provided here to assist the orientation. In Chapter 2, a generic PDE optimization problem is introduced. Based on the optimality conditions of this problem, we discuss several adjoint procedures that can be used to compute the reduced gradient vector of the objective function at a fixed computational cost. One specific procedure, what we call as the "piggyback", is used to solve primal and adjoint equations simultaneously. Extending this method to incorporate also the design updates yields the one-shot method. The conditions to achieve the convergence of the one-shot method and suitable preconditioning for the design updates are discussed in detail. Further, we present results, which help to quantify the performance of the one-shot optimization. In

Chapter 3, first essential ingredients of aerodynamic shape optimization are introduced. Based on these ingredients, we present ways how shape optimization chains can be constructed. In Chapter 4, we introduce the Euler and Navier-Stoke equations and discuss numerical solution methods used to solve these equations approximately. We also shortly introduce several turbulence modeling approaches used in simulations of high Reynolds number flows. In Chapter 5, first a review of sensitivity evaluation methods is given. Then, we briefly introduce adjoint methods for aerodynamic design optimization. A special emphasis is given to the discrete adjoint method, which is the method of choice in the present work. Further, we discuss accuracy and robustness issues of the adjoint solvers that are crucial for the success of the shape optimization. Chapter 6 is completely reserved for the AD techniques, which are used to generate discrete adjoint solvers in the present work. We discuss the implementation issues and present advanced techniques to improve the performance of the differentiated codes. In Chapter 7, we present the numerical results obtained by the application of the one-shot method applied to three different scenarios of aerodynamic shape optimization, and make a performance comparison between the one-shot method and a nested quasi-Newton method to demonstrate the efficiency of the one-shot method. In this chapter, we also provide performance measurements of the adjoint codes, which are generated using the consistent discrete adjoint approach. Finally, in Chapter 8, we draw conclusions and suggest possible extensions of the method as the future outlook.

CHAPTER 2

Theoretical framework of the one-shot method

In this chapter, we introduce the theoretical background and the methodology of the one-shot method. Even though aerodynamic shape optimization is the main focus of the present work, theoretical considerations that are given in this chapter will remain to be valid for general optimization problems with partial differential equations (PDEs). Therefore, one-shot method that is presented in this work is not only restricted to the aerodynamic shape optimization problems and the same methodology can be applied to a broad class of design optimization problems with a PDE constraint. The key assumptions made are that the state PDE can be solved with a contractive fixed-point solver and a discrete realization of the solution scheme exists, i.e., the state PDE is solvable with a numerical method that is implemented as a computer software.

In order to generalize the method, we first introduce a generic optimization problem with a PDE constraint as the starting point for the theoretical derivations. In the generic problem, the objective function is taken as a scalar quantity, which is a function of design and state vectors. Note that, the state vector itself is also a function of the design and this functional dependency is given by the state equation. The state equation describes the physics involved in the problem. Therefore, the PDE constraint guarantees that the state solution that corresponds to a so-called optimal design is a physical solution, i.e., it obeys physical laws. In classical engineering optimization, the optimization problems are usually formulated without the state PDE constraint as it is implicitly assumed that the state solution in each optimization cycle is already a converged solution satisfying the state equation. However, in one-shot context, the optimality and the convergence of the state PDE are attained simultaneously so that this assumption cannot be implicitly made. Therefore, the state PDE is explicitly stated as an equality constraint to force the state solution to a physical one. For this reason, the convergence behavior of the state PDE is very important, and several assumptions concerning the convergence are made in order to ease the theoretical analysis. These assumptions are given in the following.

Historically, the one-shot method can be considered as the extension of the so-called "piggy-back" method, which suggests solving state and adjoint equations simultaneously. Therefore, we first introduce the piggy-back method. Then, the one-shot method is covered in detail. As far as the one-shot method is concerned, attaining convergence of the coupled iterations is by far harder than piggy-back iterations that are performed without any design change. Therefore, contractivity of the coupled iterations and the derivation of a suitable design space preconditioner to ensure the contractivity are the key issues in the design of one-shot methods. Therefore, in this chapter a special emphasis is given to the theoretical analysis concerning the convergence of the coupled iterations and the preconditioner.

Another important aspect in the one-shot method is the computational cost. It is desired that the computational cost of the one-shot method remains bounded and is independent of the size of the design vector. In general, the overall aim is to keep the computational cost of the one-shot method at a small multiple of the cost of solving the state PDE. Because of this reason, bounded retardation rate concept is introduced. By using the retardation rate, it is aimed to quantify the computational cost of the one-shot method compared to a single simulation. At the end of this chapter, the characterization of the bounded retardation rate is made by considering separable problems, in which mixed second order derivatives of the Lagrangian function with respect to design and state vectors are zero. It should be pointed that, for the real applications, the corresponding optimization problems are more likely to be non-separable by the nature of the governing PDEs. Due to the theoretical difficulties that appear in the analysis of non-separable problems, however, such an attempt has not been made yet.

2.1. The generic optimization problem

We consider the generic PDE optimization problem, which is given by

$$\min_{u} \quad f(y,u) \quad \text{such that} \quad c(y,u)=0, \tag{2.1.1}$$

where $y \in Y$ is the state vector and $u \in U$ is the design vector that is restricted to a closed convex subset A of the design space U. In the following, we assume that Y, U and $X = Y \times U$ are Hilbert spaces. The constraint $c : X \to Y$ denotes the state equation that must be fulfilled. In aerodynamic shape optimization, for example, state equations are typically the discretized Euler or Navier-Stokes equations. The state vector y in this case is then simply the flow solution (velocities, pressure, etc.) and the design vector u is a parameterization of the shape that is to be optimized with respect to a given objective function $f : X \to \mathbb{R}$.

The objective function f, which is given as a scalar function in the above formulation, may be an explicit function of the design u or it may depend implicitly on u via the state PDE $c(y,u) = 0$. The functional relationship between f and u depends strictly on the chosen parameterization of the shape, and therefore on the implementation of the design chain between the design and the objective function. In the present work, we assume that f is explicitly a function of both state and control, i.e., $f = f(y,u)$. In aerodynamic shape optimization, typical objective functions are lift and drag coefficients, which result from the integration of aerodynamic forces exerted on a body of interest by the circulating flow.

We assume that the state PDE $c(y,u) = 0$ can be solved by a fixed-point iteration of the form $y_{k+1} = G(y_k, u)$. Moreover, we also assume that the state equation c is always solvable for a given (y,u), i.e., $\frac{\partial c}{\partial y}(y,u)$ is nonsingular and invertible on $Y \times A$. Note that, this is a strict assumption that is hard to satisfy since it is not really possible to solve the state PDE for an arbitrary (y,u) due to difficulties introduced by numerical methods. In addition, we assume that the functions f, G are $C^{2,1}$ on the closed convex $Y \times A$ and the fixed-point iterator G is contractive with respect to some norm $\|\cdot\|$ with the contraction rate ρ such that for all $u \in U$ and $y, \tilde{y} \in Y$ the following relation holds:

$$\|G_y(y,u)\| = \|G_y^\top(y,u)\| \leq \rho < 1 \Rightarrow \|G(y,u) - G(\tilde{y},u)\| \leq \rho \|y - \tilde{y}\|. \tag{2.1.2}$$

Note that, in general the contraction rate ρ may vary continuously as a function of u and its value is not available a priori.

2.2. The Lagrangian and the optimality conditions

The Lagrangian associated with the generic PDE optimization problem (2.1.1) is given as

$$L(y, \bar{y}, u) = f(y, u) + (G(y, u) - y)^\top \bar{y}, \tag{2.2.1}$$

where $\bar{y}$ is the vector of Lagrange multipliers. For simplicity, we introduce the "shifted Lagrangian" N, defined as

$$N(y, \bar{y}, u) = f(y, u) + G(y, u)^\top \bar{y}. \tag{2.2.2}$$

The Lagrangian can be now rewritten as

$$L(y, \bar{y}, u) = N(y, \bar{y}, u) - y^\top \bar{y}, \tag{2.2.3}$$

where $y^\top \bar{y}$ is called as the "shift term".

According to the first order necessary optimality conditions, which are known as Karush-Kuhn-Tucker (KKT) conditions, the gradient of the Lagrangian function must be zero at the optimal point:

$$\nabla_{\overline{y}} L = 0,\ \nabla_y L = 0 \text{ and } \nabla_u L = 0. \tag{2.2.4}$$

Hence, an optimal point $(u^*, y^*, \bar{y}^*)$ must satisfy:

$$y^* = G(y^*, u^*) \tag{2.2.5}$$

$$\overline{y}^* = N_y(y^*, \overline{y}^*, u^*)^\top = f_y(y^*, u^*)^\top + G_y(y^*, u^*)^\top \overline{y}^* \tag{2.2.6}$$

$$0 = N_u(y^*, \overline{y}^*, u^*)^\top = f_u(y^*, u^*)^\top + G_u(y^*, u^*)^\top \overline{y}^*. \tag{2.2.7}$$

The first equation of the above system is the state equation in fixed-point form, which corresponds to the state constraint $c(y, u) = 0$. The state equation is alternatively called as the primal equation. Any vector (u, y) that satisfies the state equation is called to be primally feasible. The second equation is called as the adjoint equation, which has also a fixed-point similar to the state equation. If the adjoint equation is satisfied, then the vector $(u, y, \bar{y})$ is called to be dual or adjoint feasible. Finally, the last equation in the KKT system is called as the design equation. These three equations state the necessary but not sufficient conditions for optimality.

2.3. Adjoint procedures

Similar to a root finding problem, the solution of the design equation can be obtained by updating the design vector iteratively using the information given by the reduced gradient N_u. Further, a design space preconditioner is used to accelerate the convergence rate of the optimization. A design update step can be written as

$$u_{k+1} = u_k - B_k^{-1}\, N_u(y_k^*, \overline{y}_k^*, u_k)^\top, \tag{2.3.1}$$

where y_k^* and $\overline{y}_k^*$ are the converged state and adjoint vectors satisfying primal and adjoint equations for a design u_k and the matrix B_k is the preconditioner. Note that, in the above equation the adjoint vector $\overline{y}_k^*$ is required to compute the

reduced gradient, which is obtained by solving the adjoint equation. Concerning the solution strategy of the adjoint equation there are two methods. The classical solution method suggests that the adjoint equation is solved only after the solution of the primal equation is obtained with a certain accuracy. In this way, the adjoint solver uses the already converged solution y_k^*, obtained from the primal solver. This method is called as the two-phase [**Gri00**] or reverse accumulation [**Chr94**] method since primal and adjoint equations are solved sequentially, one phase after the other. In the discrete setting, the adjoint equation can be easily derived by considering the total derivative of the objective function f with respect to design u at the converged state solution y^*:

$$\frac{df(y^*,u)}{du} = \frac{\partial f(y^*,u)}{\partial u} + \frac{\partial f(y^*,u)}{\partial y^*}\frac{dy^*}{du}. \tag{2.3.2}$$

If we assume that the reduced gradients of the objective function $\partial f/\partial y$ and $\partial f/\partial u$ can be easily computed, the main difficulty here is to compute the dy^*/du. From the solution of the state equation, which is written in fixed-point form $y^* = G(y^*,u)$, we have

$$\frac{dy^*}{du} = \frac{\partial G(y^*,u)}{\partial u} + \frac{\partial G(y^*,u)}{\partial y^*}\frac{dy^*}{du}, \tag{2.3.3}$$

which gives

$$\frac{dy^*}{du} = \left(I - \frac{\partial G(y^*,u)}{\partial y^*}\right)^{-1}\frac{\partial G(y^*,u)}{\partial u}. \tag{2.3.4}$$

Multiplying both sides of the above equation with $\frac{\partial f(y^*,u)}{\partial y^*}$ we obtain

$$\frac{\partial f(y^*,u)}{\partial y^*}\frac{dy^*}{du} = \frac{\partial f(y^*,u)}{\partial y^*}\left(I - \frac{\partial G(y^*,u)}{\partial y^*}\right)^{-1}\frac{\partial G(y^*,u)}{\partial u}. \tag{2.3.5}$$

If we introduce the adjoint vector $\overline{y}^*$, which is given by

$$\overline{y}^* = \left(I - \frac{\partial G(y^*,u)}{\partial y^*}\right)^{-\top}\frac{\partial f(y^*,u)^\top}{\partial y^*}, \tag{2.3.6}$$

we obtain the adjoint equation, written in fixed-point form:

$$\overline{y}^* = \frac{\partial G(y^*,u)^\top}{\partial y}\overline{y}^* + \frac{\partial f(y^*,u)^\top}{\partial y^*}. \tag{2.3.7}$$

Note that the above equation is the same equation as the second equation in the KKT system, namely $\overline{y}^* = N_y(y^*,\overline{y}^*,u^*)^\top = f_y(y^*,u^*)^\top + G_y(y^*,u^*)^\top\overline{y}^*$. Furthermore, the adjoint vector $\overline{y}$ is the vector of Lagrange multipliers of the equality constraint.

The form of Eq. (2.3.7) suggests that one can find the solution $\overline{y}^*$ by updating the adjoint vector using the fixed-point iteration scheme:

$$\overline{y}_{k+1} = \frac{\partial G(y^*,u)^\top}{\partial y}\overline{y}_k + \frac{\partial f(y^*,u)^\top}{\partial y^*}, \tag{2.3.8}$$

starting from an initial solution $\overline{y}_0$. When the adjoint solution $\overline{y}^*$ is available, the reduced gradient df/du is simply computed by

$$\frac{df(y^*,u)^\top}{du} = \frac{\partial f(y^*,u)^\top}{\partial u} + \frac{\partial G(y^*,u)^\top}{\partial u}\overline{y}^*. \tag{2.3.9}$$

Note that the term df/du is the reduced gradient of the shifted Lagrangian at the point $(u, y^*, \overline{y}^*)$. The reverse accumulation approach, which is introduced above, is summarized in the following algorithm:

ALGORITHM 1. *Reverse accumulation algorithm*

initialize $y \leftarrow y_0$ *and* $\overline{y} \leftarrow \overline{y}_0$
for $k = 0, k \leq MAXIT$ ***do***
  do primal iteration $y_{k+1} = G(y_k, u)$
  if $\|y_{k+1} - y_k\| \leq \epsilon$ ***then***
    break
  end if
end for
set $y^* \leftarrow y_{k+1}$
compute objective function $f = f(y_k, u)$
compute $\overline{z} = \frac{\partial f(y^*,u)^\top}{\partial y}$
for $k = 0, k \leq MAXIT$ ***do***
  do adjoint iteration $\overline{y}_{k+1} = \frac{\partial G(y^*,u)^\top}{\partial y}\overline{y}_k + \overline{z}$
  if $\|\overline{y}_{k+1} - \overline{y}_k\| \leq \epsilon$ ***then***
    break
  end if
end for
$\overline{y}^* \leftarrow \overline{y}_{k+1}$
compute the reduced gradient $\frac{df(y^*,u)}{du}^\top = \frac{\partial f(y^*,u)}{\partial u}^\top + \frac{\partial G(y^*,u)^\top}{\partial u}\overline{y}^*$

Note that, the matrix $G_y^\top$ has the same eigenvalues as the Jacobian of the state equation G_y, since G_y is a square matrix. On the other hand, the Jacobian of the adjoint equation is $G_y^\top$. As a result, state and the adjoint equations have the same spectral properties and both adjoint and state solvers share the same convergence properties.

Alternative to the reverse accumulation method, the second approach, which is introduced in [**GF02**], suggests that the state and the adjoint equations can be solved simultaneously. This approach is named as piggy-backing by the authors. Referring again to the KKT system, one can write coupled primal and dual steps with a fixed design u as

$$[y_{k+1}, \bar{y}_{k+1}] = [N_{\bar{y}}^\top(y_k, \bar{y}_k, u), N_y^\top(y_k, \bar{y}_k, u)] = [G(y_k, u), N_y^\top(y_k, \bar{y}_k, u)]. \tag{2.3.10}$$

The piggy-back method is summarized in the following algorithm:

ALGORITHM 2. *Piggy-back algorithm*

initialize $y \leftarrow y_0$ *and* $\overline{y} \leftarrow \overline{y}_0$
for $k = 0, k \leq MAXIT$ ***do***
  do primal iteration $y_{k+1} = G(y_k, u)$
  do adjoint iteration $\overline{y}_{k+1} = f_y^\top(y_k, u) + G_y(y_k, u)^\top \overline{y}_k$
  if (($\|y_{k+1} - y_k\| \leq \epsilon$) ***and*** ($\|\overline{y}_{k+1} - \overline{y}_k\| \leq \epsilon$)) ***then***
    break
  end if
end for
set $y^* \leftarrow y_{k+1}$
set $\overline{y}^* \leftarrow \overline{y}_{k+1}$
compute the objective function $f = f(y_k, u)$
compute the reduced gradient $\frac{df(y^*,u)}{du} = \frac{\partial f(y^*,u)}{\partial u}^\top + \frac{\partial G(y^*,u)^\top}{\partial u}\overline{y}^*$

Note that, the Jacobian of the coupled primal-dual iteration in Eq. (2.3.10) takes the form

$$J_k \equiv \frac{\partial(y_{k+1}, \bar{y}_{k+1})}{\partial(y_k, \bar{y}_k)} = \begin{bmatrix} G_y(y_k, u) & 0 \\ N_{yy}(y_k, \bar{y}_k, u) & G_y^\top(y_k, u) \end{bmatrix}. \tag{2.3.11}$$

Since the Jacobian J_k is a lower triangular block matrix, its eigenvalues are simply the eigenvalues of its diagonal block, namely G_y and $G_y^\top$. As mentioned previously the eigenvalues of G_y and $G_y^\top$ are same, thus every eigenvalue of G_y is a double eigenvalue of the Jacobian of coupled primal-dual iteration J_k. Therefore, when primal and adjoint fixed-point iterations are performed simultaneously in a piggy-back manner, they have the same asymptotical convergence rate. The only difference is that the adjoints $\bar{y}$ lag behind the the primals y, since the adjoints are heading for a moving target, namely solution of the state PDE. The convergence of the reduced gradient N_u is also similar, therefore primal feasibility is reached faster than dual feasibility. This phenomena is called as the dual retardation. As a measure of dual retardation, the number of iterations required for the primal convergence divided by the number of iterations required for the dual convergence can be used. This quotient is referred to as the dual retardation ratio. Hamdi et al. [**HG09**] found an upper bound on the dual retardation ratio and stated that dual iterates tend to catch up with primal iterates asymptotically. This upper bound is introduced in the following theorem:

THEOREM 1. *Suppose that for a constant design vector* u*, the functions* f *and* G *are Lipschitz continuously differentiable with respect to state* y *in the close neighborhood of the fixed-point solution* y^**. Furthermore, we assume that the primal fixed-point iterates converge to a solution* y^* *with a contraction rate* ρ^* *at the limit such that*

$$\lim_{k\to\infty} \frac{\|\Delta y_k\|}{\|\Delta y_{k-1}\|} = \lim_{k\to\infty} \frac{\|y_{k+1} - y_k\|}{\|y_k - y_{k-1}\|} = \rho^* \equiv \|G_y(y^*, u)\|. \tag{2.3.12}$$

If for any given tolerance $\epsilon > 0$ *and given positive real numbers* α *and* β*, we define the smallest pair of integers as* $l_p^\epsilon, l_d^\epsilon$ *such that*

$$\sqrt{\alpha}\left\|\Delta y_{l_p^\epsilon}\right\| \leq \epsilon \text{ and } \sqrt{\beta}\left\|\Delta \bar{y}_{l_d^\epsilon}\right\| \leq \epsilon, \tag{2.3.13}$$

then we have

$$\limsup_{\epsilon \to \infty} \frac{l_d^\epsilon}{l_p^\epsilon} \leq 1. \tag{2.3.14}$$

Proof: complete proof is given in [**HG09**].

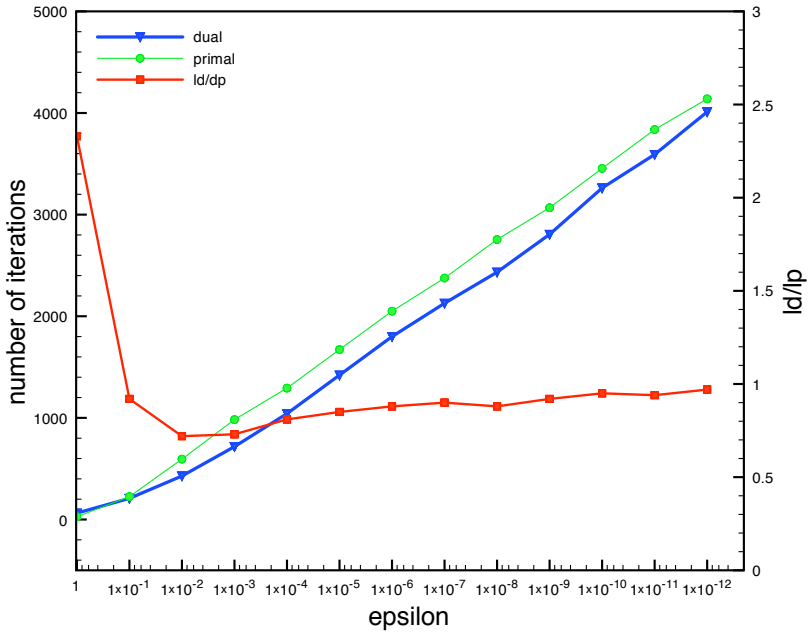

Figure 1. *Number of dual and primal iterations required to achieve different residual levels and the quotient* $l_d^\epsilon / l_p^\epsilon$

In Fig. 1, behavior of the dual retardation ratio for different tolerance values of ϵ can be observed for a transonic Euler case with drag coefficient as the objective function. Note that, as the tolerance ϵ gets smaller, the dual retardation ratio approaches to one since the number of primal and dual iterations to achieve an accuracy specified by ϵ become almost identical.

2.4. Transition from simulation to optimization

As far as the solution of the optimization problem is concerned, the KKT point can be found iteratively by updating the design using the reduced gradient N_u. The reduced gradient can be computed by employing any one of the adjoint procedures that are mentioned previously. We call this standard approach as the classical "nested" approach. The nested approach, which is by far the most frequently used method in industrial nonlinear optimization packages, is summarized in the following algorithm.

ALGORITHM 3. *The nested gradient search algorithm*

initialize $y \leftarrow y_0$, $u \leftarrow u_0$ *and* $\overline{y} \leftarrow \overline{y}_0$
for $i = 0, i \leq i_{max}$ ***do***
  for $k = 0, k \leq k_{max}$ ***do***
    do one primal iteration $y_{k+1} = G(y_k, u_i)$
    if ($(\|y_{k+1} - y_k\| \leq \epsilon)$ ***then***
      break
    end if
  end for
  set $y^* \leftarrow y_{k+1}$
  compute objective function $f = f(y^*, u_i)$
  compute partial gradient $f_y(y^*, u_i)^\top$
  for $k = 0, k \leq k_{max}$ ***do***
    do one adjoint iteration $\overline{y}_{k+1} = f_y^\top(y^*, u_i) + G_y^\top(y^*, u_i)\overline{y}_k$
    if ($(\|\overline{y}_{k+1} - \overline{y}_k\| \leq \epsilon)$ ***then***
      break
    end if
  end for
  set $\overline{y}^* \leftarrow \overline{y}_{k+1}$
  compute reduced gradient $\frac{df(y^*, \overline{y}^*, u_i)^\top}{du} = \frac{\partial f(y^*, u_i)^\top}{\partial u} + \frac{\partial G(y^*, u_i)^\top}{\partial u}\overline{y}^*$
  update design vector $u_{i+1} = u_i - \lambda_i B_i^{-1} \frac{df(y^*, \overline{y}^*, u_i)^\top}{du}$
  if $(\|\frac{df}{du}\| \leq \epsilon)$ ***then***
    STOP
  end if
end for

In the above algorithm, the nested approach is illustrated using the reverse accumulation procedure to evaluate the adjoint vector. Alternatively, piggy-back method can be also used. The reduced gradient in both cases is exactly the same and the choice of adjoint procedure does not influence the optimization history. Also note that, in each design update a positive definite preconditioner B_i with a step size λ_i is used. According to the selection of the preconditioner, one obtains a different optimization method. The most established method among the gradient search based methods is the steepest descent method with $B_i = I$. The steepest descent method is a very robust algorithm that is easy to implement, but it may suffer from slow convergence behavior. The performance of the steepest descent algorithm depends crucially on the scaling of the design parameters. An optimization problem is said to be poorly scaled if changes to u in certain directions produce much larger variations in the value of f than changes in the other directions. If the problem is well scaled, the steepest descent algorithm, combined with a line-searches, usually produce very satisfactory results. If the optimization problem is poorly scaled, however, steepest descent algorithm shows an oscillatory behavior, and therefore convergence may be very slow. A possible remedy is to apply the Newton's method,

in which the preconditioner is taken as the reduced Hessian $\Delta_{uu} f$. The Newton's method is not affected by the scaling problem since it makes use of curvature information, which are given by the second order derivatives. It has been shown that the Newton method converges locally q-quadratically [**Kel99**], but computation of the reduced Hessian may be computationally very expensive. Therefore, Newton's method may not be always feasible for PDE optimization problems due to its high computational cost. A good comprise between the steepest descent and Newton methods using the quasi-Newton type methods, which do not require second order derivative information. By measuring the changes in gradients in each optimization cycle, these methods construct a model of the objective function that is good enough to produce super-linear convergence. The most prominent methods of this class are the BFGS and DFP methods [**NW99**], which are employed in many state-of-the-art optimization packages.

Another issue is the choice of step size λ_i for the design updates. Here several strategies can be used. If a constant step size is to be taken, it should be small enough to ensure a descent in each iteration. If a too large step size is chosen, the gradient information may not be valid and the corresponding design update may rather lead to an ascent. Furthermore, with a too large step size, the design vector may get some inadmissible values, for which the primal solver may diverge. A more refined strategy is to apply a line search procedure and let the value of the step size be adapted dynamically in each iteration. Finding the optimal value for the step size may not be feasible, instead, one may apply an inexact line-search based on sufficient decrease concept given by the Wolfe conditions. For the problems, in which gradient computations are expensive, simpler strategies like successive step size reduction or cubic polynomial fitting may be also efficiently used.

As stated previously, nested methods need fully convergent primal and adjoint solution to update the design vector. In general, for PDE optimization problems, solving primal and adjoint equations at such accuracy is computationally very expensive if not impossible. Therefore, similar to the piggy-backing strategy, in which primal and adjoint vectors are updated simultaneously, the design update can be also done in a simultaneous manner without achieving full convergence. This strategy is the underlying principle of the one-shot method. As a consequence:

In order to converge to a KKT point, rather than first fully converging the state equation using

$$y_{k+1} = G(y_k, u), \quad k = 1, .., N \rightarrow \text{primal feasibility at } y_* \tag{2.4.1}$$

and then fully converging the adjoint equation applying

$$\overline{y}_{k+1} = f_y^\top(y^*, u_i) + G_y^\top(y^*, u_i)\overline{y}_k, \quad k = 1, .., N \rightarrow \text{dual feasibility at } \overline{y}_* \tag{2.4.2}$$

before finally performing a design update in "outer" optimization loop

$$u_{i+1} = u_i - \lambda_i B_i^{-1} \frac{df}{du} \rightarrow \text{optimality at } u_*, \tag{2.4.3}$$

one can use the following coupled iteration to reach the KKT point:

$$\begin{bmatrix} y_{k+1} \\ \bar{y}_{k+1} \\ u_{k+1} \end{bmatrix} = \begin{bmatrix} G(y_k, u_k) \\ N_y(y_k, \bar{y}_k, u_k)^\top \\ u_k - B_k^{-1} N_u(y_k, \bar{y}_k, u_k)^\top \end{bmatrix}, \quad k = 1, ..\bar{N}, \bar{N} > N. \tag{2.4.4}$$

Similar to a nested method, the matrix B_k is a suitable design space preconditioner to achieve the contractivity of the coupled iteration. Note that, in the

coupled iteration given in Eq. (2.4.4) all variables are updated simultaneously similar to a Jacobi iteration that is used for the iterative solution of systems of linear equations. However, in practice it is easier to implement the coupled iteration by always using the most recent variable values in Gauss-Seidel fashion. In this way, it is not required to hold the values of $u, y, \bar{y}$ in memory after updating them. The Seidel variant of the coupled iterations are given by

$$\begin{bmatrix} y_{k+1} \\ \bar{y}_{k+1} \\ u_{k+1} \end{bmatrix} = \begin{bmatrix} G(y_k, u_k) \\ N_y(y_{k+1}, \bar{y}_k, u_k)^\top \\ u_k - B_k^{-1} N_u(y_{k+1}, \bar{y}_{k+1}, u_k)^\top \end{bmatrix} \quad k = 1, ..\bar{N}, \bar{N} > N. \tag{2.4.5}$$

Finally, the Seidel variant of one-shot method is summarized in the following algorithm:

ALGORITHM 4. *Seidel variant of the one-shot algorithm*

initialize y_0, $\overline{y}_0$, u_0 *and tolerance* ϵ
for $k = 0, k \leq k_{max}$ ***do***
  do primal iteration $y_{k+1} = G(y_k, u_k)$
  do adjoint iteration $\overline{y}_{k+1} = N_y^\top(y_{k+1}, u_k, \overline{y}_k) = f_y^\top(y_{k+1}, u_k) + G_y^\top(y_{k+1}, u_k)\overline{y}_k$
  do design update $u_{k+1} = u_k - B_k^{-1} N_u(y_{k+1}, \bar{y}_{k+1}, u_k)^\top$
  if ($(\|y_{k+1} - y_k\| \leq \epsilon)$ ***and*** $(\|\overline{y}_{k+1} - \overline{y}_k\| \leq \epsilon)$) ***and*** $\|(N_u(y_k, \bar{y}_k, u_k)\| \leq \epsilon)$
  then
    STOP
  end if
end for

In Fig. 2, the fundamental difference between a nested method and the one-shot method is graphically illustrated. Note that, the blue curve $c(u, y) = 0$ shows the set of points in the variable space, which are primally feasible. The red curve $\overline{y} = N_y(u, y, \overline{y})$, on the other hand, shows the points that are dually feasible the design u. Both these equations are necessarily solved in an iterative fashion so that a nested method takes necessarily a zig-zag path to reach the optimal design u^*, starting from the initial design u_0. On the contrary, instead of initializing primal and adjoint vectors after each design update, one-shot method uses the previous values of these vectors. In this way, one-shot method makes use of the data obtained from the previous iterations. After performing an iteration of the primal and adjoint fixed-point schemes, the reduced gradient is immediately used to update the design. Hence, a short-cut, shown by the green curve in the figure, is taken to reach the optimal design u^*. If strictly one primal and adjoint iteration is executed in each coupled iteration as suggested by the Eq. (2.4.4), the method is called as "single-step one-shot" method [**Gri06**]. Alternatively, if more steps are taken, we obtain a slightly different variant of the one-shot method, which is referred to as the "multi-step one-shot" method [**BLG14**].

In the multi-step one-shot method, more than one iteration of primal and adjoint fixed-point schemes are performed after each design update. The main motivation behind the multi-step variant is the fact that each coupled iteration with a design update is computationally more expensive than a piggy-back iteration. This

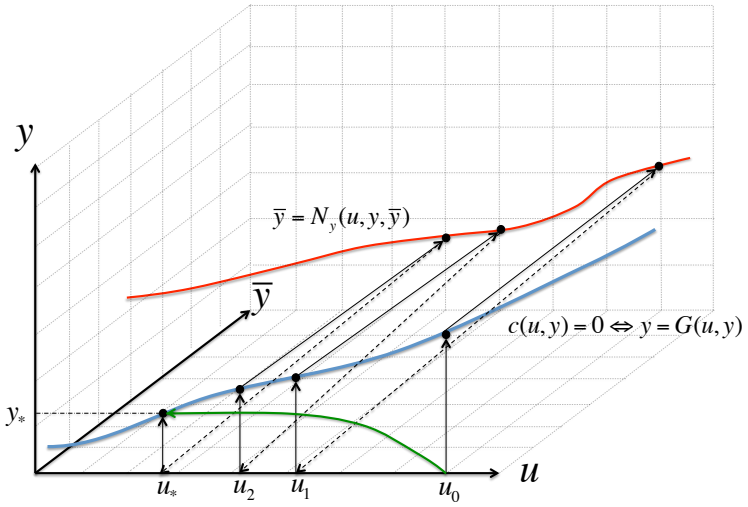

Figure 2. *Graphical illustration of the difference between the nested approach and the one-shot method.*

is due to the extra overhead related with the design update, which may be significant in may cases. In aerodynamic shape optimization, for example, each design update is associated with computationally expensive operations like mesh deformation or re-meshing. The multi-step variant performs less design update steps compared to single-step one-shot, therefore it spends less computational time for mesh deformation and metric term computations. Loosely speaking, the multi-step variant can be considered as a comprise between the single-step one-shot method and the nested approach. We give below the coupled iteration for the multi-step variant with same number of iterations for the primal and adjoint equations:

$$\begin{bmatrix} y_{k+1} \\ \bar{y}_{k+1} \\ u_{k+1} \end{bmatrix} = \begin{bmatrix} G^s(y_k, u_k) \\ N_y^s(y_k, \bar{y}_k, u_k)^\top \\ u_k - B_k^{-1} N_u(y_k, \bar{y}_k, u_k)^\top \end{bmatrix}. \tag{2.4.6}$$

In the above coupled iteration, $G^s(y_k, u_k)$ and $N_y^s(y_k, \bar{y}_k, u_k)^\top$ denote s successive iterations of the primal and adjoint fixed-point schemes. Note that, for large value of the parameter s, the multi-step variant is ultimately same as the nested method, in which primal and adjoint feasibilities are strictly satisfied after each design update. Estimating the optimal value for s a priori is difficult and a topic of ongoing research. Yet in a recent work, Bosse et al. [**BLG14**] derived a lower bound for s that is enough to ensure contractivity, which is shortly mentioned in the following.

2.5. Contractivity of the single-step one-shot method

In one-shot method, the reduced gradient N_u is evaluated at a point, which does not have primal and dual feasibilities. Only if the one-shot method is contractive, the coupled iterations converge to a unique solution. As we noticed before, the

design space preconditioner B_k is introduced in the coupled iterations for this purpose. Therefore, the choice of B_k is crucial. In the following, we first focus on the conditions that are necessary to ensure the contractivity of the coupled iterations. Then, we turn our attention to the selection of a suitable preconditioner.

The asymptotic rate of convergence of the coupled iteration, given by Eq. (2.4.4), to a KKT point $(y^*, \bar{y}^*, u^*)$ is determined by its Jacobian, which is given by the following 3×3 block matrix:

(2.5.1)
$$J^* = \frac{\partial(y_{k+1}, \bar{y}_{k+1}, u_{k+1})}{\partial(y_k, \bar{y}_k, u_k)}\Big|_{(y^*, \bar{y}^*, u^*)} \begin{bmatrix} G_y & 0 & G_u \\ N_{yy} & G_y^\top & N_{yu} \\ -B_*^{-1}N_{uy} & -B_*^{-1}G_u^\top & I - B_*^{-1}N_{uu} \end{bmatrix}.$$

Note that, the contractivity assumption made for the primal fixed-point scheme ensures that the coupled iterations converge to a KKT point $(y^*, \bar{y}^*, u^*)$ provided that the preconditioner B_k is sufficiently large. If the B_k is chosen too large, however, the design update Δu is very small and the one-shot method performs very slowly. On the other hand, if B_k is not large enough, convergence of the coupled iterations cannot be not achieved. Similar to the primal fixed-point solver, the local convergence of the coupled iterations is ensured by the condition $\rho(J^*) < 1$, where $\rho(J^*)$ is the spectral radius of Jacobian J^*. In [**Gri06**], an attempt was made to find a relation to bound eigenvalues of J_* and the authors introduced the following proposition, which characterize the eigenvalues of the Jacobian.

PROPOSITION 1. *For any nonsingular symmetric preconditioner B_*, the eigenvalues of the matrix J^* are either eigenvalues of G_y or they are the solution of the nonlinear eigenvalue problem, which is given by*

$$\det[(\lambda - 1)B_* + H(\lambda)] = 0. \tag{2.5.2}$$

In the above equation, the matrix H is defined as

$$H(\lambda) = Z(\lambda)^\top N_{xx} Z(\lambda) \quad \textit{with} \quad Z(\lambda) = \begin{bmatrix} (\lambda I - G_y)^{-1} G_u \\ I \end{bmatrix}, \tag{2.5.3}$$

where N_{xx} is the full Hessian:

$$N_{xx} = \begin{bmatrix} N_{yy} & N_{yu} \\ N_{uy} & N_{uu} \end{bmatrix}. \tag{2.5.4}$$

Proof. *The eigenvalues of Jacobian J^* solve the characteristic equation:*

$$det \begin{bmatrix} G_y - I\lambda & 0 & G_u \\ N_{yy} & G_y^\top - I\lambda & N_{yu} \\ -B_*^{-1}N_{uy} & -B_*^{-1}G_u^\top & (I - \lambda) - B_*^{-1}N_{uu} \end{bmatrix} = 0. \tag{2.5.5}$$

If the last row of J^* is multiplied with $-B_*$, the roots of the characteristic equation does not change. Hence we obtain

$$det \begin{bmatrix} G_y - I\lambda & 0 & G_u \\ N_{yy} & G_y^\top - I\lambda & N_{yu} \\ N_{uy} & G_u^\top & (\lambda - I)B_* + N_{uu} \end{bmatrix} = 0. \tag{2.5.6}$$

Note that, unless λ is an eigenvalue of G_y, the inverse of the leading 2×2 block in the above equation is given by

$$A^{-1} = \begin{bmatrix} (G_y - I\lambda)^{-1} & 0 \\ -(G_y - I\lambda)^{-\top} N_{yy} (G_y - I\lambda)^{-1} & (G_y - I\lambda)^{-\top} \end{bmatrix}. \tag{2.5.7}$$

The Schur complement of A can be then computed as

$$J^*/A \equiv (\lambda - I)B_* + N_{uu} - \begin{bmatrix} N_{uy} & G_u^\top \end{bmatrix} A^{-1} \begin{bmatrix} G_u \\ N_{yu} \end{bmatrix}. \tag{2.5.8}$$

Using the above result, the eigenvalue problem in (2.5.6) can be reduced to

$$det(A) \cdot det(J^*/A) = 0, \tag{2.5.9}$$

which leads to the Eq. (2.5.2). □

Using the above result, in the limit situation with $\lambda = 1$, we have from the Eq. (2.5.2):

$$det[H(1)] = 0, \tag{2.5.10}$$

where the matrix $H(1)$ is given by

$$H(1) = Z(1)^\top N_{xx} Z(1). \tag{2.5.11}$$

The above equation leads to

$$H(1) = [(I - G_y)^{-1} G_u \;\; I] N_{xx} \begin{bmatrix} (I - G_y)^{-1} G_u \\ I \end{bmatrix}. \tag{2.5.12}$$

Note that, the rows of $Z(1)$ span the tangent space of the manifold $G(y, u) = y$. Furthermore, $H(1)$, which must be positive semi-definite at a local minimizer according to the second order necessary condition, represents the projection of the full Hessian N_{xx} onto that m-dimensional subspace.

For the ideal case, when the Jacobian of the primal fixed point iteration G_y has n distinct real eigenvalues and the optimization problem is strongly convex such that the full Hessian N_{xx} is positive definite on the whole space, the Jacobian J^* has n double eigenvalues in the interval $(-1, 1)$. These eigenvalues come from G_y and $G_y^\top$ and belong to the primal and adjoint fixed-point schemes. For a positive definite preconditioner B_*, we can expect that two eigenvalues of J^* are close to each one of them. Out of these eigenvalues, only two eigenvalues that are beyond the largest and smallest eigenvalue of G_y are critical since the coupled iterations loose contractivity when their values go beyond 1 or -1. For this ideal situation, Griewank [**Gri06**] constituted a necessary condition on B_*, which ensures the contractivity of the coupled system. Accordingly, $H_* \succ 0$ and $det(H(1)) \neq 0$ is a necessary condition, provided that $H(1) \succeq 0$. This condition guarantees that J^* has no positive real eigenvalues greater than one. For the negative real eigenvalues, the condition $B_* \succeq \frac{1}{2} H(-1)$ must be fulfilled, such that all eigenvalues are less than -1. Using these conditions, the preconditioner can be constructed in such a way that all the real eigenvalues remain bounded between -1 and 1. There is, however, no guarantee that J^* does not have any complex eigenvalues. Therefore, these conditions on B_* are necessary but not sufficient.

2.5.1. Penalty formulation using augmented Lagrangian. Deriving conditions directly on the preconditioner B_* to ensure the contractivity of the coupled iteration has been proven to be difficult. Yet as an alternative approach, Hamdi et al. [**HG09**] proposed rather to look for a descent of a augmented Lagrangian type of merit function, which is given by

$$L^a(y,\bar{y},u) = \frac{\alpha}{2}\|G(y,u)-y\|^2 + \frac{\beta}{2}\|N_y(y,\bar{y},u)^\top - \bar{y}\|^2 + N(y,\bar{y},u) - \bar{y}^\top y, \tag{2.5.13}$$

where α and β are positive real numbers that can be considered as the penalty weights for the primal residual $\|G(y,u)-y\|^2$ and the dual residual $\|N_y(y,\bar{y},u)^\top - \bar{y}\|^2$ respectively. In this way, large steps in design update are penalized to enforce the contractivity. In the following, relations to bound the penalty weights α and β are provided.

2.5.2. Correspondence condition. In the same work, Hamdi et al. [**HG09**] derived a condition on the penalty weights α and β, which ensures that a local minimizer of the optimization problem in (2.1.1) is also a local minimizer of L^a. Further, when this condition is satisfied the augmented Lagrangian L^a is proved to be an exact penalty function. Note that, the full gradient of L^a is given as

$$\begin{bmatrix}\Delta_y L^a\\ \Delta_{\bar{y}} L^a\\ \Delta_u L^a\end{bmatrix} = \begin{bmatrix} -\alpha\Delta G_y^\top(G(y,u)-y) + (I+\beta N_{yy})(N_y(y,\bar{y},u)^\top - \bar{y})\\ G(y,u) - y - \beta\Delta G_y(N_y(y,\bar{y},u)^\top - \bar{y})\\ \alpha G_u^\top(G(y,u)-y) + \beta N_{yu}^\top(N_y(y,\bar{y},u)^\top - \bar{y}) + N_u(y,\bar{y},u)^\top\end{bmatrix}, \tag{2.5.14}$$

where $\Delta G_y = I - G_y$. The full gradient can be expressed in terms of a matrix-vector product as

$$\nabla L^a(y,\bar{y},u) = -M s(y,\bar{y},u), \text{ with } M = \begin{bmatrix}\alpha\Delta G_y^\top & -I-\beta N_{yy} & 0\\ -I & \beta\Delta G_y & 0\\ -\alpha G_u^\top & \beta N_{yu}^\top & B\end{bmatrix}. \tag{2.5.15}$$

The step increment vector s, used in the above equation, is defined as

$$s(y,\bar{y},u) = \begin{bmatrix} G(y,u)-y\\ N_y(y,\bar{y},u)-\bar{y}\\ -B^{-1}N_u(y,\bar{y},u)^\top\end{bmatrix}. \tag{2.5.16}$$

Note that, if the matrix M in Eq. (2.5.15) is nonsingular, we have a one-to-one correspondence between the stationary points of L^a and the roots of s. The non-singularity of the matrix M is ensured by the

$$\det[\alpha\beta\Delta G_y^T\Delta G_y - I - \beta N_{yy}] \neq 0. \tag{2.5.17}$$

Furthermore, for a non-zero vector $v \in \mathbb{R}^n$, we observe that

$$v^\top(\alpha\beta\Delta G_y^\top\Delta G_y - I - \beta N_{yy})v = \alpha\beta\, r^\top r - v^\top v - \beta v^\top N_{yy} v, \text{ with } r = \Delta G v. \tag{2.5.18}$$

From the above identity, we get

$$v^\top(\alpha\beta\Delta G_y^\top\Delta G_y - I - \beta N_{yy})v = \alpha\beta\,\|r\|^2 - \|v\|^2 - \beta v^\top N_{yy} v. \tag{2.5.19}$$

Therefore, the matrix M is non-singular if

$$\alpha\beta\,\|\Delta r\|^2 > \|v\|^2 + \beta v^\top N_{yy} v \tag{2.5.20}$$

is satisfied. In addition, using the Cauchy-Schwarz inequality we obtain

$$v^\top N_{yy} v \leq \|v\| \, \|N_{yy} v\| \leq \|v\|^2 \, \|N_{yy}\| \,. \tag{2.5.21}$$

In conclusion, according to inequalities in (2.5.20) and (2.5.21) it is sufficient to ensure that

$$\alpha\beta \, \|r\|^2 > (1 + \beta \, \|N_{yy}\|) \, \|v\|^2 \,. \tag{2.5.22}$$

If the contractivity assumption (Eq. (2.1.2)) is taken into account, the above inequality is implied by the condition

$$\alpha\beta(1-\rho)^2 > 1 + \beta \, \|N_{yy}\| \,, \tag{2.5.23}$$

which is called as the correspondence condition.

Moreover, at a stationary point the reduced Hessian $\nabla_{uu} L^a$ is given as

(2.5.24)
$$\begin{bmatrix} \alpha \Delta G_y^\top \Delta G_y + (I + \beta N_{yy}) N_{yy} & -(I + \beta N_{yy}) \Delta G_y^\top & -\alpha \Delta G_y^\top \Delta G_y + (I + \beta N_{yy}) N_{yu} \\ -\Delta G_y (1 + \beta N_{yy}) & \beta \Delta G_y \Delta G_y^\top & G_u - \beta \Delta G_y N_{yu} \\ -\alpha \Delta G_u^\top \Delta G_y + N_{uy}(I + \beta N_{yy}) & -\beta N_{uy} \Delta G_y^\top + G_u^\top & \alpha G_u^\top G_u + \beta N_{uy} N_{yu} + N_{uu} \end{bmatrix}.$$

One can show that the diagonalization of the $\nabla_{uu} L^a$ gives:

$$diag[\alpha \Delta G_y^\top \Delta G_y - N_{yy} - I/\beta, \beta \Delta G_y \Delta G_y^\top, H(1)], \tag{2.5.25}$$

where $H(1)$ is the reduced projected Hessian. Since the matrix $\Delta G_y \Delta G_y^\top$ is also positive definite, it follows that the Hessian of augmented Lagrangian is positive definite if the condition

$$\alpha\beta \Delta G_y^\top \Delta G_y > I + \beta \, \|N_{yy}\| \tag{2.5.26}$$

is satisfied.

2.5.3. Descent direction condition. The correspondence condition states that all stationary points of L^a are minima. Yet another condition on α and β, which is introduced also by Hamdi et al. [**HG09**], ensures that the step increment vector s defined in Eq. (2.5.16) yields descent on L^a at all its non-stationary points. Note that the step increment of the coupled iterations yields descent on augmented Lagrangian if

$$-s^\top \cdot \nabla L^a > 0. \tag{2.5.27}$$

From Eq. (2.5.15), we get the condition

$$s^\top M s > 0. \tag{2.5.28}$$

If the matrix M is symmetrized, the above condition is equivalent to

$$s^\top M s = s^\top \frac{M + M^\top}{2} s > 0. \tag{2.5.29}$$

Finally, we obtain

$$-s^\top \cdot \nabla L^a = s^\top \begin{bmatrix} \alpha \Delta \bar{G}_y & -I - \frac{\beta}{2} N_{yy} & -\frac{\alpha}{2} G_u \\ -I - \frac{\beta}{2} N_{yy} & \beta \Delta \bar{G}_y & -\frac{\beta}{2} N_{yu} \\ -\frac{\alpha}{2} G_u^\top & -\frac{\beta}{2} N_{yu}^\top & B \end{bmatrix} s > 0, \tag{2.5.30}$$

where $\Delta\bar{G}_y = \frac{1}{2}(\Delta G_y + \Delta G_y^\top)$. The symmetric matrix in the above equation can be written as

$$\tilde{M} = \frac{M + M^\top}{2} = \begin{bmatrix} A & C \\ C^\top & B \end{bmatrix}, \tag{2.5.31}$$

where the block matrices A and C are given as

$$A = \begin{bmatrix} \alpha\Delta\bar{G}_y & -I - \frac{\beta}{2}N_{yy} \\ -I - \frac{\beta}{2}N_{yy} & \beta\Delta\bar{G}_y \end{bmatrix} \text{ and } C = \begin{bmatrix} -\frac{\alpha}{2}G_u \\ -\frac{\beta}{2}N_{yu} \end{bmatrix}. \tag{2.5.32}$$

On the other hand, if the matrix $\tilde{M}$ is diagonalized by block elimination, we get

$$\begin{bmatrix} I & 0 \\ -C^\top A^{-\top} & I \end{bmatrix} \begin{bmatrix} A & C \\ C^\top & B \end{bmatrix} \begin{bmatrix} I & -A^{-1}C \\ 0 & I \end{bmatrix} = \operatorname{diag}(A, B - C^\top A^{-1} C). \tag{2.5.33}$$

Note that the descent condition, which is given in (2.5.27), is fulfilled if positive definiteness of $\tilde{M}$ is satisfied. This can be ensured by choosing proper values for α and β such that matrix A remains positive definite. In addition to this, the preconditioner B should be chosen large enough such that the matrix $B - C^\top A^{-1}C$ also remains positive definite. The positive definiteness of A is satisfied if

$$\alpha\beta\Delta\bar{G}_y > (I + \frac{\beta}{2}N_{yy})(\Delta\bar{G}_y^{-1})(I + \frac{\beta}{2}N_{yy}), \tag{2.5.34}$$

which is implied by the condition

$$\sqrt{\alpha\beta}(1 - \rho) > 1 + \frac{\beta}{2}\|N_{yy}\|. \tag{2.5.35}$$

In order to show that inequality in (2.5.34) is implied by (2.5.35), we first observe that that according to the Perturbation Lemma [**Wer92**], the inequality

$$\left\|\Delta\bar{G}_y^{-1}\right\| \leq \frac{1}{1 - \rho} \tag{2.5.36}$$

is satisfied. Then, for an arbitrary vector $v \in \mathbb{R}^n$ the inequalities in (2.5.35) and (2.5.36) imply

$$\alpha\beta(1 - \rho)\|v\|^2 > \left\|(I + \frac{\beta}{2}N_{yy})v\right\|^2 \frac{1}{1 - \rho} > \left\|(I + \frac{\beta}{2}N_{yy})v\right\|^2 \left\|\Delta\bar{G}_y^{-1}\right\|. \tag{2.5.37}$$

Taking the Cauchy-Schwarz inequality into account, the above inequality implies

$$\alpha\beta(1 - \rho)\|v\|^2 > v^\top(I + \frac{\beta}{2}N_{yy})\Delta\bar{G}_y^{-1}(I + \frac{\beta}{2}N_{yy})v. \tag{2.5.38}$$

Moreover, since $v^\top \frac{G_y + G_y^\top}{2} v \leq \rho\|v\|^2$, we obtain

$$v^\top\Delta\bar{G}_y^{-1}v = \|v\|^2 - v^\top\frac{G_y + G_y^\top}{2}v \geq (1 - \rho)\|v\|^2. \tag{2.5.39}$$

Finally, from (2.5.38) and (2.5.39), we receive

$$\alpha\beta v^\top\Delta\bar{G}_y v > v^\top(I + \frac{\beta}{2}N_{yy})\Delta\bar{G}_y^{-1}(I + \frac{\beta}{2}N_{yy})v. \tag{2.5.40}$$

This condition is referred as the descent direction condition, which is a stronger condition than the correspondence condition. As a result, any values for α and β

that satisfy the descent condition also satisfy the correspondence condition. In Fig. 3, the admissible region, where the descent condition is satisfied for different values of α and β, is illustrated for a scenario of a very slowly converging primal solver with $\rho = 0.99$ and $\|N_{yy}\| \approx 1$.

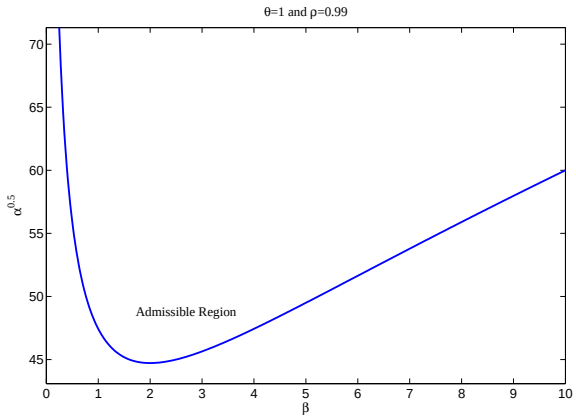

Figure 3. *The admissible region for penalty weights α and β.*

2.5.4. Particular choice of α and β. If the descent condition is used to specify the values of α and β, the limiting values can be found from the following optimization problem:

$$\min_{\beta} \sqrt{\alpha} \equiv \frac{1 + \frac{\theta}{2}\beta}{(1-\rho)\sqrt{\beta}}, \tag{2.5.41}$$

which leads to the values:

$$\alpha = \frac{2\|N_{yy}\|}{(1-\rho)^2} \text{ and } \beta = \frac{2}{\|N_{yy}\|}. \tag{2.5.42}$$

2.5.5. Estimation of ρ and $\|N_{yy}\|$. In order to calculate the weighting coefficients α and β, we need to estimate the contractivity constant ρ that is a measure of the convergence rate of the primal solver. Note that from the contractivity assumption we have:

(2.5.43)
$$\|G(y_k, u) - G(y_{k-1}, u)\| = \|y_{k+1} - y_k\| \leq \rho \|y_k - y_{k-1}\| \Rightarrow \|\Delta y_k\| \leq \rho \|\Delta y_{k-1}\|$$

for any value of the index k. Therefore, the value of ρ can be updated in each iteration starting from an initial value ρ_0 by using the formula

$$\rho_{k+1} = \max\left(\frac{\|\Delta y_k\|}{\|\Delta y_{k-1}\|}, \tau\rho_k\right), \tag{2.5.44}$$

where $\tau \in (0,1)$ is a constant. On the other hand, to estimate the value of $\theta = \|N_{yy}\|$, we consider the approximation derived from Eq. (2.4.4):

(2.5.45)
$$\Delta y_{k+1} = G(y_{k+1}, u) - y_{k+1} \text{ and } G(y_{k+1}, u) \approx G(y_k, u) + G_y(y_k, u)(y_{k+1} - y_k).$$

From the above relations it follows that $\Delta y_{k+1} \approx G_y(y_k, u)\Delta y_k$. Similarly for $\Delta \bar{y}_{k+1}$ we have:

(2.5.46)
$$\begin{aligned} \Delta \bar{y}_{k+1} &= N_y(y_{k+1}, \bar{y}_{k+1}, u) - \bar{y}_{k+1} \text{ and} \\ N_y(y_{k+1}, \bar{y}_{k+1}, u) &\approx N_y(y_k, \bar{y}_k, u) + N_{yy}(y_k, \bar{y}_k, u)(y_{k+1} - y_k) + N_{y\bar{y}}(y_k, \bar{y}_k, u)(\bar{y}_{k+1} - \bar{y}_k). \end{aligned}$$

Since $N_{y\bar{y}} = G_y$, we obtain

$$\Delta \bar{y}_{k+1} \approx N_y(y_k, \bar{y}_k, u)\Delta y_k + G_y(y_k, u)\Delta \bar{y}_k. \tag{2.5.47}$$

Writing the above approximations for Δy_{k+1} and $\Delta \bar{y}_{k+1}$ together in matrix form we obtain

$$\begin{bmatrix} \Delta y_{k+1} \\ \Delta \bar{y}_{k+1} \end{bmatrix} \approx \begin{bmatrix} G_y(y_k, u_k) & 0 \\ N_{yy}(y_k, \bar{y}_k, u_k) & G_y(y_k, u_k)^\top \end{bmatrix} \begin{bmatrix} \Delta y_k \\ \Delta \bar{y}_k \end{bmatrix}. \tag{2.5.48}$$

Therefore, we obtain

$$\begin{bmatrix} \Delta \bar{y}_k \\ \Delta y_k \end{bmatrix}^\top \begin{bmatrix} \Delta y_{k+1} \\ \Delta \bar{y}_{k+1} \end{bmatrix} \approx -(\Delta y_k)^\top N_{yy}(y_k, \bar{y}_k, u_k)\Delta y_k, \tag{2.5.49}$$

which leads to the following approximation

$$(\Delta y_k)^\top N_{yy}(y_k, \bar{y}_k, u_k)\Delta y_k \approx (\Delta y_k)^\top \Delta \bar{y}_{k+1} - (\Delta \bar{y}_k)^\top \Delta y_{k+1}. \tag{2.5.50}$$

In conclusion, the above approximation can be used to estimate the value of $\|N_{yy}\|$ in each iteration as

$$\theta_{k+1} = \max\left(\frac{(\Delta y_k)^\top \Delta \bar{y}_{k+1} - (\Delta \bar{y}_k)^\top \Delta y_{k+1}}{\|y_k\|^2}, \gamma\theta_k\right), \tag{2.5.51}$$

where γ is a constant between zero and one.

2.5.6. Preconditioning for design updates. Hamdi et al. [**HG11**] have shown that any preconditioner B that satisfies

$$B \succeq B_0 \equiv \frac{1}{\sigma}(\alpha G_u^\top G_u + \beta N_{yu}^\top N_{yu}) \quad \text{with} \quad \sigma = 1 - \rho - \frac{(1 + \frac{\|N_{yy}\|}{2}\beta)}{\alpha\beta(1-\rho)} \tag{2.5.52}$$

must yield a descent on the augmented Lagrangian L^a. Note that the matrix B_0 can be related to the Hessian of the augmented Lagrangian with respect to design. Indeed, the solution of the optimization problem

$$\min_{\Delta u} L^a(y + \Delta y, \overline{y} + \Delta \overline{y}, u + \Delta u) \tag{2.5.53}$$

can be used to identify B from $\Delta u = -B^{-1}N_u$. Furthermore, if we consider a quadratic approximation of L^a, the above optimization problem can be written as

$$\min_{\Delta u} s^T \nabla L^a(y, \overline{y}, u) + \frac{1}{2} s^T \nabla^2 L^a(y, \overline{y}, u)s, \tag{2.5.54}$$

where s is the increment vector given in Eq. (2.5.16). The above minimization problem can be transformed into the equivalent minimization problem

$$\min_{\Delta u} \varphi(\Delta u), \tag{2.5.55}$$

where φ is the a quadratic function given as

(2.5.56)

$$\begin{aligned} \varphi(\Delta u) &= \Delta u^T(\nabla_u L^a + \nabla_{uy} L^a \Delta y + \nabla_{u\overline{y}} L^a \Delta \overline{y}) + \frac{1}{2}\Delta u^T \nabla_{uu} L^a \Delta u \\ &\approx \Delta u^T \nabla_u L^a(y + \Delta y, \overline{y} + \Delta \overline{y}, u) + \frac{1}{2}\Delta u^T \nabla_{uu} L^a \Delta u. \end{aligned}$$

Whenever the Hessian $\nabla_{uu} L^a$ is positive definite, the solution of the minimization problem (2.5.55) is defined by

$$\Delta u = -\nabla_{uu} L^a(y, \overline{y}, u) \nabla_u L^a(y + \Delta y, \overline{y} + \Delta \overline{y}, u). \tag{2.5.57}$$

Thus, it follows that preconditioner can be identified as $B = \nabla_{uu} L^a$. Note that the exact reduced Hessian $\nabla_{uu} L^a$ is given by

$$\nabla_{uu} L^a = \alpha G_{uu}^\top (G - y) + \beta N_{yuu}(N_y - \overline{y}) + \alpha G_u^\top G_u + \beta N_{yu}^\top N_{yu} + N_{uu}. \tag{2.5.58}$$

Hence, if $(y, \overline{y}, u)$ is almost primally and dually feasible ($G \approx y$ and $N_y \approx \overline{y}$) we obtain

$$B = \nabla_{uu} L^a \approx \alpha G_u^\top G_u + \beta N_{yu}^\top N_{yu} + N_{uu}. \tag{2.5.59}$$

If we assume that the matrix N_{uu} is positive definite, the preconditioner B that is suggested in the above equation satisfies the condition in (2.5.52) and therefore s yields descent provided that the chosen values for α and β fulfill the descent condition.

2.5.7. BFGS update as an approximation to B. The evaluation of preconditioner B as given in Eq. (2.5.59) involves derivative matrices, which may lead to expensive computations for large-scale problems. Alternatively, one can use an approximate by using BGFS updates. Since $B \approx \nabla_{uu} L^a$, we have

$$B\Delta u = \nabla_{uu} L^a(y, \overline{y}, u)\Delta u \approx \nabla_u L^a(y, \overline{y}, u + \Delta u) - \nabla_u L^a(y, \overline{y}, u)\ . \tag{2.5.60}$$

Thus, one may employ the above approximation as a secant equation into the update of $H \equiv B^{-1}$. Therefore, we may impose the secant condition

(2.5.61)

$$H_{k+1} R_k = \Delta u_k, \text{ where } R_k := \nabla_u L^a(y_k, \overline{y}_k, u_k + \Delta u_k) - \nabla_u L^a(y_k, \overline{y}_k, u_k)\ .$$

The above secant equation has a solution only if the so-called curvature condition

$$R_k^\top \Delta u_k > 0 \tag{2.5.62}$$

is satisfied. Therefore, curvature condition is checked in all iterations and BFGS update is performed whenever it is satisfied. Otherwise, one can defer the update by setting $B_k = B_{k-1}$. It should be also noted that, as far as the BFGS update is concerned, there is no need to first update the matrix B and then invert it. Instead, one can directly update B^{-1} by using the Sherman-Morrison formula [**NW99**].

2.6. Contractivity of the multi-step one-shot method

In case of multi-step one-shot method, the Jacobian of the coupled iterations, which can be derived by differentiating $y_{k+1}, \overline{y}_{k+1}, u_{k+1}$ with respect to $y_k, \overline{y}_k, u_k$, is given as

(2.6.1)
$$J_*^s = \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ N_{yu} & N_{yy} & G_y^\top \end{bmatrix}^s \begin{bmatrix} I & 0 & 0 \\ G_u & G_y & 0 \\ 0 & 0 & I \end{bmatrix}^s \begin{bmatrix} I - B^{-1}N_{uu} & -B^{-1}N_{uy} & -B^{-1}G_u^\top \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix}.$$

Bosse et al. [**BLG14**] showed that unless they are not in the spectrum of G_y^s all eigenvalues of the J^* satisfy the equation:

$$\det[M(\lambda)] = 0 \tag{2.6.2}$$

with

$$M(\lambda) = (1-\lambda)B - H_s(\lambda). \tag{2.6.3}$$

In the above equation, the matrix H_s is the parameterized projected Hessian of the multi-step one-shot scheme using s iterations. It is given by

(2.6.4)
$$H_s(\lambda) = Z_s(\lambda)^\top \begin{bmatrix} N_{yy} & N_{yu} \\ N_{uy} & N_{uu} \end{bmatrix} Z_s(\lambda) \quad \text{with} \quad Z_s(\lambda) = \begin{bmatrix} (\lambda I - G_y^s/\lambda)(\lambda I - G_y)^{-1} G_u \\ I \end{bmatrix}.$$

Note that if the number of iterations s goes to infinity, the parameterized projected Hessian approaches to $H_* = H_s(1)$ for all eigenvalues that are not in the spectrum of G_y^s. When s is sufficiently large and the preconditioner B is close to H_*, all the eigenvalues of J^* that do not belong to G_y^s must render the matrix $M(\lambda) \approx \lambda B$ singular and therefore close to zero. In other words, rapid convergence rate is achieved when the primal solver with multiple steps is close to a Newton iteration and B is close to the projected Hessian. This describes an ideal situation that is hard to achieve in reality but serves as a bound for the convergence rate of the multi-step one-shot scheme.

The contractivity of the multi-step one-shot scheme is assured if the eigenvalues of the J_*^s can be bounded. This can be achieved by adjusting the iteration number s. A lower bound for s to assure the contractivity is found by the same authors as

$$s \geq s_0 = \log_{(1/\rho_0)} \left[1 + 2(\sqrt{d(1-\gamma)/\nu + c^2} + c)\nu/(1-\gamma) \right]. \tag{2.6.5}$$

If number of iterations s is selected to be grater than s_0, it is ensured that the spectral radius of J_s^* is less than one. Therefore, multi-step one-shot scheme becomes contractive. In the above equation, γ and ν are the scalar quality measures for the preconditioner B. Further, the variables c and d are the sensitivity measures of the adjoint equation. The variable ρ_0 denotes the contraction rate of the fixed-point iterator G with one iteration only ($s = 1$). The adjoint sensitivity measures are given by

$$d = \|N_{yy}\|\|\tilde{Z}\|^2 \quad \text{and} \quad c = \|N_{yu} + N_{yy}\tilde{Z}\|\|\tilde{Z}\|^2. \tag{2.6.6}$$

Note that, the matrix $\tilde{Z}$ in the above equations represents the corresponding null-space of the reparameterized design space $\tilde{u} = T^{-1}u$, and T being the transformation matrix for which $\tilde{H}_* = T^\top H_* T = I$ holds. In other words, the design parameters can be reparameterized in such a way that the new projected Hessian becomes the identity matrix. In this case, we have $\tilde{Z} = (\lambda I - G_y^s/\lambda)(\lambda I - G_y)^{-1} G_u T$ and $\tilde{B} = T^\top B T$. The variables d and c measure the sensitivity of the adjoint equation with respect to state and design respectively.

In order to measure the quality of the preconditioner, the measures γ and ν are used. These are given by

$$\gamma = \|I - \tilde{B}\| \quad \text{and} \quad \mu = \|\tilde{B}^{-1}\|. \tag{2.6.7}$$

In the ideal case, the preconditioner B is the projected Hessian such that $\tilde{B} = \tilde{H}_* = I$. In this case, we have $\gamma = 0$ and $\nu = 1$.

2.7. Characterization of the bounded retardation rate

In general, it is aimed to construct the one-shot method such that the coupled iterations convergence within a reasonable number of iterations compared to the number of iterations required to obtain primal convergence, i.e., steady-state solution of the state PDE. The ultimate goal is of course to perform the complete optimization has a run-time, which is only a small multiple of a single simulation. In other words, it is desired to minimize the slowdown factor of the one-shot method measured in run-time, what we call as the retardation ratio R:

$$R = \frac{\text{run-time of an optimization}}{\text{run-time of a single simulation}}. \tag{2.7.1}$$

Note that the actual retardation ratio based on run-time measurements in the above formula highly depends on the performance of the chosen sensitivity evaluation method and initial conditions. Therefore, making retardation ratio comparisons between different optimizations based on run-time measurements can be misleading since adjoint evaluation methods and initial conditions cannot be generalized. Instead, one can use the estimate given by the number of coupled iterations required for the optimization divided by the number of iterations required for the pure simulation. A good indicator for this quotient is the ratio, which can be measured with the spectral radii of the Jacobians of the primal fixed-point solver G_y and the coupled iterations J_*:

$$Retardation factor: \quad r = \frac{(1 - \rho(G_y))}{(1 - \rho(J_*))} \approx \frac{\ln(\rho(G_y))}{\ln(\rho(J_*))}. \tag{2.7.2}$$

The retardation factor is only an idealized measure of the slowdown factor of the one-shot method. In the above definition, not only the initial conditions are neglected but also the fact that performing a coupled iteration is computationally more expensive as that of a primal fixed-point iteration.

In this section, first the key problem characteristics that determine the retardation factor of the one-shot method are introduced. To simplify the analysis involved to characterize r, first a model scenario, in which the state equation is solved by the Newton method is introduced. For this ideal case, the adjoint equation is separable, i.e., the $L_{uy} = 0$, and the projected Hessian can be evaluated exactly. Then, the minimal retardation factors for Jacobi and multi-grid on a $1D$ test problem are

analyzed. The theory presented in this section follow essentially those in [**GHO13**] and all theoretical and numerical results obtained from the different test cases are reprinted here with permission from the authors.

2.7.1. Key problem parameters. We consider again the generic optimization problem given in (2.1.1). The most important quantities, which are used in characterizing the generic optimization problem are:

- The Jacobian of the primal fixed-point iteration G_y and the Hilbert norm with respect to which the fixed-point scheme is contractive such that $\|G_y\| \leq \rho$. Note that, in real applications involving stiff PDEs, the contraction factor ρ may be very close to 1 and we speak of a slowly converging fixed-point scheme in this case. On the other extreme, we have the fast converging Newton iteration $G(y,u) = y - c_y(y,u)^{-1}c(y,u)$. This situation will be considered as the limiting scenario for all methods that have a rapid convergence rate such as full multi-grid scheme.
- The Jacobian of the fixed-point iteration with respect to design G_u, which represents the sensitivity of the state equation with respect to the changes in the design vector. In the following, we assume that G_u has full rank and the design space can be reparameterized, at least theoretically, in such a way that $G_u^\top G_u = I$.
- The partial Hessian L_{yy} or N_{yy}, which represents both the nonlinearity of the fixed-point solver and the sensitivity of the adjoint with respect to state. Therefore, the norm $p \equiv \|L_{yy}\|$ can be considered as a measure of the coupling between the two.
- The mixed derivative L_{yu}, which represents the sensitivity of adjoint equation with respect to design. If we have $L_{yu} = 0$, then the optimization problem is called as a separable problem. Although, the design optimization problems are generally non-separable, studying separable problems serves as an important tool to understand and quantify the retardation rate. Generally, the ratio $q \equiv \max_v \|L_{yu}v\|/\|G_u v\|$ can be used as measure of separability. Note that, if G_u is orthogonal, we have then simply $q \equiv \|L_{yu}\|$.
- The positive definiteness condition $H(1) \succ 0$ in the neighborhood of the solution $((y,u) \approx (y^*,u^*))$ represents the second order sufficiency condition, a mild condition, which is completely independent of the chosen fixed-point iterator G.
- The global positive definiteness condition $\nabla^2_{y,u}L \succ 0$ for $(y,u) \approx (y^*,u^*)$ on the full Hessian of the Lagrange functional implies $H(\lambda) \succ 0$ for all λ. The matrices $H(\lambda)$ are for $\lambda \neq 1$ highly dependent on G and the condition seems unreasonably strong especially if $\dim(y) \gg \dim(u)$.
- The partial Hessian with respect to design L_{uu} need not be positive definite for second order sufficiency condition $H(1) \succ 0$. However, the positive definiteness is often guaranteed by a regularization of the design vector u. When L_{uu} is gradually scaled up to infinity, we are likely to have $u^* \to 0$ and $r \to 0$. Conversely, one has $\|u^*\| \to \infty$ and $r \to \infty$ as L_{uu} becomes small and $H(1)$ almost singular.

2.7.2. Preconditioner for the separable case. As introduced previously, separable problems are of great interest to characterize the bounded retardation

rate. In the following, we introduce a suitable design preconditioner B for these problems:

PROPOSITION 2. *Preconditioner for the general separable case*

Let the matrix P be defined as $P(\lambda) = (\lambda - 1)B + H(\lambda)$, then for separable problems, in which the mixed derivatives are zero, the preconditioner

$$B \equiv \alpha G_u G_u^\top + L_{uu} \;\; \textit{with} \;\; \alpha = \|L_{yy}\|/(1 - \|G_y\|)^2 \tag{2.7.3}$$

ensures that the matrix $P(\lambda)$ can not be singular for $\lambda \leq -1$ or $\lambda = 1$.

Proof. If Eq. (2.5.3) is taken into account, we get for all $v \in \mathbb{C}^n$ and $\lambda \in \mathbb{C}$ such that $|\lambda| \geq 1$,

$$\begin{aligned} \overline{v}^\top H(\lambda)v - \overline{v}^\top L_{uu}v &= \overline{v}^\top G_u(\lambda I - G_y)^{-1}L_{yy}(\lambda I - G_y)^{-1}G_u v \\ &\leq \|L_{yy}\| \, \|(\lambda I - G_y)^{-1}G_u v\|^2 \leq \|L_{yy}\|\|G_u v\|^2/(1-\rho)^2, \end{aligned} \tag{2.7.4}$$

where $\rho = \|G_y\|$. Then, the preconditioner B as given in Eq. (2.7.3) ensures

$$\overline{v}^\top H(\lambda)v \leq \overline{v}^\top B v, \quad \text{for all } \; v \in \mathbb{C}^n \; \text{ and } \; \lambda \in \mathbb{C} \text{ with } \; |\lambda| \geq 1. \tag{2.7.5}$$

Therefore, using Eq. (2.5.2), for all real λ such that $\lambda \leq -1$ we get,

$$P(\lambda) = (\lambda - 1)B + H(\lambda) \preceq \lambda B \prec 0, \tag{2.7.6}$$

which implies that $P(\lambda)$ cannot be singular for $\lambda \leq -1$. Therefore, λ cannot be an eigenvalue of J_*, and since the second order sufficiency ensures that $P(1) = H(1) \succ 0$, $\lambda = 1$ cannot be an eigenvalue of J_*. $\square$

As stated before, the efforts to derive a preconditioner B such that all eigenvalues of J_* are inside the unit ball have not been successful. If B is conservatively taken very large, this condition is satisfied but the resulting convergence rate of the optimization may be slow. Note that for non-separable problems, the preconditioner B contains also the term $\beta L_{uy}L_{yu}$, where β is a suitable parameter to penalize rapid changes in the design parameters that strongly influence the adjoint equation. Similarly, the term $\alpha G_u G_u^\top$ is meant to be penalize rapid changes in the design that have a strong effect on the state equation. Both these penalty terms are quantities, which are relative to the convergence rate of the primal fixed-point scheme ρ. The convergence rate is also a measure of the ability of the solver to recover primal feasibility. In the following, first the use of reduced Hessian $H(1)$ is considered.

2.7.3. The Newton scenario for separable adjoints. In this section, we consider the limit case namely the Newton scheme used for the state equation. For this case, the primal fixed-point scheme for the state PDE $c(y, u) = 0$ can be written as $G(y, u) = y - c_y(y, u)^{-1}c(y, u)$. Moreover, it is also assumed that the optimization problem is separable such that $L_{yu} = 0$. This limiting case serves as a model for all fast converging solvers since similar convergence behavior can be expected when the primal fixed-point iterator represents an inner iteration like several multi-grid cycles that solves the state equation to a good accuracy before the design parameters are updated. From the separability assumption it implies

that $H(-1) = H(1)$ and the eigenvalues of the Jacobian J_*, unless they belong to the spectrum of G_y, are characterized by singularity of the matrix

$$P(\lambda) = (\lambda-1)B + G_u^\top L_{yy} G_u/\lambda^2 + L_{uu} = (\lambda-1)B + G_u^\top L_{yy} G_u(1/\lambda^2 - 1) + H(1). \tag{2.7.7}$$

Note that, the above equation is simply a special case of Eq. (2.5.2). Under the second order sufficiency condition $H(1) \succ 0$, the matrix $P(\lambda)$ can now be transformed into

$$\tilde{P}(\lambda) = (\lambda-1)\tilde{B} + (1/\lambda^2 - 1)\Gamma + I, \tag{2.7.8}$$

where $\Gamma = diag(\gamma_i)_{i=1}^n$ denotes the diagonal matrix that results from the diagonalization of $G_u^\top L_{yy} G_u$ with respect to $H(1)$. Since any positive definite matrix can be selected as the preconditioner B, we take it as $B = H(1)/\eta$, i.e., the projected Hessian scaled by the reciprocal of the step size η. In this case, we obtain $\tilde{B} = I/\eta$ that leads to the diagonal matrix

$$\tilde{P}(\lambda) = \big[(\lambda-1)/\eta + 1\big] I + (1/\lambda^2 - 1)\Gamma. \tag{2.7.9}$$

The matrix $\tilde{P}$ in the above equation is singular if one of its diagonal elements vanishes. Therefore, we obtain the set of rational equations given by

$$\frac{(\lambda-1)/\eta + 1}{1 - 1/\lambda^2} = \gamma_i \in I\!R, \quad i = 1 \ldots n. \tag{2.7.10}$$

The following proposition investigates the situation when full-steps are taken, i.e., the step size η is 1.

PROPOSITION 3. *Full step convergence:*

For the full step size $\eta = 1$, the maximal modulus ρ_* of any solution to Eq. (2.7.10) is less than 1 if and only if $\gamma = \|\Gamma\| < 1/\sqrt{2}$. In this case, we have $\rho_* < \sqrt[3]{2\gamma}$.

Proof. Taking the reciprocal of Eq. (2.7.10) with $\eta = 1$, we get

$$\frac{1}{\gamma_i} = \frac{1 - 1/\lambda^2}{\lambda}, \tag{2.7.11}$$

which gives for $\hat{\lambda} = 1/\lambda$ the cubic equation $1/\gamma_i = \hat{\lambda} - \hat{\lambda}^3$. Note that, this cubic equation has more than one real root if $\Delta = (4 - 27/\gamma_i^2) > 0$. This happens only if $|\gamma_i| \geq \sqrt{6.75} \approx 2.6$. In that case, one or two of the three roots lie in the interval $(-1, 1)$ such that $\hat{\lambda} = 1/\lambda$ is larger than 1 in modulus and $\rho > 1$. On the other hand, if $\hat{\lambda} = (\cos\varphi + i \sin\varphi)/\rho$ is a complex root with $\sin\varphi \neq 0$, one can obtain $|\lambda| = 1/|\hat{\lambda}| = \rho = 1 + 2\cos(2\varphi)$. Hence, ρ can be smaller than one only if φ lies in the interval given by $\left(\frac{\pi}{4}, \frac{3\pi}{4}\right) \cup \left(\frac{5\pi}{4}, \frac{7\pi}{4}\right)$. In fact, for $\varphi = \pm\frac{\pi}{4}$, we have exactly $\hat{\lambda} = \frac{1}{\sqrt{2}}(1 \pm i) = \frac{1}{\sqrt{2}}(1 \pm i)^{-1} = \lambda^{-1}$ with

$$\frac{\lambda^3}{\lambda^2 - 1} = \frac{(-1 \mp i)\dfrac{1}{\sqrt{2}}}{-1 \mp i} = \frac{1}{\sqrt{2}} = \frac{-(-\lambda)^3}{(-\lambda)^2 - 1}. \tag{2.7.12}$$

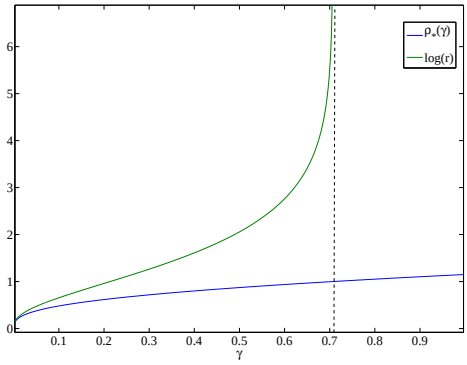

Figure 4. *Dependence of the convergence rate $\rho_*(\gamma)$ on the cross-term size (reprinted from* [**GHO13**] *with permission).*

Thus, the condition $\gamma < 1/\sqrt{2}$ must be fulfilled to ensure the convergence. The final assertion is obtained from

$$\frac{|\lambda|^3}{2} \leq \frac{|\lambda|^3}{1+|\lambda|^2} \leq \left|\frac{\lambda^3}{\lambda^2-1}\right| = |\gamma_i| \leq \gamma. \ \square \tag{2.7.13}$$

According to this proposition, the optimization problem must be regular enough such that γ is less than $1/\sqrt{2}$. The actual retardation factor in this case is given by the relation $r = 1/(1-\rho_*(\gamma))$, which is plotted together with ρ_* in Fig. 4. From the figure, it can be observed that only for rather small values of γ fast convergence is achieved. When the generalized eigenvalues γ_i lie outside $\left(-1/\sqrt{2}, 1/\sqrt{2}\right)$ but are bounded above by 1, convergence can be ensured by using a step multiplier of size $1/(\gamma+1)$. The next proposition examines the dependency of the convergence rate with respect to γ.

PROPOSITION 4. *Convergence with step size control:*
If the preconditioner is selected as $B = H(1)/(1+\gamma)$ with $\gamma = \|\Gamma\|$ and $L_{uu} \succ 0$, then the spectral radius ρ_* is always contained in the interval $\left[\gamma/(\gamma+1), 1\right)$. Moreover, ρ_* is given by the relation

$$\rho_* = \sqrt[3]{\frac{\gamma}{1+\gamma}} \approx 1 - \frac{1}{3\gamma} \tag{2.7.14}$$

Proof. Substituting $\eta = 1+\gamma$ into Eq. (2.7.10), we obtain set of equations:

$$Q_i(\lambda) = \lambda + \frac{\gamma_i}{(1+\gamma)\lambda^2} - \frac{\gamma_i+\gamma}{1+\gamma} = 0, \qquad i = 1, .., n. \tag{2.7.15}$$

If we add this set $Q_0(\lambda)$ with $\gamma_0 \equiv -\gamma$ if $max\{|\gamma_i|\}$ is attained for $\gamma_i > 0$, the values of γ_i can be ordered as $-\gamma = \gamma_0 \leq \gamma_1 \leq \gamma_i \leq ... \leq \gamma_n \leq \gamma$. We have then for all $i \geq 0$,

$$Q_i(-1) = -1 - \frac{\gamma}{1+\gamma} = \frac{-(1+2\gamma)}{1+\gamma} < \ 0 \ < \ Q_i(1) = 1 - \frac{\gamma}{1+\gamma} = \frac{1}{1+\gamma}. \tag{2.7.16}$$

We can denote by $P_i(\lambda) = \lambda^2 Q_i(\lambda)$ the corresponding cubic polynomial. If $\gamma_i = 0$, there exists a double root at $\lambda = 0$ and a nontrivial root at $\lambda_i = \gamma/(1+\gamma) \in (0,1)$.

For all $\gamma_i \neq 0$, we find that $Q_i(0) \to \infty$ if $\gamma_i > 0$ and $Q_i(0) \to -\infty$ if $\gamma_i < 0$. Therefore, by using the mean value theorem and the observation regarding $Q_i(\pm 1)$, there must be real roots

$$\lambda_i \in (-1, 0), \text{ if } \gamma_i > 0 \quad \text{and} \quad \lambda_i \in (0, 1), \text{ if } \gamma_i < 0. \tag{2.7.17}$$

In particular, we have $Q_0(\lambda_0) = \lambda_0 - \gamma/(1+\gamma)\lambda_0^2$ and thus, $\lambda_0 = \sqrt[3]{\gamma/(1+\gamma)}$.

Furthermore, if $\gamma_i > 0$ then we have

$$Q_i\left(\frac{-\gamma_i}{1+\gamma}\right) = \frac{-(2\gamma_i+\gamma)}{1+\gamma} + \frac{1+\gamma}{\gamma_i} = \frac{1+\gamma^2+2\gamma-2\gamma_i^2-\gamma_i\gamma}{(1+\gamma)\gamma_i} \geq \frac{1+2\gamma(1-\gamma)}{(1+\gamma)\gamma_i} > 0. \tag{2.7.18}$$

On the other hand, we have

$$Q_i(-\lambda_0) = -\lambda_0 + \frac{\gamma_i}{(1+\gamma)\lambda_0^2} - \frac{\gamma_i+\gamma}{1+\gamma} = \frac{\gamma_i-\gamma}{(1+\gamma)\lambda_0^2} - \frac{\gamma_i+\gamma}{1+\gamma} \leq 0. \tag{2.7.19}$$

Hence, we conclude that

$$\lambda_i \in [-\lambda_0, -\gamma_i/(1+\gamma)), \qquad \text{if } \gamma_i > 0 \tag{2.7.20}$$

Similarly, for $\gamma_i < 0$ we derive

$$Q_i(\frac{\gamma}{1+\gamma}) = \frac{\gamma_i(1+\gamma)}{\gamma^2} - \frac{\gamma_i}{1+\gamma} = \frac{\gamma_i(1+2\gamma)}{\gamma^2(1+\gamma)} < 0. \tag{2.7.21}$$

and

$$\begin{aligned} Q_i(\lambda_0) &= \lambda_0 + \frac{\gamma_i}{(1+\gamma)\lambda_0^2} - \frac{\gamma_i+\gamma}{1+\gamma} \\ &= \frac{\gamma+\gamma_i}{(1+\gamma)\lambda_0^2} - \frac{\gamma_i+\gamma}{1+\gamma} = \frac{\gamma_i+\gamma}{1+\gamma}\left(\frac{1}{\lambda_0^2}-1\right) \geq 0. \end{aligned} \tag{2.7.22}$$

if $\lambda_0 \in (0,1)$ is used for the last inequality. Therefore, we obtain

$$\lambda_i \in (\gamma/(1+\gamma), \lambda_0], \qquad \text{if } \gamma_i < 0. \tag{2.7.23}$$

As a result, the modulus $|\lambda_i|$ of the real roots λ_i is bounded above by $\lambda_0 = \sqrt[3]{\gamma/(1+\gamma)}$, which motivates the conjecture. However, each cubic polynomial $P_i(\lambda)$ has another pair of roots $\lambda^{\pm}$ which must satisfy

$$|\lambda_i^+||\lambda_i^-||\lambda_i| = |P_i(0)| = |\gamma_i|/(1+\gamma) \leq \gamma/(1+\gamma) \tag{2.7.24}$$

and hence using the common lower bound $|\gamma_i|/(1+\gamma)$ on $|\lambda_i|$ we find that $\min(|\lambda_i^-||\lambda_i^+|) < 1$. Finally, it can be observed from the sign conditions that if one of the two roots is real, the other root must be also real and both roots must be smaller than 1 in size. The same condition applies if the roots is a complex conjugate pair so that also $|\lambda_i^-| = |\lambda_i^+| < 1$. $\square$

Even when the above conjecture is valid, the convergence rate $\sqrt[3]{\gamma/(1+\gamma)}$ is not satisfactory unless $\gamma \approx 0$, which indicates that L_{uu} must be very large. If $H(1) = G_u \top L_{yy} G_u + L_{uu}$ is only just positive definite with the negative curvature of the first term being just balanced by the second, then $\gamma = -\lambda_1$ can be arbitrarily large and the same holds for the retardation factor

$$r = 1/(1-\rho_*) = 1/(1-\sqrt[3]{\gamma/(1+\gamma)}) \approx \gamma/3. \tag{2.7.25}$$

This means that the effort required to solve the state equation with a good accuracy at each inner loop of the optimization does not pay off, unless the problem

is strongly regularized. We conclude that, the apparent natural optimization step $-H(1)^{-1}L_u$ is too large and lead to divergence unless the cross term γ is quite small. One can remedy the situation by cutting the step size back by a factor of order $1/(1+\gamma)$ but the resulting retardation factor grows proportional to γ. Hence, solving the state equation accurately at each optimization step does not really pay off even if the projected Hessian $H(1)$ can be evaluated at a reasonable numerical cost. It is also remarkable that a positive semi-definite L_{uu} is required such that the elements of vector Γ are not greater than 1. For this reason, the results apply only to regularized problems in this sense.

2.7.4. Jacobi method on an elliptic Problem. In this part, we consider the scenario of a slowly converging primal solver. The test problem chosen for this purpose is the standard elliptic regulator problem in $1D$, in which the primal solution is obtained with Jacobi method. The preconditioner is selected to be the multiple of the identity matrix and the optimal scaling factor in this case can be found by solving a system of three cubic polynomials, which at the end can be reduced to a single polynomial in the convergence factor ρ_*. As the objective function, we use tracking type function:

$$f(y,u) = \frac{1}{2}\int_0^1 (y(t)-z(t))^2 dt + \frac{\mu}{2}\int_0^1 u^2(t)dt, \tag{2.7.26}$$

where the dependency between state y and control u is given by the state equation

$$-y''(t) = u(t), \quad \text{for } t \in [0,1]. \tag{2.7.27}$$

The boundary conditions of the problem are $y(0) = 0$ and $y(1) = 0$. The parameter μ is the regularization parameter that is strictly a positive real number and z denotes the desired target state. The Laplacian term is discretized using central finite differences using an n point equidistant grid with grid size $h = 1/(n+1)$. Based on the given target state $z \in I\!R^n$, we obtain, as for example in [**GO91**], the following discretized optimization problem:

$$\min_{(y,u)\in I\!R^{2n}} f(y,u) = \frac{h}{2}\|y-z\|^2 + \frac{\mu h}{2}\|u\|^2 \quad \text{such that } \; Cy = u, \tag{2.7.28}$$

where C is the tridiagonal matrix given by $C = -\operatorname{tridiag}(1,-2,1)/h^2$. In order to solve the linear system $Cy = u$, we use the Jacobi type of fixed-point iteration:

$$y = G(y,u) \equiv My + \frac{h^2}{2}u, \tag{2.7.29}$$

where M is the iteration matrix of the classical Jacobi method. It is well known its eigenvalues are given by

$$c_i \equiv -\cos(i\pi h), \quad i = 1,..,n. \tag{2.7.30}$$

Hence, the spectral radius of the symmetric matrix M is given as $\rho(M) = \cos(\pi h) \approx 1 - \frac{1}{2}h^2\pi^2 < 1$. Note that even for small values of n (e.g., $n \approx O(10)$), we have $h \approx O(10^{-2})$, which gives to $\rho \approx 0.99$. Therefore, we can conclude that the test problem with Jacobi method is a good model problem for slowly converging fixed-point schemes. The Lagrangian as defined in (2.2.1) becomes now

$$L(y,\overline{y},u) = \frac{h}{2}\|y-z\|^2 + \bar{y}^T My + \frac{h^2}{2}\bar{y}^T u + \frac{\mu h}{2}\|u\|^2 - \overline{y}^\top y. \tag{2.7.31}$$

Therefore, the fixed-point iteration scheme in (2.4.4) takes the form

$$y_{k+1} = My_k + 0.5h^2 u_k \tag{2.7.32}$$

$$\bar{y}_{k+1} = hy_k + J\bar{y}_k \tag{2.7.33}$$

$$u_{k+1} = u_k - B_k^{-1}(\mu h u_k + 0.5h^2 \bar{y}_k). \tag{2.7.34}$$

Note that, characteristic quantities of the optimization problem, as discussed previously, are given by

$$G_u = 0.5h^2 I,\ G_y = M,\ L_{yy} = hI,\ L_{uu} = \mu hI,\ L_{yu} = 0,\ q = 0, \tag{2.7.35}$$

and the projected Hessian is

$$H(\lambda) = \mu hI + (\lambda I - M)^{-1} h^5/4. \tag{2.7.36}$$

As the preconditioner, we set $B = Ih/\eta$ and determine the scaling factor η such that it yields the optimal convergence rate. The coupled one-shot iteration now takes the form

$$y_{k+1} = My_k + 0.5h^2 u_k \tag{2.7.37}$$

$$\bar{y}_{k+1} = h(y_k - z) + M\bar{y}_k \tag{2.7.38}$$

$$u_{k+1} = -0.5\eta h \bar{y}_k + (1 - \eta\mu)\, u_k. \tag{2.7.39}$$

The Jacobian matrix for this case is given by

$$J_* = \begin{bmatrix} M & 0 & \frac{h^2}{2} I \\ hI & J & 0 \\ 0 & -\frac{\eta h}{2} I & (1-\eta\mu) I \end{bmatrix}. \tag{2.7.40}$$

The matrix $P(\lambda) = (\lambda - 1)B + H(\lambda)$ can be diagonalized by the eigenvectors of J, which yields the matrix transformed matrix

$$\tilde{P}(\lambda) = (\lambda - 1)h/\eta I + 0.25h^5 diag(1/(\lambda - c_i)^2)_{i=1}^n + \mu hI. \tag{2.7.41}$$

Hence, the eigenvalues of J_* satisfy the set of equations:

$$P_i(\lambda) = (\lambda + \eta\mu - 1)(\lambda - c_i)^2 + h^4\eta/4 = 0, \quad i = 1, .., n, \tag{2.7.42}$$

where c_i are the eigenvalues of M as given in Eq. (2.7.30). The above equation can be rewritten as

$$\frac{1}{(\lambda - c_i)^2} = \frac{\lambda + \eta\mu - 1}{-h^4\eta/4}. \tag{2.7.43}$$

Note that, the left hand side of the above equation has a quadratic pole at $\lambda = c_i$ and the right hand side, which does not depend on c_i, is linear with respect to λ. These form a set of cubic equation in λ for each value of c_i. In Fig. 5, right hand and left hand sides of these equations are shown for the case $n = 4$ and $\eta = 0.1$. The descending blue line on the figures represents the common right hand side $(\lambda + \eta\mu - 1)/(-h^4\eta/4)$, which intersects each pole at some $\lambda_i < c_i$ on its ascending left branch. Moreover, there will be an extra pair of roots $\lambda_i^{\pm} \in P_i^{-1}(0)$, which are real for some values of c_i. The optimal step size can now be determined by the solution of the optimization problem:

$$\rho_* \equiv \max_{1 \le i \le n} \{|\lambda_i|, |\lambda_i^-|, |\lambda_i^+|\} \tag{2.7.44}$$

In the following, we examine the conditions such that the step size η is optimal, which lead to the solution of the above optimization problem.

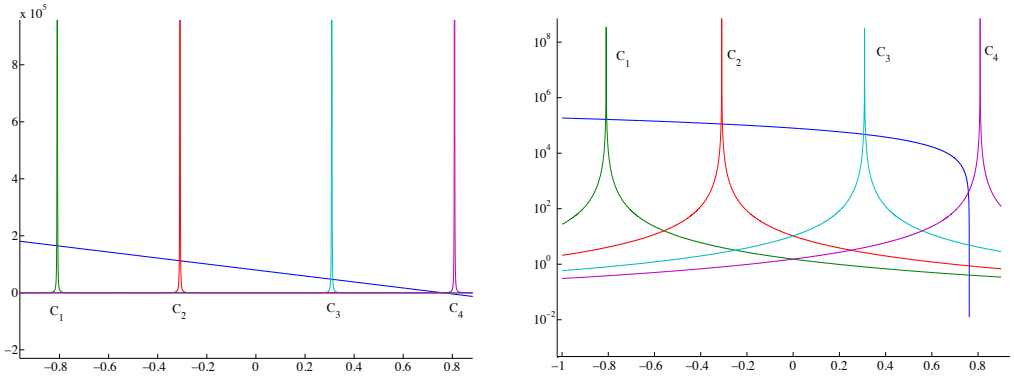

Figure 5. *Cubic equations for $n = 4$ and $\eta = 0.1$ (left figure: normal scaling, right figure: logarithmic scaling, reproduced from* [**GHO13**] *with permission).*

PROPOSITION 5. *Algebraic characterization of optimal step size*

The optimal step size η, the resulting minimal convergence factor ρ_* and an auxiliary eigenvalue λ can be computed by solving the system of three cubic polynomials given by

$$(-\rho_* + 1 - \Delta c)^2(-\rho_* - 1 + \eta\mu) + \frac{1}{4}h^4\eta = 0 \tag{2.7.45}$$

$$(\lambda - 1 + \Delta c)^2(\lambda - 1 + \eta\mu) + \frac{1}{4}h^4\eta = 0 \tag{2.7.46}$$

$$-(1 - \Delta c)^2 + \lambda\rho_*^2 + \eta(\frac{1}{4}h^4 - \mu(1 - \Delta c)^2) = 0, \tag{2.7.47}$$

where $\Delta c = 1 - c_n = 1 + c_1 \leq 0.5\pi^2h^2$.

Proof. For $\eta = 0$, we have $\rho_* = 1$ and for $\eta \to \infty$ the products of eigenvalues $\lambda_i\lambda_i^-\lambda_i^+ = -P_i(0)$ go to infinity so that ρ_* becomes very large. Therefore, a minimizer $\eta_* \geq 0$ must exist by continuity. Furthermore, this optimum can only be attained where there is a tie between at least two moduli. When all roots are real, then they are contained in the interval formed by the smallest and largest root of P_1. However, it can be easily checked that these cannot have the same modulus so that a tie between a real eigenvalue of P_1 and the complex pair $\lambda_n^\pm$ of P_n must exist. Therefore, rather than computing $\lambda_n^\pm$ directly, we impose the following conditions:

$$-P_n(0) = \lambda_n\lambda_n^-\lambda_n^+ = \lambda\rho^2 \quad \text{and} \quad P_n(\lambda_n) = 0. \tag{2.7.48}$$

These conditions, together with the equation $P_1(-\rho_*) = 0$ lead to the system of three cubic equations given above. □

Due to the linearity with respect to η, the system of three cubic polynomials introduced in Proposition 5 can be rewritten as

$$a_{11} + \eta a_{12} = 0 \ , a_{21} + \eta a_{22} = 0 \ , \ a_{31} + \eta a_{32} = 0. \tag{2.7.49}$$

The existence of a real solution requires that these vectors (a_{1j}, a_{2j}) for $j = 1, 2, 3$ are pairwise linearly dependent such that the system of two equations $a_{11}a_{32} = a_{12}a_{31}$ and $a_{11}a_{22} = a_{12}a_{21}$ in λ and ρ_* can be equivalently written. Since the first

determinant equation is linear in λ, it can be used to express λ in terms of ρ_* and can be substituted into the second equation that yields a polynomial in ρ. Finally, this polynomial can be solved by employing standard mathematical software.

In Fig. 6, the resulting retardation factors as a function $1/\mu$ for grids with $n = 32, 64, 128$ points are plotted. As it can be seen from the blue, green and red curves in the figure, the retardation factors are very small until the value of $1/\mu$ is about 10^2. Beyond this critical value, retardation factors grow quite rapidly until they become a linear function of $1/\mu$. Finally, for very large values of $1/\mu$, retardation factors become constant. It should be noted that the curves in Fig. 6 are verified by computing and optimizing the spectral radius of J_* directly as a function of the scaling parameter η using MATLAB.

The optimal values, which are obtained from the cubic system introduced in Proposition 5, contains exactly one pair of complex conjugate eigenvalues that are roots of P_n. Since their argument was observed to be rather small, we probably do not lose much by picking η as the value for which P_n has $\lambda = \lambda_n^+ = \lambda_n^-$ as the real root such that that all $3n$ eigenvalues are in fact real. Moreover, by inspection of Fig. 5, it can be seen that all these eigenvalues must be contained in the interval $\left[-\lambda_1^+, \lambda_1^+\right]$, where $P_1(\lambda_1^+) = 0$. Hence, it follows that the contraction factor must be $\rho_* = \lambda_1^+$. More specifically, an upper bound on the retardation factor is provided by the next proposition.

PROPOSITION 6. *Upper retardation bound*
The polynomial P_n has a double root at

$$\lambda = \frac{2}{3}(1 - \mu\eta) + \frac{1}{3}c_n \tag{2.7.50}$$

if $\tilde{\eta} = \eta/\Delta c$ satisfies the cubic equation, given by

$$(1 - \mu\tilde{\eta})^3 = \tilde{\eta}\frac{27h^4}{16(\Delta c)^2} \geq \tilde{\eta}\frac{27}{4\pi^4}, \tag{2.7.51}$$

which yields the retardation bound

$$r = \frac{\Delta c}{(1 - \rho_*)} \;\leq\; \frac{1}{\tilde{\eta}(\mu + h^4/16)} \;\leq\; \overline{r} \equiv \frac{1}{\tilde{\eta}\mu}. \tag{2.7.52}$$

Proof. If the polynomial P_n is differentiated with respect to λ, we obtain

$$P_n'(\lambda) = 2(\lambda - c_n)(\lambda - 1 + \mu\eta) + (\lambda - c_n)^2. \tag{2.7.53}$$

Hence, a double root $\lambda \neq c_n$ must satisfy

$$0 = 2(\lambda - 1 + \eta\mu) + \lambda - c_n = 3\lambda - 2 + 2\eta\mu - c_n, \tag{2.7.54}$$

which yields the first assertion. Substituting this value into $P_n(\lambda)$, we get

$$P_n(\lambda) = \frac{-4}{27}(\Delta c - \mu\eta)^3 + \frac{h^4}{4}\eta = 0. \tag{2.7.55}$$

Division of the above expression by Δc yields the second assertion. Since $\Delta c/h^2 = \pi^2/2 - O(h^2)$, the relation between μ and $\tilde{\eta}$ is essentially independent of h for sufficiently small h. Moreover, we find for the resulting $\rho_* = \lambda_1^+ \in (c_n, 1)$ that $0 = (\rho - c_1)^2(\rho + \eta\mu - 1) + h^4/4$ and hence with $\rho - c_1 \leq 2$

$$1 - \rho_* = \mu\eta + \frac{h^4\eta/4}{(\rho_* - c_1)^2} \geq \eta\mu + \frac{h^4\eta}{16} \tag{2.7.56}$$

Multiplication of the reciprocal by Δc yields the last assertion. □

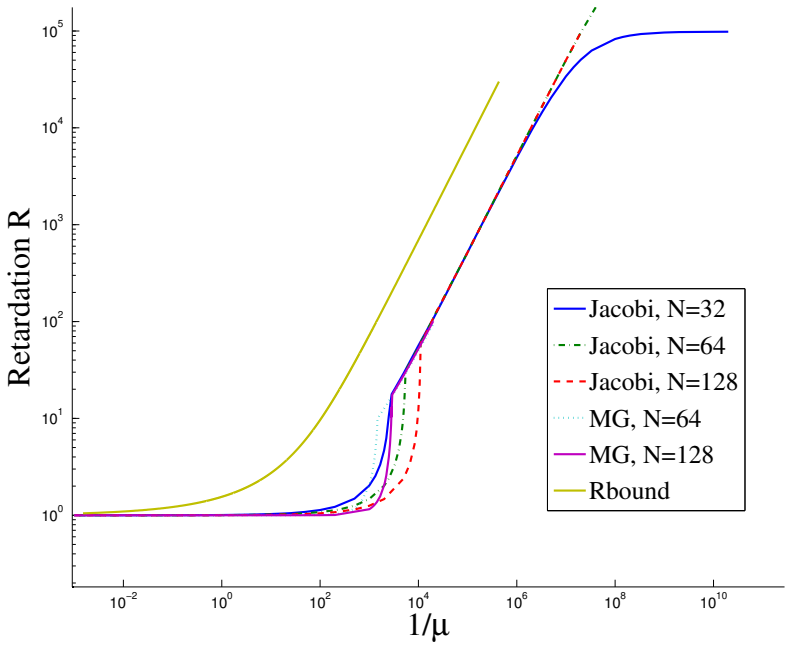

Figure 6. *Retardation factors obtained for Jacobi and multi-grid methods (reprinted from* [**GHO13**] *with permission).*

Note that the last inequality in the preposition is almost an equality as long as $\mu >> h^4/16$. By solving for μ from the cubic equation introduced in Proposition 6, we obtain the expression

$$\mu(\tilde{\eta}) \equiv \left(1 - \frac{3}{\sqrt[3]{4\pi^4}}\sqrt[3]{\tilde{\eta}}\right)/\tilde{\eta} \quad \text{for} \quad \tilde{\eta} \in (0, 4\pi^4/27). \tag{2.7.57}$$

The yellow curve in Fig. 6 is obtained by plotting the curve $(1/\mu(\tilde{\eta}), \overline{r}(\tilde{\eta}))$ parameterized by $\tilde{\eta}$. As it can be seen from the figure, $\overline{r}$ is an upper bound on the retardation and almost proportional to $1/\mu$ in a medium range, where the optimized step parameter η yields a similar slope. However, the optimized version is always faster by a factor of more than 10. Depending on the grid size h, there is an extra gain when the regularization parameter μ is relatively large. When the term $h^4/4$ dominates μ, the retardation factor reaches the constant given by $\Delta c/(1-\rho_*)$ with $\tilde{\eta} = 4\pi^4/27$, and the ρ_* the largest root of $(\rho_* - 1)(\rho - c_1)^2 + h^4\tilde{\eta}/(4\Delta c)$. This contraction ratio is of size $1 - O(h^2)$, as expected for the Jacobi method.

2.7.5. Multi-grid method. In this section, a standard V-cycle multi-grid algorithm with a Jacobi smoother is employed for primal and adjoint iterations to study the behavior of the already established retardation factor for the same optimization problem. The primal fixed-point iterations in this case are given as

$$y_{k+1} = G(y_k, u_k) \equiv C_{MG} y_k + K u_k, \tag{2.7.58}$$

where C_{MG} and K being two matrices. Then, the corresponding adjoint fixe-point iteration is

$$\bar{y}_{k+1} = hy_k + C_{MG}^{\top}\bar{y}_k. \tag{2.7.59}$$

Since the state equation is linear, all second derivatives of the Jacobian depend only on the objective, and are exactly the same as in Jacobian method. The preconditioner is given by the Proposition 2, which is scaled by a step multiplier η:

$$B = \frac{1}{\eta}(\frac{\alpha}{2}K^{\top}K + \mu h), \tag{2.7.60}$$

where $K = G_u$ and $\alpha = h/(1-\rho(C_{MG}))$. The Jacobian of the one-shot iteration is given by

$$J_* = \begin{bmatrix} C_{GM} & 0 & K \\ hI & C_{GM}^{\top} & 0 \\ 0 & -B^{-1}K & 1 - B^{-1}\mu h \end{bmatrix}. \tag{2.7.61}$$

The spectral radius of the Jacobian can be computed for small-scale problems using computer packages like MATLAB. On the same elliptic problem, as introduced in the previous section, the matrices C_{MG} and K are computed for the two level coarse and fine grids with mesh-widths $(1/32, 1/64)$ and $(1/64, 1/128)$. The spectral radius of the coupled Jacobian matrix for a range of different η values is computed to determine the minimal ρ_* approximately. The resulting retardation factors are shown in Fig. 6 in violet and light-blue colors. As one can see from the figure, the dependency on the regularization coefficient μ is almost identical to the one observed for the Jacobi method. Note that, the multi-grid solver and the resulting optimizations are much faster but the ratio between their contraction factors seems to be the same and is largely independent of the grid size used.

CHAPTER 3

Basic ingredients for aerodynamic shape optimization

In this chapter, we present the essential ingredients of an aerodynamic design chain. Based on the design chain, later we introduce the corresponding aerodynamic shape optimization chain. What is meant by design chain is the set of processes that are sequentially executed to evaluate the objective function value for the given set of design parameters. Based on the design chain, the optimization chain is built to compute the sensitivities of the objective function with respect to the shape parameters and to update the design based on this information automatically. The set of shape parameters, which is referred to as the design vector, is a discrete representation of the shape that is to be optimized.

3.1. Aerodynamic design and shape optimization chains

The shape under interest is initially specified by a vector of design parameters. The functional relationship between the design parameters and the actual shape is specified by the geometric modeling algorithm used in the shape parameterization. Using these shape parameters, a surface grid is generated by employing a Computer-Aided Design (CAD) tool. The physics of the problem is resolved by solving a state PDE, and for this purpose a computational grid is generated using the surface data. In this way, the domain, in which the physical phenomena is simulated, is divided into finite number of pieces. Note that, for practically relevant configurations, the grid generation process is computationally expensive and the grid generation software that are involved are fairly complex. Furthermore, grid quality and grid spacing have a strong influence on the accuracy of the numerical solution. After the grid is generated, boundary conditions and solver parameters are given as an input to the design chain by a preprocessor tool. Using the grid file and the configuration data, state solver is executed to solve the equations that govern the physical phenomena. Note that, the solution step is the most complex and computationally expensive step in the design chain. Once the simulation results are ready, the objective function is computed by a post-processing tool using the converged state solution. In Fig. 1, the data flow and the ingredients in a typical aerodynamic design chain are illustrated.

In many cases, it may be impractical to generate the surface and volume grids from scratch each time the shape is modified. In yet another approach, deformed surfaces are first generated using a surface deformation tool for a given vector of shape deformations. The deformed volume grid is then generated by using a grid deformation tool, which adapts the given initial grid according to the new shape using the point data of the deformed surface grid. The result is a deformed grid, which has the same topology as the initial grid. The advantage of such a design

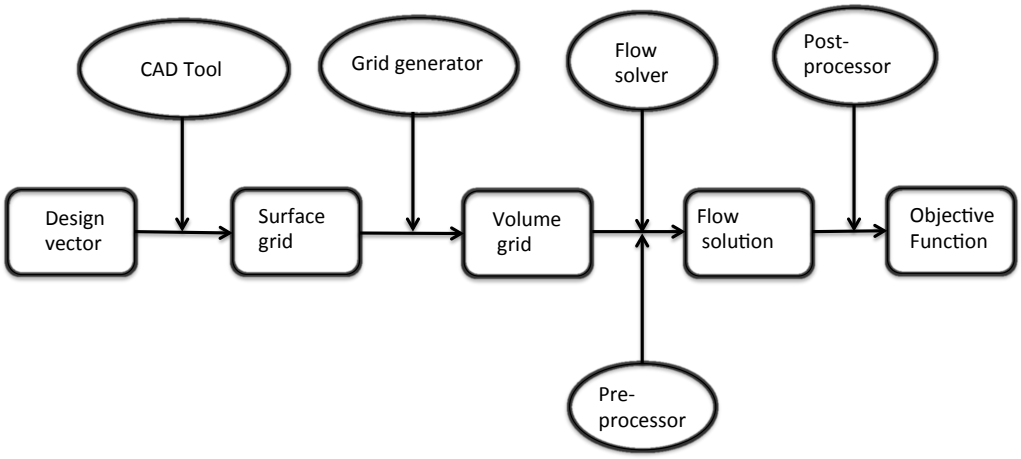

Figure 1. *The data flow in a typical aerodynamic design.*

chain is that one does not require the CAD tool and the grid generator in the design chain. Instead, grid deformation tools, which are less complex than the grid generators, are used. In this alternative design chain, the CAD tool and the grid generator are used only to generate the initial surface and volume grids. If the shape deformations are relatively small and the grid topology is to be preserved, this approach is computationally much cheaper than the mesh generation, and therefore very efficient. If the deformations are large, then the grid quality may deteriorate and re-meshing may be necessary to achieve a reasonable accuracy in simulation. The data flow and the ingredients in such an alternative design chain are illustrated in Fig. 2.

As far as the shape optimization is concerned, the optimization chain is constructed based on the same tools of the given design chain. When a gradient-based optimization algorithm is used, the sensitivities of the objective function with respect to the design parameters are required, and therefore must be computed in the optimization chain. Note that, the data flow between the design parameters and the objective function is given as

design $u \to$ surface geometry $x_s \to$ grid points $m \to$ state $y \to$ objective function J.

The sensitivities of the objective function J with respect to design u are then calculated using the chain rule as

$$\frac{dJ}{du} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial m}\frac{\partial m}{\partial x_s}\frac{\partial x_s}{\partial u}. \tag{3.1.1}$$

For the ith element of the design vector u, the above equation simply reduces to

$$\frac{dJ}{du_i} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial m}\frac{\partial m}{\partial x_s}\frac{\partial x_s}{\partial u_i}. \tag{3.1.2}$$

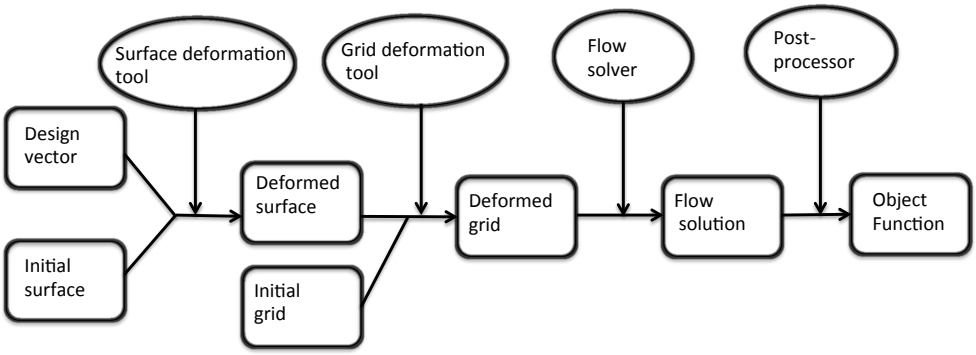

Figure 2. *An alternative aerodynamic design chain based on grid deformation.*

If the vector of design parameters has M elements, one needs to evaluate the Eq. (3.1.2) M times ($i = 1, .., M$) to be able compute all the elements of the gradient vector. Alternatively, one can use the transpose of Eq. (3.1.1):

$$\left(\frac{dJ}{du}\right)^\top = \left(\frac{\partial x_s}{\partial u}\right)^\top \left(\frac{\partial m}{\partial x_s}\right)^\top \left(\frac{\partial y}{\partial m}\right)^\top \left(\frac{\partial J}{\partial y}\right)^\top, \tag{3.1.3}$$

If we define the vector $\lambda = \left(\frac{\partial m}{\partial x_s}\right)^\top \left(\frac{\partial y}{\partial m}\right)^\top \left(\frac{\partial J}{\partial y}\right)^\top$, the above equation reduces to

$$\left(\frac{dJ}{du}\right)^\top = \left(\frac{\partial x_s}{\partial u}\right)^\top \lambda. \tag{3.1.4}$$

Note that, the vector λ does not depend on u and therefore the computational cost of the gradient vector obtained from the above equation is independent from M. As a result, first the vector λ is computed and then each element of the gradient vector can be computed by

$$\left(\frac{dJ}{du_i}\right)^\top = \left(\frac{\partial x_s}{\partial u_i}\right)^\top \lambda, \; i = 1, .., M. \tag{3.1.5}$$

The method described above is the "discrete adjoint method" and it is used in the present work to evaluate the sensitivity information, which is necessary for the one-shot algorithm. Different adjoint approaches and issues concerning adjoint solver development using Automatic Differentiation are discussed in detail in Chapters 5 and 6.

Note that, in Eq. (3.1.3), the term $(\partial J/\partial y)^\top$ is computed by an adjoint post-processor tool. This vector is then given input to an adjoint flow solver, which generates the metric sensitivities $(\partial J/\partial m)^\top$. These metric sensitivities are then used by an adjoint grid deformation tool to compute sensitivities of the objective function with respect to surface grid deformations. Finally, this result is used by

an adjoint surface grid deformation tool to compute the sensitivities with respect to design parameters. The data flow for the gradient vector computation is in the reverse order as the original design chain. The sensitivity information, generated by the adjoint chain in the reverse order, is used by the optimizer to update the design vector. Such an optimization chain is illustrated in Fig. 3.

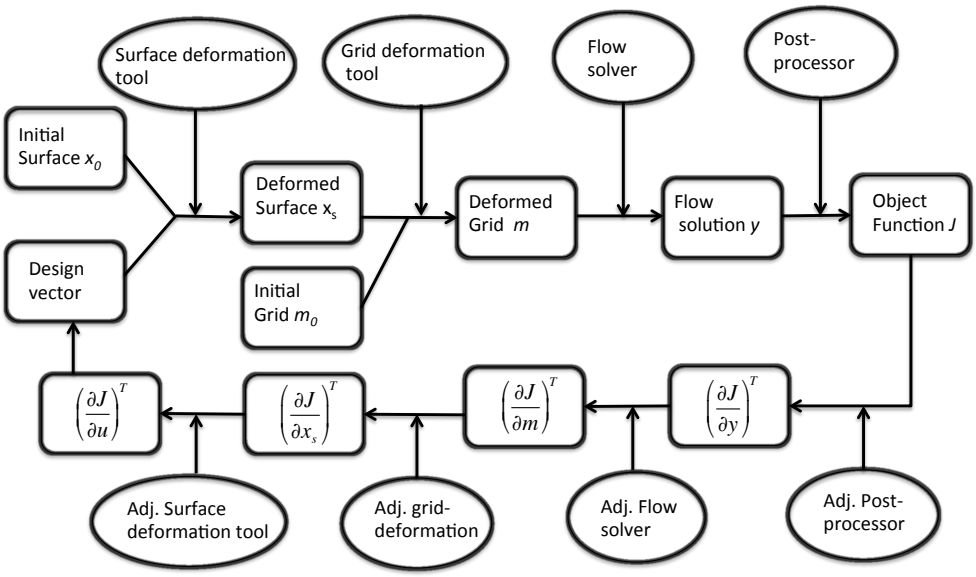

Figure 3. *Optimization chain based on the adjoint method.*

3.2. Shape parameterization techniques

Shape parameterization is one of the most important issues in aerodynamic shape optimization. The success of an optimization study highly depends on the capabilities of the chosen parameterization. In general, conceptual design optimization studies are usually restricted to small set of design parameters, which feature the global characteristics of a shape. On the other hand, detailed shape optimization studies using PDEs are usually associated with a high degree of freedom. Small scale optimizations are usually easier to perform and there is more possibility to use engineering insight and experience gained from the previous designs. If a parameterization with very few design parameters is used, one can completely abandon the gradient based optimization algorithms in many cases. For these cases, typically gradient-free methods such as Simplex method [**NM65**] or Genetic Algorithms [**Per02, Oba95, OON97**] are used. Compared to the adjoint methods, it is

easy to implement an optimization chain with gradient-free methods since no pre-knowledge over the simulation tool is required. On the other hand, these kinds of methods are quite disadvantageous if the number of design parameters used in the optimization becomes large and simulation step is computationally expensive. In conclusion, there is no general parameterization technique that works for all cases. The suitable choice in many cases is a compromise, which necessitates considering different aspects of the optimization problem.

The number of design parameters may significantly effect the result obtained from the optimization. Apart from its simplicity, small-scale optimization may lead to designs that are worse than the "real optimum" since the number of possible shapes that can be captured by the optimizer are restricted to a limited subset. Therefore, there is always a natural tendency to increase the number of parameters when the small-scale optimizations reach their limits. Therefore, it is a realistic estimate that that degrees of freedom in shape optimization studies will always increase in the future and using efficient gradient evaluation methods will become more attractive.

In this section, we briefly introduce several shape parameterization techniques that are commonly used in aerodynamic shape optimization. Detailed information about different shape parameterization techniques in aerodynamic optimization can be found in [**Sam01**].

3.2.1. Parameterization based on a set of reference shapes. This is probably the simplest type of shape parameterization, in which shape deformations are restricted to set of reference shapes, usually defined by a few parameters. For some shape sets, the optimization problem can be reduced to a pure sizing problem. The biggest advantage of this kind of parameterization is of course its simplicity. However, the major drawback is that the parameterization is specific to the particular optimization problem and the result of optimization may not be a necessarily good one. As a typical example, for airfoil shape optimization, airfoil shapes that are generated by Kutta-Joukowsky transformation can be given. This transformation is a conformal map, which was historically used to understand some principles of airfoil design. It is given by

$$z = \zeta + \frac{\lambda^2}{\zeta}, \tag{3.2.1}$$

where ζ is a complex number and λ is a real number.

Using this transformation, a circle in the ζ plane is transformed into an airfoil in the z plane, as shown in Fig. 4. By varying the center of the circle, it is possible to obtain different airfoil shapes. The airfoils that are generated by this transformation are called as Joukowsky airfoils.

3.2.2. Parameterization of the shape deformation. For some cases it makes more sense to use shape deformation rather than parameterizing the shape itself. As an example, consider the airfoil geometry, which is illustrated in the Fig. 5. In this figure, thickness-camberline decomposition of an airfoil is shown. The camberline is the loci of points, which are located in the middle of upper and lower surfaces of the airfoil. The exact shape of camberline depends on how the thickness distribution $d(x)$ is specified. The chord line, on the other hand, is the straight line that connects the leading and trailing edges of the airfoil, and can be taken as the characteristic length scale of an airfoil. For a symmetric airfoil, camberline

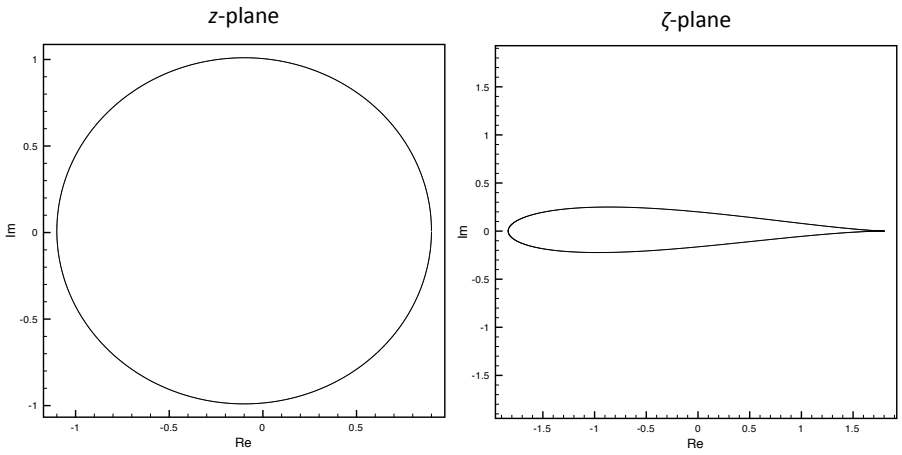

Figure 4. *Airfoil parameterization through Kutta-Joukowsky transformation ($\lambda^2 = 0.8091$ and $c = -0.1 + 0.001i$).*

and chord line coincide with each other. The distance between the leading and the trailing edges is called as chord length, denoted by c. The complete airfoil shape can be parameterized by specifying the coordinates of leading and trailing edges, thickness distribution $d(x)$ and camberline of the airfoil.

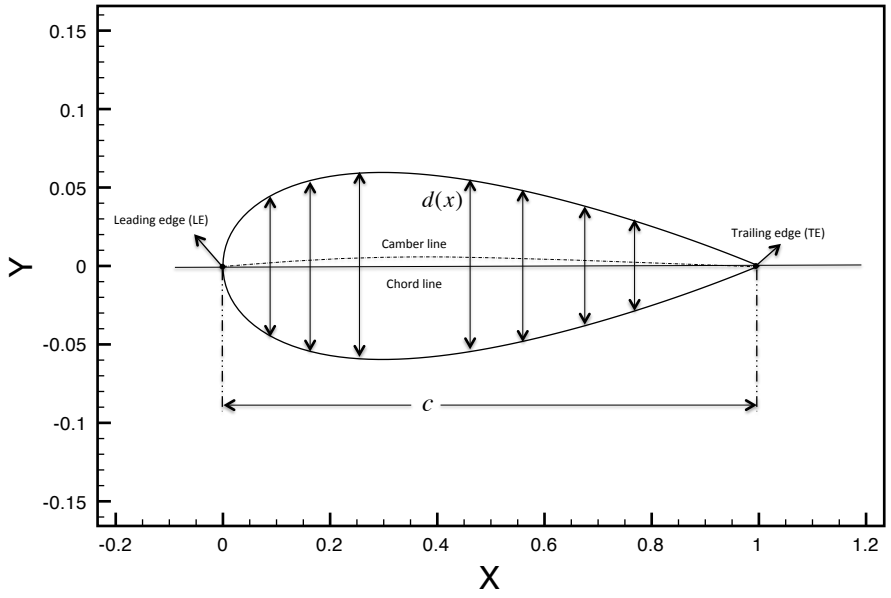

Figure 5. *Thickness-camberline decomposition of an airfoil.*

One possibility to modify the airfoil geometry is to use some basis functions for the deformation of the camberline. The basic idea is to evaluate the basis functions scaled with certain design parameters and to deform the camberline by adding these deformations. The new shape is obtained by using the deformed camberline and the initial thickness distribution. The result is a surface deformation that maintains the airfoil thickness.

As far as the basis functions are concerned, one can use for example the Hicks-Henne functions [**HH78**] for the camberline deformation. The Hicks-Henne functions, which are widely used in aerospace applications, are defined as

$$h_{i,b} : [0,1] \to [0,1] : h_{i,b}(t) = (\sin(\pi t^{\frac{\log 0.5}{\log t_i}}))^b, \quad t_i = \frac{i}{n+4}, \tag{3.2.2}$$

where b is a constant. In Fig. 6, as an example, Hicks-Henne functions for $b = 2$ and $t_i = \{0.2, 0.4, 0.6, 0.8\}$ are illustrated. Note that these functions get always zero at the endpoints and have their maximums at the position t_i. The initial camberline of an airfoil, which is denoted by $cam(0,t)$, can be deformed by a set of n Hicks-Henne basis functions $h_{i,b}$, scaled by the components of the design vector $P = (p_i) \in I\!R^n$. The resulting deformed camberline is then given by

$$cam(P,t) = cam(0,t) + \sum_{i=1}^{n} p_i h_{i,b}(t). \tag{3.2.3}$$

In conclusion, an airfoil shape can be fully parameterized by the vector P. Note that, in this kind of parameterization, leading and trailing edges of the airfoil as well as thickness distribution are kept constant. The initial value of the vector P is set to zero initially, such that the deformed shape is same as the initial shape at the first optimization iteration. In the successive iterations, the vector P is updated iteratively by the optimization algorithm to deform the airfoil shape.

The Hicks-Henne functions are smooth, therefore this kind of parameterization has the advantage that the airfoil shapes that are generated during the optimization process do not have kinks.

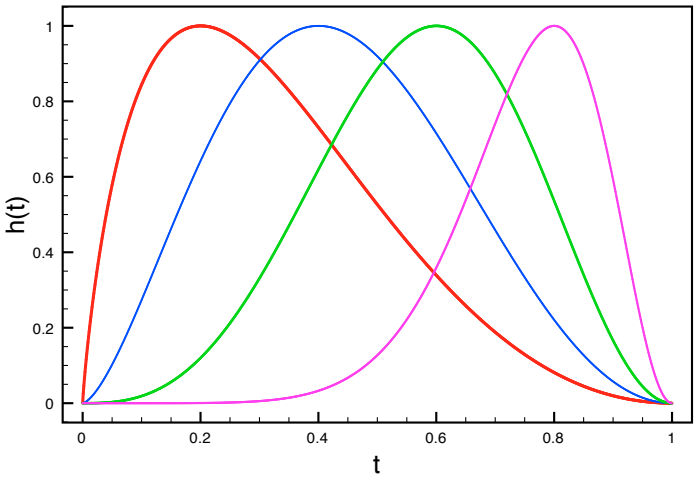

Figure 6. *Hicks-Henne basis functions.*

3.2.3. Polynomial and spline methods. Polynomial and spline methods are the most frequently used techniques to parameterize airfoils. For example, Bezier and B-splines [**Far96**] are used to represent curves using minimum number of control points. In the optimization process, the coordinates of these control points are simply shifted. In Fig. 7, such a parameterization of an airfoil using cubic Bezier splines is illustrated.

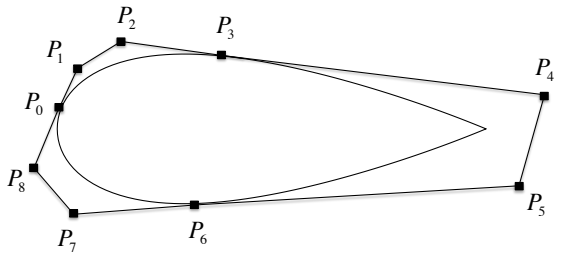

Figure 7. *The representation of an airfoil by using the control points of cubic Bezier splines.*

3.2.4. Free form deformation (FFD). FFD is a technique to model deformations of a rigid body, which originally comes from computer graphics [**SP86**]. The FFD method is based on the idea of enclosing a body within a convex hull, represented by a lattice, and modifying the shape of the body within the same hull as the lattice is deformed during the optimization. In $3D$, the lattice is usually a cube shape, which encloses the body under interest. In $2D$, a rectangle is used instead. The lattice is composed of usually Bezier patches in $3D$ and Bezier points in $2D$. Alternatively, B-splines or NURBS can also be used. When the control points of the lattice are shifted in the space, the body inside the lattice is also deformed. The advantage of the FFD method is that the computational grid is also deformed automatically by deforming the lattice. Examples of FFD method in aerodynamic shape optimization can be found in [**Sam04, AJD03**].

3.2.5. CAD-based parameterization. In this kind of parameterization, shape is defined through a few control points by using a CAD tool. The number of CAD parameters are usually much less than the number of grid points. The biggest advantage of this parameterization is its flexibility and easy usage. Furthermore, the realization of a design is much easier since the manufacturing processes are performed based on the CAD data. On the other hand, CAD based parameterization has the disadvantage that the CAD tool or geometric modeler should also be integrated in the optimization chain, which introduces extra complexity during the implementation.

3.2.6. CAD-free parameterization. This parameterization is also referred to as the free-node parameterization in the literature. In this method, all surface grid points are chosen as shape parameters in the optimization. This is the simplest kind of parameterization since a geometric modeler or a CAD tool is not required in the optimization chain. In aerodynamic shape optimization applications, the geometric modeler is usually quite complex and the source code may not be available.

Therefore, an adjoint chain cannot be easily built. This difficulty can be avoided if free-node parameterization is used. In this type of parameterization, the number of shape parameters is equal to the number surface grid points, and thus very large. Therefore, using an adjoint approach for the evaluation of the gradient vector is the only feasible choice.

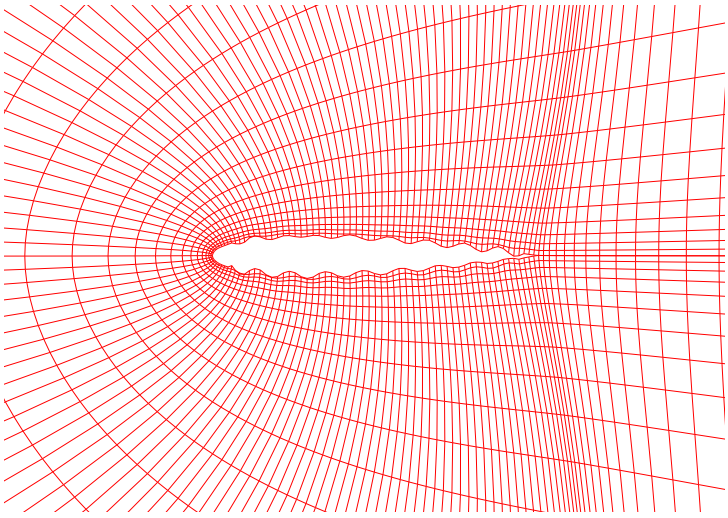

Figure 8. *A Wavy airfoil geometry, which may occur if the CAD-free parameterization is used.*

A major drawback is that the free parameterization does not guarantee smooth shapes, any wavy shapes may occur during the optimization. Such an airfoil shape is illustrated in Fig. 8. In general, it is very difficult to properly resolve the flow physics around wavy shapes due to computational difficulties. Therefore, such kind of shapes should be avoided during the optimization. This can be achieved by smoothing the gradient vector before deforming the shape. The smoothing can be considered as a projection of the gradient vector into the function space of the parameterization.

As the gradient smoothing, Sobolev smoother can be used [**Jam04**]. For a gradient vector G, Sobolev smoothing is applied by solving the following Laplace problem to obtain a smoothed gradient vector $\bar{G}$

$$\bar{G} - \frac{\partial}{\partial \xi} \epsilon \frac{\partial}{\partial \xi} \bar{G} = G \ . \tag{3.2.4}$$

In the above equation, ξ denotes the tangential component of the curvilinear coordinates. In Fig. 9, one can observe the effect of Sobolev smoothing on the shape sensitivities calculated by Automatic Differentiation. With the Sobolev smoother, the sensitivities of the drag coefficient with respect to y coordinates of an airfoil geometry are smoothed using different values of ϵ. Note that large values of ϵ produce smoother shape sensitivities.

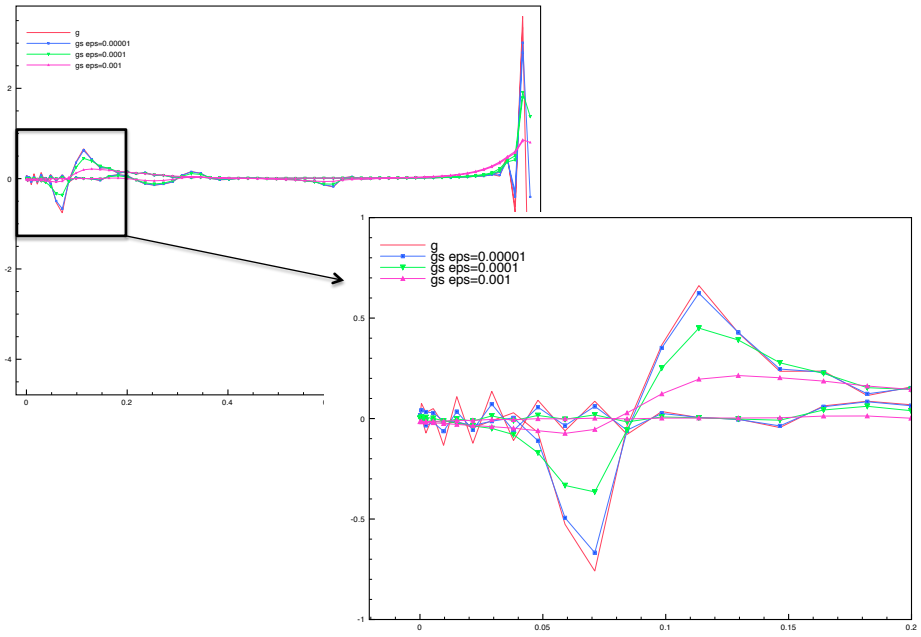

Figure 9. *Sobolev smoothing of the gradient.*

3.3. Shape and grid deformation

Shape deformation can be considered as the process of generating deformed surfaces from an initial shape. The shape deformation strategy strongly depends on the chosen shape parameterization. When free-node parameterization is used, shape deformation is trivial and the shape sensitivities, after they are multiplied with a preconditioner, can be simply added to the shape parameters. If Hicks-Henne type of parameterization is used, first points on the camberline must be deformed by adding shape functions that are scaled with design parameters. After this step, surface points are computed by using initial thickness distribution and deformed camberline. Once the deformed shape is obtained, computational grid should be adapted to the deformed shape. In case body-fitted grids are used, variations of inner nodes are directly effected from the variations on the wall boundary. One key issue here is that the grid deformation should be performed in such a way that grid connectivity is kept fixed. In addition to this, deformation process should not generate negative control volumes. In the literature, several mesh deformation techniques can be found. An overview of the most common strategies in aerodynamic design optimization can be found in [**YM05, MP02**].

In the present work, the grid deformation technique is based on the volume spline method, which was introduced by Prananta et al. [**PHZ96**]. The grid deformation tool used in the optimization studies is the DLR deformation tool `meshdefo` [**GN08**]. For the sake of completeness, in the following we introduce the volume spline method shortly as given in [**Gau08**].

The volume spline method is a general method for interpolating curves. For n different points $(x_i, y_i, z_i), i = 1, .., n$, and n different functional values $f_i(x_i, y_i, z_i), i = 1, .., n$, the interpolation formula is given by

$$f(x,y,z) = a_1 + a_2 x + a_3 y + a_4 z + \sum_{i=1}^{n} b_i \sqrt{(x-x_i)^2 + (y-y_i)^2 + (z-z_i)^2}. \tag{3.3.1}$$

In the above formula, there are $n + 4$ unknowns, therefore $n + 4$ relations are required to have an unique solution of interpolation factors. Totally n conditions are given by the functional values:

$$f(x_i, y_i, z_i) = f_i, \ i = 1, ..., n. \tag{3.3.2}$$

Additional four equations come from the extra equilibrium conditions, which are given as

$$\sum_{i=1}^{n} b_i = 0, \quad \sum_{i=1}^{n} b_i x_i = 0, \quad \sum_{i=1}^{n} b_i y_i = 0 \quad \text{and} \quad \sum_{i=1}^{n} b_i z_i = 0. \tag{3.3.3}$$

As a result, one can obtain the following $(n+4)\times(n+4)$ system of linear equations:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & . & . & 1 \\ 0 & 0 & 0 & 0 & x_1 & x_2 & . & . & x_n \\ 0 & 0 & 0 & 0 & y_1 & y_2 & . & . & y_n \\ 0 & 0 & 0 & 0 & z_1 & z_2 & . & . & z_n \\ 1 & x_1 & y_1 & z_1 & 0 & \epsilon_{12} & . & . & \epsilon_{1n} \\ 1 & x_2 & y_2 & z_2 & \epsilon_{21} & 0 & . & . & \epsilon_{2n} \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ 1 & x_n & y_n & z_n & \epsilon_{n1} & \epsilon_{n2} & . & . & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ b_1 \\ b_2 \\ . \\ . \\ b_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ f_1 \\ f_2 \\ . \\ . \\ f_n \end{bmatrix}, \tag{3.3.4}$$

where, ϵ_{ij} denotes the Euclidean distance between any two points (x_i, y_i, z_i) and (x_j, y_j, z_j), given by

$$\epsilon_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}. \tag{3.3.5}$$

The volume spline interpolation formula is applied by taking the surface deformations as the data points f_i. If we have the deformed surface grid points $(x_{i,new}, y_{i,new}, z_{i,new})$ and the initial surface grid points $(x_{i,old}, y_{i,old}, z_{i,old})$, the interpolated deformation functions $dx(x, y, z)$, $dy(x, y, z)$ and $dz(x, y, z)$ can be found by using the above interpolation formula. Then, the inner grid coordinates are deformed by using the interpolated deformation functions. As an example, for the x coordinates, one can set the functional values as $f_i = x_{i,new} - x_{i,old}, \ i = 1, .., n$, and then deform all the internal nodes using $dx(x, y, z)$:

$$x_{j,new} = x_{j,old} + dx(x_{j,old}, y_{j,old}, z_{j,old}). \tag{3.3.6}$$

Similarly, y and z coordinates of the interior grid nodes are deformed using the interpolated deformation functions $dy(x, y, z)$ and $dz(x, y, z)$. In Fig. 10, a deformed grid that is obtained by applying the volume spline method to the initial grid for a RAE 2822 airfoil is shown.

Note that, in the optimization process sensitivities of grid coordinates with respect to deformations are required. For this purpose, the grid deformation tool can be differentiated by using the reverse mode of Automatic Differentiation (AD).

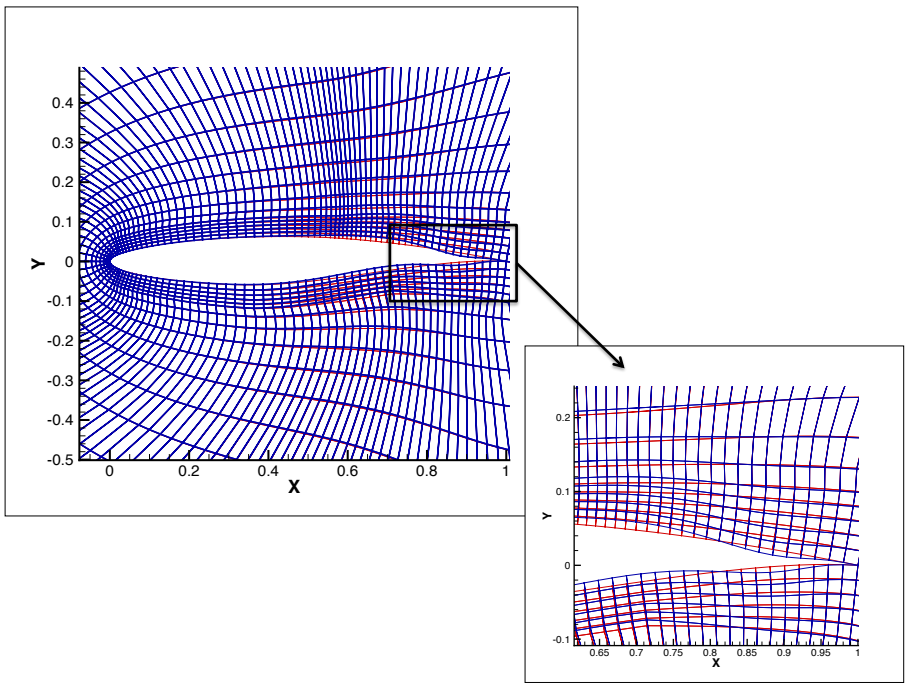

Figure 10. *Grid deformation for the RAE 2822 airfoil.*

The number of surface deformations used in the optimization might be quite large, therefore it is not feasible to compute the derivatives of the grid points with respect to surface points using finite differences. In fact, AD appears to be the most suitable choice if the source code of the grid deformation tool is available.

3.4. Simulation tools

In order to resolve the fluid flow around a body, the governing state equations must be solved. The state equations are derived from the physical laws, which describe the fluid flow phenomena. In general, Euler or Navier-Stokes equations are solved to determine the distribution of state variables like velocity components and pressure around the object of interest.

Apart from simple configurations like Poiseuille and Couette flows, it is not possible to solve the Euler or Navier-Stokes equations analytically. Therefore, numerical methods are employed to find an approximation. The branch of fluid mechanics that is involved in numerical methods and algorithms to simulate fluid flow is called as Computational Fluid Dynamics (CFD). Starting from linearized potential equations in 1930s, CFD methods have now reached such a maturity level that unsteady RANS simulations for highly complex $3D$ geometries such as a complete aircraft can be performed. Thanks to the significant progress of computational facilities and development of efficient numerical algorithms, CFD has become an essential part of the design process in many industrial applications. Aerospace, turbomachinery, chemical and automotive industries count as the major application

fields of CFD. In the following chapter, we introduce the Euler and Navier-Stokes equations as well as CFD methods in detail.

CHAPTER 4

Solution of Flow Equations

In this chapter, we introduce the Euler and Navier-Stokes equations that are used as state PDEs in aerodynamic shape optimization problems. Euler and Navier-Stokes equations form a set of PDEs, which are derived from the transport equations for conservation of mass, momentum and energy. Since the analytical solution can be obtained only for simple configurations, these equations are generally solved by using numerical methods.

In integral form, the continuity equation is written as

$$\frac{\partial}{\partial t}\int_{\Omega} \rho \, d\Omega + \oint_{\partial\Omega} \rho(\vec{U}\cdot\vec{n}) \, dS. \tag{4.0.1}$$

In the above equation, ρ and $\vec{U}$ denote the density of the fluid and the Cartesian velocity vector respectively. The outward pointing unit normal vector is represented by $\vec{n}$. The continuity equation states that the net increase or decrease of mass in a fluid element Ω must be balanced by the net mass flux through the boundary $\partial\Omega$ (see. Fig. 1). Note that, continuity equation is valid for any arbitrary fluid element Ω. If the surface integral term is transformed into a volume integral using the Gauss divergence theorem, we get

$$\frac{\partial}{\partial t}\int_{\Omega} \rho \, d\Omega + \int_{\Omega} \nabla\cdot(\rho\vec{U}) \, d\Omega. \tag{4.0.2}$$

Since the control volume can be chosen arbitrarily, Eq. (4.0.2) is also valid for any point in the flow domain Ω. Therefore, differential form of the continuity equation is

$$\frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\vec{U}) = 0. \tag{4.0.3}$$

Using tensor notation, the above equation can be written as

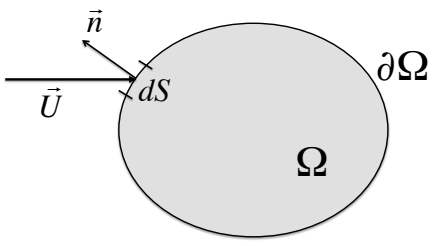

Figure 1. *A fluid element in 2D.*

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho U_i)}{\partial x_i} = 0, \tag{4.0.4}$$

where x_i denotes the Cartesian coordinate in i direction. For an incompressible flow (if the density ρ is assumed to be constant), continuity equation can be further simplified as

$$\frac{\partial U_i}{\partial x_i} = 0. \tag{4.0.5}$$

The above equation states that the velocity field in an incompressible flow must be divergence-free (i.e., $\nabla \cdot \vec{U} = 0$). Therefore, continuity equation is a kinematic constraint on the flow.

The momentum equation in integral form (if external forces are neglected) is given as

$$\frac{\partial}{\partial t} \int_\Omega \rho \vec{U} d\Omega + \oint_{\partial\Omega} \rho \vec{U} (\vec{U} \cdot \vec{n}) \, dS = - \oint_{\partial\Omega} p \, \vec{n} \, dS + \oint_{\partial\Omega} \tau_{ij} \cdot \vec{n} \, dS, \tag{4.0.6}$$

where p and τ_{ij} denote pressure and viscous shear stress tensor respectively. For a Newtonian fluid, second order tensor τ_{ij} is given using the Stokes's hypothesis as

$$\tau_{ij} = \mu \left(\frac{\partial U_j}{\partial x_i} + \frac{\partial U_i}{\partial x_j} \right) - \frac{2}{3} \delta_{ij} \mu \frac{\partial U_k}{\partial x_k}, \tag{4.0.7}$$

where μ denotes the dynamic viscosity of the fluid. The Kronecker-Delta δ_{ij} in the above equation is defined as

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{4.0.8}$$

In differential form, momentum equations can be written in tensor notation as

$$\frac{\partial (\rho U_j)}{\partial t} + \frac{\partial (\rho U_i U_j)}{\partial x_i} = - \frac{\partial p}{\partial x_j} + \frac{\partial \tau_{ij}}{\partial x_i}. \tag{4.0.9}$$

The momentum equations is a set of equations, which states that the linear momentum in each coordinate direction must be conserved. The sum of terms on the left hand side of the above equation denotes the total or material derivative of the momentum component (ρU_j). The total derivative can be represented as the time derivative along the streamline of the fluid element. The terms on the right hand side denote the surface forces exerted on the flow element. The first term represents the pressure forces acting on the fluid element. The second term, on the other hand, is the sum of viscous forces, which represents the molecular transport of j-momentum component in the i direction.

The momentum equation given in Eqs. (4.0.9) and (4.0.6) are the most general forms and valid for all type of flows in continuous media. Further simplifications based on different assumptions are possible. If the effect of viscosity is neglected (inviscid flow assumption), one can derive the Euler equations from Navier-Stokes equations. The Euler equations are used to simulate high Mach number flows, in which inviscid forces are dominant and effect of viscosity is negligible. Another well established assumption is the incompressible flow assumption, which is used in low speed aerodynamics and hydrodynamics. For incompressible flows, the momentum equations can be simplified by taking constant density and viscosity, which lead to the incompressible Navier-Stokes equations:

$$\frac{\partial U_i}{\partial x_i} = 0 \tag{4.0.10}$$

$$\rho\left(\frac{\partial U_j}{\partial t} + U_i\frac{\partial U_j}{\partial x_i}\right) = -\frac{\partial p}{\partial x_j} + \frac{\partial \tau_{ij}}{\partial x_i}. \tag{4.0.11}$$

These equations form a closed set that consists of four equations and four unknowns (p, U_i) in 3D. Therefore, velocity and pressure fields at any time t can be computed only by solving these equations using any additional information. For compressible flows, however, two additional equations are required to close the system. Note that, in compressible flows, density can be represented as a function of temperature T and pressure p, i.e., $\rho = f(p, T)$. This functional relationship is called as the state equation. For a perfect gas, for example, the state equation is given by

$$p = \rho R T, \tag{4.0.12}$$

where R denotes the specific gas constant. It is given by the difference of specific heats at constant pressure and constant volume, $R = c_p - c_v$. To be to write the state equation in terms of conservative variables, one can use the relation for the specific internal energy from thermodynamics. It is given in differential form as

$$de = c_v dT. \tag{4.0.13}$$

If the reference internal energy is taken as zero for $T = 0$, we obtain

$$e = c_v T. \tag{4.0.14}$$

On the other hand, total internal energy E is given by

$$E = e + \frac{\vec{U}^2}{2}. \tag{4.0.15}$$

Using the above relation in state equation, we finally get

$$p = (\gamma - 1)\rho(E - \frac{1}{2}(v^2 + u^2)), \tag{4.0.16}$$

where γ is the ratio of specific heats c_p/c_v (e.g., $\gamma = 1.4$ for air). In the above equation the internal energy is also an unknown, therefore energy equation must be also included to close the equation system. The energy equation is derived from the first law of Thermodynamics, which states that the net energy change of a fluid element is equal to the net amount of heat flux plus the net amount of work done. In integral form, if external forces and heat sources are neglected, the energy equation is given as

$$\frac{\partial}{\partial t}\int_\Omega \rho E \, d\Omega + \oint_{\partial\Omega} \rho E(\vec{U}\cdot\vec{n})dS = \oint_{\partial\Omega} k(\nabla T\cdot\vec{n})\, dS + \oint_{\partial\Omega}(\sigma_{ij}\cdot\vec{U})\cdot\vec{n}\, dS, \tag{4.0.17}$$

where k is the thermal conductivity coefficient. For simplicity, pressure and viscous terms are written as a summation of a surface integral, which is denoted by the second order tensor $\sigma_{ij} = -pI + \tau_{ij}$. In differential form, the energy equation is given as

$$\frac{\partial(\rho E)}{\partial t} + \frac{\partial(\rho E U_i)}{\partial x_i} = -\frac{\partial \dot{q}_i^t}{\partial x_i} - p\frac{\partial U_j}{\partial x_j} + \tau_{ij}\frac{\partial U_j}{\partial x_i}, \tag{4.0.18}$$

where $\dot{q}_i^t$ denotes the thermal conduction term and can be calculated using the Fourier's law: $\dot{q}_i^t = -k\frac{\partial T}{\partial x_i}$.

Although both compressible and incompressible Navier-Stokes equations are derived from the same conservation laws, solution schemes used for them are different. This is due to the fact that compressible and incompressible equations are used to simulate different flow regimes, in which the character of the equations also changes. For example, in steady-state subsonic flows incompressible Navier-Stokes equations are used and they have an elliptic character. When the flow becomes unsteady subsonic, these equations have a mixed parabolic-elliptic character. In transonic and supersonic flows compressibility effects cannot be neglected, therefore compressible Euler and Navier-Stokes equations are used for simulation. In contrast to incompressible equations, compressible equations have a hyperbolic character in these flow regimes. As a result, spatial discretization and solution schemes used for compressible and incompressible equations are substantially different from each other. In the following, we first introduce the compressible Euler/Navier-Stokes equations and the density-based solution methods following the textbook of Blazek [**Bla01**]. Then, we turn our attention to the incompressible Navier-Stokes equations, which are mainly solved by employing pressure-based schemes. As far as the incompressible Navier-Stokes equations and numerical treatment of pressure-based schemes are concerned, we follow the textbook of Ferziger and Peric [**FP01**]. In both cases, we stick to the Finite Volume Method (FVM), which is by far the most popular discretization approach in CFD.

4.1. Density-based schemes

If the integral forms of continuity, momentum and energy equations, one can write the compressible Navier-Stokes equations as

$$\frac{\partial}{\partial t}\int_\Omega \vec{q}\, d\Omega + \oint_{\partial\Omega} (\vec{F_c} - \vec{F_v})\, dS = \int_\Omega \vec{Q}\, d\Omega. \tag{4.1.1}$$

In the above equation, $\vec{q}$ is called as the vector of conservative variables and it has the following four components in $2D$:

$$\vec{q} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{bmatrix}. \tag{4.1.2}$$

The vectors $\vec{F_c}$ and $\vec{F_v}$ the convective and diffusive fluxes, which are given by

$$\vec{F_c} = \begin{pmatrix} \rho V \\ \rho u V + n_x p \\ \rho v V + n_y p \\ \rho V H \end{pmatrix}, \vec{F_v} = \begin{pmatrix} 0 \\ n_x\tau_{xx} + n_y\tau_{xy} \\ n_x\tau_{yx} + n_y\tau_{yy} \\ n_x\Theta_x + n_y\Theta_y \end{pmatrix}, \tag{4.1.3}$$

where

$$\Theta_x = u\tau_{xx} + v\tau_{xy} + k\frac{\partial T}{\partial x} \tag{4.1.4}$$

and

$$\Theta_y = u\tau_{yx} + v\tau_{yy} + k\frac{\partial T}{\partial y}. \tag{4.1.5}$$

In the above equations, V and H denote the contravariant velocity and the enthalpy respectively. The contravariant velocity V is always normal to the surface element dS and it is given by the scalar product of the velocity and the normal vector $V = \vec{U} \cdot \vec{n} = u\ n_x + v\ n_y$. The enthalpy H, on the other hand, is given by $H = E + p/\rho$.

If external forces and heat sources are neglected, the source term $\vec{Q}$ is zero. If some external forces ($\vec{f_e}$) and/or heat sources ($\dot{q_h}$) are present in the flow, this vector is then given as

$$\vec{Q} = \begin{bmatrix} 0 \\ \rho f_{e_x} \\ \rho f_{e_y} \\ \rho f_e \cdot \vec{U} + \dot{q_h} \end{bmatrix}. \tag{4.1.6}$$

Note that, one can obtain the compressible Euler equations from the compressible Navier-Stokes equations if viscous fluxes are neglected ($\vec{F_v} = 0$).

In order to solve Eq. (4.1.1) using FVM, the flow domain is partitioned into smaller elements, which are called control volumes. If the control volumes do not change in time and volume averaged quantities are used, Eq. (4.1.1) can be rewritten as

$$\frac{\partial \vec{q}}{\partial t} = -\frac{1}{\Omega}\left[\oint_{\partial\Omega} (\vec{F_c} - \vec{F_v})\ dS - \int_{\Omega} \vec{Q}\ d\Omega\right]. \tag{4.1.7}$$

The surface integral of convective and diffusive fluxes, which shows up in the above equation, is approximated by a sum of fluxes crossing each face of the control volume. This approximation is known as the spatial discretization. For the spatial discretization, it is usually assumed that convective and diffusive fluxes are constant along each control volume face. The source term, on the other hand, is assumed to have a constant value inside the control volume. If we consider a single control volume Ω_i, which has N_f boundary faces, we have

$$\frac{\partial \vec{q_i}}{\partial t} = -\frac{1}{\Omega_i}\left[\sum_{m=1}^{N_f} (\vec{F_c} - \vec{F_v})_m\ \Delta S_m - \vec{Q}\ \Omega_i\right]. \tag{4.1.8}$$

In the above equation ΔS_m denotes the face area and Ω_i is the volume of the ith cell. The number of cell faces N_f depends on the shape of control volumes. Note that, the terms inside the bracket is referred to as the cell residual, which may be denoted with $\vec{R_i}$. The Eq. (4.1.8) is then abbreviated as

$$\frac{\partial \vec{q_i}}{\partial t} = -\frac{1}{\Omega_i}\vec{R_i}. \tag{4.1.9}$$

If the above equation is written for each control volume, one gets a coupled system of ODEs that are hyperbolic in time. This system can be then solved by applying a suitable time-marching scheme with appropriate initial and boundary conditions. In the following, we first introduce two spatial discretization schemes commonly used in compressible Euler and Navier-Stokes simulations. Then, the temporal discretization schemes will be covered.

4.1.1. Discretization of convective fluxes. In order to solve the Eq. (4.1.8), the convective fluxes across each face of a control volume must be discretized. In compressible flows, the discretization scheme used for the convective fluxes is very

crucial to be able to resolve the flow physics correctly. Convective schemes can be broadly classified into five categories:

- central schemes
- flux-vector splitting schemes
- flux-difference splitting schemes
- total variation diminishing (TVD) schemes
- fluctuation-splitting schemes

Out of these schemes, only central scheme with artificial dissipation and Roe's flux-difference splitting scheme are introduced in the following. For detailed information about the convective schemes, the reader is advised to refer [**Bla01**].

4.1.1.1. *Central schemes.* The central scheme with artificial dissipation is one of the simplest convective scheme compared to other methods. Since its implementation is easy, it has been very popular in the CFD community. It was developed by Jameson et al. [**JST81**] for the Euler equations, therefore it is abbreviated as the JST (Jameson-Schmidt-Turkel) scheme.

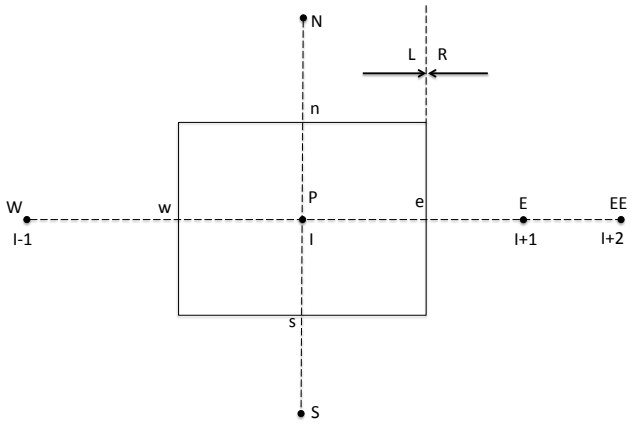

Figure 2. *A CV in 2D and the cell face notations, left and right states at a cell face in a cell-centered scheme.*

The basic idea in a central scheme is to compute the convective flux at a cell face simply by averaging the conservative variables on both sides of the face. An artificial dissipation term, which is similar to a viscous term, is added to stabilize the scheme and prevent overshooting problems at shock positions. The convective flux of a conservative variable q at one cell face, for example east face 'e' of a control volume (see Fig. 2), is then approximated as

$$F_{c,e}S_e \approx F_{c,e}(q_e)S_e - D_e, \tag{4.1.10}$$

where D_e denotes the dissipation term. The conservative variables on the east face are simply averaged by using the nodal values obtained from the locations E (east) and P:

$$q_e = \frac{q_P + q_E}{2}. \tag{4.1.11}$$

The artificial dissipation term D_e is given by

$$D_e = \lambda_{S_e}[\epsilon^{(2)}(q_E - q_P) - \epsilon^{(4)}(q_{EE} - 3q_E + 3q_P - q_W)], \tag{4.1.12}$$

where λ_{S_e} is the sum of spectral radii of the convective flux Jacobian in all directions. It is used to scale dissipation terms. In $2D$, for example, we have

$$\lambda_{S_e} = \lambda_{x,e} + \lambda_{y,e}. \tag{4.1.13}$$

The spectral radius in one direction at a cell face can also be found by averaging between the neighboring nodes. Similarly, for the east face in the x direction, we have

$$\lambda_{x,e} = \frac{\lambda_{x,E} + \lambda_{x,P}}{2}. \tag{4.1.14}$$

The spectral radius at the nodal locations can be evaluated by considering the spectral radius of the convective flux Jacobian. In $2D$, it is computed by

$$\lambda_{x,P} = (|V| + c)\ \Delta S, \tag{4.1.15}$$

where V is the contravariant velocity and c is the speed of sound.

Furthermore, in flows with shocks, a pressure based sensor is used to switch off the fourth order term in the locations, where the pressure gradient is large. For the smooth regions in the flow, however, second order terms are switched off to reduce the dissipation and increase the accuracy of the scheme. The coefficients in Eq. (4.1.12) are given by the relations below, where Γ denotes the pressure sensor.

$$\epsilon_e^{(2)} = k^{(2)} \max(\Gamma_P, \Gamma_E),\ \epsilon_e^{(4)} = \max(0, k^{(4)} - \epsilon_e^{(2)}). \tag{4.1.16}$$

The nodal value of the pressure sensor in the above equation is given as

$$\Gamma_P = \frac{|p_E - 2p_P + p_W|}{p_E - 2p_P + p_W}. \tag{4.1.17}$$

Typical values for the dissipation coefficients are given as: $k^{(2)} = 0.5$ and $1/128 < k^{(4)} < 1/64$.

The central scheme, which is described above, is as a scalar dissipation scheme since a scalar coefficient λ_{S_e} is used to scale the dissipation terms. Alternatively, a matrix can be also used. In this way, each equation is separately scaled by the corresponding eigenvalue. This alternative method is called as the matrix dissipation scheme [**ST92**].

4.1.1.2. *Flux-difference schemes.* The flux-difference schemes, which are based on a idea initially introduced by Godunov, evaluate the convective fluxes at a cell face from the left and right states by solving the Riemann problem. In order to reduce the computational effort of the Godunov's scheme, which is used for the exact solution of the Riemann problem, approximate Riemann solvers were developed in the past. In particular, Roe's method [**Roe81**] is employed frequently in the density-based solvers because of its high accuracy and good resolution of shocks.

Roe's approximate Riemann solver is based on the decomposition of the flux difference into a sum of wave contributions, while ensuring the conservation properties of the Euler equations at the same time. For example, on the 'east' face, flux difference is given as

$$(\vec{F_c})_R - (\vec{F_c})_L = \bar{A}_{roe,e}(\vec{q}_R - \vec{q}_L). \tag{4.1.18}$$

In the above expression, the subscripts 'R' and 'L' denote right and left states respectively (see Fig. 2). The matrix $\bar{A}_{roe,e}$, which is referred to as the Roe matrix, is identical to the convective flux Jacobian, in which flow variables are replaced by the so-called Roe-averaged variables. In 2D, it is given by

$$\bar{A}_{roe} = \begin{bmatrix} 0 & n_x & n_y & 0 \\ n_x\tilde{\phi} - \tilde{u}\tilde{V} & \tilde{V} - (\gamma-2)n_x\tilde{u} & n_y\tilde{u} - (\gamma-1)n_x\tilde{v} & (\gamma-1)n_x \\ n_y\tilde{\phi} - \tilde{v}\tilde{V} & n_x\tilde{v} - (\gamma-1)n_y\tilde{u} & \tilde{V} - (\gamma-2)n_y\tilde{v} & (\gamma-1)n_y \\ \tilde{V}(\tilde{\phi} - a) & n_x a - (\gamma-1)\tilde{u}\tilde{V} & n_y a - (\gamma-1)\tilde{v}\tilde{V} & \gamma\tilde{V} \end{bmatrix}, \tag{4.1.19}$$

where $a = \gamma E - \tilde{\phi}$, $\tilde{V} = n_x\tilde{u} + n_y\tilde{v}$ and $\tilde{\phi} = \frac{1}{2}(\gamma-1)(\tilde{u}^2 + \tilde{v}^2)$. The Roe-averaged values are evaluated using the left and the right states according to the following formulae:

$$\tilde{\rho} = \sqrt{\rho_L \rho_R}, \tag{4.1.20}$$

$$\tilde{u} = \frac{u_L\sqrt{\rho_L} + u_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}}, \tag{4.1.21}$$

$$\tilde{v} = \frac{v_L\sqrt{\rho_L} + v_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}}, \tag{4.1.22}$$

$$\tilde{H} = \frac{H_L\sqrt{\rho_L} + H_R\sqrt{\rho_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}}, \tag{4.1.23}$$

$$\tilde{c} = \sqrt{(\gamma-1)(\tilde{H} - \frac{\tilde{u}^2 + \tilde{v}^2}{2})}. \tag{4.1.24}$$

The convective flux on the east face of a cell is then given by

$$\vec{F}_{ce} = \frac{1}{2}[\vec{F}_c(\vec{q}_R) + \vec{F}_c(\vec{q}_L) - |\bar{A}_{roe,e}|(\vec{q}_R - \vec{q}_L)]. \tag{4.1.25}$$

The product $|\bar{A}_{roe,e}|(\vec{q}_R - \vec{q}_L)$ can be evaluated as

$$|\bar{A}_{roe,e}|(\vec{q}_R - \vec{q}_L) = |\Delta\vec{F}_1| + |\Delta\vec{F}_{2,3}| + |\Delta\vec{F}_4|, \tag{4.1.26}$$

where the terms $|\Delta\vec{F}_1|$, $|\Delta\vec{F}_{2,3}|$ and $|\Delta\vec{F}_4|$ are given by

$$|\Delta\vec{F}_1| = |\tilde{V} - \tilde{c}|\frac{\Delta p - \tilde{\rho}\tilde{c}\Delta V}{2\tilde{c}^2}\begin{bmatrix} 1 \\ \tilde{u} - \tilde{c}n_x \\ \tilde{v} - \tilde{c}n_y \\ \tilde{H} - \tilde{c}\tilde{V} \end{bmatrix}, \tag{4.1.27}$$

$$|\Delta\vec{F}_{2,3}| = |\tilde{V}|\left(\Delta\rho - \frac{\Delta p}{\tilde{c}^2}\right)\begin{bmatrix} 1 \\ \tilde{u} \\ \tilde{v} \\ (\tilde{u}^2 + \tilde{v}^2)/2 \end{bmatrix} + |\tilde{V}|\tilde{\rho}\begin{bmatrix} 0 \\ \Delta u - \Delta V n_x \\ \Delta v - \Delta V n_y \\ \tilde{u}\Delta u + \tilde{v}\Delta v - \tilde{V}\Delta V \end{bmatrix} \tag{4.1.28}$$

and

$$|\Delta\vec{F}_4| = |\tilde{V} + \tilde{c}|\frac{\Delta p + \tilde{\rho}\tilde{c}\Delta V}{2\tilde{c}^2}\begin{bmatrix} 1 \\ \tilde{u} + \tilde{c}n_x \\ \tilde{v} + \tilde{c}n_y \\ \tilde{H} + \tilde{c}\tilde{V} \end{bmatrix}. \tag{4.1.29}$$

In the above expressions, the operator Δ denotes the jump condition between right and the states (i.e., $\Delta(\cdot) = (\cdot)_R - (\cdot)_L$). These states are typically computed using the Van Leer's MUSCL (Monotone Upstream-Centered Schemes for Conservation Laws) scheme [**vL79**]. If the flow region contains strong gradients, the MUSCL interpolation usually has to be enhanced by the so-called limiter functions (e.g., Van-Albada limiter [**vAvLR82**], Venkatakrishnan limiter [**Ven95**], etc.). The purpose of these limiters is to suppress the non-physical oscillations of the solution, which may occur especially in the shock region.

Note that, the Roe flux replaces nonlinear waves of gas dynamics (i.e., rarefactions and shock waves) by linear waves that are the contact discontinuities. If sufficiently weak shock waves occur for a discontinuity between two states $\vec{q}_R$ and $\vec{q}_L$, then the Roe flux can be considered as a good approximation. However, if a rarefaction containing a sonic point is present among the nonlinear waves that solves the discontinuity problem between $\vec{q}_R$ and $\vec{q}_L$, then the Roe flux does not satisfy the entropy condition, and therefore the solution need not to be a physical solution [**MT09**]. In order to solve this problem, modulus of the eigenvalues $|\tilde{V} \pm \tilde{c}|$ are modified using Harten's entropy correction [**HLvL83**].

4.1.2. Discretization of diffusive fluxes. In general, control volumes chosen for the discretization of the viscous fluxes are the same as these chosen for convective fluxes so that both discretization schemes must be consistent with each other. In the most common way, viscous fluxes in the discretized governing equations are evaluated from the averaged variables at the cell faces. This is in accordance with the elliptic nature of the viscosity driven molecular momentum transport. Therefore, velocity components u, v, viscosity μ, and heat conduction coefficient k, which are required for the computation of the viscous terms, are simply averaged at a cell face. For the cell-centered scheme, value of a flow variable ϕ at the east face is computed as

$$\phi_e = \frac{1}{2}(\phi_P + \phi_E). \tag{4.1.30}$$

Based on averaging, evaluation of velocity and temperature gradients, which are required to compute viscous fluxes, can be accomplished using the Green's theorem or finite differences based on the interpolated values.

4.1.3. Temporal discretization. If we consider the semi-discrete form of the Navier-Stokes equations, which can be obtained by applying method of lines, i.e., applying separate spatial and temporal discretization, we get

$$\Omega_i \frac{d\vec{q}_i}{dt} = -\vec{R}_i, \tag{4.1.31}$$

where the vector $\vec{R}_i$ denotes the cell residual. It is computed by the summation of convective and diffusive fluxes over the cell faces of a control volume. In the following, vector notation will be dropped for simplicity and the cell residual and the vector of conservative variables will be simply denoted by R_i and q_i.

The time derivative in the semi-discrete form can be approximated by the following non-linear scheme [**Hir07**]:

$$\frac{\Omega_i \bar{M}}{\Delta t_i}\Delta q_i^n = -\frac{\beta}{1+\omega}R_i^{n+1} - \frac{1-\beta}{1+\omega}R_i^n + \frac{\omega}{1+\omega}\frac{\Omega_i \bar{M}}{\Delta t_i}\Delta q_i^{n-1}, \tag{4.1.32}$$

where the state vector at the time iteration $n+1$ is updated as

$$q_i^{n+1} = q_i^n + \Delta q_i^n. \tag{4.1.33}$$

The matrix $\bar{M}$ in Eq. (4.1.32) is called as the mass matrix, which can be replaced by the identity matrix without disturbing the temporal accuracy in a cell-centered scheme. Depending on the value of parameters β and ω, the temporal scheme given in Eq. (4.1.32) has an explicit or an implicit character. For an explicit scheme, value of Δq_i^n is only a function of the state vector at time iteration n. Therefore, it can be easily evaluated by using the values from the previous iteration. For an implicit scheme, however, this term also depends on the values at the new time iteration $n+1$, and can be evaluated by solving a system of linear equations.

Depending on the type of problem, the time steps used in the time integration are either real or pseudo time-steps. If the flow has a steady-state solution, the time history is not important, and therefore largest possible values for the time steps can be taken. In this case, time steps are called as pseudo time-steps since their values are only important for the numerical stability. On the other hand, if the flow has an unsteady character, then the time history must be resolved correctly. As a result, real time-steps must be taken small enough to resolve the transient flow phenomena. For unsteady compressible flows, the dual time-stepping approach [**VM96**] is frequently used in density-based solvers. In this method, a steady-state problem is solved using pseudo time-stepping at each physical time step.

A basic explicit temporal scheme can be derived by setting $\beta = 0$ and $\omega = 0$, which gives the explicit Euler scheme:

$$\Delta q_i^n = -\frac{\Delta t_i}{\Omega_i}R_i^n. \tag{4.1.34}$$

The explicit Euler scheme is easy to implement, but it is difficult to use it for practical CFD problems due stability problems. In general, multistage (Runge-Kutta) schemes are more widely used. For example, a Runge-Kutta scheme with m stages is given below:

(4.1.35)

$$\begin{aligned}
q_i^{(0)} &= q_i^n \\
q_i^{(1)} &= q_i^{(0)} - \alpha_1 \frac{\Delta t_i}{\Omega_i}R_i^{(0)} \\
&.. \\
&.. \\
q_i^{(m)} &= q_i^{(0)} - \alpha_m \frac{\Delta t_i}{\Omega_i}R_i^{(m-1)} \\
q_i^{n+1} &= q_i^{(m)}
\end{aligned}$$

Using the Runge-Kutta scheme, the state solution at time iteration $n+1$ is found by using m intermediate stages. In the above scheme, values of $\alpha_1, ..., \alpha_m$ are the

stage coefficients of the Runge-Kutta scheme and $R_i^{(m-1)}$ denotes the cell residual, which is evaluated by using the current value of state vector at the stage $m-1$.

In CFD solvers, explicit temporal schemes are very practical to use since they are easy to implement and they have less memory and run-time requirements per time iteration. The major disadvantage of them is their restricted numerical stability. Regarding the stability of the explicit schemes, the time step Δt_i plays the the most important role. It is chosen according to the so-called CFL (Courant-Friedrich-Levy) condition, which states that the domain of dependence of the numerical scheme has to include the domain of dependence of the partial differential equation. The meaning of the CFL condition for a basic explicit scheme is that the time step should be equal to or smaller than the time required to transport the information across the stencil of the spatial discretization scheme. For example, according to the CFL condition, one can choose the time-step for the $2D$ Euler equations using

$$\Delta t_i = \sigma \frac{\Omega_i}{(\lambda_c^x + \lambda_c^y)}. \tag{4.1.36}$$

In the above equation, σ denotes the CFL number and its value depends on the temporal and spatial discretization scheme. The parameters λ_c^x and λ_c^y are the maximum eigenvalues of the convective flux Jacobian in the x and y directions respectively.

Although computational cost of explicit schemes per time iteration is less than the implicit schemes, implicit time-schemes are also used frequently, especially for steady-state problems. The main reason for this is the fact that implicit schemes have better stability properties. If the value of β in Eq. (4.1.32) is set to value other than zero, one obtains an implicit time integration scheme. For steady-state problems, a simple implicit scheme is obtained by setting $\omega = 0$ and $\beta = 1$:

$$\Delta q_i^n = -\frac{\Delta t_i}{\Omega_i} R_i^{n+1}. \tag{4.1.37}$$

Note that, in the above equation R_i^{n+1} is also unknown. If it is linearized about the current time iteration n, we get

$$R_i^{n+1} \approx R_i^n + \frac{\partial R_i}{\partial q_i} \Delta q_i^n. \tag{4.1.38}$$

If the linearization is substituted in Eq. (4.1.37), we obtain the implicit Euler scheme:

$$\left(\frac{\Omega}{\Delta t} I + \frac{\partial R_i}{\partial q_i} \right) \Delta q_i^n = -R_i^n. \tag{4.1.39}$$

The matrix $\partial R_i / \partial q_i$ in the above equation is obtained by linearizing the cell residual with respect to state variables and is referred to as the cell flux Jacobian. It depends only on the spatial discretization scheme. Furthermore, it is a sparse matrix since the stencil of the spatial discretization contains only cells in the close neighborhood of the ith cell. As a result, the above equation represents a sparse system of linear equations, which can be solved most efficiently by using iterative methods. The popular methods, which are used for solving sparse linear systems in compressible Navier-Stokes solvers, are: Alternating Direction Implicit (ADI)

[**BM77**], Lower-Upper Symmetric Gauss-Seidel (LU-SGS) [**JY87**], Conjugate Gradient Squared (CGS) [**Son89**], Bi-Conjugate Gradient Stabilized (Bi-CGSTAB) [**vdV92**] and Generalized Minimal Residual (GMRES) [**SS86**] methods.

In conclusion, implicit time discretization schemes have the advantage that they have less stringent stability restrictions, therefore, large time steps can be taken in order to reach the steady-state as quick as possible. On the other hand, the main disadvantage is the high computational cost for the solution of the linear system in each pseudo time-step and the high memory requirements that are needed to store the flux Jacobian matrix.

4.1.4. Convergence acceleration techniques. In order to accelerate the convergence rate for steady-state problems, several techniques have been suggested in the past. A review of these methods can be found in [**Mav98**]. One of the acceleration techniques is the local time-stepping. In this method, integration in time is made using the largest possible time step for each control volume in the flow domain instead of using a fixed time-step. As a result, convergence of the flow solver can be accelerated, but the transient solution is no longer accurate. Therefore, this method is used only for steady-state problems.

Another powerful method to increase the convergence rate is the implicit residual smoothing (IRS), which was introduced by Jameson and Baker [**JB83**]. The aim of IRS method is to give an explicit numerical scheme implicit character, and hence to increase the maximum allowable CFL number. For example, the central implicit residual smoothing scheme for $2D$ structured grids is given by

$$\begin{aligned} -\epsilon_x R^*_{I-1,J} + (1+2\epsilon_x) R^*_{I,J} - \epsilon_x R^*_{I+1,J} &= R_{I,J} \\ -\epsilon_y R^{**}_{I,J-1} + (1+2\epsilon_y) R^{**}_{I,J} - \epsilon_y R^{**}_{I,J+1} &= R^*_{I,J} \end{aligned}$$

Here the indices I and J are cell indices in x and y directions respectively and $R^*_{I,J}$ and $R^{**}_{I,J}$ are the smoothed cell residuals for the cell with the index (I, J). The parameters ϵ_x and ϵ_y are called as the smoothing coefficients and suitable values for them can be found in literature (e.g, [**TSVW91**]). The resulting system of equations, which is obtained from the above scheme, is solved by most efficiently using the Thomas (TDMA) algorithm.

4.1.5. Boundary conditions. The boundary conditions, which are most frequently used for airfoil simulations are:

- Far-field boundary condition: Since the flow past an airfoil is an external flow, far-field boundary condition is most suitably used for the outer domain. The far-field boundary can be thought as the outer part of the computational domain, where the disturbances on the airfoil geometry does not have any influence. In order to satisfy this requirement, the distance from the airfoil to the far-field boundary should be large enough. Typically, a distance of $15-20$ times of the chord length is used. Another important issue is the local Mach number of the inflow and the outflow. For supersonic inflow and outflow, all the eigenvalues of the flux Jacobian have the same sign and there are four incoming characteristics. In this case, conservative variables on the far-field boundary can be set equal to the free-stream values. If the inflow is subsonic, one characteristics is

outgoing, and therefore one of the characteristic variables should be interpolated from the flow domain whereas the remaining three can be set equal to the free-stream values.

- Wall boundary condition: As far as the wall boundary condition is concerned, the treatment for Euler equations differs significantly from the Navier-Stokes equations. Since the viscous terms are neglected in Euler equations, boundary layer development is not captured. Therefore, slip boundary condition is used in Euler equations. In this case, normal velocity component is set to zero since the wall surface is assumed to be inpermeable. Consequently, the flow must be tangential at the wall, i.e., the contravariant velocity must be zero:

$$\vec{U} \cdot \vec{n} = 0 \Rightarrow V = 0. \tag{4.1.40}$$

Applying the above condition, convective flux vector on the wall reduces to:

$$F_c = \begin{pmatrix} 0 \\ n_x p \\ n_y p \\ 0 \end{pmatrix}. \tag{4.1.41}$$

For the viscous fluid past a solid body, on the other hand, the relative velocity between the wall surface and the fluid is directly taken as zero at the surface. Therefore, this boundary condition is called as the no-slip boundary condition. If the wall surface stationary, Cartesian velocity components become zero there:

$$u = v = 0. \tag{4.1.42}$$

- Cut boundary condition: The cut boundary condition is used only for structured grids. The coordinate cut represents an artificial boundary, which is composed of grid points with different computational coordinates but same physical location. This means that the computational grid is folded in such a way that it touches itself on the cut boundary. This is typically the case when grids with the so-called C- or 0-grid topology are used. The state variables and their gradients have to stay continuous across the cut for a consistent spatial discretization. The cut boundaries are typically implemented using dummy cells on the boundary. These dummy cells are the computational cells, which do not exist in the physical domain. They are only used to ease boundary treatment. In Fig. 3, cut and far-field boundaries of a structured C-grid around an airfoil geometry are illustrated.

4.1.6. Aerodynamic coefficients. In order to compute the aerodynamic coefficients of an airfoil, pressure is transferred into the dimensionless pressure coefficient C_p, defined as

$$C_p = \frac{2(p - p_\infty)}{\gamma M_\infty^2 p_\infty}, \tag{4.1.43}$$

where M_∞ is the free-stream Mach number and p_∞ is the free-stream pressure. Pressure coefficient is then used to evaluate drag and lift coefficients, which are

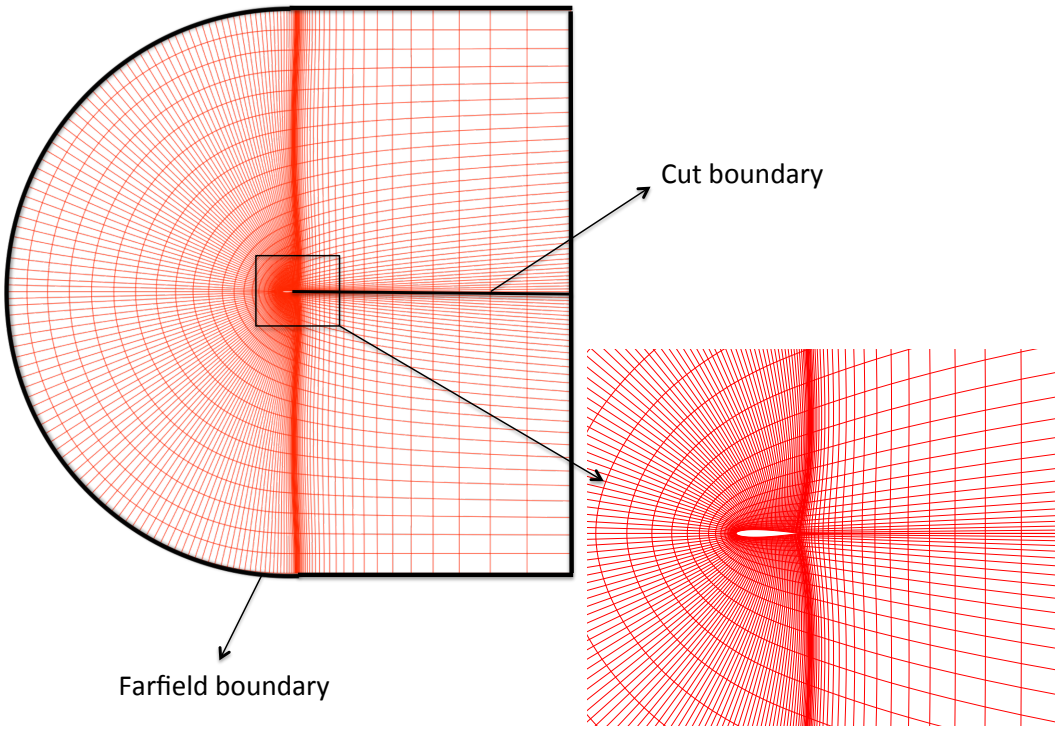

Figure 3. *Structured C-type grid for the airfoil geometry.*

given by

$$C_D = \frac{1}{C_{\text{ref}}} \int_{\partial\Omega_{wall}} C_p(n_x \cos\alpha + n_y \sin\alpha)\, dS \tag{4.1.44}$$

and

$$C_L := \frac{1}{C_{\text{ref}}} \int_{\partial\Omega_{wall}} C_p(n_y \cos\alpha - n_x \sin\alpha)\, dS, \tag{4.1.45}$$

where, α is the angle of attack and C_{ref} is the reference area of airfoil.

In case, viscous effects are taken into account, the skin friction coefficient should be added to the form drag. The local skin friction coefficient is given by

$$C_f = \frac{2\tau_w}{\gamma M_\infty^2 p_\infty}, \tag{4.1.46}$$

where τ_w is the local wall shear stress. In order to find the contribution to the overall drag coefficient, local skin friction coefficient is integrated along the wall boundary.

4.2. Pressure-based schemes

In aerodynamics, air can be assumed to be incompressible for low speeds ($Ma < 0.3$). In this case, the Navier-Stokes equations, which consist of three equations and three unknowns in $2D$, can be solved without the energy equation. In $2D$, the incompressible Navier-Stokes equations are composed of continuity equation and two momentum equations in x and y directions. These equations are decoupled from each other since there exists no transport equation for the pressure p. For this reason, incompressible equations cannot be solved in a strong coupled way as the compressible Navier-Stokes equations are solved. In fact, transport equation for each variable is discretized and solved separately by treating the other variables as constants. In the following, basic discretization and solution schemes using FVM method for the incompressible Navier-Stokes equations are covered. Since each equation is treated separately, it is easier to introduce first the generic transport equation in conservation form and illustrate the discretization schemes on this equation. The generic transport equation for a flow quantity ϕ is given as

$$\frac{\partial(\rho\phi)}{\partial t} + \frac{\partial(\rho U_i\phi)}{\partial x_i} = \frac{\partial}{\partial x_i}\left(\Gamma\frac{\partial\phi}{\partial x_i}\right) + Q_\phi, \tag{4.2.1}$$

where Γ is the diffusion coefficient of the quantity ϕ and Q_ϕ is the source term. Note that, from the above equation one can easily obtain the continuity equation by taking $\phi = 1$, $\Gamma = 0$ and $Q_\phi = 0$. If the pressure term is treated non-conservatively, momentum equations can be similarly obtained by taking $\phi = U_j$, $\Gamma = \mu$ and $Q_\phi = -\frac{\partial p}{\partial x_i}$.

In particular, integral form of the generic transport equation is more useful for the FVM. It can be easily derived by taking the volume integral of the differential transport equation over a control volume Ω and applying Gauss theorem:

$$\frac{\partial}{\partial t}\int_\Omega(\rho\phi)d\Omega + \int_S \rho\phi\vec{U}\cdot\vec{n}dS = \int_S \Gamma\frac{\partial\phi}{\partial x_i}\cdot\vec{n}dS + \int_\Omega Q_\phi\, d\Omega. \tag{4.2.2}$$

In the following, we shortly introduce the most common ways of discretizing surface and volume integrals, which appear in the above generic transport equation.

4.2.1. Discretization of surface and volume integrals. Surface integrals can be computed by the summation of the flux integrals over the faces of a control volume:

$$\int_S \rho\phi\vec{U}\cdot\vec{n}\, dS = \sum_k \int_{S_k} (\rho\phi\vec{U})\cdot\vec{n}\, dS_k. \tag{4.2.3}$$

For a rectangular CV in $2D$ (see Fig. 4), we have the summation of four integrals given by:

$$\int_S \rho\phi\vec{U}\cdot\vec{n}\, dS = \sum_{k=w,e,n,s} \int_{S_k} (\rho\phi\vec{U})\cdot\vec{n}\, dS_k, \tag{4.2.4}$$

where $(\rho\phi\vec{U})\cdot\vec{n}$ is the flux vector that represents the transport of ϕ across the face. Note that, this term can be considered as a convective flux. In contrast to the compressible methods, in which resolution of shocks are of extreme importance, usually much simpler discretization schemes for convective terms are used in incompressible equations.

One method to approximate the surface integral is the midpoint rule. In this method, the flux is approximated by using the values located in the middle of the face. For example, for the east cell face ('e'), we have:

$$\int_{S_e} (\rho\phi\vec{U}\cdot\vec{n}) \approx (\rho\phi\vec{U}\cdot\vec{n})_e \, S_e. \tag{4.2.5}$$

If the velocity field is know, we need only the value of ϕ_e to compute $(\rho\phi\vec{U}\cdot\vec{n})_e$. This value can be approximated by using the central difference scheme (CDS) between the nodal values at E and P:

$$\phi_e = \phi_E \frac{\Delta x_e}{\Delta x_P} + \phi_P \left(1 - \frac{\Delta x_e}{\Delta x_P}\right). \tag{4.2.6}$$

The central differencing scheme is second order accurate, but it may produce oscillatory solutions. Another widely used interpolation method is the upwind interpolation or upwind difference scheme (UDS). In contrast to CDS, UDS takes the direction of the velocity into account:

$$\phi_e = \phi_P \text{ if } (\vec{U}\cdot\vec{n})_e > 0, \tag{4.2.7}$$

$$\phi_e = \phi_E \text{ if } (\vec{U}\cdot\vec{n})_e < 0. \tag{4.2.8}$$

The UDS method has the boundedness property, and therefore it is especially used for the convective terms to avoid oscillatory solutions. On the other hand, UDS is a first order scheme and the leading truncation error is in the same order as a diffusive flux. For this reason, this error term is also known as the "false diffusion" term. To diminish the accuracy loss due to false diffusion, central differencing and upwind differencing schemes are blended. For example, the deferred correction method, in which upwind term is treated implicitly and central differencing term is treated explicitly, is widely used. Interpolation of ϕ_e using deferred correction is given as

$$\phi_e = \beta\phi_e^{CDS} + (1-\beta)\phi_e^{UDS}, \tag{4.2.9}$$

where β is the scalar blending factor.

Alternative to UDS and CDS, there exists also high order schemes such as the quadratic upwind interpolation (QUICK) [**Leo79**] scheme, in which a larger stencil is used for the interpolation to increase the accuracy.

The volume integrals can be approximated by the product of the mean value of ϕ and the cell volume. Usually, value at the cell center is taken as the mean value since it is easily available without using any interpolation. As a result, the volume integral in Eq. (4.2.2), can be approximated by:

$$\int_\Omega Q_\phi \, d\Omega \approx Q_{\phi,P} \, \Omega_i. \tag{4.2.10}$$

4.2.2. Implicit pressure-correction methods. The fact that pressure does not appear in the continuity equation and the presence of nonlinear terms in the momentum equations causes extra difficulties for the solution of incompressible Navier-Stokes equations. Although, compressible Navier-Stokes equations can be solved in a strong coupled way, this is not the case for the incompressible Navier-Stokes system. For the compressible case, density and pressure are strongly linked by the state equation (e.g., ideal gas law), and the continuity equation is used as

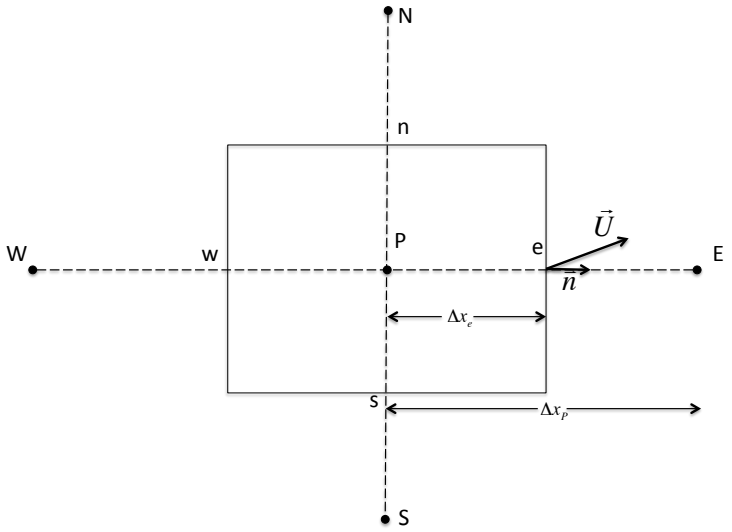

Figure 4. *A rectangular CV in* $2D$.

a transport equation for the density. For the incompressible case, however, density is taken as constant, and therefore is not an independent state variable. For this reason, there is no separate transport equation for the pressure in the incompressible Navier-Stokes system. Since nonlinear convective terms are present in the momentum equations, using iterative solution schemes is the proper strategy to construct a coupling mechanism between the momentum equations and the continuity equation. The most prominent example of velocity-pressure coupling schemes is the SIMPLE (Semi-Implicit Method for Pressure Linked Equations) scheme, which was suggested by Caretto et al. [**CGPS73**]. The derivation of the SIMPLE method, which is outlined shortly in the following, can be found in more detail in [**FP01**].

Consider the discrete form of the momentum equations written for the velocity component U_i at the time iteration $t+1$:

$$A_P U_{i,P}^{t+1} + \sum_l A_l U_{i,l}^{t+1} = Q^{t+1} - \left(\frac{\delta p}{\delta x_i}\right)^{t+1}, \tag{4.2.11}$$

where A_P is the entry in the coefficient matrix for the unknown velocity U_i for the cell P and A_l are the entries for the velocity components that belong to the neighboring cells. The source term, which is denoted by Q, contains all constant terms, e.g., any linearized term that can be explicitly computed. In Eq. (4.2.11), the pressure gradient term is written separately to emphasize the influence of the pressure in the momentum equations. Note that, in contrast to compressible equations, only the gradient, not the absolute value of pressure is important. Since pressure gradient at time $t+1$ is not known, the momentum equation cannot be solved directly. At this point, we introduce the *outer iterations* within a single time

iteration, which give an estimate of U_i at the time iteration $t+1$:

$$A_P U_{i,P}^{m*} + \sum_l A_l U_{i,l}^{m*} = Q^{m-1} - \left(\frac{\delta p}{\delta x_i}\right)^{m-1}. \tag{4.2.12}$$

In the above equation, we use an outer iteration counter m instead of time-steps. The momentum equations are solved sequentially so that for each velocity component a system of linear equations must be solved. The iterations of these linear systems are called as the *inner iterations*. In Eq. (4.2.12), source and pressure gradient terms are calculated by using the values from the previous outer iteration $m-1$, therefore, the solution method is semi-implicit. Note that, the velocity components that are calculated from the Eq. (4.2.12) do not satisfy the continuity equation. Therefore, they must be corrected to enforce the mass conservation. The way how one can achieve this is to derive an equation for the pressure correction terms by combining momentum equations and continuity equation. The estimated velocity component at the node P is obtained by solving (4.2.12):

$$U_{i,P}^{m*} = U_{i,P}^{\tilde{m}*} - \frac{1}{A_P}\left(\frac{\delta p}{\delta x_i}\right)^{m-1}, \tag{4.2.13}$$

where

$$U_{i,P}^{\tilde{m}*} = \frac{Q^{m-1} - \sum_l A_l U_{i,l}^{m*}}{A_P}. \tag{4.2.14}$$

The next step is to correct the velocity field such that the new velocity field at the outer iteration m satisfies the continuity equation. This can be achieved if the pressure field is constructed in such that way that the equation:

$$U_{i,P}^{m} = U_{i,P}^{\tilde{m}*} - \frac{1}{A_P}\left(\frac{\delta p}{\delta x_i}\right)^{m} \tag{4.2.15}$$

is satisfied. Note that the pressure field is now constructed by exchanging the pressure term in Eq. (4.2.13) with the constructed pressure field p^m, which enforces mass conservation. Applying divergence operator to the Eq. (4.2.15), we obtain the Poisson's equation for pressure:

$$\frac{\delta(U_{i,P}^{m})}{\delta x_i} = \frac{\delta(U_{i,P}^{\tilde{m}*})}{\delta x_i} - \frac{\delta}{\delta x_i}\left[\frac{1}{A_P}\left(\frac{\delta p}{\delta x_i}\right)^{m}\right] = 0 \tag{4.2.16}$$

$$\Rightarrow \frac{\delta(U_{i,P}^{\tilde{m}*})}{\delta x_i} = \frac{\delta}{\delta x_i}\left[\frac{1}{A_P}\left(\frac{\delta p}{\delta x_i}\right)^{m}\right]. \tag{4.2.17}$$

The Poisson's equation can be discretized in the same way as one can do with momentum equations, and can be solved using any of the iterative methods for linear equations. After solving the pressure, a velocity field that satisfies the continuity equation can be constructed by using Eq. (4.2.15). At this step, mass conservation is satisfied but the new velocity and pressure fields do not satisfy momentum equations. Therefore, an iterative process is necessary to make sure that outer iterations are executed until both momentum and continuity equations are satisfied. These class of methods, in which first a velocity field is constructed and then it is corrected to enforce mass conservation are known as projection methods.

In SIMPLE method, the key idea is to use pressure-correction terms instead of the actual pressures. The correction for pressure and velocity fields can be written as

$$p^m = p^{m-1} + p^c \text{ and } U_i^m = U_i^{m*} + U_i^c, \tag{4.2.18}$$

where p^c and U_i^c are the correction terms for pressure and velocities respectively. If the momentum equations are written the above expressions, we get:

$$U_i^c = \tilde{U}_i^c - \frac{1}{A_P}\left(\frac{\delta p^c}{\delta x_i}\right), \tag{4.2.19}$$

where $\tilde{U}_c$ is defined as

$$\tilde{U}_i^c = -\frac{\sum_l A_l U_{i,l}^c}{A_P}. \tag{4.2.20}$$

Since it is aimed to enforce the mass conservation for the outer iteration m, we obtain from the continuity equation:

$$\frac{\delta U_i^m}{\delta x_i} = \frac{\delta(U_i^{m*})}{\delta x_i} + \frac{\delta(U_i^c)}{\delta x_i} = 0. \tag{4.2.21}$$

Taking the divergence of the Eq. (4.2.19) and substituting it into the continuity equation, we obtain the Poisson equation for the pressure corrections:

$$\frac{\delta(U_i^{m*})}{\delta x_i} + \frac{\delta \tilde{U}_i^c}{\delta x_i} = \frac{\delta}{\delta x_i}\left[\frac{1}{A_P}\left(\frac{\delta p^c}{\delta x_i}\right)^m\right]. \tag{4.2.22}$$

The velocity corrections $\tilde{U}_i^c$ in the above equation are unknown so they are simply neglected in SIMPLE algorithm. After the pressure correction term is computed by solving Eq. (4.2.22), velocity components can be corrected by using Eq. (4.2.19):

$$U_i^c = -\frac{1}{A_P}\left(\frac{\delta p^c}{\delta x_i}\right) \tag{4.2.23}$$

Note that, neglecting $\tilde{U}_i^c$ term in the pressure-correction equation may lead to large oscillations in pressure corrections, which may cause a slowdown in convergence. To improve the convergence of the method, under-relaxation for pressure corrections is used:

$$p^m = p^{m-1} + \alpha p^c, \tag{4.2.24}$$

where $\alpha \in [0, 1]$. Finding the optimal value of the under-relaxation parameter α is not straightforward since its influence on the convergence is highly problem dependent. This difficulty is one of the major drawbacks of the SIMPLE method.

There are other more refined algorithms than SIMPLE, which approximate the term $\tilde{U}_i^c$ by a weighted average of the neighboring points rather than completely neglecting it (e.g., SIMPLEC [**vDR84**]). Other algorithms (e.g., SIMPLER [**Pat80**]) use another corrector step for the pressure corrections to improve the convergence.

Another important issue in pressure-velocity coupling schemes is the choice of grid arrangement. If all the state variables are saved in the cell center, we speak about the *collocated arrangement*. The other approach is to use a *staggered arrangement*, where different variables are kept at different locations. Collocated and staggered arrangements are shown in the Figs. 5 and 6 for a simple $2D$ control volume. The centers of the neighboring cells are denoted with N(north), S(south),

W(west) and E(east) respectively. In a staggered arrangement, the y component of the velocity is saved at the north face (n). The x component, on the other hand, is saved on the east face (e). The pressure p is saved in the cell center in both arrangements. The idea behind using a staggered arrangement is to avoid errors, which may appear if central differencing scheme (CDS) is used for spatial discretization. As an example, consider the pressure term $\partial p/\partial x_i$ in the momentum equations. If a uniform grid with equidistant grid spacing ($\Delta x = const, \Delta y = const, S_e = S_w = \Delta y$) is used, pressure term in the x direction is given as

$$\int_\Omega \frac{\partial p}{\partial x} = \int_S pi_x \cdot ndS = (p_e - p_w)\Delta y. \tag{4.2.25}$$

If the pressure values in the middle of the east and west faces are approximated by using CDS, we have

$$\int_S pi_x \cdot ndS = \frac{(p_E + p_P)}{2}\Delta y - \frac{(p_P + p_W)}{2}\Delta y = \frac{p_E - p_W}{2}\Delta y. \tag{4.2.26}$$

Similarly in the y direction we have

$$\int_S pi_y \cdot ndS = \frac{p_N - p_S}{2}\Delta x. \tag{4.2.27}$$

If the pressure field is oscillating, we may have $p_E \approx p_W$ and $p_N \approx p_S$. In this case, both pressure terms are very close as zero, which may lead to serious stability problems of the pressure-velocity coupling scheme. By using a staggered arrangement and different control volumes for each momentum equation, this problem can be avoided. In Fig. 7, different control volumes for each momentum equation in $2D$ are illustrated. Note that, for this arrangement pressure values on the control volume faces are available and therefore can be directly used. This is very advantageous since no interpolation in the spatial discretization is necessary. The staggered arrangement also prevents the unphysical behavior of the solution scheme with a highly oscillatory pressure field. As a result, a strong pressure-velocity coupling is achieved.

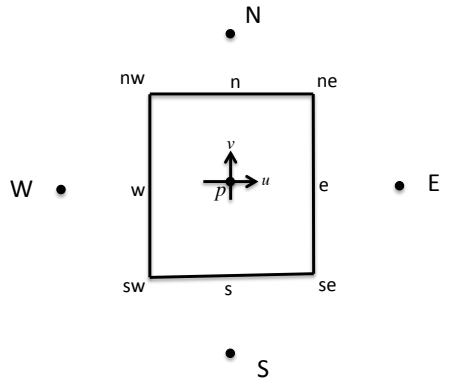

Figure 5. *Collocated arrangement.*

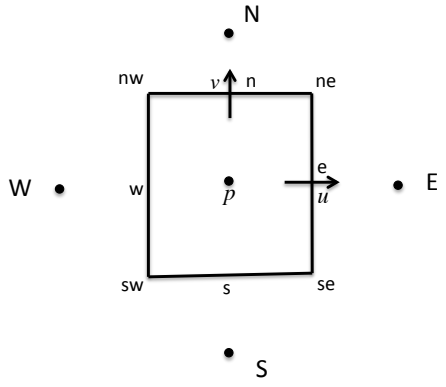

Figure 6. *Staggered arrangement.*

Because of its good stability, staggered arrangement was favored in the past for implicit pressure-velocity coupling schemes. One of the major disadvantages

of this arrangement is the high effort required for the implementation. Therefore, thanks to its simplicity, collocated arrangement gained its popularity over the past decades. In order to solve the stability problems of the collocated arrangement, an alternative interpolation method is necessary, which does not suffer from the errors due to an oscillatory pressure field. Different methods are available in the literature, out of which Rhie - Chow interpolation [**RC83**] is worth to mention here.

In Rhie - Chow interpolation method, interpolated velocity at the cell face is corrected by using the difference between the pressure gradient and the interpolated pressure gradient at the cell face. Therefore, oscillations in pressure are better smoothed out. For example, at the east cell face "e", the velocity is corrected by [**FP01**]

$$U_{i,e}^{m*} = \overline{U}_{i,e}^{m*} - \Delta\Omega_e \overline{\left(\frac{1}{A_p}\right)}_e \left[\left(\frac{\delta p}{\delta x_i}\right)_e - \overline{\left(\frac{\delta p}{\delta x_i}\right)}_e\right]^{m-1}. \tag{4.2.28}$$

In the above equation the bar operator $(\overline{\cdot})$ denotes the interpolated value and $\Delta\Omega_e$ is the volume that is centered around a cell face, i.e., $\Delta\Omega_e = (x_E - x_P)\Delta y$ for Cartesian grids.

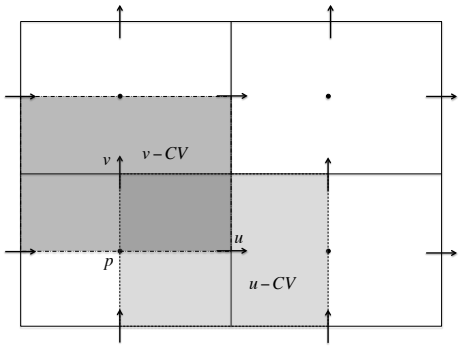

Figure 7. *Different control volumes for the staggered arrangement.*

Finally, we summarize the general structure of a pressure-based solver with pressure-velocity coupling in the following pseudo-code.

ALGORITHM 5. *Pressure-based solution method for incompressible Navier-Stokes equations*

initialize *velocity field* U_i *and pressure field* p
for $t = T_0, t \leq T_N$ ***do***
  for $m = 0, m \leq m_{max}$ ***do***
    SOLVE discretized x-momentum equation for u
    SOLVE discretized y-momentum equation for v
    Compute uncorrected mass fluxes at cell faces
    SOLVE pressure correction equation
    Correct pressure field

Correct face mass fluxes
Correct cell velocities
if *(* $(\|U_i^m - U_i^{m-1}\| \leq \epsilon)$ ***and*** $(\|p^m - p^{m-1}\| \leq \epsilon)$ *)* ***then***
break
end if
end for
end for

In the above algorithm, each SOLVE statement corresponds to the inner iterations of the linear solver with fixed coefficients A_P and A_l. In each outer iteration of the m loop, these coefficients are sequentially updated by using velocity and pressure fields from the previous outer iteration. The linear equation systems, which are obtained in this way for each transport variable, can be solved with an iterative method. A popular iterative method for the pressure-based solvers is the SIP (Strongly Implicit Procedure) or Stone's method [**Sto68**], which is specially designed for PDEs. The SIP method is based on the incomplete LU decomposition, which approximates the exact LU decomposition of the coefficient matrix. As far as the implementation is concerned, the coefficients are saved as one dimensional arrays for each diagonal A_P, A_W, A_E, A_S and A_N to reduce the memory demand. In each outer iteration, the linear systems are solved inexactly, and therefore only a moderate number of inner iterations are performed. The outer iterations are performed until a reasonable convergence is achieved (i.e., correction terms are close to zero). Therefore, on the outer iteration level, we have a fixed point solution of the state vector $y = (U_i, p)$, which can be denoted by $y_* = G(y_*)$. The fixed point iterator G includes all the steps that are performed in one outer iteration (SIMPLE scheme, Rhie-Chow interpolation, inner iterations etc.). At the outermost loop, we have the time iterations denoted by t. Note that in contrast to outer iterations, there exists no fixed-point on that level. If the flow has a steady-state solution, the unsteady terms in the spatial discretization can be omitted in the momentum equations. In this case, only outer the iterations are performed until the convergence of the pressure-velocity coupling loop is achieved within a single time-step.

4.2.3. Boundary conditions. Although physical boundary conditions does not change in the incompressible equations, numerical treatment is different compared to compressible equations. At the inflow boundary usually the convective and diffusive fluxes are specified for the momentum equations. At the outflow, either these values can be specified or the normal component of the velocity gradient can be set to zero, similar to the treatment of the symmetry boundaries.

At the wall boundary, no-slip boundary condition is used. This means that velocity components are zero at the wall (i.e., fluid particles stick to the wall). For the Finite Volume method, the diffusive flux can be set to zero since the normal viscous stress is zero at the wall. If we consider a control volume the near the wall boundary as shown in Fig. 8, we have:

$$\frac{\partial u}{\partial x} = 0. \tag{4.2.29}$$

From the continuity equation, it follows:

$$\frac{\partial v}{\partial y} = 0 \Rightarrow \tau_{yy} = 0. \tag{4.2.30}$$

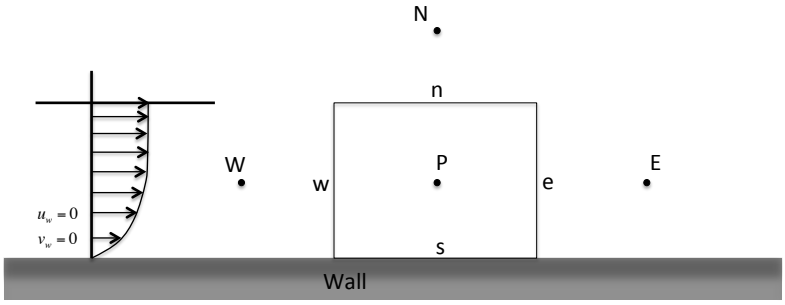

Figure 8. *Velocity profile at the wall boundary and a near boundary control volume.*

The diffusive flux for the y momentum equation at the wall boundary (south boundary in the figure) is then:

$$\int_s \tau_{yy} dS_s = 0. \tag{4.2.31}$$

The shear stress term that is needed for the x momentum equation can be computed using one-sided approximation of the derivative term $\partial u/\partial y$:

$$\frac{\partial u}{\partial y} \approx \frac{u_P}{y_P - y_S}. \tag{4.2.32}$$

Note that, for the pressure-correction equation boundary conditions are also needed. At the inlet usually the mass flux is specified so the mass flux correction is also zero there. This condition can directly be used for the derivation of the pressure-correction equation, which correspond to the homogenous Neumann boundary condition.

At the wall boundaries, the mass flux is zero since the wall boundaries are assumed to be impermeable. Therefore the pressure and pressure correction values can be linearly extrapolated to the wall boundary.

At the outlet, the velocities are usually extrapolated from the inner cells to the boundary when the outflow boundary is far enough from the region of interest. The extrapolated velocities are then corrected to satisfy the mass balance between inlet and the outlet, which cannot be guaranteed by the extrapolation. In case, homogenous Neumann boundary condition is applied for the pressure correction equation the solution is not unique. Because of this reason usually a reference pressure at some point is kept fixed and pressure correction calculated at this point is subtracted from the pressure correction in each cell.

4.3. Turbulence modeling

A very important issue in CFD is the presence of turbulence in the flow, which brings a great deal of extra complexity to the numerical treatment. Especially high Reynolds number flows are associated with an unsteady and chaotic behavior of the state variables, which is an indication of turbulence. Even though giving an exact definition of turbulence is hard, some symptoms associated with turbulence can be given. These are: presence of vorticity with diverse length scales, high amount of mixing and molecular transport, unsteady fluctuations of velocity and pressure in

all three-dimensions and presence of a broad range of length and time scales in the flow.

Based on the application, turbulence might be desired or undesired phenomena. For example, drag of an airfoil increases significantly when the flow regime changes from laminar to turbulent, and therefore has a negative influence on the performance of airfoil. On the other hand, in the flow around a blunt body, turbulence reduces the drag by shifting the separation point further to downstream. For example a golf ball with dimples has less drag than a smooth ball since the dimples on the ball surface promote turbulence. In majority of the industrial applications, flow regimes are turbulent, therefore predicting turbulent flows with a reasonable accuracy is an important issue in CFD.

Turbulence and turbulence modeling is a very broad subject, therefore only a short summary is provided in the following. For detailed information, reader may refer to [**Pop00, Wil04, Jov04**]. In general, there are three main approaches used simulating turbulent flows:

- Direct Numerical Simulation (DNS) : In this approach, unsteady Navier-Stokes equations are directly solved such that all length and time scales that are present in the flow are resolved. The computational domain must be at least large enough as the largest turbulent eddy, which is quantified with the integral scale of turbulence (L). On the other hand, computational grid must be fine enough such that kinetic energy dissipation caused by the smallest turbulent eddies is captured. This length scale is called as the Kolmogorov scale, which is denoted by η. Tennekes and Lumley [**TL72**] showed that L/η in a flow is proportional to $Re_L^{3/4}$, Re_L being the Reynolds number based on the integral length scale. Using this relation, one can estimate that the number of grid points in one coordinate dimension must be at the same order with $Re_L^{3/4}$. Since the same number of grid points must be used in all three dimensions, and number of time steps is also related to the grid size due to stability reasons, the total cost of simulation is proportional with Re_L^3. As a consequence, the cost of direct numerical simulation is beyond the capabilities of the available computing power for high Reynolds number flows. For this reason, direct numerical simulations are performed for the Reynolds numbers up to some thousands and for relatively simple geometries, mainly for the purpose of to collecting data for turbulence research.
- Large eddy simulation (LES): The main idea behind the LES is the assumption that small-scale eddies are isotropic and have a universal character, as suggested by the Kolmogorov's hypothesis of local isotropy. For this reason, it is easier to model small scales of turbulence than the large scales. In LES approach, a filtering operation is applied to the Navier-Stokes equations to filter out the small-scaled eddies. Therefore, only contributions of large, energy-carrying structures to momentum and energy transfer are simulated without using any turbulence model. The effects of the small structures, which are not resolved by the numerical scheme, are modeled. The main advantage of the LES is that coarser grids compared to DNS can be used, and therefore LES simulations are numerically cheaper than DNS. In particular, LES is usually employed for the high Reynolds number flows or complex geometries that cannot be resolved by

DNS. Although LES is a very good research tool, computational cost is still too high for practical usage in industrial applications.

- Reynolds-Averaged-Navier-Stokes (RANS) models: Usually in engineering design, the mean flow quantities are of interest rather than the instantaneous flow quantities. However, DNS and LES simulations provide a very detailed information about the flow, much more than what is actually required. In RANS approach, all of the unsteadiness due to turbulence is averaged out by applying the so-called Reynolds averaging. The resulting RANS equations are then solved using a turbulence model. The main advantage of the RANS approach is that much coarser grids compared to DNS and LES can be used. Furthermore, steady-state simulations can be performed, which have a computational cost significantly lower than the unsteady simulations. Thanks to these merits, RANS approach is extensively used in simulating turbulent flows for industrial problems. In the following, a brief summary of the RANS approach is provided.

The main philosophy behind the RANS models is the fact that one can decompose a flow variable ϕ into a mean value and a fluctuation about that value. This approach is called as the Reynolds averaging and it is given by

$$\phi(x_i,t) = \overline{\phi}(x_i) + \phi'(x_i,t). \tag{4.3.1}$$

In the above equation, $\overline{\phi}$ is the mean value of ϕ and it is found by:

$$\overline{\phi}(x_i) = \lim_{T\to\infty} \frac{1}{T}\int_0^T \phi(x_i,t)\ dt. \tag{4.3.2}$$

Note that by definition mean value of the fluctuations is zero, i.e., $\overline{\phi'} = 0$.

If the flow is unsteady, ensemble averaging is used instead of time-averaging:

$$\overline{\phi}(x_i) = \lim_{N\to\infty} \frac{1}{N}\sum_{n=1}^{N} \phi^n(x_i,t)\ dt, \tag{4.3.3}$$

where N is the number of members of the ensemble, which is ideally large enough to eliminate the effect of fluctuations.

If the density is not constant, e.g., as in compressible flows, it is advised using the Favre-averaging to certain quantities instead of Reynolds-averaging. The motivation for this to simplify the equations, which would otherwise become remarkably complex due to the additional correlations caused by density fluctuations. Favre averaging is given by

$$\tilde{\phi} = \frac{1}{\overline{\rho}} \lim_{T\to\infty} \frac{1}{T}\int_0^T \rho(x_i,t)\phi(x_i,t)\ dt, \tag{4.3.4}$$

where $\overline{\rho}$ denotes the Reynolds-averaged density. Favre-averaging leads to Favre-averaged Navier-Stokes equations, which are used for compressible flow simulations.

Fundamental rules of Reynolds-averaging can be derived using Eq. (4.3.2):

$$\overline{\psi+\phi} = \overline{\psi} + \overline{\phi},\ \overline{\psi\phi'} = 0\ , \overline{\overline{\psi}\phi'} = 0 \ \text{ and } \ \overline{\overline{\psi}\,\overline{\phi}} = \overline{\psi}\,\overline{\phi}, \tag{4.3.5}$$

where ϕ and ψ are two arbitrary flow quantities. Using the above identities, averaging applied to a multiplication results in

$$\begin{aligned} \overline{\psi\phi} &= \overline{(\overline{\psi}+\psi')(\overline{\phi}+\phi')} \\ &= \overline{\overline{\psi}\,\overline{\phi}+\overline{\psi}\phi'+\psi'\overline{\phi}+\psi'\phi'} \\ &= \overline{\psi}\,\overline{\phi}+\overline{\psi'\phi'}. \end{aligned} \tag{4.3.6}$$

As it can be seen from the above identity, averaging of a nonlinear term produces extra terms, which is one of the main difficulties in numerical treatment.

The RANS equations are obtained by averaging the Navier-Stokes equations, in which the velocity and pressure terms are replaced with mean values plus fluctuating parts, i.e., $U_j = \overline{U_J} + U'_J$ and $p = \overline{p} + p'$. The incompressible RANS equations (if body forces are neglected) are given as

$$\frac{\partial(\overline{U}_i)}{\partial x_i} = 0 \tag{4.3.7}$$

$$\frac{\partial(\rho\overline{U}_i)}{\partial t} + \frac{\partial}{\partial x_j}(\rho\overline{U}_i\overline{U}_j + \rho\overline{U'_iU'_j}) = -\frac{\partial\overline{p}}{\partial x_i} + \frac{\partial\overline{\tau}_{ij}}{\partial x_j}, \tag{4.3.8}$$

where $\overline{\tau}_{ij}$ denotes the mean viscous stress tensor that is given by

$$\overline{\tau}_{ij} = \mu\left(\frac{\partial\overline{U}_i}{\partial x_j} + \frac{\partial\overline{U}_j}{\partial x_i}\right) . \tag{4.3.9}$$

It can be observed that, due to the time-averaging of the nonlinear convective term, additional terms ($\rho\overline{U'_iU'_j}$) in RANS equations appear. These unknown terms are referred to as the Reynolds stresses. Hence, in RANS equations there are more unknowns than the number of equations leading to an underdetermined system. This problem is known as the closure problem of turbulence.

One of the main research fields of turbulence research is related with constructing turbulence models to model Reynolds stresses using mean flow quantities. Many turbulence models have been proposed in the past, and the majority of them make use of the Bousinessq eddy-viscosity assumption. According to eddy-viscosity assumption, momentum transport mechanism of turbulent velocity fluctuations is analog to the molecular momentum transport for an isotropic Newtonian fluid. Therefore, Reynolds stresses and mean flow quantities are supposed to be correlated as:

$$\overline{U'_iU'_j} = \mu_T\left(\frac{\partial\overline{U}_i}{\partial x_j} - \frac{\partial\overline{U}_j}{\partial x_i}\right) + \frac{2}{3}\delta_{ij}k, \tag{4.3.10}$$

where, μ_T denotes the eddy viscosity and k is the turbulent kinetic energy that is defined as

$$k = \frac{1}{2}\overline{U'_iU'_i}. \tag{4.3.11}$$

The turbulent kinetic energy k can be used to compute the turbulent intensity I of a flow, which is given by

$$I = \sqrt{\frac{2k}{3}}/\overline{U},\ \overline{U} \equiv \sqrt{U_iU_i}. \tag{4.3.12}$$

If the turbulence intensity is around 1%, we can speak about of a flow with low turbulent intensity. If the value is around 10% and more, the flow is said to be

highly turbulent. Turbulence intensity is a useful quantity especially to specify boundary conditions on turbulent kinetic energy based on experimental data.

In general, the turbulence can be characterized by using the turbulent kinetic energy k and a length scale l. Using dimensional analysis it is possible to derive an expression for μ_T in terms of k and l:

$$\mu_T = C_\mu \rho \sqrt{2k} l, \tag{4.3.13}$$

where C_μ is a dimensionless constant.

For simple flows, it is possible to use an analytical expression for turbulent kinetic energy k in terms of mean flow quantities and length scale l. For example, Prandlt's mixing length hypothesis states that

$$\sqrt{2k} \sim l \left| \frac{\partial \overline{U}_j}{\partial x_i} \right|. \tag{4.3.14}$$

Combining these relations in (4.3.14) and in (4.3.13), we obtain

$$\mu_T = l_m^2 \left| \frac{\partial \overline{U}_j}{\partial x_i} \right|, \tag{4.3.15}$$

where l_m denotes a constant that is referred to as the mixing length. The mixing length l_m can be prescribed for simple flows (e.g., for fully developed pipe flow with pipe radius R, it is given as a function of the radial coordinate r as $l_m = R[0.14 - 0.08(1 - r/R)^2 - 0.06(1 - r/R)^4]$ [**VM95**]). For more complex flows (e.g., detached flows), however, such kind of relations does not exist.

Turbulence models like Prandtl's mixing length model are called as algebraic or zero equation models since they do not require solution of additional PDEs. In algebraic models, turbulent eddy viscosity μ_T can be directly computed using only mean flow quantities. Therefore, these models are easy to implement and computationally cheap. However, their usage is restricted to simple flows. Other important algebraic models are Baldwin-Lomax [**BL78**] and Cebeci-Smith models [**SC67**].

Existence of turbulent kinetic energy in Eq. (4.3.13) energy suggests that one can also derive a PDE for the turbulent kinetic energy itself using momentum equations, continuity equation and energy equation. The transport equation for the turbulent kinetic energy, as it is given in [**Dur08**], is:

$$\begin{aligned} \rho \overline{U}_i \frac{\partial}{\partial x_i} \left(\frac{1}{2} \overline{U_j'^2} \right) &= -\frac{\partial}{\partial x_j} (\overline{p' U_j'}) + \frac{\partial}{\partial x_j} \left(\overline{\mu U_j' \frac{\partial U_j'}{\partial x_i}} \right) - \frac{\rho}{2} \frac{\partial}{\partial x_i} (\overline{U_i' U_j'^2}) \\ &- \rho \overline{U_i' U_j'} \frac{\partial \overline{U}_j}{\partial x_i} - \overline{\mu \frac{\partial U_j'}{\partial x_i} \frac{\partial U_j'}{\partial x_i}}. \end{aligned} \tag{4.3.16}$$

In the above PDE:

- $\rho \overline{U}_i \frac{\partial}{\partial x_i} \left(\frac{1}{2} \overline{U_j'^2} \right)$ represents convective transport of turbulent kinetic energy.
- $\frac{\partial}{\partial x_j} (\overline{p' U_j'})$ represents pressure-velocity correlation, which is responsible for the redistribution of kinetic energy from the jth component to the other components of velocity fluctuations.

- $\frac{\partial}{\partial x_j}\left(\mu\overline{U_j'\frac{\partial U_j'}{\partial x_i}}\right)$ describes the molecular transport of the turbulent kinetic energy.

- $\rho\overline{U_i'U_j'}\frac{\partial \overline{U}_j}{\partial x_i}$ represents production of turbulent kinetic energy by the mean flow. This is basically equivalent to the energy transfer from the mean flow to the smaller scaled eddies in the energy cascade.

- $\mu\overline{\frac{\partial U_j'}{\partial x_i}\frac{\partial U_j'}{\partial x_i}}$ is the viscous dissipation of turbulent kinetic energy, the rate at which turbulent kinetic energy is converted into internal energy by the small-scaled eddies.

- $\frac{\rho}{2}\frac{\partial}{\partial x_i}(\overline{U_i'U_j'^2})$ represents the diffusive transport of the turbulent kinetic energy by velocity fluctuations.

The transport equation for the turbulent kinetic energy can be rewritten in terms of k as

$$\rho\overline{U}_i\frac{\partial k}{\partial x_i} = \frac{\partial}{\partial x_j}\left(\mu\frac{\partial k}{\partial x_j}\right) - \frac{\partial D_j}{\partial x_j} + P_k - \epsilon, \tag{4.3.17}$$

where D_j is referred to as the turbulent diffusion term, given by

$$D_j = \overline{p'U_j'} + \frac{\rho}{2}\overline{U_j'U_i'U_i'} \tag{4.3.18}$$

In Eq. (4.3.17), P_k and ϵ denote the production and dissipation terms respectively. These terms as well as the turbulent diffusion term must be modeled since they are unknown. The turbulent diffusion can be modeled by using gradient diffusion assumption [**FP01**], given by

$$D_j \approx -\frac{\mu_T}{\sigma_T}\frac{\partial k}{\partial x_j}, \tag{4.3.19}$$

where σ_T is the turbulent Prandtl number.

The production term P_k can be estimated by using the Boussinesq hypothesis as

$$P_k = -\rho\overline{U_i'U_j'}\frac{\partial \overline{U}_j}{\partial x_j} \approx \mu_T\left(\frac{\partial \overline{U}_i}{\partial x_j} + \frac{\partial \overline{U}_j}{\partial x_i}\right)\frac{\partial \overline{U}_j}{\partial x_j} \tag{4.3.20}$$

The dissipation term is usually modeled by using the expression:

$$\epsilon \approx \frac{k^{3/2}}{l}, \tag{4.3.21}$$

which is suggested by considering equilibrium flows with $P_k \approx \epsilon$.

As it can be seen from the Eq. (4.3.13), a length scale l is also needed for the estimation of turbulent eddy-viscosity μ_T. In so-called one-equation turbulence models (e.g., Spalart-Almaras model [**SA92**]), only one PDE (usually a PDE for k) is solved and the other characteristic quantity (usually l) is estimated using analytical expressions. In other class of turbulence models, which are called two-equation turbulence models, another transport equation for the length scale is solved. One approach is to write a transport equation for the dissipation, which is related with k and length scale l. In the most common form, transport equation for dissipation

ϵ is given as

$$\rho\frac{\partial\epsilon}{\partial t}+\rho\overline{U}_i\frac{\partial\epsilon}{\partial x_i}=C_{\epsilon1}P_k\frac{\epsilon}{k}-\rho C_{\epsilon2}\frac{\epsilon^2}{k}+\frac{\partial}{\partial x_i}\left(\frac{\mu_T}{\sigma_\epsilon}\frac{\partial\epsilon}{\partial x_i}\right). \tag{4.3.22}$$

In order to compute the eddy-viscosity, by using assumption (4.3.21), we obtain from Eq. (4.3.13) the relation:

$$\mu_T=\rho C_\mu\frac{k^2}{\epsilon}. \tag{4.3.23}$$

This model is called as $k-\epsilon$ model and has been widely used for simulating turbulent flows. Constants $C_\mu, C_{\epsilon1}, C_{\epsilon2}, \sigma_\epsilon$ and σ_T are the model parameters and their values are found empirically by studying experimental data and DNS results. The ideal values for these constants are different for each type of flow and universal values does not exist. The most frequently used values are:

$$C_\mu=0.09,\ C_{\epsilon1}=1.44,\ C_{\epsilon2}=1.92,\ \sigma_\epsilon=1.3,\ \sigma_T=1.0. \tag{4.3.24}$$

Among two-equation models, an alternative approach is to write a PDE for the inverse of the time scale, which is denoted by ω. The ω equation, which is given by Wilcox is

$$\rho\frac{\partial\omega}{\partial t}+\rho\overline{U}_i\frac{\partial\omega}{\partial x_i}=\alpha\frac{\omega}{k}P_k-\rho\beta\omega^2+\frac{\partial}{\partial x_j}\left[\left(\mu+\frac{\mu_T}{\sigma^*_\omega}\right)\frac{\partial\omega}{\partial x_j}\right], \tag{4.3.25}$$

with

$$\mu_T=\rho\frac{k}{\omega}. \tag{4.3.26}$$

Standard values for the model parameters are given as

$$\alpha=5/9,\ \beta=3/40,\ \sigma^*_\omega=2. \tag{4.3.27}$$

The dissipation term is given by the relation

$$\epsilon=0.09\omega k. \tag{4.3.28}$$

This model is known as the standard $k-\omega$ or Wilcox $k-\omega$ model [**Wil88**]. Other well known variants of the $k-\omega$ model are the modified Wilcox $k-\omega$ [**Wil04**] and SST (Shear Stress Transport) $k-\omega$ model of Menter [**Men94**].

4.3.1. Treatment of wall boundaries. In turbulent flows, a special care is necessary for the boundary conditions in the wall region. Especially for high Reynolds number flows, boundary layer is extremely thin and it is not feasible to generate a grid point that is fine enough to resolve the high velocity gradients close to the wall. Also turbulence quantities like kinetic energy and dissipation have their peak values very close to the wall surface. Since same grid resolution is used for all quantities, grid spacing in the boundary layer may be insufficient for the turbulence quantities and computations may generate unphysical values (e.g., negative values of turbulent kinetic energy) in this region.

Note that, on the wall surface, it is appropriate to set $k=0$ since no-slip boundary condition is used for velocities. At high Reynolds numbers, however, it is not possible to resolve the boundary layer such that this boundary condition can be applied directly. For these cases, wall function approach is used. These functions are based on the existence of a universal velocity profile in the wall region. Close to the wall, viscous effects are important and flow does not depend on the free-stream

parameters. In the near wall-region, mean flow velocity $\overline{U}$ is a function of near-wall region parameters only:

$$\overline{U} = f(y, \rho, \mu, \tau_w), \tag{4.3.29}$$

where y and τ_w denote the normal distance to the wall and the wall shear stress respectively. By employing dimensional analysis, it is possible to derive the relationship:

$$u^+ = g(y^+), \tag{4.3.30}$$

where $u^+ = \overline{U}/u_\tau$, u_τ is called as the friction velocity and it is related with the wall shear stress τ_w via the relation:

$$u_\tau = \sqrt{\frac{\tau_w}{\rho}}. \tag{4.3.31}$$

The other dimensionless number y^+ is known as the dimensionless wall distance and it is given as

$$y^+ = \frac{u_\tau y}{\nu}. \tag{4.3.32}$$

The dimensionless wall distance y^+ is a similarity parameter, which is very similar to the Reynolds number. By introducing y^+, coordinate normal to the wall is made dimensionless. In general, the Eq. (4.3.30) is known as the "law of the wall". The near-wall region is divided into three sublayers according to the value of y^+:

- viscous sublayer ($0 < y^+ < 5$): In this region, viscous effects are important, therefore viscous stresses dominate over the Reynolds stresses. In this sublayer, it can be assumed that shear stress is constant and approximately equal to the wall shear stress τ_w. By using this assumption, it is possible to derive a linear relationship between the wall distance and the mean velocity by using dimensional analysis:

$$u^+ = y^+. \tag{4.3.33}$$

- buffer layer ($5 < y^+ < 30$): In this sublayer, viscous and Reynolds stresses are equally important.
- the log-law sublayer ($30 < y^+ < 300$): In the log-law sublayer, Reynolds stresses are dominant and relationship between u^+ and y^+ is given as

$$u^+ = \frac{1}{\kappa} \ln y^+ + B, \tag{4.3.34}$$

where the values of κ (von Karman constant) and B are determined from experimental results. These are universal constants that are valid for all wall bounded turbulent flows at high Reynolds numbers. For smooth walls, the values for them are given as $\kappa = 0.41$ and $B = 5.5$. The Eq. (4.3.34) is known as "log-law" in the literature.

If the wall functions are used, the first grid point can be placed in the log-law layer instead of the viscous sublayer ($y^+ > 100$). This leads to a significant reduction in the number of cells used for the boundary layer, thus making RANS simulations computationally feasible for high Reynolds number flows. Moreover, improvement in the cell aspect ratio near the wall helps to reduce the computational

stiffness. By using the log-law, it is possible to derive the boundary conditions for ϵ and ω:

$$k_p = \frac{u_\tau^2}{C_\mu} \text{ and } \epsilon = \frac{C_\mu^{3/4} {k_P}^{3/2}}{\kappa y}, \tag{4.3.35}$$

for the standard $k-\epsilon$ model. Similarly, for the standard $k-\omega$ turbulence model, boundary conditions are given as

$$k_p = \frac{u_\tau^2}{\sqrt{\beta}} \text{ and } \omega = \frac{{k_P}^{1/2}}{\beta^{1/4}\kappa y}. \tag{4.3.36}$$

In above equations, k_P denotes the turbulent kinetic energy in the first cell adjacent to the wall.

For low Reynolds number flows, the log-law is not valid and therefore wall functions cannot be used. For these cases, first grid point is usually located in the viscous sublayer (at about $y^+ = 1$) and a modified low Reynolds number turbulence model is used [**KMID05**]. Furthermore, damping functions are introduced into the turbulence model in the near-wall region. These functions are tuned by using DNS data to reproduce the actual near-wall behavior. The major difficulty with this approach is that the boundary layer should be resolved fine enough, which means much finer grids must be used compared to the high Reynolds number models using wall functions. If low Reynolds number version is used, the value of ω at the first cell is given in the LLR $k-\omega$ model ([**RT96**]) as

$$\omega = \frac{6\nu_w}{0.83y^2}. \tag{4.3.37}$$

Similarly, in the Low Reynolds number version of the $k-\epsilon$ model [**LL93**], the value of ϵ is given as

$$\epsilon = \frac{2\nu_w k_P}{y}. \tag{4.3.38}$$

CHAPTER 5

Sensitivity evaluation methods for gradient-based optimization

Gradient-based optimization methods, including the one-shot method, require computation of gradient vector of an objective function. In this chapter, various sensitivity evaluation methods in aerodynamic design optimization are reviewed and discussed in detail. We present various aspects of these methods according to the following criteria, which are especially of interest for aerodynamic shape optimization problems that are characterized by a large number of design parameters:

- Accuracy: Simple methods like the steepest descent method may work with inaccurate gradients reasonably well. However, more sophisticated and faster optimization algorithms like quasi-Newton methods require accurate gradient information. In fact, inaccurate gradient information may slow down the convergence behavior of the overall optimization process and even lead to erroneous results.
- Robustness: A good method for computing gradients must be able to produce reliable results for a wide range of flow conditions, different boundary conditions and grid qualities. Only robust methods can be employed efficiently inside an optimization framework.
- Computational cost : Yet another important criteria is the computational cost of evaluating gradients. A gradient evaluation method for large-scale optimization should be able to compute sensitivities at a computational cost, which is independent of the number of design parameters. In one-shot framework, this plays a key role in ensuring the bounded retardation property.
- Easiness of implementation: This criteria concerns how promptly a method can be implemented or modified to account for various add-ons, which are implemented as an extension of the underlying flow solver. This is of particular importance when applicability of a method to engineering problems is concerned. If a method is hard to implement and modify, most likely the gap between the functionality of the flow solver and the gradient evaluation tool will increase in time. This leads to an undesirable situation when the long term feasibility of the tool is concerned.

In the following subsections, we present various methods for evaluating gradients that are used in aerodynamic shape optimization problems. Each method performs better than others in one or more of the above criteria. The choice of the most suitable method depends on many factors like, number of design parameters, availability of the source codes that makes up the simulation chain, type of optimization problem, long term goals etc. Therefore, selection of the gradient evaluation method is then necessarily a compromise between various objectives.

5.1. Finite difference method

The easiest way of approximating derivatives is to use finite differences. This approach is straightforward to implement since it does not require any additional programming effort. Furthermore, finite differences can be computed in a black-box manner without the necessity of accessing the underlying source code. In general, compared to other methods, it requires less expertise and implementation effort, which makes this method attractive for complex problems. Moreover, it is often used to validate other sensitivity evaluation methods.

One can compute the gradient of a scalar objective function f with respect to a design parameter vector $u \in \mathbb{R}^m$ using first order accurate forward differences as

$$\frac{df}{du_i} \approx \frac{f(u_i + \epsilon \vec{e_i}) - f(u_i)}{\epsilon}, \quad i = 1, .., m. \tag{5.1.1}$$

Alternatively second order accurate central differences can be also used:

$$\frac{df}{du_i} \approx \frac{f(u + \epsilon \vec{e_i}) - f(u - \epsilon \vec{e_i})}{2\epsilon}, \quad i = 1, .., m. \tag{5.1.2}$$

In the above equations, $\vec{e_i}$ is the basis vector with ith entry is one while all other entries are zero.

Even though it is simple and well established, there exist well-known drawbacks associated with the finite difference method:

- The computational cost for the calculation of the gradient vector df/du depends on the size of the design vector u. For example, for forward differences, $m + 1$ function evaluations are necessary. On the other hand, central differences require $2m$ function evaluations. Therefore, if m is a large number, finite difference method is not feasible due to its high computational cost.
- The choice of the perturbation parameter ϵ plays a critical role in accuracy. If ϵ is chosen too large, sensitivity results are not accurate due to truncation error. On the other hand, if ϵ is chosen too small, cancellation error due to the subtraction may lead to wrong results. As a result, the perturbation parameter ϵ must be ideally tuned for each design parameter u_i, which drastically increases the computational cost.

In Fig. 1, a comparison of finite difference results using different perturbations parameters are shown. In this study, grid point sensitivities of a NACA 0012 airfoil on the wall boundary (only y coordinates) are computed point-wise by using forward finite differences and forward mode of AD respectively. The objective function is selected as the drag coefficient and the sensitivity values are plotted along the x coordinates scaled with the chord length c. Note that, the sensitivities that are computed with the forward AD are exact. Therefore, forward AD results are taken as the reference solution. It can be easily seen that grid sensitivities have an oscillatory character especially close to the leading edge of the airfoil and this behavior cannot be captured by finite difference method if a too large perturbation (e.g., 1% of u_i) is taken. On the other hand, a too small value of perturbation (e.g., 0.001%) leads to zero values on the leading edge due to cancellation errors. Optimum result is achieved by taking a value of 0.01%. From this study, it can be argued that the tuning of finite differences may be inevitable in some cases.

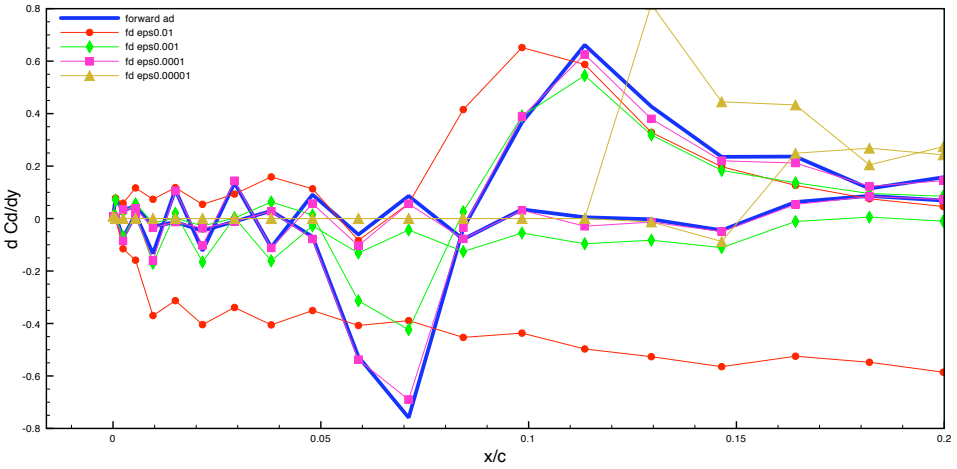

Figure 1. *Exact forward AD sensitivities and finite difference sensitivities generated using different perturbation values.*

5.2. Complex Taylor series expansion method

In order to avoid the subtractive cancellation errors in finite differences, one can use Complex Taylor Series Expansion (CTSE) method [**ST98**]. In this method, the objective function f is expanded in Taylor series with an imaginary perturbation, given by

$$f(u + i\epsilon e_i) = f(u) + i\epsilon f' - \frac{\epsilon^2}{2} f'' - i\frac{\epsilon^3}{6} f''' + O(\epsilon^4). \tag{5.2.1}$$

Equating the imaginary parts on both sides of the above equation, we get the derivative approximation

$$f' \approx \frac{Im(f(u + i\epsilon e_i)}{\epsilon} + O(\epsilon^2). \tag{5.2.2}$$

The CTSE method results in a second order approximation. Furthermore, it does not have any cancellation errors due to step size as the value of the perturbation parameter ϵ can be chosen small enough so that the derivative is accurate to machine precision. Thus, CTSE method is step size independent in the sense that the sensitivity evaluated is constant over a large range of step sizes once the second-order error is smaller than machine zero [**BN03**]. Although CTSE method has this nice feature, it has also two major drawbacks:

- Similar to the finite difference method, computational cost grows linearly with m, thus making it infeasible for optimization problems associated with large values of m.
- Underlying source code of f should be modified so that all the floating point operations are performed using complex numbers. This can be easily done for languages that support operator overloading. The programmer should then only change the data types of the input/output and intermediate variables to complex numbers and overload all subroutines and data structures in the call tree. The memory requirement in that case is approximately twice as the original code since each overloaded variable must

hold an extra imaginary part. As far as the implementation is concerned, an obvious disadvantage is that the source code of f must be available, and therefore evaluating derivatives in a black-box manner is not possible.

5.3. Linearization of the state equation

In design optimization problems that are characterized by a state PDE, one can linearize the state equation to compute the derivatives of an objective function with respect to design parameters. The state PDE linearization approach is also referred as the discrete direct method in the literature [**TKH91, BE91, BE92**]. For a given continuous state PDE, $c(u, y) = 0$, the discrete version in steady-state can be written as

$$R(u, y) = 0, u \in I\!R^m \text{ and } y \in I\!R^n, \tag{5.3.1}$$

where R denotes the residual corresponding to the discretized spatial terms in $c(u, y) = 0$. Linearizing the above discrete state equation using variational calculus, we get

$$\frac{\partial R}{\partial y}\delta y = -\frac{\partial R}{\partial u}\delta u. \tag{5.3.2}$$

Approximating the term $\frac{\partial R}{\partial u}\delta u$ using forward finite differences, we obtain

$$\frac{\partial R}{\partial u}\delta u \approx R(u + \delta u, y) - R(u, y). \tag{5.3.3}$$

Substituting the above approximation in Eq. (5.3.2), we obtain the system of linear equations

$$\frac{\partial R}{\partial y}\delta y = R(u, y) - R(u + \delta u, y). \tag{5.3.4}$$

On the other hand, if design vector is perturbed by δu, total variation of the scalar objective function $f(u, y)$ is given by

$$\begin{aligned} \delta f &= \frac{\partial f}{\partial u}\delta u + \frac{\partial f}{\partial y}\delta y \\ \Rightarrow \frac{df}{du}\delta u &= \frac{\partial f}{\partial u}\delta u + \frac{\partial f}{\partial y}\delta y. \end{aligned} \tag{5.3.5}$$

As a result, a perturbation in design by δu causes a perturbation in state by δy, which in return perturbs the objective function by δf. For computation of the gradient of f, the linearized state equation in (5.3.4) must be solved for each perturbation $\delta u_i, i = 1, .., m$. The solution of this system gives δy_i, which is then used to compute the ith element of the gradient vector df/du_i. Note that solving the linearized state equation requires the same computational effort as solving the primal PDE. The primal solution of $R(u, y) = 0$ can be obtained by a pseudo-time stepping scheme of the semi-discrete state equation

$$\frac{dy}{dt} + R(u, y) = 0. \tag{5.3.6}$$

For example, if the backward Euler implicit scheme is used for the transient term, we get the system of linear equations for the pseudo time iteration n

$$\frac{\partial R}{\partial y}(y^n - y^{n-1}) = -R(u, y^{n-1}). \tag{5.3.7}$$

It can be observed that the above solution method is very similar to the one presented in (5.3.4). Therefore, linearized state equation can be solved using the same iterative solver used for the state equation. Since the coefficient matrix of the linear system (i.e., the flux Jacobian $\partial R/\partial y$) is same for both equations, spectral properties of the linearized equation are exactly the same as the primal equation. Furthermore, the linearized equation inherits the convergence behavior of the primal solution.

Similar to FD and CTSE methods, the major drawback associated with the state PDE linearization method is its computational cost, which is a linear function of m.

5.4. Adjoint methods

All the sensitivity evaluation methods that are presented in the previous sections have one thing in common: Their computational cost increases linearly with m. In contrast to these methods, an efficient way of computing functional gradients is by employing the adjoint methods. Adjoint methods compute the complete gradient vector at a fixed expense, which is independent of the number of design parameters. Owing to their computational efficiency, adjoint methods have been widely used in aerodynamic shape optimization, optimal flow control, uncertainty quantification and data assimilation over the past two decades. The adjoint methods can be classified into two: continuos and discrete adjoint methods.

In the continuous adjoint approach, the adjoint PDE and its boundary conditions are first derived using the variational form of the state PDE and the objective function. The adjoint PDE is then discretized and solved using existing numerical methods. Since exactly the same PDE can also be derived from the optimality conditions, in which state PDE is treated as an equality constraint, the continuous adjoint method is also known as "first optimize then discretize" [**HUPU09**].

In contrast to the continuos approach, the discrete adjoint equation is derived based on the discrete realization of the state PDE. In this case, optimality conditions are written using the discrete state equation. Therefore, this method is known as "first discretize then optimize". In the following, we introduce these methods in detail. A detailed comparison of the continuous and discrete adjoint methods is presented in [**NJ00, NJ01**].

5.4.1. Continuos adjoint method. After being introduced by Pironneau [**Pir74**] in fluid mechanics, the continuous adjoint method became rapidly popular as it was successfully employed by Jameson [**Jam88**] for aerodynamic design optimization. Initially, Jameson derived the continuous adjoint equations for potential flow and compressible Euler equations. Later, the same methodology was extended to the compressible Navier-Stokes equations by Jameson et al. [**JMP98**] and applied to aerodynamic design of wing and wing-body configurations [**RJ95, RJF$^+$96**]. In the following, the derivation of continuos adjoint Euler equations is presented. The derivation of the adjoint PDE and the adjoint boundary

conditions, which are outlined below, can be found in more detail in [**Gau03**].

We consider first the $2D$ Euler equations in a flow domain Ω:

$$\frac{\partial w}{\partial t} + \frac{\partial f}{\partial x} + \frac{\partial g}{\partial y} = 0, \tag{5.4.1}$$

with

$$w = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix}, \quad f = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uH \end{pmatrix} \quad and \quad g = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vH \end{pmatrix}. \tag{5.4.2}$$

In the above equation, ρ is the density, u and v are the x and y components of the velocity, p is the static pressure, H is the enthalpy and E is the internal energy. For a perfect gas the relation between the static pressure and the internal energy is given by

$$p = (\gamma - 1)\rho(E - \frac{1}{2}(u^2 + v^2)), \tag{5.4.3}$$

where γ is the isentropic expansion factor. If the Euler equations in Cartesian coordinates are transformed using the body-fitted coordinates (see Fig.2), we get

$$\frac{\partial W}{\partial t} + \frac{\partial F}{\partial \xi} + \frac{\partial G}{\partial \eta} = 0, \tag{5.4.4}$$

where

$$W = J \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix}, \quad F = J \begin{pmatrix} \rho U \\ \rho U u + \frac{\partial \xi}{\partial x} p \\ \rho U v + \frac{\partial \xi}{\partial y} p \\ \rho U H \end{pmatrix} \quad and \quad G = J \begin{pmatrix} \rho V \\ \rho V u + \frac{\partial \eta}{\partial x} p \\ \rho V v + \frac{\partial \eta}{\partial y} p \\ \rho V H \end{pmatrix}. \tag{5.4.5}$$

In the above equations, J denotes the determinant of the metric Jacobian, given by

$$J = det \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{pmatrix}. \tag{5.4.6}$$

Further, U and V are the transformed velocity components, given by

$$\begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{pmatrix}^{-1} \begin{pmatrix} u \\ v \end{pmatrix} = \frac{1}{J} \begin{pmatrix} \frac{\partial y}{\partial \eta} & -\frac{\partial x}{\partial \eta} \\ -\frac{\partial y}{\partial \xi} & \frac{\partial x}{\partial \xi} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}. \tag{5.4.7}$$

The flux functions in body-fitted coordinates can be written as

$$F = J \left(f \frac{\partial \xi}{\partial x} + g \frac{\partial \xi}{\partial y} \right) \tag{5.4.8}$$

and

$$G = J \left(f \frac{\partial \eta}{\partial x} + g \frac{\partial \eta}{\partial y} \right). \tag{5.4.9}$$

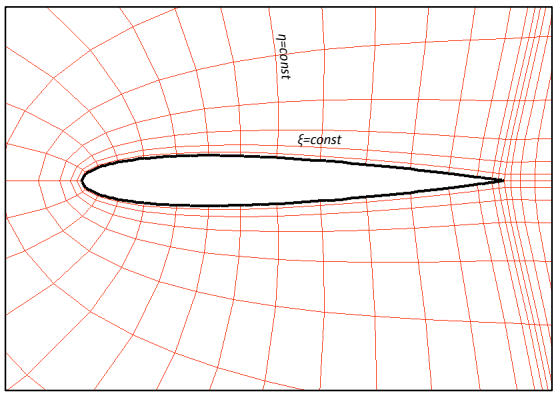

Figure 2. *Computational grid around an airfoil in body-fitted coordinates.*

The objective function that is to be minimized is the drag coefficient, given by

$$C_D = \frac{1}{C_{\text{ref}}} \int_{\partial\Omega_w} C_p(n_x \cos\alpha + n_y \sin\alpha)\mathrm{d}S, \tag{5.4.10}$$

where n_x and n_y are the x and y components of the normal vector, α is the angle of attack and C_{ref} is the reference length. The pressure coefficient C_p is defined by

$$C_p = \frac{2(p - p_\infty)}{\gamma M_\infty^2 p_\infty}, \tag{5.4.11}$$

where p, p_∞ and M_∞ are the static pressure, free-stream pressure and free-stream Mach number respectively. In the following, all free-stream variables are treated as constant. We now consider a perturbation in the airfoil geometry $\delta\Omega_w$, which causes a variation in the drag coefficient δC_D. If angle of attack is taken as a constant, using calculus of variations we get

$$\delta C_D = \frac{1}{C_{\text{ref}}} \int_{\partial\Omega_w} \left(C_p(\delta n_x \cos\alpha + \delta n_y \sin\alpha) + \delta C_p(n_x \cos\alpha + n_y \sin\alpha)\right) \mathrm{d}S, \tag{5.4.12}$$

where the variation in pressure coefficient is given by

$$\delta C_p = \frac{2\delta p}{\gamma M_\infty^2 p_\infty}. \tag{5.4.13}$$

Note that, the variation $\delta\Omega_w$ causes a variation in pressure δp and thus results in a variation in the objective function C_D. A less significant variation is caused by the variations δn_x and δn_y due to the change of airfoil geometry. At the perturbed state, steady-state Euler equations can be written as

$$\frac{\partial}{\partial\xi}(F + \delta F) + \frac{\partial}{\partial\eta}(G + \delta G) = 0. \tag{5.4.14}$$

$$\Rightarrow \underbrace{\frac{\partial F}{\partial\xi} + \frac{\partial G}{\partial\eta}}_{0} + \frac{\partial}{\partial\xi}(\delta F) + \frac{\partial}{\partial\eta}(\delta G) = 0.$$

Consequently, we obtain the variational Euler equations in body-fitted coordinates as

$$\frac{\partial}{\partial \xi}(\delta F) + \frac{\partial}{\partial \eta}(\delta G) = 0. \tag{5.4.15}$$

In weak form the above equation can be written as

$$\int_\Omega \psi^\top \left(\frac{\partial}{\partial \xi}(\delta F) + \frac{\partial}{\partial \eta}(\delta G) \right) d\Omega = 0, \tag{5.4.16}$$

where ψ is an arbitrary differentiable test function. Using the Green's identity, we get

$$\begin{aligned} \int_\Omega \psi^\top \left(\frac{\partial}{\partial \xi}(\delta F) + \frac{\partial}{\partial \eta}(\delta G) \right) d\Omega = &- \int_\Omega \left(\frac{\partial \psi^\top}{\partial \xi}(\delta F) + \frac{\partial \psi^\top}{\partial \eta}(\delta G) \right) d\Omega \\ &+ \int_{\partial\Omega} (n_1 \psi^\top \delta F + n_2 \psi^\top \delta G) dS. \end{aligned} \tag{5.4.17}$$

In the above equation, n_1 and n_2 are the components of the normal vector in ξ and η coordinates respectively. The boundary of the flow domain Ω is denoted by $\partial\Omega$, which is composed of wall and far-field boundaries ($\partial\Omega = \partial\Omega_w \cup \Omega_f$). Since body-fitted coordinates are used, we have $n_1 = 0$ and $n_2 = -1$ on the wall boundary $\partial\Omega_w$. The Eq. (5.4.17) then reduces to

$$\begin{aligned} \int_\Omega \psi^\top \left(\frac{\partial}{\partial \xi}(\delta F) + \frac{\partial}{\partial \eta}(\delta G) \right) d\Omega = &- \int_\Omega \left(\frac{\partial \psi^\top}{\partial \xi}(\delta F) + \frac{\partial \psi^\top}{\partial \eta}(\delta G) \right) d\Omega \\ &+ \int_{\partial\Omega_f} (n_1 \psi^\top \delta F + n_2 \psi^\top \delta G) dS \\ &+ \int_{\partial\Omega_w} (-\psi^\top \delta G) dS. \end{aligned} \tag{5.4.18}$$

The variations in δF and δG can be obtained from Eqs. (5.4.8) and (5.4.9) :

$$\delta F = \delta \left(J \frac{\partial \xi}{\partial x} \right) f + \delta \left(J \frac{\partial \xi}{\partial y} \right) g + J \frac{\partial \xi}{\partial x} \frac{\partial f}{\partial w} \delta w + J \frac{\partial \xi}{\partial y} \frac{\partial g}{\partial w} \delta w \tag{5.4.19}$$

$$\delta G = \delta \left(J \frac{\partial \eta}{\partial x} \right) f + \delta \left(J \frac{\partial \eta}{\partial y} \right) g + J \frac{\partial \eta}{\partial x} \frac{\partial f}{\partial w} \delta w + J \frac{\partial \eta}{\partial y} \frac{\partial g}{\partial w} \delta w. \tag{5.4.20}$$

On the wall boundary, we have $V = 0$ due to the slip boundary condition. Therefore, the flux function G is simplified as

$$G = J \begin{pmatrix} 0 \\ \frac{\partial \eta}{\partial x} p \\ \frac{\partial \eta}{\partial y} p \\ 0 \end{pmatrix}. \tag{5.4.21}$$

The variation in G along the wall boundary Ω_w is then given as

$$\delta G = J \begin{pmatrix} 0 \\ \frac{\partial \eta}{\partial x}\delta p \\ \frac{\partial \eta}{\partial y}\delta p \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \delta\left(\frac{\partial \eta}{\partial x}J\right)p \\ \delta\left(\frac{\partial \eta}{\partial y}J\right)p \\ 0 \end{pmatrix}. \tag{5.4.22}$$

As a result, Eq. (5.4.18) can be written as

(5.4.23)

$$\begin{aligned}\int_\Omega \psi^\top \left(\frac{\partial}{\partial \xi}(\delta F) + \frac{\partial}{\partial \eta}(\delta G)\right) d\Omega = &- \int_\Omega \left(\frac{\partial \psi^\top}{\partial \xi}\left(\delta\left(J\frac{\partial \xi}{\partial x}f\right) + \delta\left(J\frac{\partial \xi}{\partial y}g\right) + J\frac{\partial \xi}{\partial x}\frac{\partial f}{\partial w}\delta w + J\frac{\partial \xi}{\partial y}\frac{\partial g}{\partial w}\delta w\right)\right) \\ &+ \left(\frac{\partial \psi^\top}{\partial \eta}\left(\delta\left(J\frac{\partial \eta}{\partial x}f\right) + \delta\left(J\frac{\partial \eta}{\partial y}g\right) + J\frac{\partial \eta}{\partial x}\frac{\partial f}{\partial w}\delta w + J\frac{\partial \eta}{\partial y}\frac{\partial g}{\partial w}\delta w\right)\right) d\Omega \\ &+ \int_{\partial\Omega_f} (n_1\psi^\top \delta F + n_2 \psi^\top \delta G) dS \\ &- \int_{\partial\Omega_w} \psi_2 \left(J\frac{\partial \eta}{\partial x}\delta p + \delta\left(J\frac{\partial \eta}{\partial x}\right)p\right) dS \\ &- \int_{\partial\Omega_w} \psi_3 \left(J\frac{\partial \eta}{\partial y}\delta p + \delta\left(J\frac{\partial \eta}{\partial y}\right)p\right) dS.\end{aligned}$$

If the above equation is added to the variation in the objective function δC_d, we obtain

(5.4.24)

$$\begin{aligned}\delta C_D = &- \int_\Omega \left(\frac{\partial \psi^\top}{\partial \xi}\left(\delta\left(J\frac{\partial \xi}{\partial x}f\right) + \delta\left(J\frac{\partial \xi}{\partial y}g\right) + J\frac{\partial \xi}{\partial x}\frac{\partial f}{\partial w}\delta w + J\frac{\partial \xi}{\partial y}\frac{\partial g}{\partial w}\delta w\right)\right) \\ &+ \left(\frac{\partial \psi^\top}{\partial \eta}\left(\delta\left(J\frac{\partial \eta}{\partial x}f\right) + \delta\left(J\frac{\partial \eta}{\partial y}g\right) + J\frac{\partial \eta}{\partial x}\frac{\partial f}{\partial w}\delta w + J\frac{\partial \eta}{\partial y}\frac{\partial g}{\partial w}\delta w\right)\right) d\Omega \\ &+ \int_{\partial\Omega_f} (n_1\psi^\top \delta F + n_2 \psi^\top \delta G) dS \\ &- \int_{\partial\Omega_w} \psi_2 \left(J\frac{\partial \eta}{\partial x}\delta p + \delta\left(J\frac{\partial \eta}{\partial x}\right)p\right) dS \\ &- \int_{\partial\Omega_w} \psi_3 \left(J\frac{\partial \eta}{\partial y}\delta p + \delta\left(J\frac{\partial \eta}{\partial y}\right)p\right) dS \\ &+ \frac{1}{C_{\text{ref}}}\int_{\partial\Omega_w} C_p(\delta n_x \cos\alpha + \delta n_y \sin\alpha)\mathrm{d}S + \delta C_p(n_x\cos\alpha + n_y\sin\alpha)\mathrm{d}S.\end{aligned}$$

Since the variation of the objective function depends on δw and δp, one flow simulation must be performed for each design parameter if Eq. (5.4.24) is directly used to compute δC_D. However, one can chose the adjoint vector ψ in such a way that δw and δp terms vanish in the variational equation. From the fact that δw terms should vanish in Ω, we obtain the equation:

$$-\frac{\psi^\top}{\partial \xi}\left(J\frac{\partial \xi}{\partial x}\frac{\partial f}{\partial w} + J\frac{\partial \xi}{\partial y}\frac{\partial g}{\partial w}\right) - \frac{\psi^\top}{\partial \eta}\left(J\frac{\partial \eta}{\partial x}\frac{\partial f}{\partial w} + J\frac{\partial \eta}{\partial y}\frac{\partial g}{\partial w}\right) = 0, \tag{5.4.25}$$

which can be written as

$$-\frac{\psi^\top}{\partial\xi}\left(\frac{\partial\xi}{\partial x}\frac{\partial f}{\partial w}+\frac{\partial\xi}{\partial y}\frac{\partial g}{\partial w}\right)-\frac{\psi^\top}{\partial\eta}\left(\frac{\partial\eta}{\partial x}\frac{\partial f}{\partial w}+\frac{\partial\eta}{\partial y}\frac{\partial g}{\partial w}\right)=0. \tag{5.4.26}$$

Using the above equation, the adjoint PDE can be written as

$$-\left(\frac{\partial f}{\partial w}\right)^\top\frac{\partial\psi}{\partial x}-\left(\frac{\partial g}{\partial w}\right)^\top\frac{\partial\psi}{\partial y}=0 \quad \text{in} \quad \Omega. \tag{5.4.27}$$

Note that in the above PDE all terms have a minus sign, which indicates that convection of adjoint quantities occur in the opposite direction compared to the convection of conservative variables in the Euler equations.

Boundary conditions for the adjoint PDE are derived from the fact that δp terms should vanish at the wall and far-field boundaries. At the wall boundary Ω_w, we get

$$-\psi_2 J\frac{\partial\eta}{\partial x}\delta p-\psi_3 J\frac{\partial\eta}{\partial y}\delta p+\frac{1}{C_{ref}}\delta C_p(n_x\cos(\alpha)+n_y\sin(\alpha))=0. \tag{5.4.28}$$

Using the identity for δC_p (Eq. 5.4.13), we obtain

$$\psi_2 J\frac{\partial\eta}{\partial x}\delta p+\psi_3 J\frac{\partial\eta}{\partial y}\delta p=\frac{2\delta p(n_x\cos(\alpha)+n_y\sin(\alpha)}{C_{ref}\gamma M_\infty^2 p_\infty}. \tag{5.4.29}$$

From the coordinate transformation, one can write

$$\frac{\partial\eta}{\partial x}=-\frac{1}{J}\frac{\partial y}{\partial\xi} \quad \text{and} \quad \frac{\partial\eta}{\partial y}=\frac{1}{J}\frac{\partial x}{\partial\xi}. \tag{5.4.30}$$

Therefore, the adjoint wall boundary condition can be derived using the above relations as

$$-\psi_2\frac{\partial y}{\partial\xi}+\psi_3\frac{\partial x}{\partial\eta}=\frac{2(n_x\cos(\alpha)+n_y\sin(\alpha)}{C_{ref}\gamma M_\infty^2 p_\infty}. \tag{5.4.31}$$

Since far-field boundary is supposed to be far enough from the airfoil geometry, the variations in metric terms and state variables due to perturbation $\delta\Omega_w$ should not effect the far-field. Consequently, we force that the far-field boundary terms in the variation of objective function should be equal to zero:

$$\int_{\partial\Omega_f}(n_1\psi^\top\delta F+n_2\psi^\top\delta G)dS=0. \tag{5.4.32}$$

The above equation is satisfied if the far-field boundary is kept fixed, i.e.,

$$\delta\left(J\frac{\partial\xi}{\partial x}\right)\to 0,...,\ \delta\left(J\frac{\partial\eta}{\partial y}\right)\to 0, \tag{5.4.33}$$

and the variation δw should vanish, i.e.,

$$\psi^\top J\frac{\partial\xi}{\partial x}\frac{\partial f}{\partial w}\delta w=0,...,\ \psi^\top J\frac{\partial\eta}{\partial y}\frac{\partial g}{\partial w}\delta w=0. \tag{5.4.34}$$

Finally, δC_D reduces to

$$\begin{aligned}\delta C_D = &-\int_\Omega \left(\frac{\partial \psi^\top}{\partial \xi}\left(\delta\left(J\frac{\partial \xi}{\partial x}f\right)+\delta\left(J\frac{\partial \xi}{\partial y}g\right)\right)\right.\\ &\left.+\left(\frac{\partial \psi^\top}{\partial \eta}\left(\delta\left(J\frac{\partial \eta}{\partial x}f\right)+\delta\left(J\frac{\partial \eta}{\partial y}g\right)\right)\right)d\Omega\right.\\ &-\int_{\partial\Omega_w}\psi_2\delta\left(J\frac{\partial \eta}{\partial x}p\right)dS\\ &-\int_{\partial\Omega_w}\psi_3\delta\left(J\frac{\partial \eta}{\partial y}p\right)dS\\ &+\frac{1}{C_{ref}}\int_{\partial\Omega_w}C_p(\delta n_x\cos\alpha+\delta n_y\sin\alpha)\mathrm{d}S.\end{aligned} \tag{5.4.35}$$

Note that, the above variational formulation does not include any variations in state variables. As a result, if δC_D is to be computed m times (for each design parameter) it is enough to solve the adjoint PDE only once and then evaluate each time the variational objective function by using the same adjoint vector ψ. The variations in metric terms, which are also required in the above formulation, can be computed easily by finite differences. The only disadvantage is that the computational cost increases significantly as m increases. Therefore, using finite differences for metric terms can be a bottleneck for large-scale shape optimization. In yet another approach, shape calculus is used to derive the gradient expression in Hadamard form. The Hadamard form enables the efficient computation of the gradient without computing the metric variations in the domain Ω. This is especially the case for objective functions such as drag or lift coefficients, which are boundary integrals. The shape derivatives for these quantities can be found for example in [**SISG11**] for a compressible fluid model and in [**SS10**] for incompressible Navier-Stokes equations.

As far as the solution method for the adjoint PDE is concerned, a similar procedure that is used for the primal PDE can also be used for the adjoint equations. For steady-state flows, a pseudo time-stepping algorithm can be used for the adjoint equations combined with a suitable spatial discretization scheme. More information about the solution schemes for the adjoint Euler and Navier-Stokes equations can be found in [**JMP98, GB02, CLPZ06**].

To summarize, the continuous adjoint method is known to have minimum memory and CPU requirements and therefore enables very efficient computation of gradient vectors. The major drawback is the fact that it may suffer from inconsistency problems caused by some assumptions that are often made while deriving the adjoint PDE/boundary conditions. These assumptions often result from the difficulties introduced by various physical models and/or non-differentiabilities of some boundary conditions. A frequently used assumption, for example, is the constant eddy viscosity (CEV) or the so-called frozen μ_T assumption [**AV99**]. In this assumption, the eddy viscosity is taken as constant while deriving the adjoint PDE. In this approach, one does not need to derive the adjoint scalar transport PDEs and adjoint boundary conditions for the turbulence model. As a result, adjoint derivation is eased considerably but the resulting adjoint solver is inconsistent with the primal flow solver, which may cause inaccuracy problems. Although some effort has been done in the past for the derivation of adjoint turbulence models

[**ZPGO09, ZPGO10**], a consistent treatment is still missing for the majority of the models. As an another source of error, the inconsistency caused by the finite grid spacing can be given. The gradients based on continuous adjoints may not be consistent with objective function evaluations on the given computational grid. By using grid refinement, one can obtain adjoints that are consistent with primal solutions, but this is coupled with the extra overhead in CPU run-time. In conclusion, due to the inconsistencies that are mentioned above, continuous approach lacks accuracy and robustness in the computation of functional gradients in RANS simulations [**CTÖG10**].

5.4.2. Discrete adjoint method. The other possibility for the development of an adjoint method is to use the discrete state equation in the derivation of adjoints. This method is referred to as the discrete adjoint method. The way how a discrete adjoint solver is generated is not unique and different approaches can be found in literature. In a broad sense, these approaches can be classified under three categories:

- A well known practice of developing a discrete adjoint solver, which has been very popular in the past two decades, is by using the so-called "hand-discrete" approach. In this approach, adjoint system of linear equations are derived by hand based on the spatial discretization scheme of the underlying nonlinear flow solver. A computer code is then implemented to solve the adjoint linear system and to compute the sensitivities.
- In the second approach, the adjoint equations are derived based on the fixed-point iterator used in the primal solver. Note that, the primal fixed-point iterator includes not only spatial discretization terms but also other parts of the primal solution scheme like interpolation schemes, convergence acceleration techniques, etc. The adjoint fixed-point iterator is then automatically generated by applying the reverse mode of AD to the computer code that represents the primal fixed-point iterator. Finally, adjoint solver is constructed by hand using the adjoint fixed-point iterator.
- In yet another approach, adjoint code is generated by applying the reverse mode of AD in a black-box fashion to the complete source code of the primal solver. This approach does not require any analytical derivation or hand implementation as the adjoint solver is generated by the AD tool.

In the following subsections, these three methods will be presented. The details concerning the application of AD techniques is covered in the following chapter.

5.4.2.1. *The hand-discrete approach.* We consider an optimization problem, for which the derivatives of the objective function f with respect to the design u are required. Since f is a function of both state y and design u, using the chain rule we get

$$\frac{df}{du} = \frac{\partial f}{\partial u} + \frac{\partial f}{\partial y}\frac{dy}{du}. \tag{5.4.36}$$

On the other hand, the steady-state Euler or Navier-Stokes equations in discrete form are given by

$$R(u, y) = 0,\ y \in I\!R^n,\ u \in I\!R^m, \tag{5.4.37}$$

where R is the vector of discrete cell residuals. Note that the cell residual of a control volume having N_f boundary faces is given by

$$R_i = \sum_{l=1}^{N_f} (F_c^l - F_v^l) - Q_i, \tag{5.4.38}$$

where F_c^l and F_v^l denote the convective and viscous fluxes through the cell face l and Q_i represents the source term. From the Eq. (5.4.37), one can derive

$$\frac{\partial R}{\partial u} + \frac{\partial R}{\partial y}\frac{dy}{du} = 0 \Rightarrow \frac{dy}{du} = -\left(\frac{\partial R}{\partial y}\right)^{-1}\frac{\partial R}{\partial u}. \tag{5.4.39}$$

Substituting dy/du into the Eq. (5.4.36), we obtain the sensitivity equation:

$$\frac{df}{du} = \frac{\partial f}{\partial u} - \frac{\partial f}{\partial y}\left(\frac{\partial R}{\partial y}\right)^{-1}\frac{\partial R}{\partial u}. \tag{5.4.40}$$

In order to compute the complete gradient vector $(df/du) \in I\!R^m$, there are two possibilities. In one approach, one can compute

$$\lambda_i = \left(\frac{\partial R}{\partial y}\right)^{-1}\frac{\partial R}{\partial u_i}, \quad i = 1, .., m, \tag{5.4.41}$$

where λ_i is the solution of the linear system:

$$\left(\frac{\partial R}{\partial y}\right)\lambda_i = \frac{\partial R}{\partial u_i}. \tag{5.4.42}$$

The values of λ_i are then used to compute the sensitivities

$$\frac{dC_D}{du_i} = \frac{\partial C_D}{\partial u_i} - \frac{\partial C_D}{\partial y}\lambda_i, \quad i = 1, ..., m. \tag{5.4.43}$$

This way of computing sensitivities is same as the direct discrete method, which is presented previously. Alternatively, one can compute first

$$\psi^\top = \frac{\partial C_D}{\partial y}\left(\frac{\partial R}{\partial y}\right)^{-1}, \tag{5.4.44}$$

where ψ is the solution of the linear system

$$\left(\frac{\partial R}{\partial y}\right)^\top \psi = \frac{\partial C_D}{\partial y}^\top, \tag{5.4.45}$$

and then finally compute the sensitivities

$$\frac{dC_D}{du_i} = \frac{\partial C_D}{\partial u_i} - \psi^\top\frac{\partial R}{\partial u_i}, \quad i = 1, ..., m. \tag{5.4.46}$$

The second approach is known as the discrete adjoint method and the Eq. (5.4.45) is referred to as the discrete adjoint equation.

In order to compute the gradient vector for different m design parameters, the direct discrete method requires the solution of the linear system in Eq. (5.4.42) m times. On the contrary, if the adjoint method is used the linear system in (5.4.45) is solved only once. Therefore, for large values of m, the computational cost of the adjoint method is significantly cheaper than the direct discrete method.

Note that the term $\partial f/\partial u_i$ in Eq. (5.4.40) must also be computed for each design parameter u_i. These sensitivities depend on the shape parameterization and can be calculated most suitably either by using finite differences or by applying

reverse mode of AD to the post-processing tool in a black-box manner. Since post-processing step is computationally much cheaper than solving the state PDE, the computational cost of these sensitivities are negligible compared to the solution of the adjoint equation.

As far as the implementation of the hand-discrete approach is concerned, the most important issue is the way how the adjoint linear system in Eq. (5.4.45) is assembled. In order to compute the flux Jacobian $\partial R/\partial y$, discrete residual R (convective and viscous fluxes plus source terms) must be differentiated with respect to the state y. In literature, various approaches of computing the flux Jacobian can be find. Below, we give the most popular ones:

- Analytical derivation of the flux Jacobians:

  This method is also known as "hand differentiation" in the literature [**NA99**], since the linearization process for the subroutines that compute the fluxes is totally done by hand using symbolic differentiation rules. This approach results in adjoint solvers that are efficient in terms of memory and run-time and has been in wide usage in aerodynamic shape optimization [**CKvT07, NDY09, EP97**] and optimal flow control [**RZ10**]. The major disadvantage is the difficulty in linearizing the convective and viscous fluxes by hand, which is cumbersome and error-prone. Further, the linearization step has to be repeated whenever the spatial discretization scheme is modified. Another complication caused by the analytical approach is the difficulty in linearizing the turbulence models and incorporating them into the adjoint solver. There are mainly two approaches as the treatment of the turbulence models in the adjoint solvers is concerned.

  The first approach, as it is done for continuous adjoint method, is to use the constant eddy viscosity (CEV). In this approach, the eddy viscosity μ_T is assumed to be constant in the derivation of adjoint equation so that the turbulence equations are used only in the nonlinear flow solver but are neglected in the adjoint solver. Although CEV assumption is known to be valid for some problems, it produces erroneous results in other cases. A detailed study regarding the effect of CEV assumption on the accuracy of adjoint solvers can be found in [**KKR02**].

  The second approach is to linearize the turbulence model by hand. This is by far the most difficult approach since a lot of effort is needed for deriving analytical derivatives for highly complex turbulence models. Therefore, only a very limited number of work in this direction can be found in literature. For example, the Spalart-Allmaras model was linearized e.g., by Giles et al. [**GDM01**]. In another work, Kim et al. linearized the standard $k-\epsilon$, SST $k-\omega$ and Wilcox $k-\omega$ models and applied them to subsonic and transonic airfoil design, as well as high-lift configurations [**KKR01, KKR02**]. However, it should be noted that there is a notable lack of applications that are significantly more complex than isolated wings in fully attached steady-state flows or 2D high-lift configurations. According to Peter et al. [**PD10**], the reason for that is possibly not the difficulty of performing the linearization of the turbulence model but the convergence problems of the adjoint solver caused by the

adjoint turbulence models.

- Approximation using finite differences:

  Since the flux Jacobian matrix is sparse, the entries of it can be approximated by using finite differences at a reasonable computational cost. This method is surely the easiest one to implement and, once computed, the flux Jacobian can be saved in a compressed sparse matrix format. As an example, the discrete adjoint version of the DLR solver TRACE has been developed using this strategy [**FKN09**]. An obvious disadvantage of this approach is that the finite differencing results in inaccurate entries of the Jacobian matrix and therefore may lead to erroneous sensitivity gradients. The reason for this is the large condition number of the flux Jacobian, in which small truncation errors in the computation of flux Jacobian tend to amplify and produce large errors in the adjoint solution. Because of this drawback, finite difference method is not popular for the implementation of discrete adjoint solvers.

- Exact computation using AD:

  Based on the implemented source code of the discrete cell residual R_i, forward or reverse mode of AD can be employed to generate automatically a differentiated source code that exactly evaluates $\partial R/\partial y$. This approach has been used by several researchers in developing discrete adjoint solvers. In the aeronautical context, perhaps the first application was by Mohammadi [**Moh97**]. Later Courty et al. [**CKH03**] used the reverse mode of AD to compute the transposed Jacobian vector products for the shape optimization of a $2D$ nozzle. In yet another work, Mader et al. [**MMAvdW08**] used AD for the development of a discrete adjoint solver for the compressible Navier-Stokes equations. The usage of AD for an industrial flow solver was first initiated by Giles et al. [**GGD08**] and realized in the adjoint version of the HYDRA code, which is used in the design of turbomachines [**GDMP03**].

A second issue, which is important for the implementation, is the storage cost of the Jacobian matrix. In most of the existing adjoint solvers, the Jacobian is stored in a compressed format by exploiting the sparsity structure. The sparsity comes from the fact that the discretized fluxes at a cell face are influenced only by a few neighboring cells. Alternatively, some researches suggested assembling the left hand side of the adjoint system on-the-fly [**CKvT07**]. In this way, the transposed Jacobian vector product can be computed without storing the Jacobian itself. The resulting adjoint linear system is then solved using a matrix-free method. In terms of memory requirements, this approach is more efficient, especially for $3D$ problems, but the analytical derivation and implementation of the transposed Jacobian vector product needs a lot of effort.

In order to increase the sparsity pattern of the Jacobian and also to ease the linearization process, several Jacobian approximations are proposed. The most well known Jacobian approximations, which are frequently used in discrete adjoint solvers are:

- In one approximation, only first order terms of the cell residual are linearized and taken into account for the adjoint equation. The derivatives of the higher order terms in the numerical stencil are simply neglected. The aim of this approximation is to reduce the stencil of the discretization to immediate neighbors of a node and thus reduce the fill-in of the Jacobian and the stiffness of the linear adjoint system. However, this reduction comes at the cost of loss in accuracy and may lead to significant deviations in results. Nielsen et al. [**NA99**] examined the effect of this approximation for a high-lift configuration and observed significant errors in the sensitivities.

- Another assumption, which is frequently made in approximating the flux Jacobian, is treating some model parameters of the underlying numerical scheme as constant. Therefore, the functional relationship between the model parameters and the state variables are neglected in the linearization step. As an example to this approach, treating artificial dissipation terms for the JST scheme can be given [**Mav06**].

- The stencil of the viscous fluxes can be reduced further if an alternative discretization for the adjoint part is employed, which is inspired by the thin shear-layer (TSL) approximation used in structured grids. In this approach, only the normal component of the gradient is used to compute the viscous flux on a cell face. It should be noted that this simplification is not the same as the common TSL approximation used in flow solvers, in which only viscous fluxes normal to wall boundaries are considered. On the contrary, when used for the adjoint equation, the viscous fluxes in all directions are taken into account. This approach has been used, for example, in the discrete adjoint part of the ONERA elsA code [**Pet06**].

In order to summarize, each Jacobian approximation introduces an error in the adjoint solution up to some extent, leading to inaccuracies in sensitivity gradients. The error is highly test case dependent, therefore it is hard to quantify it in general. The influence of Jacobian approximations to the accuracy and robustness of the adjoint solvers is still being investigated.

The last important issue in the discrete adjoint solver development is the solution scheme chosen for adjoint equation. Note that the form of the adjoint system

$$\left(\frac{\partial R}{\partial y}\right)^{\top} \psi = \frac{\partial f}{\partial y}^{\top} \tag{5.4.47}$$

suggests at the first glance the solution can be easily obtained using an iterative linear equation solver. However, this approach may lead to serious convergence problems in many applications and therefore lacks robustness [**NA99**]. In order to increase the robustness of the adjoint solver, Nielsen and Anderson suggested solving the adjoint equation using a pseudo time-stepping scheme similar to the one employed for the primal nonlinear solver [**NA02**], which is introduced shortly in the following.

The adjoint residual can be defined as

$$R_{\psi} = \left(\frac{\partial R}{\partial y}\right)^{\top} \psi - \frac{\partial f}{\partial y}^{\top}. \tag{5.4.48}$$

Instead of directly solving the equation $R_\psi = 0$, one can obtain the same solution by solving the equation

$$\Omega \frac{d\psi}{dt} + R_\psi = 0 \tag{5.4.49}$$

using a pseudo time-step method. If the time derivative is discretized using backward Euler scheme, we get

$$\Omega \frac{(\psi^{n+1} - \psi^n)}{\Delta t} + R_\psi^{n+1} = 0, \tag{5.4.50}$$

which can be written as

$$\Omega \frac{(\psi^{n+1} - \psi^n)}{\Delta t} + \left(\frac{\partial R}{\partial y}\right)^\top \psi^{n+1} - \frac{\partial f}{\partial y}^\top = 0. \tag{5.4.51}$$

The above equation can be rearranged such that

$$\left(\frac{\Omega}{\Delta t} I + \frac{\partial R}{\partial y}^\top\right) \Delta\psi^n = \frac{\partial f}{\partial y}^\top - \frac{\partial R}{\partial y}^\top \psi^n \tag{5.4.52}$$

where $\Delta\psi^n$ is the adjoint update at the pseudo time-iteration n and it is given by

$$\Delta\psi^n = \psi^{n+1} - \psi^n. \tag{5.4.53}$$

The above solution procedure mimics the solution procedure for the state equation using backward Euler time discretization and pseudo time-stepping, given by

$$\left(\frac{\Omega}{\Delta t} I + \frac{\partial R}{\partial y}\right) \Delta y^n = -R^n. \tag{5.4.54}$$

The pseudo time-stepping scheme for the solution of the adjoint equation is outlined in the algorithm below:

ALGORITHM 6. *Pseudo time-stepping method for adjoint equation*

initialize $\psi \leftarrow \psi_0$ *and* $y \leftarrow y^*$ *(converged flow solution)*
for $n = 1, n \leq N_{max}$ ***do***
  Compute the adjoint residual $R_\psi^n = \frac{\partial R}{\partial y}^\top \psi^n - \frac{\partial f}{\partial y}^\top$
  Solve the linear system $\left[\frac{\Omega}{\Delta t} I + \frac{\partial R}{\partial y}^\top\right] \Delta\psi^n = -R_\psi^n$
  Update the adjoint vector $\psi^{n+1} = \psi^n + \Delta\psi^n$
  if $\|\Delta\psi^n\| \leq \epsilon$ ***then***
    $\psi^* \leftarrow \psi^{n+1}$
    break
  end if
end for
Compute the sensitivities $\frac{df}{du} = \frac{\partial f}{\partial u} - \left(\frac{\partial R}{\partial u}\right)^\top \psi^*$

It should be noted that the term $\frac{\Omega}{\Delta t} I$, which is added to the transposed flux Jacobian, increases the diagonal dominance of the coefficient matrix of the linear system and hence enhances the convergence and robustness behavior of the adjoint solver.

Nielsen et al. generalized this approach and based on the Giles' exact dual method presented a modified algorithm that preserves discrete duality [**NLPD04**]. The authors suggested solving the adjoint equation using exactly the same time-marching scheme used in the primal solver. In general, a time-marching scheme for the primal state equation can be written as

$$y^{n+1} = y^n - P(y,u)R(y,u),\ n = 1, ..., N. \tag{5.4.55}$$

Note that, if an implicit time discretization scheme is chosen, the term $\Delta y^n = -P(y^n, u)R(y^n, u)$ in the above equation corresponds to solution of a linear system for the pseudo time iteration n. Since inexact methods are usually chosen, the preconditioner P corresponds to an approximation of the inverse of the coefficient matrix. Further, the accuracy of the approximation depends on the method used for the solution of the linear system of equations and on the number of iterations performed. For example, if backward Euler time-marching scheme is used, the preconditioner is given by

$$P = \left[\frac{\Omega}{\Delta t} I + \left(\frac{\partial R}{\partial y}\right)\right]^{-1}, \tag{5.4.56}$$

if the system of linear equations

$$\left[\frac{\Omega}{\Delta t} I + \frac{\partial R}{\partial y}\right] \Delta y^n = -R(y^n, u) \tag{5.4.57}$$

is solved exactly in each pseudo time iteration n. However, since the system of linear equations are solved iteratively only up to some accuracy, the preconditioner is a poor approximate of the inverse matrix. Following the exact dual scheme, a pseudo-time integration scheme for the adjoint equations can therefore be written in a general form as

$$\psi^{n+1} - \psi^n = \Delta\psi^n = P^\top R_\psi, \tag{5.4.58}$$

where the adjoint residual R_ψ is given by

$$R_\psi = \frac{\partial f}{\partial y}^\top - \frac{\partial R}{\partial y}^\top \psi^n. \tag{5.4.59}$$

5.4.2.2. *Discrete adjoint approach based on the fixed-point iterator.* Another approach while deriving the adjoint equation is to use the fixed-point form of the primal state PDE instead of the cell residual. The primal state update in Eq. (5.4.55) can be written in the fixed-point form as

$$y^{n+1} = G(y^n, u),\ n = 1, ..., N. \tag{5.4.60}$$

Differentiating the fixed-point iterator G with respect to the design vector u, we get

$$\frac{dG}{du} = \frac{\partial G}{\partial u} + \frac{\partial G}{\partial y}\frac{dy}{du} \Rightarrow \frac{dy}{du} = \left(I - \frac{\partial G}{\partial y}\right)^{-1} \frac{\partial G}{\partial u}. \tag{5.4.61}$$

Similar to the hand-discrete method, substituting the term dy/du into the Eq. (5.4.36), we obtain the sensitivity equation based on G as

$$\frac{df}{du} = \frac{\partial f}{\partial u} + \frac{\partial f}{\partial y}\left(I - \frac{\partial G}{\partial y}\right)^{-1} \frac{\partial G}{\partial u}. \tag{5.4.62}$$

We introduce now the adjoint vector Ψ, defined by

$$\Psi^\top = \frac{\partial f}{\partial y}\left(I - \frac{\partial G}{\partial y}\right)^{-1}. \tag{5.4.63}$$

From the above equation, we get the adjoint equation,

$$\Psi = \frac{\partial f}{\partial y}^\top + \frac{\partial G}{\partial y}^\top \Psi. \tag{5.4.64}$$

Note that, similar to the primal equation, the above equation is in fixed-point form that can be solved using the fixed-point iterations

$$\Psi^{n+1} = \frac{\partial f}{\partial y}^\top + \frac{\partial G}{\partial y}^\top \Psi^n, n = 1, ..., N \tag{5.4.65}$$

Using the converged solution of the adjoint equation Ψ_* the sensitivities are then computed by

$$\frac{df}{du} = \frac{\partial f}{\partial u} + \Psi_*^\top \frac{\partial G}{\partial u}. \tag{5.4.66}$$

The adjoint fixed-point scheme in Eq. (5.4.65) converges to a fixed-point for every starting point Ψ^0 if and only if $\rho(G_y^\top) < 1$. Note that the spectral radius of G_y and the $G_y^\top$ are the same since both matrices have the same eigenvalues. Therefore, the adjoint fixed-point scheme is guaranteed to converge if the underlying primal fixed-point scheme converges. Moreover, the adjoint solver will have the same convergence rate as the primal solver at the termination.

Alternative to the solution procedure, which is given in Eq. (5.4.65), the state vector y and the adjoint vector Ψ can be updated simultaneously in a piggy-back manner. In this case, we have then the solution procedure

$$\begin{bmatrix} y^{n+1} \\ \Psi^{n+1} \end{bmatrix} = \begin{bmatrix} G(y^n, u) \\ \frac{\partial f}{\partial y}^\top + \frac{\partial G}{\partial y}^\top \Psi^n \end{bmatrix}, \; n = 1, ..., N \tag{5.4.67}$$

Using the above derivation of the adjoint equation based on the fixed-point iterator, it is also possible to derive the adjoint equation in the hand-discrete approach. Since the sensitivities of the design u must be equal in both formulations, we have

$$\frac{\partial f}{\partial u} - \psi^\top \frac{\partial R}{\partial u} = \frac{\partial f}{\partial u} + \Psi^\top \frac{\partial G}{\partial u}, \tag{5.4.68}$$

which reduces to

$$-\frac{\partial R}{\partial u}^\top \psi = \frac{\partial G}{\partial u}^\top \Psi. \tag{5.4.69}$$

On the other hand, the primal fixed-point iterator G can be written as

$$G(y^n, u) = y^n - P(y^n, u)R(y^n, u). \tag{5.4.70}$$

Differentiating the above expression with respect to design u, we get

$$\frac{\partial G}{\partial u} = -P\frac{\partial R}{\partial u} \Rightarrow \frac{\partial G}{\partial u}^\top = -\frac{\partial R}{\partial u}^\top P^\top. \tag{5.4.71}$$

Submitting the above expression in Eq. (5.4.69), we obtain the transformation for the adjoint vectors Ψ and ψ as

$$\psi = P^\top \Psi. \tag{5.4.72}$$

Differentiating $G(y^n, u)$ with respect to y^n, we get

$$\frac{\partial G(y^n,u)}{\partial y} = I - \frac{\partial P(y^n,u)}{\partial y} R(y^n,u) - P(y^n,u)\frac{\partial R(y^n,u)}{\partial y}. \tag{5.4.73}$$

If it is assumed that the residual $R(y^n, u)$ is exactly zero at the convergence of primal iterations, we obtain

$$\frac{\partial G(y^n,u)}{\partial y}^\top = I - \frac{\partial R(y^n,u)}{\partial y}^\top P(y^n,u)^\top. \tag{5.4.74}$$

Substituting the above expression into the Eq. (5.4.65), we get

$$\Psi^{n+1} = \frac{\partial f}{\partial y}^\top + \left(I - \frac{\partial R(y^n,u)}{\partial y}^\top P(y^n,u)^\top \right) \Psi^n \tag{5.4.75}$$

Multiplying this equation with $P(y^n, u)^\top$ and reordering the terms results in

$$P(y^n,u)^\top \Psi^{n+1} = P(y^n,u)^\top \Psi^n + P(y^n,u)^\top \left(\frac{\partial f}{\partial y}^\top - P(y^n,u)^\top \frac{\partial R(y^n,u)}{\partial y}^\top \Psi^n \right). \tag{5.4.76}$$

The above equation can be written in terms of ψ using the Eq. (5.4.72) as

$$\psi^{n+1} = \psi^n - P(y^n,u)^\top \left(\frac{\partial f}{\partial y}^\top - \frac{\partial R(y^n,u)}{\partial y}^\top \psi^n \right), \tag{5.4.77}$$

which results in Giles' exact dual method.

If the primal PDE is solved exactly by using the Newton method, the preconditioner is given by

$$P(y,u) = \frac{\partial R}{\partial y}^{-1}. \tag{5.4.78}$$

In this case, the above fixed point scheme reduces to

$$\psi^{n+1} = \psi^n - \frac{\partial R}{\partial y}^{-\top} \frac{\partial f}{\partial y}^\top - \frac{\partial R}{\partial y}^{-\top} \frac{\partial R(y^n,u)}{\partial y}^\top \psi^n \Rightarrow \psi^{n+1} = \frac{\partial R}{\partial y}^{-\top} \frac{\partial f}{\partial y}, \tag{5.4.79}$$

which leads to the linear system

$$\frac{\partial R}{\partial y}^\top \psi^{n+1} = \frac{\partial f}{\partial y}. \tag{5.4.80}$$

Note that for Newton case the adjoint solution is obtained by only solving a single system of linear equations, which is identical with the adjoint equation given in Eq. (5.4.45).

In conclusion, we state that the adjoint solution procedure given by the fixed point iteration in Eq. (5.4.65) is fully consistent with the solution method of the primal equations.

The adjoint fixed-point scheme given in Eq. (5.4.65) requires the computation of $G_y^\top \Psi$ instead of $R_y^\top \psi$. Although, the Jacobian R_y based on the local discrete residual is sparse, the Jacobian G_y based on the fixed-point iterator has a much denser structure. Therefore, it is not feasible to compute first $G_y^\top$ and then multiply it by Ψ. Instead, the matrix vector product $G_y^\top \Psi$ should be computed on-the-fly without explicitly storing the elements of G_y. Constructing a procedure by hand for this purpose is extremely laborious and error prone since the fixed point iterator G includes not only spatial discretization terms but also the other elements of the time-marching scheme. These are typically convergence acceleration and stability improvement techniques (e.g., implicit residual smoothing, local time stepping, Rhie-Chow interpolation etc.). Therefore, a more practical way of computing $G_y^\top \Psi$ is by applying the reverse mode of AD to the source code that represents G. When applied to the primal fixed-point iterator G, AD tool generates automatically another source that computes $G_y^\top \Psi$ on-the-fly. Furthermore, usage of AD ensures that all elements of the solution scheme are differentiated properly without using any Jacobian approximation. As a result, an efficient and robust discrete adjoint solver with an excellent accuracy is obtained. In discrete adjoint solver development, this approach has been successfully used for compressible Navier-Stokes [**GWMW07**] and incompressible RANS [**OG09**] equations.

5.4.2.3. *The Blackbox AD approach.* A discrete adjoint solver can also be developed by applying the chain rule to the complete trajectory of computations that make up the computational chain of the primal solution scheme. Note that the objective function f is calculated using the final value of the state y^N, which is the result of a solution procedure with N fixed-point iterations:

$$y^n = G^n(y^{n-1}, u),\ n = 1, ..., N, \tag{5.4.81}$$

where the final solution y^N is not necessarily a fixed point such that

$$y^N = G^N(y^N, u) \tag{5.4.82}$$

holds. Using the chain rule, the derivatives of f with respect to u can be written as

$$\frac{df}{du} = \frac{\partial f}{\partial u} + \frac{\partial f}{\partial y^N}\frac{dy^N}{du} \tag{5.4.83}$$

On the other hand, applying the chain rule to the fixed-point iterations yields

(5.4.84)
$$\begin{aligned}
\frac{dy^1}{du} &= \frac{\partial G^1}{\partial u} + \frac{\partial G^1}{\partial y^0}\frac{dy^0}{du} = \frac{\partial G^1}{\partial u} \\
\frac{dy^2}{du} &= \frac{\partial G^2}{\partial u} + \frac{\partial G^2}{\partial y^1}\frac{dy^1}{du} = \frac{\partial G^2}{\partial u} + \frac{\partial G^2}{\partial y^1}\frac{\partial G^1}{\partial u} \\
. &= . \\
. &= . \\
\frac{dy^N}{du} &= \frac{\partial G^N}{\partial u} + \frac{\partial G^N}{\partial y^{N-1}}\frac{dy^{N-1}}{du} = \frac{\partial G^N}{\partial u} + \frac{\partial G^N}{\partial y^{N-1}}\frac{\partial G^{N-1}}{\partial u} \\
&+ \frac{\partial G^N}{\partial y^{N-1}}\frac{\partial G^{N-1}}{\partial y^{N-2}}\frac{\partial G^{N-2}}{\partial u} + ... \\
&+ \frac{\partial G^N}{\partial y^{N-1}}\frac{\partial G^{N-1}}{\partial y^{N-2}}\frac{\partial G^{N-2}}{\partial y^{N-3}}...\frac{\partial G^1}{\partial u}.
\end{aligned}$$

In the above equations, G^n is the short hand notation of $G(y^n, u)$. Using the expression for dy^N/du from the above equation, df/du is now given as

(5.4.85)
$$\begin{aligned}
\frac{df}{du} &= \frac{\partial f}{\partial u} + \frac{\partial f}{\partial y^N}\frac{\partial G^N}{\partial u} + \frac{\partial f}{\partial y^N}\frac{\partial G^N}{\partial y^{N-1}}\frac{\partial G^{N-1}}{\partial u} \\
&+ \frac{\partial f}{\partial y^N}\frac{\partial G^N}{\partial y^{N-1}}\frac{\partial G^{N-1}}{\partial y^{N-2}}\frac{\partial G^{N-2}}{\partial u} + ... \\
&+ \frac{\partial f}{\partial y^N}\frac{\partial G^N}{\partial y^{N-1}}\frac{\partial G^{N-1}}{\partial y^{N-2}}\frac{\partial G^{N-2}}{\partial y^{N-3}}...\frac{\partial G^1}{\partial u}.
\end{aligned}$$

Taking the transpose of the above equation yields

(5.4.86)
$$\begin{aligned}
\frac{df}{du}^\top &= \frac{\partial f}{\partial u}^\top + \frac{\partial G^N}{\partial u}^\top \frac{\partial f}{\partial y^N}^\top + \frac{\partial G^{N-1}}{\partial u}^\top \frac{\partial G^N}{\partial y^{N-1}}^\top \frac{\partial f}{\partial y^N}^\top \\
&+ \frac{\partial G^{N-2}}{\partial u}^\top \frac{\partial G^{N-1}}{\partial y^{N-2}}^\top \frac{\partial G^N}{\partial y^{N-1}}^\top \frac{\partial f}{\partial y^N}^\top + ... \\
&+ \frac{\partial G^1}{\partial u}^\top ...\frac{\partial G^{N-2}}{\partial y^{N-3}}^\top \frac{\partial G^{N-1}}{\partial y^{N-2}}^\top \frac{\partial G^N}{\partial y^{N-1}}^\top \frac{\partial f}{\partial y^N}^\top .
\end{aligned}$$

The above equation can be rearranged as

$$\frac{df}{du}^\top = \frac{\partial f}{\partial u}^\top + \frac{\partial G^N}{\partial u}^\top \frac{\partial f}{\partial y^N}^\top + \sum_{i=N-1}^{1} \frac{\partial G^i}{\partial u}^\top \phi_i, \tag{5.4.87}$$

where ϕ_i are given recursively by

$$\phi_{N-1} = \frac{\partial G^N}{\partial y^{N-1}}^\top \frac{\partial f}{\partial y^N}^\top \text{ and } \phi_i = \frac{\partial G^{i+1}}{\partial y^i}^\top \phi_{i-1}, i = N-2, .., 1. \tag{5.4.88}$$

In the above scheme, all expressions in Eq. (5.4.87) are to be evaluated from right to left order. In this way, always matrix vector products are computed at each time. Note that the computations are in the reverse order so that the values

of the state vector before performing each primal fixed-point iteration must be saved. This introduces an extra storage overhead, which may easily dominate the overall run-time for large-scale applications. This specific problem will be discussed in detail in the following chapter.

As mentioned previously, the adjoint code that performs the solution procedure given in Eqs. (5.4.87) and (5.4.88) can be generated automatically by applying AD on the source code of the primal solver in a black-box fashion. In this way, one obtains a very robust discrete adjoint solver that gives exact sensitivity results at any residual level achieved by the primal solver, however at the expense of increased memory and run-time requirements.

CHAPTER 6

Automatic differentiation

In this chapter, we introduce the basic principles of Automatic Differentiation (AD) [**Gri00**], a technique used to differentiate functions that are implemented as computer programs. AD is alternatively referred to as "Algorithmic Differentiation" to emphasize that it is a way of differentiating algorithms symbolically at the source code level. Using AD, any kind of sensitivity information of a function can be computed, provided that there exists a discrete realization of that function at the software level. Therefore, AD is a valuable technique that can be employed for the generation of discrete adjoint codes. Further, the differentiation process can be performed almost in an automated fashion by using an AD package, which reduces the effort of code development significantly.

In a broad sense, AD can be seen as the application of the chain rule to the operations that make up a computer code in a mechanical fashion. The main idea is based on the simple fact that every computer code, no matter how complicated, is composed of elementary operations like $*, +, -, \sin$. If the chain rule is applied repeatedly to these operations, derivative of a function is then a concatenation of simple derivative expressions that are performed sequentially.

In general, chain rule can be applied in two ways to a given set of elementary operations. The first way, which appears to be more natural, is the so-called forward mode of AD. In the forward mode, chain rule is applied to every operation in a sequence that starts from the input parameters and ends with the output parameters. Therefore, each operation in the data flow is differentiated with respect to the input parameters. The resulting derivative expressions are then evaluated simultaneously with the operations of the original function. In contrast to the forward mode, the reverse or adjoint mode of AD applies the chain rule in the reverse order, in which the operations are performed in the original computer program. Note that, both the forward and reverse modes produce exactly the same result.

The application of chain rule in either forward or reverse mode can be done efficiently by using AD tools. These tools generate a differentiated source code based on the underlying original source code. In general, there are two methods how these tools achieve this task. The first approach uses the operator overloading, which is supported by some programming languages. Using operator overloading, all the elementary operations and functions are overloaded with a new data type, which evaluates both functional and derivative expressions. All floating point operations in the source code can be recoded using the new data type. This results in an overloaded source code that evaluates both function and its derivatives. The alternative method is the source transformation technique, which enables generating a differentiated code augmented with derivative expressions. A source transformation tool operates very similar to a compiler. It first parses the underlying original code and performs a dependency analysis between independent and dependent variables that

are specified by the user. Based on the dependency graph, it then generates the derivative expressions and inserts them into the original source code. The result is then a modified code, which computes the functional values and derivatives.

This chapter is organized as follows: We first introduce the forward mode since it is simpler to understand than the reverse mode. We demonstrate the working principle of the forward mode on a simple example. Then, we focus on the more interesting reverse mode, which is employed to generate discrete adjoint codes. Finally, we discuss several issues concerning the implementation, and present computational complexity analysis of the forward and reverse modes.

6.1. Forward mode

As an example, we consider the coordinate transformation from spherical coordinates (r, θ, φ) of a point to its Cartesian coordinates (x, y, z) in $3D$:

$$f : I\!R^3 \to I\!R^3 = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} rsin(\theta)cos(\varphi) \\ rsin(\theta)sin(\varphi) \\ rcos(\theta) \end{bmatrix}.$$

The coordinate transformation can be implemented as a Fortran 77 subroutine as

```
SUBROUTINE TRANSFORM(r,theta,varphi,x,y,z)
REAL*8 r,theta,varphi
REAL*8 x,y,z

x=R*SIN(theta)*COS(varphi)
y=R*SIN(theta)*SIN(varphi)
z=R*COS(theta)
END
```

For simplicity, we introduce some additional temporary variables and rewrite the above subroutine in such a way that each line in the source code is a single elementary operation (single assignment code):

```
SUBROUTINE TRANSFORM(r,theta,varphi,x,y,z)
REAL*8 r,theta,varphi
REAL*8 x,y,z
REAL*8 temp1,temp2,temp3,temp4,temp5
temp1=SIN(theta)
temp2=r*temp1
temp3=COS(theta)
temp4=COS(varphi)
temp5=SIN(varphi)
x=temp2*temp4
y=temp2*temp5
z=r*temp3
END
```

In Fig. 1, the dependency graph of the subroutine `TRANSFORM` is illustrated, which shows data flow between the program variables. An arrow between any two variables means that there exists a direct functional relationship between these variables. For example, the variable `temp1` directly depends on the value of `theta`

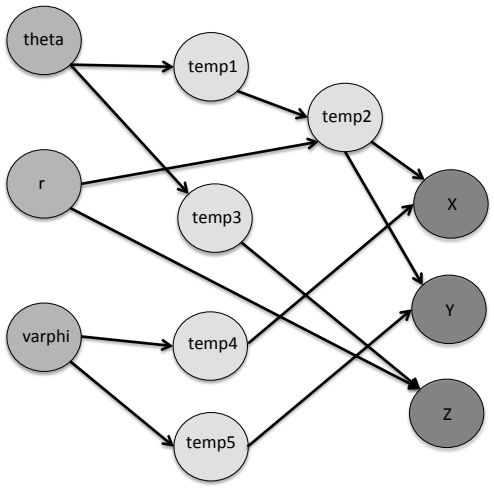

Figure 1. *The dependency graph of the subroutine TRANSFORM.*

via the relationship `temp1=SIN(theta)`. Accordingly, there is an arrow in the dependency graph between the `theta` and `temp1`.

Consider now, for instance, the derivative of the output variable x with respect to the input variable θ, $dx/d\theta$. The dependency graph shows that the intermediate variables `temp1` and `temp2` are computed in the data flow between x and θ. In the AD terminology, the variables like `temp1` and `temp2` are called "active" variables since their derivatives influence the sensitivity evaluation. The other program variables are named as "passive" variables. Note that, values of some passive variables are still required in the sensitivity evaluation although their derivatives do not play any role. These variables, which can be treated as constant in the differentiation, are called as "useful" variables. In our example, `temp4` is a useful variable. Applying the chain rule to each operation in the code, which takes an active parameter as an argument, with respect to θ, we obtain the derivative expressions:

(6.1.1)
$$\frac{d(temp1)}{d\theta} = cos(\theta), \quad \frac{d(temp2)}{d\theta} = \frac{d(temp1)}{d\theta} * r, \quad \frac{dx}{d\theta} = \frac{d(temp2)}{d\theta} * temp4.$$

If we denote the derivative of an arbitrary variable with respect to θ by the "dot" notation, such that $\dot{p} := dp/d\theta$ for a variable p, we can rewrite the above expressions as

$$\dot{temp1} = cos(\theta), \quad \dot{temp2} = \dot{temp1} * r, \quad \dot{x} = \dot{temp2} * temp4.$$

The above expressions finally give the desired result $\dot{x} = r\cos(\theta)\cos(\varphi)$. One can generalize this result such that $\dot{x}$ gives the directional derivative

$$\begin{aligned} \dot{temp1} &= cos(\theta) * \dot{\theta} \\ \dot{temp2} &= \dot{temp1} * r + temp1 * \dot{r} \\ \dot{temp4} &= -sin(\varphi) * \dot{\varphi} \\ \dot{x} &= \dot{temp2} * temp4 + temp2 * \dot{temp4}. \end{aligned}$$

operation	primal	forward AD
$\pm$	$x = y \pm z$	$\dot{x} = \dot{y} \pm \dot{z}$
$*$	$x = y * z$	$\dot{x} = \dot{y} * z + \dot{z} * y$
$/$	$x = y/z$	$\dot{x} = (\dot{y} * z - \dot{z} * y)/(z * z)$
c	$x = c$	$\dot{x} = 0$
y^c	$x = y^c$	$\dot{x} = c * y^{c-1} * \dot{y}$
sin	$x = sin(y)$	$\dot{x} = cos(y) * \dot{y}$
log	$x = log(y)$	$\dot{x} = \dot{y}/y$

Table 1. *Basic elementary operations and their forward AD counterparts.*

If the differentiation direction is given as $[\dot{\theta}, \dot{r}, \dot{\varphi}]^\top = [1, 0, 0]^\top$, we obtain

$$\begin{aligned} \dot{temp1} &= cos(\theta) \\ \dot{temp2} &= \dot{temp1} * r \\ \dot{temp4} &= 0 \\ \dot{x} &= \dot{temp2} * temp4, \end{aligned}$$

which gives the result $\dot{x} = \partial x/\partial\theta = r\ cos(\theta)cos(\varphi)$ as expected. Note that, in this example we have only used the derivative of three elementary operations, namely $+, *$ and sin. Similarly, derivative expressions involving other elementary operations can be derived using the chain rule. In Table 1, basic elementary operations and their forward AD counterparts are listed. Using these differentiation rules, derivative expressions can be generated for any code.

The above expressions can be written as a subroutine using the notation such that the suffix "d" after a variable name denotes the "dot" value of that variable:

```
SUBROUTINE TRANSFORM_D(r,rd,theta,thetad,varphi,varphid,x,xd,y,z)
REAL*8 r,theta,varphi,x,y,z
REAL*8 rd,thetad,varphid,xd
REAL*8 temp1,temp2,temp3,temp4,temp5
REAL*8 tempd,temp2d,temp4d

temp1=SIN(theta)
temp1d=COS(theta)*thetad
temp2=r*temp1
temp2d=rd*temp1+temp1d*r
temp3=COS(theta)
temp4=COS(varphi)
temp4d=-SIN(varphi)*varphid
temp5=SIN(varphi)

x=temp2*temp4
xd=temp2d*temp4+temp4d*temp2
y=temp2*temp5
z=r*temp3
```

```
END
```

The new subroutine `TRANSFORM_D` evaluates not only the values of primal variables x, y, z but also the derivative $\dot{x}$. The initializations of `rd`, `thetad` and `varphid` should be done by the user. In this example, we chose `rd`=0, `thetad`=1 and `varphid`=0, which yields $dx/d\theta$. Note that, one can also compute the derivatives of x with respect to r and φ as well in any mixed differentiation direction specified by the vector $[\dot{r}\ \dot{\theta}\ \dot{\varphi}]^\top$.

Similar to the derivative of x, one can easily derive the expressions for $\dot{y}$ and $\dot{z}$. If these expressions are augmented into the source code, the differentiated subroutine is

```
SUBROUTINE TRANSFORM_D(r,rd,theta,thetad,varphi,varphid,x,xd,y,yd,z,zd)
REAL*8 x,y,z,r,theta,varphi
REAL*8 xd,yd,zd,rd,thetad,varphid
REAL*8 temp1,temp2,temp3,temp4,temp5
REAL*8 temp1d,temp2d,temp3d,temp4d,temp5d

temp1=SIN(theta)
temp1d=COS(theta)*thetad
temp2=r*temp1
temp2d=rd*temp1+temp1d*r
temp3=COS(theta)
temp3d=-SIN(theta)*thetad
temp4=COS(varphi)
temp4d=-SIN(varphi)*varphid
temp5=SIN(varphi)
temp5d=COS(varphi)*varphid
x=temp2*temp4
xd=temp2d*temp4+temp4d*temp2
y=temp2*temp5
yd=temp2d*temp5+temp5d*temp2
z=r*temp3
zd=rd*temp3+temp3d*r

END
```

Note that, the above subroutine evaluates exactly the Jacobian vector product

(6.1.2)
$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} = \begin{bmatrix}\frac{\partial x}{\partial r} & \frac{\partial x}{\partial \phi} & \frac{\partial x}{\partial \varphi}\\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial \phi} & \frac{\partial y}{\partial \varphi}\\ \frac{\partial z}{\partial r} & \frac{\partial z}{\partial \phi} & \frac{\partial z}{\partial \varphi}\end{bmatrix} \begin{bmatrix}\dot{r}\\ \dot{\theta}\\ \dot{\varphi}\end{bmatrix} = \begin{bmatrix}\dot{r}\sin(\theta)\cos(\varphi) + \dot{\theta} r\cos(\theta)\cos(\varphi) - \dot{\varphi} r\sin(\theta)\sin(\varphi)\\ \dot{r}\sin(\theta)\sin(\varphi) + \dot{\theta} r\cos(\theta)\sin(\varphi) + \dot{\varphi} r\sin(\theta)\cos(\varphi)\\ \dot{r}\cos(\theta) - \dot{\theta} r\sin(\theta)\end{bmatrix}.$$

Note that, in this example, the derivative expressions can be easily derived by hand. In reality, however, a computer code may contain thousands of lines distributed over many subroutines and files. In such cases, to derive derivative expressions first by hand and then code them as a source code is not a feasible task. Therefore, in practical cases, derivative code is generated efficiently by using an AD package, which executes the described procedure automatically.

The forward mode of AD gives the derivatives of all outputs with respect to only one differentiation direction. Therefore, in order to compute the complete gradient vector of a single output, one has to run the forward code for all input directions. Hence, forward mode is computationally expensive if the number of input variables is large.

In general, for a function $f : X \in I\!R^n \rightarrow Y \in I\!R^m$, forward mode of AD gives the Jacobian vector product $\dot{Y} = J\dot{X}$, where the Jacobian matrix is defined as

$$J = \begin{bmatrix} \frac{\partial Y_1}{\partial X_1} & \frac{\partial Y_1}{\partial X_2} & . & . & \frac{\partial Y_1}{\partial X_n} \\ . & . & . & . & . \\ . & . & . & . & . \\ \frac{\partial Y_m}{\partial X_1} & . & . & . & \frac{\partial Y_m}{\partial X_n} \end{bmatrix}. \tag{6.1.3}$$

Typically for functions with a scalar output ($m = 1$) and many input variables ($n >> 1$), J reduces to a large vector with dimension n. If f is composed of p many sub-functions $f_p, f_{p-1}, f_{p-2}, ..., f_1$ such that

$$Y = f(X) = f_p(f_{p-1}(f_{p-2})(...(f_1(X)))), \tag{6.1.4}$$

where the vector Y is updated iteratively by each subfunction, f' is given as

$$\begin{aligned} f' &= f'_p(f_{p-1}(f_{p-2})(..(f_1(X)))) \\ &\quad \cdot f'_{p-1}(f_{p-2})(..(f_1(X))) \\ &\quad .... \\ &\quad \cdot f'_1(X) \\ &= f'_p(Y_{p-1}) \cdot f'_{p-1}(Y_{p-2}) \cdot ... \cdot f'_1(Y_0). \end{aligned} \tag{6.1.5}$$

Here, Y_k is defined recursively as $Y_0 = X$ and $Y_k = f_k(Y_{k-1})$. Multiplying f' with a differentiation direction $\dot{X}$, we get

$$\dot{Y} = f'_p(Y_{p-1}) \cdot f'_{p-1}(Y_{p-2}) \cdot ... \cdot f'_2(Y_1) \cdot f'_1(Y_0) \cdot \dot{X}. \tag{6.1.6}$$

It is important to note that derivatives in the above expression are always propagated in forms of matrix vector products rather than computing each Jacobian separately and multiplying with $\dot{X}$. In other words, forward AD first computes the matrix vector product $f'_1(Y_0).\dot{X}$ and then this result is multiplied with $f'_2(Y_1)$ and so on.

The reverse mode of AD, in contrast to the forward mode, is advantageous when the number input variables is significantly more than than the output variables, i.e., $n >> m$. This is typically the case in aerodynamic design optimization problems, which are characterized by a large number of design parameters. For these problems, often it is desired to compute the derivative of a scalar objective function (e.g., drag coefficient, mass flow rate etc.) with respect to a set of design parameters, which may be of the order $O(10^6)$ for large-scale applications (e.g., see [**SSIG11**]). The most efficient way is then use an adjoint solver, which computes the gradient at a computational cost independent of problem size. The reverse mode of AD can be utilized to generate such an adjoint solver.

We consider again the coordinate transformation example, which is already introduced in the previous section. This time, instead of selecting one input parameter and calculating the sensitivity of each intermediate variable with respect to that input, we select one output variable and calculate the sensitivity of that output with respect to all intermediate variables. Naturally, this process should be

in the reverse order, in which the intermediate variables are computed in the original code. The name "reverse mode" is originated from this basic fact. For example, if the sensitivities of x with respect to each intermediate variable are computed, we get:

$$\begin{aligned}
\frac{\partial x}{\partial temp2} &= temp4 \\
\frac{\partial x}{\partial temp4} &= temp2 \\
\frac{\partial x}{\partial \varphi} &= \frac{\partial x}{\partial temp4} * \frac{\partial temp4}{\partial \varphi} = \frac{\partial x}{\partial temp4} * (-sin(\varphi)) \\
\frac{\partial x}{\partial temp1} &= \frac{\partial x}{\partial temp2} * \frac{\partial temp2}{\partial temp1} = \frac{\partial x}{\partial temp2} * r \\
\frac{\partial x}{\partial r} &= \frac{\partial x}{\partial temp2} * \frac{\partial temp2}{\partial r} = \frac{\partial x}{\partial temp2} * temp1 \\
\frac{\partial x}{\partial \theta} &= \frac{\partial x}{\partial temp1} * \frac{\partial temp1}{\partial \theta} = \frac{\partial x}{\partial temp1} * cos(\theta)
\end{aligned}$$

Note that the last expression is exactly the derivative that we wanted to compute. In the following, we introduce a notation for simplicity: for any variable p, we associate another variable $\bar{p} := \partial x/\partial p$, referred to as the adjoint variable. We can now rewrite the above expressions using the adjoint variables as

$$\begin{aligned}
\overline{temp2} &= temp4 \\
\overline{temp4} &= temp2 \\
\overline{\varphi} &= \overline{temp4} * \frac{\partial temp4}{\partial \varphi} = \overline{temp4} * (-sin(\varphi) \\
\overline{temp1} &= \overline{temp2} * \frac{\partial temp2}{\partial temp1} = \overline{temp2} * r \\
\overline{r} &= \overline{temp2} * \frac{\partial temp2}{\partial r} = \overline{temp2} * temp1 \\
\overline{\theta} &= \overline{temp1} * \frac{\partial temp1}{\partial \theta} = \overline{temp1} * cos(\theta)
\end{aligned}$$

Finally, we obtain a chain of derivative expressions, which gives the sensitivities of x with respect to all input variables, i.e., derivatives of x with respect to r, θ and φ. In a similar way, adjoint derivative expressions for other elementary operations can also be generated by the chain rule. In Table 2, adjoint expressions for basic elementary operations are given.

The adjoint expressions, which are given for the coordinate transformation example, can be implemented as a Fortran subroutine:

```
SUBROUTINE TRANSFORM_B(r,rb,theta,thetab,varphi,varphib,x,xb,y,z)
REAL*8 x,y,z,r,theta,varphi
REAL*8 xb,rb,thetab,varphib
REAL*8 temp1,temp2,temp3,temp4,temp5
REAL*8 temp1b,temp2b,temp4b

temp1=SIN(theta)
```

operation	primal	adjoint
$\pm$	$x = y \pm z$	$\bar{t} = \bar{x},\ \bar{x} = 0$ $\bar{y} += \pm\bar{t},\ \bar{z} += \pm\bar{t}$
$*$	$x = y * z$	$\bar{t} = \bar{x},\ \bar{x} = 0$ $\bar{y} += \bar{t} * z,\ \bar{z} += \bar{t} * y$
$=$	$x = y$	$\bar{x} = \bar{x},\ \bar{x} = 0$
c	$x = c$	$\bar{x} = 0$
y^c	$x = y^c$	$\bar{t} = (\bar{x}c)x$ $\bar{x} = 0$ $\bar{y} += \bar{t}/y$
sin	$x = sin(y)$	$\bar{t} = \bar{x} * \cos(y)$ $\bar{x} = 0, \bar{y} += \bar{t}$
log	$x = log(y)$	$\bar{t} = \bar{x}/y$ $\bar{x} = 0, \bar{y} += \bar{t}$

Table 2. *Some elementary operations and their reverse AD counterparts.*

```
  temp2=R*temp1
  temp3=COS(theta)
  temp4=COS(varphi)
  temp5=SIN(varphi)

  x=temp2*temp4
  y=temp2*temp5
  z=r*temp3

  temp2b=temp4*xb
  temp4b=temp2*xb
  varphib=temp4b*(-SIN(varphi))
  temp1b=temp2b*r
  rb=temp2b*temp1
  thetab=temp1b*COS(theta)

END
```

Note that, adjoint variables are named by adding the suffix "`b`" to the variable names. In the subroutine `TRANSFORM_B`, first the operations of the original program are performed, which evaluate `X,Y` and `Z`. This initial part of the adjoint code, where only primal variables are computed, is called as the "forward sweep" in the AD terminology. After completing the forward sweep, in the successive part the adjoint variables are evaluated in reverse order. This part, which corresponds to the actual differentiation process by chain rule, is called as the "reverse sweep".

Similar to the forward mode of AD, memory must be also allocated for adjoint variables in the reverse mode. Therefore, adjoint code requires twice the memory of the original source code. Furthermore, there is another aspect of reverse mode

that may cause extra memory overhead. In the following, we shortly illustrate this problem using the previous example.

Suppose that the subroutine `TRANSFORM` is slightly modified, in which `theta` is assigned to some value immediately after the expression `temp1=SIN(theta)`:

```
      SUBROUTINE TRANSFORM(r,theta,varphi,x,y,z)
      REAL*8 r,theta,varphi
      REAL*8 x,y,z
      REAL*8 temp1,temp2,temp3,temp4,temp5
      REAL*8 theta_new

      theta_new=0.785
      temp1=SIN(theta)
C Here we change theta!
      theta=theta_new
      temp2=r*temp1
      temp3=COS(theta)
      temp4=COS(varphi)
      temp5=SIN(varphi)

      x=temp2*temp4
      y=temp2*temp5
      z=R*temp3

      END
```

Consider now the adjoint expression `thetab=temp1b*COS(theta)` in the reverse sweep, which corresponds to `temp1=SIN(theta)`. In the adjoint part, the value of `theta` before the assignment `theta=theta_new` is required. However, this value is lost in memory after the assignment is executed. This problem can be solved by saving the value of `theta` before changing its value in the forward sweep. This value, which is saved in the forward sweep, is then used for the adjoint expression in the reverse sweep. This mechanism can be implemented using a stack data structure and two stack functions, namely `PUSH` and `POP`. For our example, the `PUSH(theta)` operator adds the value of `theta` on top of the stack and the `POP(theta)` operator removes it back from the stack. Using these two functions, the modified differentiated subroutine is given as

```
      SUBROUTINE TRANSFORM_B(r, rb, theta, thetab, varphi, varphib, x,
     +                       xb, y, z)

      REAL*8 x, y, z, r, theta, varphi
      REAL*8 theta_new
      REAL*8 xb,rb, thetab, varphib
      REAL*8 temp1, temp2, temp3, temp4, temp5
      REAL*8 temp1b, temp2b, temp4b
      temp1 = SIN(theta)
      CALL PUSH(theta)
```

```
theta = theta_new
temp2 = r*temp1
temp3 = COS(theta)
temp4 = COS(varphi)
temp5 = SIN(varphi)
x = temp2*temp4
y = temp2*temp5
z = r*temp3
temp2b = temp4*xb
temp4b = temp2*xb
varphib = -(SIN(varphi)*temp4b)
rb = temp1*temp2b
temp1b = r*temp2b
CALL POP(theta)
thetab = COS(theta)*temp1b
END
```

In order to increase the computational efficiency, one can remove some of the primal operations from the differentiated code without effecting the derivative results. The part of the primal code, which does not influence any of the adjoint computations, is called as the "dead adjoint" code [**HAP05**]. The removal of the dead adjoint part from the adjoint code can be done most efficiently using an AD package that is based on source transformation method. For the given example, adjoint source code can be further simplified as

```
SUBROUTINE TRANSFORM_B(r, rb, theta, thetab, varphi, varphib, x,
+                       xb, y, z)

REAL*8 x, y, z, r, theta, varphi
REAL*8 xb,rb, thetab, varphib
REAL*8 temp1, temp2, temp4
REAL*8 temp1b, temp2b, temp4b
temp1 = SIN(theta)
temp2 = r*temp1
temp4 = COS(varphi)
x = temp2*temp4
temp2b = temp4*xb
temp4b = temp2*xb
varphib = -(SIN(varphi)*temp4b)
rb = temp1*temp2b
temp1b = r*temp2b
thetab = COS(theta)*temp1b
END
```

The above subroutine computes the gradient of x without fully performing the operations of the original subroutine `TRANSFORM`. In the general form, one can also choose `Y` and `Z` as the output parameters and obtain the adjoint subroutine:

```
SUBROUTINE TRANSFORM_B(r, rb, theta, thetab, phi, x, xb, y, yb, z
+                       , zb)
REAL*8 x, y, z, r, theta, phi
REAL*8 xb, yb, zb, rb, thetab
```

```
REAL*8 temp1, temp2, temp3, temp4, temp5
REAL*8 temp1b, temp2b, temp3b
temp1 = SIN(theta)
temp2 = r*t1
temp3 = COS(theta)
temp4 = COS(varphi)
temp5 = SIN(varphi)
x = temp2*temp4
y = temp2*temp5
z = r*temp3
t2b = temp4*xb + temp5*yb
t1b = r*temp2b
rb = temp1*temp2b + temp3*zb
t3b = r*zb
t5b = temp2*yb
t4b = temp2*xb
phib = COS(phi)*temp5b - SIN(varphi)*t4b
thetab = COS(theta)*temp1b - SIN(theta)*t3b
END
```

Note that, the above subroutine evaluates the transposed Jacobian vector product (6.1.7)

$$\begin{bmatrix}\bar{r}\\ \bar{\theta}\\ \bar{\varphi}\end{bmatrix} = \begin{bmatrix}\frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} & \frac{\partial z}{\partial r}\\ \frac{\partial x}{\partial \theta} & \frac{\partial y}{\partial \theta} & \frac{\partial z}{\partial \theta}\\ \frac{\partial x}{\partial \varphi} & \frac{\partial y}{\partial \varphi} & \frac{\partial z}{\partial \varphi}\end{bmatrix} \begin{bmatrix}\bar{x}\\ \bar{y}\\ \bar{z}\end{bmatrix} = \begin{bmatrix}\bar{x}\sin(\theta)\cos(\varphi) + \bar{y}\sin(\theta)\sin(\varphi) + \bar{z}\cos(\theta)\\ r\cos(\theta)\sin(\theta)(\bar{x}\cos(\varphi) + \bar{y}\sin(\varphi) - \bar{z})\\ r\sin(\theta)\cos(\varphi)\sin(\varphi)(\bar{y} - \bar{x})\end{bmatrix}.$$

To summarize, for a given function $f : X \in I\!R^n \to Y \in I\!R^m$, reverse mode of AD gives $\bar{X} = J^\top \bar{Y}$. If the function f is composed of p sub-functions, we have then

$$\begin{aligned} f'^\top &= f_1'^\top(X) \\ &\quad \cdot f_2'^\top(f_1(X)) \\ &\quad .... \\ &\quad \cdot f_{p-1}'^\top(f_{p-2})(..(f_1(X))) \\ &\quad \cdot f_p'^\top(f_{p-1}(f_{p-2})(..(f_1(X)))) \\ &= f_1'^\top(Y_0) \cdot f_2'^\top(Y_1) ... f_{p-1}'^\top(Y_{p-2}) \cdot f_p'^\top(Y_{p-1}), \end{aligned} \tag{6.1.8}$$

which leads to

$$\bar{X} = f_1'^\top(Y_0).f_2'^\top(Y_1)...f_{p-1}'^\top(Y_{p-2}).f_p'^\top(Y_{p-1}).\ \bar{Y}. \tag{6.1.9}$$

If the source code of f is differentiated using reverse mode, the derivative code exactly performs the operations given by the above formula. Note that, matrix-vector products are always computed from right to left, given by the recurrence:

$$\psi_0 = f_p'^\top(Y_{p-1}).\bar{Y} \tag{6.1.10}$$

$$\psi_i = f_{p-i}'^\top(Y_{p-1-i})\psi_i,\ i = 1, .., p-1 \tag{6.1.11}$$

$$\bar{X} = \psi_{p-1}. \tag{6.1.12}$$

In this way, adjoint values are always propagated as matrix-vector products and sub-Jacobians f_i' are not explicitly stored in memory. Note that, similar to the previous example, values of Y vectors must be available in the reverse order during

the adjoint computation. One way of retaining them in reverse order is to save the values in $Y_i, i = 1, .., p$ in the forward sweep. In AD terminology, this process is called as "taping", which is done by the stack function `PUSH`. The stored values are then retrieved back using the `POP` function. This strategy is called as "store-all" method in the AD terminology [**Has09**]. If the dimension of Y is large, the taping process demands large amount of memory. The other alternative to store-all is to recompute the value of Y_i from the initial state Y_0 as they are needed in the reverse sweep. This approach is called as the "recompute-all" method [**GK98**]. In contrast to save-all, recompute-all approach does not need to save the values of Y_i in the forward sweep. Therefore, it results in very efficient code in terms of memory demand. On the other hand, the number of recomputations grows quadratically with p, thus increasing the run-time required for the reverse sweep.

6.1.1. Checkpointing strategies. In applications, in which large vectors are updated iteratively, neither store-all nor recompute-all approach performs well. The store-all method requires a huge amount of memory, which may not be available. On the other hand, the recompute-all method suffers from high computational cost. To circumvent this problem, a compromise between the two can be made. Instead of storing the Y vector at each step, it can be stored at some selected locations and recomputations are performed starting from these locations. This compromise is called as the checkpointing strategy and the locations, where the Y vectors are stored, are known as checkpoints. In the following, we illustrate the checkpointing strategy with a simple example.

Consider a case, in which the state vector Y is updated iteratively by a fixed-point iterator, given by

$$Y_k = f_k(Y_{k-1}),\ k = 1, .., 6 \tag{6.1.13}$$

In Fig. 2, the store–all approach is illustrated. In the forward sweep, the intermediate values $Y_1, .., Y_5$ are taped. These values are then retrieved in the reverse sweep to compute the adjoints. In this case, the memory requirement increases by a factor of 5. In Fig. 3, the adjoint scheme using recompute–all approach is shown. In this case, no taping is done in the forward sweep. All the intermediate values of Y, which are needed in the reverse sweep, are recomputed again starting from the initial value Y_0. The recomputations, which are shown by the black arrows in the reverse sweep, are done repeatedly for each adjoint step in the reverse sweep. From this figure it can be clearly seen that the recompute-all approach requires 15 extra recomputations, which increases the run-time by a factor of 2.5. In Fig. 4, checkpointing strategy is illustrated for the same problem. In this example, only a single checkpoint is used such that in the forward sweep, Y_3 is saved. During the reverse sweep, the recomputations are done starting from the checkpoint Y_3 instead of Y_0. When the adjoint value $\overline{Y_3}$ is computed, this checkpoint is no more needed and can be reused to save Y_1. In conclusion, the memory demand has increased by a factor of 2 at the expense of 8 recomputations, which means a run-time increase by a factor of 1.33.

A critical issue in the checkpointing strategy is the distribution of checkpoints. The simplest strategy is the equidistant distribution. However, this method does not give the minimum number of recomputations. In fact, if the number of iterations and available checkpoints are known a priori, one can compute efficient checkpointing schedules in advance to decrease the number of recomputations. This procedure

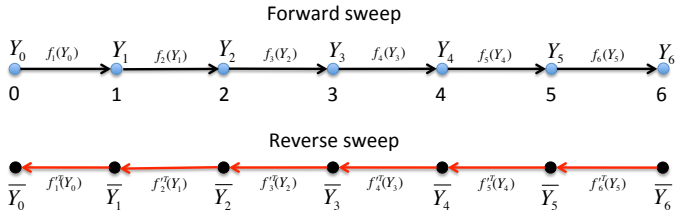

Figure 2. *The store–all approach.*

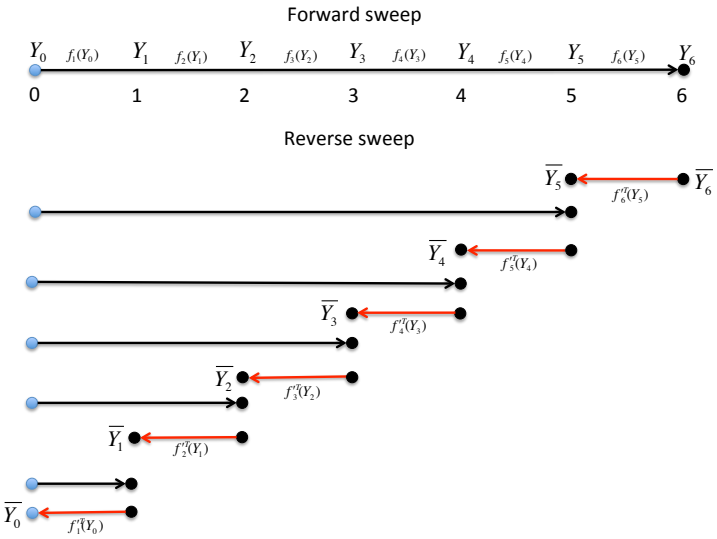

Figure 3. *The recompute–all approach.*

is referred to as the offline checkpointing. A well known offline checkpointing strategy is the binomial checkpointing, implemented in the algorithm *revolve* [**GW00**]. The binomial checkpointing strategy leads to the optimal distribution of checkpoints in such a way that the number of recomputations is minimal. On the other hand, if the computational scheme is adaptive, the number of iterations is not known beforehand. In this case, online checkpointing strategies are employed [**SW10**].

Similar to the iterative schemes, checkpointing can be performed in a nested fashion by processing the subroutines call tree data. Two basic call tree reversal schemes exists in the literature. In the so-called "split" mode, the adjoint computations are preceded by an execution of the forward sweep with taping. This version of forward sweep is called as the "recording forward sweep" to distinguish

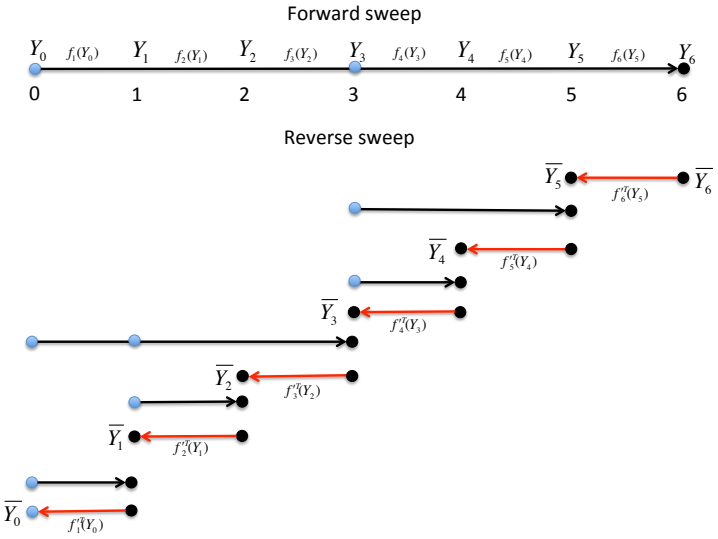

Figure 4. *The comprise using checkpointing.*

it from the forward sweep used in the recomputation step. The recording forward sweep is done only once for each subroutine during the adjoint evaluation. All values that are required for the correct reversal of the data flow are stored in tape. This strategy can be regarded as the store-all approach applied at the subroutine level and might produce tapes, which require prohibitively large amount of memory. This problem can be alleviated by the use of "joint" reversals such that a comprise is made between run-time and memory by using checkpointing at the subroutine level. The recording motion of the forward sweep is immediately followed by the adjoint computations so that the memory for the taping can be reclaimed again for other subroutines. This results in a much lower peak memory usage for the adjoint code at the expense of extra recomputations. Further information about call tree reversal schemes and illustrative examples can be found in [**Nau12**].

6.2. Computational complexities of the forward and reverse modes

The computational cost of AD is a very important issue as far as the applicability of the differentiated codes to the practical problems is concerned. For the assessment of the computational efficiency, the slowdown factor of the differentiated code is the most important quantity. It can be defined as the run-time of the differentiated code divided by the run-time of the underlying primal source code. The slowdown factor of the AD generated codes can be estimated a priori by analyzing the complexity of the forward and reverse modes. In [**Gri00**], a complexity analysis for both these modes is given, and is briefly introduced in the following. In his work, Griewank introduced a four component measure to determine the complexity

of a task:

$$W(task) \equiv \begin{bmatrix} \text{Number of memory fetches and stores} \\ \text{Number of additions} \\ \text{Number of multiplications} \\ \text{Number nonlinear operations.} \end{bmatrix} \tag{6.2.1}$$

In the following, we assume that the memory is flat, i.e., no distinction is made between reading and writing data from and to registers, cache etc. Moreover, we assume that the implementation of a task consists of only these four elementary operations:

- Initialization of a variable to a constant, denoted by c.
- Addition or subtraction, denoted by $\pm$.
- Multiplication, denoted by $*$.
- Nonlinear operation (e.g., sin, log, etc.), denoted by ψ.

Note that, the division operator is treated as taking reciprocal followed by a multiplication. The reciprocal itself is considered as a nonlinear operation.

The multiplication operator $*$ requires two memory fetches for the operands and one memory store for the result. Therefore, using the above complexity measure, complexity of performing a multiplication can be written as

$$W(*) = (3\ 0\ 1\ 0)^\top. \tag{6.2.2}$$

Similarly, addition/subtraction '$\pm$' operations have the complexity

$$W(\pm) = (3\ 1\ 0\ 0)^\top. \tag{6.2.3}$$

Performing a single nonlinear operation of a univariate function like $y = \psi(x)$ needs one memory fetch and one memory store. Therefore, the nonlinear operation ψ has the complexity

$$W(\psi) = (2\ 0\ 0\ 1)^\top. \tag{6.2.4}$$

Finally, the initialization operation "c" requires only one memory store,

$$W(c) = (1\ 0\ 0\ 0)^\top. \tag{6.2.5}$$

As a result, for the evaluation of a function F, which consists of l_1 initializations, l_2 additions, l_3 multiplications and l_4 nonlinear operations, the complexity is then given by

$$W(eval(F)) = \begin{bmatrix} 1 & 3 & 3 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} l_1 \\ l_2 \\ l_3 \\ l_4 \end{bmatrix}. \tag{6.2.6}$$

Here, it is assumed that the temporal additivity of a task holds, i.e., complexity of evaluating F is the sum of the complexities of the elemental operations that are performed to evaluate F. In other words, for the function F that involves l elementary operations $\varphi_i,\ i = 1, .., l$, the following equality is satisfied:

$$W(eval(F)) = \sum_{i=1}^{l} W(eval(\varphi_i)). \tag{6.2.7}$$

The actual run-time $T(F)$, which is required to evaluate F, can be estimated by

$$T(F) = w^\top W(eval(F)), \tag{6.2.8}$$

where w is a vector with four components that represents the number of clock cycles required for a memory operation (either fetch or store), an addition, a multiplication and a nonlinear operation. If the addition is taken as reference, w can be represented as $w = [\mu\ 1\ \pi\ \nu]^\top$. Reasonable assumptions for the values of μ, π and ν are:

$$\pi \geq 1,\ \nu \geq 2\pi \text{ and } \mu \geq \max(1, \pi/2). \tag{6.2.9}$$

In other words, we assume that a multiplication (π) is more expensive than an addition or subtraction, a nonlinear operation (ν) is at least as expensive as two multiplications and a memory operation (μ) is at least as expensive as one addition or half of a multiplication.

We now seek a scalar measure, which shows the factor of the run-time increase associated with performing a task (e.g., differentiating F in forward or reverse mode) with respect to the pure evaluation of F. The following proposition provides this scalar measure.

PROPOSITION 7. *Let $task(F)$ denote a task that is based on a function F and $\varphi_i \in \Psi,\ i = 1, .., l$ denote the set of elemental operations that are performed to evaluate F with Ψ being the set of all elementary operations. Assume that $task'(\varphi)$ denotes the related task to be performed for each elemental operation. Furthermore, we assume that the temporal additivity of a $task(F)$ exists such that the inequality*

$$W(task(F)) \leq \sum_{i=1}^{l} W(task'(\varphi_i) \tag{6.2.10}$$

holds. We also assume that elemental task boundedness holds, i.e., there exists a matrix C_{task} such that

$$W(task'(\varphi_i) \leq C_{task} W(eval(\varphi_i)),\ \forall\ \varphi_i \tag{6.2.11}$$

is satisfied. Then for the run-time functional $T(\cdot)$ the following inequality holds

$$T(task(F)) \leq \omega T(eval(F)), \tag{6.2.12}$$

where

$$\omega \equiv \max_{\varphi \in \Psi} \frac{w^\top W(task'(\varphi))}{w^\top W(eval(\varphi))} \leq \|DC_{task}D^{-1}\|_1 \text{ with } D \equiv diag(w) \tag{6.2.13}$$

and $\|\cdot\|_1$ denoting the 1-norm of matrices.

Proof: Complete proof is given in [**Gri00**].

By using the four element measure of the complexity for the upper bound ω, we have

$$\omega \equiv \max_{1 \leq i \leq 4} \frac{w^\top W_{task}\ e_i}{\omega^\top W_{eval}\ e_i}, \tag{6.2.14}$$

where $e_i \in I\!R^4$ is the ith Cartesian basis vector.

6.2.1. Complexity of the forward mode. As far as the forward mode is concerned, we refer to the Table 1 to find the complexities of the differentiated elementary operations.

For the assignment operator $x = c$, the differentiation gives two operations namely $\dot{x} = 0$ followed by $x = c$. Therefore, number of memory operations is doubled in the differentiated code, which results in

$$W(forward(c)) = \begin{pmatrix} 1+1 & 0 & 0 & 0 \end{pmatrix}^\top. \tag{6.2.15}$$

For the addition/subtraction operations like $x = y \pm z$, the forward code has $\dot{x} = \dot{y} \pm \dot{z}$ followed by $x = y \pm z$. In this case, both the number of memory and addition/subtraction operations are doubled. We then have the complexity

$$W(forward(\pm)) = \begin{pmatrix} 3+3 & 1+1 & 0 & 0 \end{pmatrix}^\top. \tag{6.2.16}$$

For the multiplication operator $x = y*z$, the differentiated code has $\dot{x} = \dot{y}*z+y*\dot{z}$ followed by $x = y * z$. We now have 6 memory operations, 1 addition and 3 multiplications and 3 assignments. In this case, the complexity is given by

$$W(forward(*)) = \begin{pmatrix} 3+3 & 0+1 & 1+2 & 0 \end{pmatrix}^\top. \tag{6.2.17}$$

Finally, for a nonlinear operation like $x = \psi(y)$, the differentiated code consists of $\dot{x} = \psi'(y) * \dot{y}$ followed by $x = \psi(y)$. Since the derivative of a nonlinear operation is also a nonlinear operation, we have totally 4 memory operations, 1 multiplication and 2 nonlinear operations. The complexity is then

$$W(forward(\psi)) = \begin{pmatrix} 2+2 & 0 & 0+1 & 1+1 \end{pmatrix}^\top. \tag{6.2.18}$$

Similar to Eq. (6.2.6), the complexity of the forward mode can be written as

$$W(forward(F)) = W(\dot{F}) = \underbrace{\begin{bmatrix} 2 & 6 & 6 & 4 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 2 \end{bmatrix}}_{W_{\dot{F}}} \begin{bmatrix} l_1 \\ l_2 \\ l_3 \\ l_4 \end{bmatrix} \tag{6.2.19}$$

If the bounded complexity of the forward mode is assumed, we have

$$W(forward(F)) = W(\dot{F}) \leq C * W(eval(F)), \tag{6.2.20}$$

where C is a matrix and the operator "$\leq$" should be understood component-wise, i.e., each element of the vector $W(\dot{F})$ should be less than or equal to the corresponding entry of the vector $C * W(F)$. Therefore, the matrix C can be thought as an upper bound for the complexity and as a measure of the computational cost of the differentiated code with respect to the primal code. For the forward mode, the upper bound is given from the equality condition as

$$W_{\dot{F}} = C * W_F \Rightarrow C = W_{\dot{F}} W_F^{-1}. \tag{6.2.21}$$

From the above equation, the matrix C is easily computed as

$$C = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 2 \end{bmatrix} \tag{6.2.22}$$

Although C gives a valuable measure of the complexity, a scalar measure of the run-time is desired to estimate the run-time performance. For this purpose, we employ Eq. (6.2.14) using the vector $w = (\mu, 1, \pi, \nu)^\top$ and matrices W_F, $W_{\dot{F}}$. Finally we get the slowdown factor:

$$\omega = \max\left(2, \frac{6\mu + 1 + 3\pi}{3\mu + \pi}, \frac{4\mu + \pi + 2\nu}{2\mu + \nu}\right). \tag{6.2.23}$$

By using the inequalities in (6.2.9), we get the upper bound

$$\omega \leq 2.8. \tag{6.2.24}$$

In conclusion, the differentiated code is slower than the underlying original code by a factor varying between 2 and 2.8. Compared to finite difference method, we can conclude that the forward AD is slower by a factor of 2 compared to the one-sided differences and almost as expensive as the central differences.

6.2.2. Complexity of the reverse mode. The complexity analysis of the reverse mode is more involved than the forward mode due to taping, which may require fetching/storing of large amount of data from/to the memory. To simplify the complexity analysis, we distinguish between the complexities of forward sweep, reverse sweep and the taping, i.e.,

$$W(reverse) = W(forward\ sweep) + W(reverse\ sweep) + W(taping). \tag{6.2.25}$$

We first focus on the complexities of forward and reverse sweeps and neglect the taping. Note that the elementary operations performed in the forward sweep are exactly the same as those in the original program. For the reverse sweep, on the other hand, the complexities of the adjoint elementary operations can be calculated by using Table 1.

The adjoint counterpart of the assignment $x = c$ is another assignment $\bar{x} = c$ in the reverse sweep. We then have in total 2 assignments in the differentiated code, and the complexity is given by

$$W(reverse(c)) = \big(1 + 1, \quad 0, \quad 0, \quad 0\big)^\top. \tag{6.2.26}$$

For the addition and subtraction operations $x = y \pm z$, we have in the reverse sweep $\bar{y} = \bar{y} \pm \bar{x}, \bar{z} = \bar{z} \pm \bar{x}$ and $\bar{x} = 0$. In this case the adjoint part consists of 3 assignments and 3 additions/subtractions. The complexity of $\pm$ operation in the reverse mode is then given by

$$W(reverse(\pm)) = \big(3 + 6, \quad 1 + 2, \quad 0, \quad 0\big)^\top. \tag{6.2.27}$$

The differentiation a multiplication $x = y * z$ in reverse mode gives $\bar{y} = \bar{y} + \bar{x} * z, \bar{z} = \bar{z} + \bar{x} * y$ and $\bar{x} = 0$. In this case the complexity can be written as

$$W(reverse(*)) = \big(3 + 8, \quad 0 + 2, \quad 1 + 2, \quad 0\big)^\top. \tag{6.2.28}$$

Finally, the adjoint counterpart of a nonlinear operation $x = \psi(y)$ is given as $\bar{y} = \bar{y} + \bar{x} * \psi'(y)$ and $\bar{x} = 0$. In total, we have 3 assignments, 1 addition, 1 multiplication and 2 nonlinear operations. The complexity is then given by

$$W(reverse(\psi)) = \big(2 + 5, \quad 0 + 1, \quad 0 + 1, \quad 1 + 1\big)^\top. \tag{6.2.29}$$

Similar to the Eq. (6.2.19), the complexity of the reverse mode can be written as

$$W(reverse(F)) = \underbrace{\begin{bmatrix} 2 & 9 & 11 & 7 \\ 0 & 3 & 2 & 1 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 2 \end{bmatrix}}_{W_{\bar{F}}} \begin{bmatrix} l_1 \\ l_2 \\ l_3 \\ l_4 \end{bmatrix}. \tag{6.2.30}$$

The matrix C for this case can be computed as

$$C = \begin{bmatrix} 2 & 3 & 5 & 3 \\ 0 & 3 & 2 & 1 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 2 \end{bmatrix}. \tag{6.2.31}$$

As a result, increase in the run-time of the reverse mode is given as

$$\omega = \max\left(3, \frac{11\mu + 2 + 3\pi}{3\mu + \pi}, \frac{7\mu + 1 + \pi + 2\nu}{2\mu + \nu}\right), \tag{6.2.32}$$

which leads to the scalar upper bound

$$\omega \leq 4. \tag{6.2.33}$$

As stated earlier, the above result is valid for the cases, in which the computational cost of taping can be neglected compared to adjoint operations. However, for a computer codes that result from the discretization of a PDE, run-time required for taping may easily dominate the overall run-time. In this case, the actual run-time increase is much higher than the theoretical upper bound.

In Fig. 5, run-time measurements made with a tangent linear Euler solver, generated by the forward mode are presented. The run-times of the differentiated and primal codes are plotted for different number of pseudo time-steps used in the time integration. From this figure, it can be seen that the run-time of the differentiated solver increases linearly and the slowdown factor stays almost constant around a value of 2.6, which is very close to the theoretical bound.

In Fig. 6, run-time measurements of the adjoint code, which is generated by differentiating the same solver in reverse mode, are presented. All computations are performed on the same grid such that a direct comparison with Fig. 5 can be made. As expected, due to excessive amount of taping, slowdown factor is much higher than the theoretical results if large number of time steps are taken for the time integration. It can be observed that, run-time of the adjoint code increases initially linearly, and then it drastically increases after 350 iterations. Since the bandwidth of physical memory is limited, taping operations become a bottleneck in this case for the adjoint evaluation. From these results, we can conclude that the reverse mode, when applied to a PDE solver in a black-box fashion, results in an adjoint solver with prohibitively large memory and run-time requirements.

6.3. AD Tools and implementation

Generating tangent linear or adjoint codes by hand is a difficult and error prone task. Using AD packages, this task can be significantly eased. In general, these packages employ one of the two different approaches to automatically generate the differentiated codes. These are namely the operator overloading and the source transformation methods, which are briefly presented in the following.

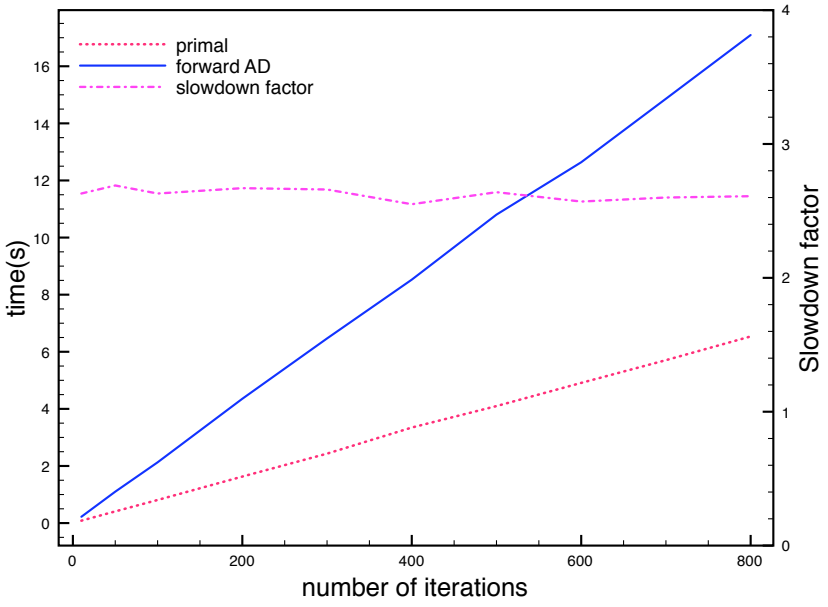

Figure 5. *Run-time measurements and slowdown factor for forward AD differentiated Euler code.*

6.3.1. Operator overloading. If the language of the input source code permits, one can replace the floating-point types in the source code with a new type that contains differential expressions, and then overload all the arithmetic operations using this type to propagate the derivative information. In this method, the AD tool is actually a library, which is a collection of the arithmetic operations of the overloaded type. The AD user has to only change the declaration of the active variables with the new data type and link the object code with the AD library. Therefore, Automatic Differentiation by the operator overloading approach is elegant and powerful in the sense that differentiation can be easily done even for a complex code. Furthermore, overloading concept can be extended to cover the features of languages like C/C++ and the overloaded library can be quickly modified to compute higher-order derivatives providing an immense flexibility to the AD user.

The overloading approach, despite its advantages that are mentioned above, has a serious drawback in the adjoint mode. The AD tool must be able to follow the control flow and the execution order of the original program. Since the adjoint mode performs the derivative computations in the reverse order, they cannot be simply run by the overloaded operations as done in the forward mode. The solution to this problem is to store the required derivative computations themselves on a stack, which are then recalled in the correct order in the reverse sweep after the overloaded program terminates. In other words, the overloaded statements crate a new program, named as the "code-list", which is then executed to evaluate adjoint values. Note that, storing the derivative computations may be quite memory intensive since the code-list grows with the execution time of the underlying program.

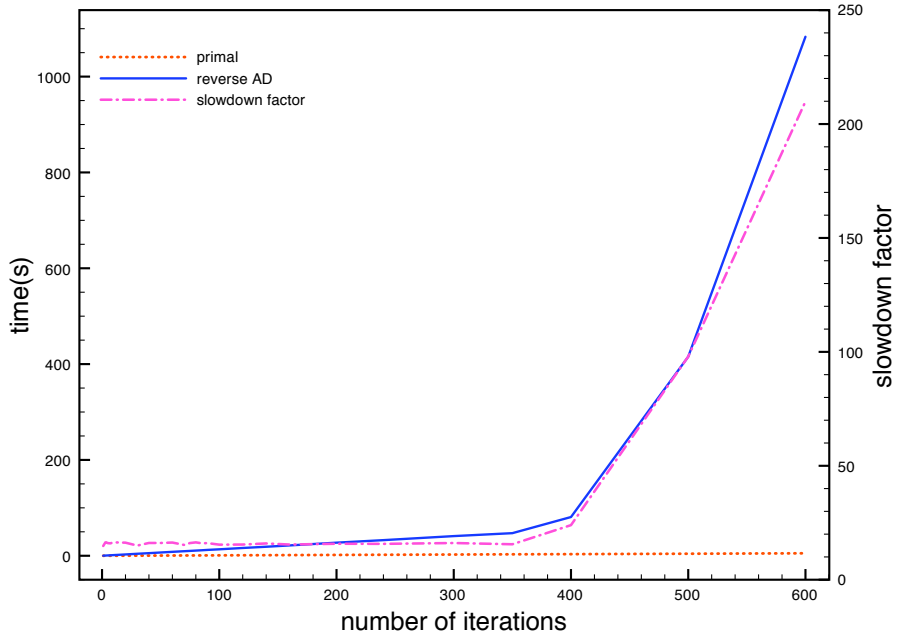

Figure 6. *Run-time measurements and slowdown factor for the reverse mode of AD.*

A number of refinements, which are mostly based on the source transformation approach, can help to diminish this storage cost but for most of the cases the overloading approach still results in adjoint codes that are significantly slower than the theoretical estimates. As a result, typical size of applications that can be efficiently handled by operator overloading is remarkably smaller than the source transformation [**HD08**]. To give an idea about the performance of the adjoint solvers that are generated by the operator overloading tools, performance measurements are made using the adjoint version of the compressible Euler code *TAUij*. The TAUij code is a structured quasi $2D$ version of the TAU code, developed at the German Aerospace Center in Braunschweig (DLR) [**Hei06**]. Two different adjoint versions of *TAUij* have been generated by using operator overloading based AD packages *ADOL-C* [**GJU96**] and *DCO* [**LLN11**]. In both cases, adjoint solvers use piggybacking strategy. As the test case, transonic flow around the RAE 2822 airfoil is taken. The computational grid has 5313 control volumes. The increase in run-time and memory demand of the adjoint codes with respect to the original code are presented in Table 3. From the results, it can be observed that slowdown factors of the adjoint codes are much higher than the theoretical results. Furthermore, the tape size goes beyond 1Gb and increase in memory demand is around factor of 90. As a result, it can be argued that operator overloading approach is not feasible for large-scale problems of PDE optimization problems due to prohibitively large run-time and memory requirements.

6.3.2. Source transformation. As an alternative to the operator overloading, one can instead decide to explicitly implement a new source code that computes

Case	run-time	memory demand	tape size (MB)
Euler adjoint (*ADOL-C*)	102.0	93.2	1240
Euler adjoint (*DCO*-chunk tape)	34.4	92.4	1229
Euler adjoint (*DCO*)	23.86	92.4	1229

Table 3. *The run-time and memory increase of the adjoint codes that are generated by operator overloading tools for the Euler Solver TAUij.*

package	language	strategy	adjoint mode
ADOL-C	C/C++	operator overloading	yes
ADIC	C/C++	source transformation	no
TAPENADE	Fortran/C	source transformation	yes
CppAD	C/C++	operator overloading	yes
ADIFOR	Fortran	source transformation	no
TAF	Fortran	source transformation	yes
FADBAD	C/C++	operator overloading	yes

Table 4. *The most established AD packages.*

the derivatives. This means that the original source code must be first parsed to build an internal representation. By using this representation and user specification, the differentiated code is then generated by the AD tool. This approach allows the AD tool to perform a global analysis of the source code, which helps to produce more efficient adjoint codes. The development of a source transformation tool is more involved and time consuming than the operator overloading approach. This is the main reason why operator overloading based AD tools appeared earlier and are more numerous. Furthermore, source transformation tools are fragile and more effort is needed to maintain them to account for the new programming constructs and styles.

In Table 4, the most prominent AD packages are listed. For the time being, source transformation tools can only handle `Fortran` and `C` only. For `Fortran 77/95` programs, these tools are quite mature and can be applied to complex simulation codes. For `C` programs, however, their usage is rather restricted to simple codes. In conclusion, source transformation tools generate very efficient adjoint codes compared to operator overloading.

6.4. Performance improvement techniques for the adjoint solvers

The adjoint codes that are generated by an AD tool may initially show a poor performance in terms of run-time and memory. However, computational efficiency can be enhanced significantly by applying simple techniques. In the following, we present three strategies that are particularly useful for the adjoint flow solvers, generated by AD tools. The first strategy is the elimination of redundant taping from the differentiated source code. This is possible only in source transformation method and cannot be applied when an operator overloading tool is used. The second strategy is the special treatment of independent loops, which may exist in

the source code. This technique can be easily used for the source transformation method, but its usage for the operator overloading requires more effort. The third strategy is the replacement of AD generated code with the hand differentiated code for linear equation solvers, which can be used in both source transformation and operator overloading methods.

6.4.1. Elimination of redundant taping. When the source transformation method is used to generate an adjoint code, taping mechanism inserted into the derivative code by the AD tool may include a lot of unnecessary taping, and thus may be far from being optimal. These taping operations can be removed from the source code by hand without influencing the actual adjoint computations. The simplified code has then less run-time and memory requirements, which may result in a significant speedup. For example, consider the following `Fortran` example:

```
      PROGRAM REDUNDANT_TAPING_EXAMPLE
      REAL*8 x(100000),y(100000)
      CALL G(X,Y)
      END

      SUBROUTINE G(X,Y)
      REAL*8 x(100000),y(100000)
      DO I=1,100000
          X(I)=1.0
      ENDDO
      CALL F(X,Y)
      DO I=1,100000
          Y(I)=2.0*SIN(X(I))
      ENDDO
      END

      SUBROUTINE F(X,Y)
      REAL*8 x(100000),y(100000)
      x(5)=x(2)+x(4)*COS(x(6))
      END
```

If the subroutine `G`, the differentiated subroutine `G_B` as it is generated by the AD tool is:

```
      SUBROUTINE G_B(x, xb, y, yb)
      REAL*8 x(100000), y(100000)
      REAL*8 xb(100000), yb(100000)
      INTEGER i
C forward sweep
      DO i=1,100000
        x(i) = 1.0
        CALL PUSHREAL(x(i))
      ENDDO

      CALL F(x, y)
```

```
C reverse sweep
      DO i=100000,1,-1
        xb(i) = xb(i) + 2.0*COS(x(i))*yb(i)
        yb(i) = 0.0
        CALL POPREAL(x(i))
      ENDDO

      CALL F_B(x, xb, y)
      DO i=100000,1,-1
        xb(i) = 0.0
      ENDDO
      END

      SUBROUTINE F_B(x, xb, y)
      REAL*8 x(100000), y(100000)
      REAL*8 xb(100000)
      xb(2) = xb(2) + xb(5)
      xb(4) = xb(4) + COS(x(6))*xb(5)
      xb(6) = xb(6) - x(4)*SIN(x(6))*xb(5)
      xb(5) = 0.0
      END
```

Note that in the adjoint subroutine `G_B`, all entries of vector `X` are taped before calling the subroutine `F`, which requires 800KB space in memory. However, taping the complete vector is unnecessary since the subroutine `F` changes only the value of `X(5)`, not the complete vector. Therefore, it is sufficient to tape only `X(5)`. As a result, memory requirement of the adjoint code and run-time overhead required for the taping are reduced significantly. In order to give a brief idea on how much improvement can be achieved by removing the redundant taping, we present run-time and memory requirements of adjoint Euler and Navier-Stokes solvers before and after the code optimization in Table 5. From the run-time and memory measurements, it is clear that a significant improvement can be achieved by a slight modification the adjoint code to eliminate redundant taping.

Case	run-time factor	memory factor
Euler adjoint (not optimized)	25.3	53
Euler adjoint (optimized)	7.2	4.6
Navier-Stokes adjoint (not optimized)	34.9	110.4
Navier-Stokes adjoint (optimized)	6.4	5.9

Table 5. *Factor of run-time and memory increase of the adjoint code before and after code optimization.*

6.4.2. Independent loops. Another powerful method to reduce the run-time and memory requirements of adjoint solvers is the special treatment of independent loops. A loop is called as an independent loop if the loop order can be arbitrarily changed without effecting the result. By identifying such independent loops in

the underlying source code, source transformation tools can produce much more efficient adjoint codes [**HFH01**]. For example, the following loop is an independent loop:

```
DO i=1,n
   temp1= 2.0*x(i)**2.0
   temp2= (y(i)+x(i))/temp1
   sum=sum+temp2
ENDDO
```

On the other hand, the following loop is not an independent loop since the value of `x(i)` depends on `x(i-1)`:

```
DO i=2,n
   x(i)=3*x(i-1)
   temp1= 2.0*x(i)**2.0
   temp2= (y(i)+x(i))/temp1
   sum=sum+temp2
ENDDO
```

To give a concrete idea on how much gain in computational efficiency can be achieved, we consider the following example subroutine:

```
SUBROUTINE F(X,Y,sum)
REAL*8 x(100000),y(100000),temp1,temp2,temp3,sum
INTEGER I

sum=0.0
DO I=1,100000
temp1=x(i)*cos(x(i))
x(i)=x(i)*2.0
temp2=x(i)*x(i)
x(i)=sin(x(i))
temp3=x(i)*1.2
y(i)=temp1+temp2+temp3
sum=y(i)* y(i)
ENDDO

END
```

Now suppose we want to evaluate the derivative of `sum` with respect to `X`. In this case, differentiation in reverse mode generates the following subroutine:

```
      SUBROUTINE F_B(x, xb, y, sum, sumb)
      REAL*8 x(100000),y(100000),temp1, temp2, temp3, sum
      REAL*8 xb(100000),yb(100000),temp1b, temp2b, temp3b, sumb
      INTEGER i

C forward sweep
      DO i=1,100000
        temp1 = x(i)*COS(x(i))
        CALL PUSHREAL8(x(i))
        x(i) = x(i)*2.0
```

```
        temp2 = x(i)*x(i)
        CALL PUSHREAL8(x(i))
        x(i) = SIN(x(i))
        temp3 = x(i)*1.2
        y(i) = temp1 + temp2 + temp3
      ENDDO

C reverse sweep
      DO i=100000,1,-1
        yb(i) = yb(i) + 2*y(i)*sumb
        temp1b = yb(i)
        temp2b = yb(i)
        temp3b = yb(i)
        yb(i) = 0.0
        xb(i) = xb(i) + 1.2*temp3b
        CALL POPREAL8(x(i))
        xb(i) = 2*x(i)*temp2b + COS(x(i))*xb(i)
        CALL POPREAL8(x(i))
        xb(i) = (COS(x(i))-x(i)*SIN(x(i)))*temp1b + 2.0*xb(i)
      ENDDO
      sumb = 0.0
      END
```

Note that, in the differentiated subroutine the values of `X(i)` is saved 2×10^5 times on a stack in the primal sweep. In the reverse sweep, these values are then retrieved from the stack in the reverse order for the adjoint computation. The peak memory usage for the taping in this case is 1.6 MB. However, if the `DO` loop in is treated as an independent loop, the same AD tool generates a different differentiated code, which produces exactly the same result:

```
      SUBROUTINE F_B(x, xb, y, sum, sumb)

      REAL*8 x(100000),y(100000), temp1, temp2, temp3, sum
      REAL*8 xb(100000),yb(100000),temp1b, temp2b, temp3b, sumb
      INTEGER i

      DO i=1,100000
        temp1 = x(i)*COS(x(i))
        CALL PUSHREAL8(x(i))
        x(i) = x(i)*2.0
        temp2 = x(i)*x(i)
        CALL PUSHREAL8(x(i))
        x(i) = SIN(x(i))
        temp3 = x(i)*1.2
        y(i) = temp1 + temp2 + temp3
        yb(i) = yb(i) + 2*y(i)*sumb
        temp1b = yb(i)
        temp2b = yb(i)
```

```
      temp3b = yb(i)
      yb(i) = 0.0
      xb(i) = xb(i) + 1.2*temp3b
      CALL POPREAL8(x(i))
      xb(i) = 2*x(i)*temp2b + COS(x(i))*xb(i)
      CALL POPREAL8(x(i))
      xb(i) = (COS(x(i))-x(i)*SIN(x(i)))*temp1b + 2.0*xb(i)
      sumb = 0.0
    ENDDO
    sumb = 0.0
    END
```

Note that for an independent loop, forward and reverse sweep loops can joined without changing the result. The new subroutine has the same number of taping operations in total but the peak memory usage is reduced to 16 Bytes.

6.4.3. Linear equation solvers. Another technique, which can be used in the adjoint solver development is the special treatment of the linear equation solvers. For a CFD solver, which has an implicit time discretization, linear equation solving is the most time consuming part in the computation. In this case, it may not be ideal to use the AD-generated version of the adjoint linear solver. For example, consider the following subroutine that solves a system of linear equations $Ax = b$ by Gauss elimination method:

```
SUBROUTINE GAUSS_ELIMINATION(A,x,b,n)
      PARAMETER (N=2000)
      REAL*8 A(N,N), X(N), B(N), F, SUM
      INTEGER I,J,K
C Decomposition (Elimination)
      DO K = 1, N-1
         DO I = K+1, N
            F = A(I,K) / A(K,K)
            DO J = K+1, N
               A(I,J) = A(I,J) - F * A(K,J)
            ENDDO
           B(I) = B(I) - F * B(K)
         ENDDO
      ENDDO
C  Back-substitution
       X(N) = B(N) / A(N,N)
       DO I = N-1, 1, -1
       SUM = 0.0
       DO J = I+1, N
          SUM = SUM + A(I,J) * X(J)
       ENDDO
      X(I) = (B(I) - SUM) / A(I,I)
      ENDDO
      END
```

If the above subroutine is differentiated in a black-box fashion using reverse mode, the AD tool generates the following adjoint subroutine:

```
      SUBROUTINE GAUSS_ELIMINATION_B(a, ab, x, xb, b, bb)
      PARAMETER (n=2000)
      REAL*8 a(n, n), x(n), b(n), f, sum,tmp,tmp0
      REAL*8 ab(n, n), xb(n), bb(n), fb, sumb
      INTEGER i, j, k
C Forward sweep
      DO k=1,n-1
        DO i=k+1,n
          CALL PUSHREAL8(f)
          f = a(i, k)/a(k, k)
          DO j=k + 1,n
            CALL PUSHREAL8(a(i, j))
            a(i, j) = a(i, j) - f*a(k, j)
          ENDDO
          CALL PUSHREAL8(b(i))
          b(i) = b(i) - f*b(k)
        ENDDO
      ENDDO

      x(n) = b(n)/a(n, n)
      DO i=n-1,1,-1
        CALL PUSHREAL8(sum)
        sum = 0.0
        DO j=i+1,n
          sum = sum + a(i, j)*x(j)
        ENDDO
        CALL PUSHREAL8(x(i))
        x(i) = (b(i)-sum)/a(i, i)
      ENDDO

C reverse sweep starts here
      DO i=1,n-1,1
        CALL POPREAL8(x(i))
        bb(i) = bb(i) + xb(i)/a(i, i)
        sumb = -xb(i)/a(i, i)
        ab(i, i) = ab(i, i) - (b(i)-sum)*(xb(i)/a(i, i))/a(i, i)
        xb(i) = 0.0
        DO j=i,i+1,-1
          ab(i, j) = ab(i, j) + x(j)*sumb
          xb(j) = xb(j) + a(i, j)*sumb
        ENDDO
        CALL POPREAL8(sum)
      ENDDO

      tempb0 = xb(n)/a(n, n)
      bb(n) = bb(n) + xb(n)/a(n, n)
```

```
      ab(n, n) = ab(n, n) - b(n)*(xb(n)/a(n, n))/a(n, n)
      xb(n) = 0.0

      DO k=n-1,1,-1
        DO i=n,k+1,-1
          CALL POPREAL8(b(i))
          fb = -(b(k)*bb(i))
          bb(k) = bb(k) - f*bb(i)
          DO j=n,k+1,-1
            CALL POPREAL8(a(i, j))
            fb = fb - a(k, j)*ab(i, j)
            ab(k, j) = ab(k, j) - f*ab(i, j)
          ENDDO
          CALL POPREAL8(f)
          ab(i, k) = ab(i, k) + fb/a(k, k)
          ab(k, k) = ab(k, k) - a(i, k)*(fb/a(k, k))/a(k, k)
        ENDDO
      ENDDO
      END
```

Note that, AD generated adjoint subroutine has high memory and run-time requirements due to excessive amount of taping operations. To reduce the memory and run-time, one can implement a hand-discrete version of the adjoint linear solver and use it instead of the AD generated code. If it is assumed that the residual $r = b - Ax$ is zero, one can derive the expressions for the adjoint variables $\bar{A}$ and $\bar{b}$ as

$$\bar{b} = A^{-\top}\bar{x} \Rightarrow A^{\top}\bar{b} = \bar{x}, \tag{6.4.1}$$

$$\bar{A} = -x^{\top}\bar{b}. \tag{6.4.2}$$

The first expression is a system of linear equations and it can be solved by the same linear equation solver. The second expression is a dyadic product of two vectors. The hand-discrete version of the subroutine `GAUSS_ELIMINATION_B` is then can be written as

```
      SUBROUTINE GAUSS_ELIMINATION_B(a, ab, x, xb, b, bb)
      PARAMETER (n=2000)
      REAL*8 a(n,n), ab(n,n),atranspose(n,n)
      REAL*8 x(n), xb(n), b(n), bb(n),incrbb(n)
      INTEGER i,j

      CALL GAUSS_ELIMINATION(a,x,b)

C transpose of the matrix A
      DO i=1,n
        DO j=1,n
          atranspose(i,j) = a(j,i)
        ENDDO
      ENDDO
```

```
CALL GAUSS_ELIMINATION(atranspose, incrbb, xb)
DO j=1,n
  bb(j) = bb(j) + incrbb(j)
ENDDO

DO i=1,n
  DO j=1,n
    ab(i,j) = ab(i,j) - x(j)*incrbb(i)
  ENDDO
ENDDO
END
```

The hand-discrete version of the adjoint linear solver performs adjoint computations without doing any taping. In this way, memory usage and run-time requirements are reduced significantly. In Fig. 7, factor of run-time increase of the hand-discrete and AD versions of the `GAUSS_ELIMINATION_B` are presented for different problem sizes. From the results it can be observed that the hand-discrete version is much more efficient than the AD generated one.

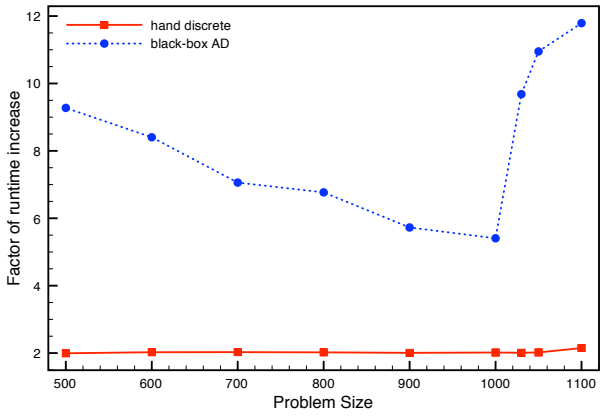

Figure 7. *The factor of run-time increase of the black-box AD and hand-discrete adjoints of the linear solver.*

Although hand-discrete version is significantly faster than the AD generated code, it has a serious drawback. If the hand-discrete approach is used for iterative solvers, it may lead to serious errors if residual is not close to zero. In many practical cases, accurate solution is not required and only a few iterations in the linear solver are performed. For example, in a density-based solver, it is often not necessary to resolve each pseudo time-step with a high accurately to be able to get an accurate steady-state solution. In these cases, the main assumption $x = A^{-1}b$ while deriving the Eqs. (6.4.2) and (6.4.2) is not valid, and therefore hand-discrete version may lead to significant errors in the adjoint solution. In Fig. 8, error introduced by the hand-discrete approach is shown. As the test problem, a $2D$ Laplace problem that is solved by the SIP (Strongly Implicit Procedure) method is taken. In the figure,

average error (relative error divided by the total number of cells) is plotted for 10×10 and 20×20 grids along with the number of iterations. Note that, black-box AD code gives exact results, which perfectly match with finite difference results at any residual level. The results of the hand-discrete version, however, strongly depend on the residual level. If the primal equation system $Ax = b$ is solved with an accuracy of three decade residue fall, adjoint results are quite accurate. On the other hand, if only a few iterations are performed in the primal solver, hand-discrete results deviate significantly from the black-box AD results and error associated with neglecting the residual is large. From these results, it can be argued that hand-discrete method is suitable only if good accuracy in the primal solver can be achieved. In other cases with poor convergence or inexact solution, however, this method may lead to significant errors.

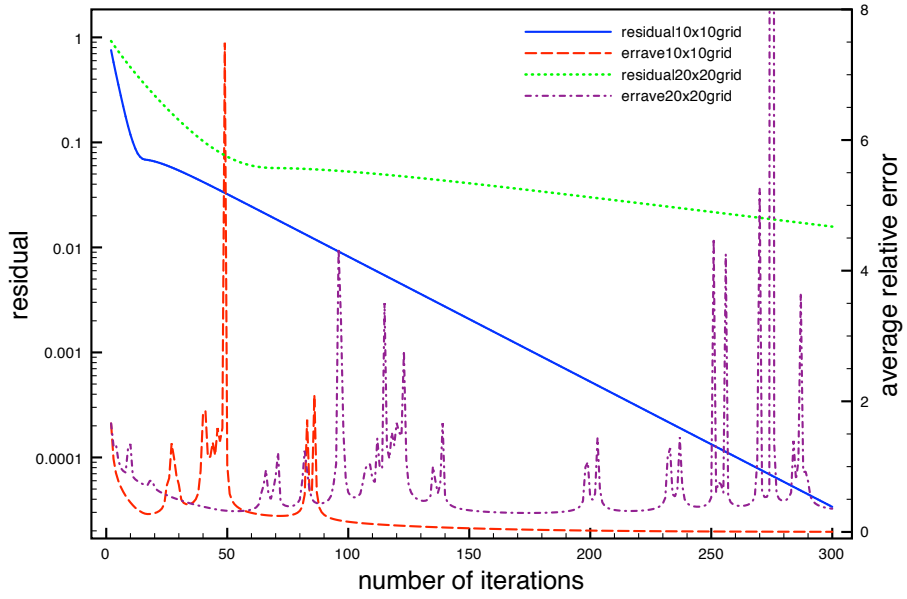

Figure 8. *The relative error between the black-box AD and hand-discrete adjoints for a 2D Laplace problem.*

CHAPTER 7

Numerical results and discussion

In this chapter, we present the results that are obtained by the one-shot method for three different airfoil shape optimization scenarios. In order to demonstrate the flexibility of the one-shot method based on the consistent discrete adjoint approach, various established flow solvers are used for these optimization studies. The first scenario is the airfoil optimization in transonic flow conditions. For this scenario, two different test cases are taken, on which single-step and multi-step variants of the one-shot method are tested. In both cases, structured compressible Euler solvers are used to compute state vectors. The second scenario is the airfoil optimization in supersonic flow conditions. This test case is especially important to demonstrate the advantages of the free-node parameterization and unstructured grids. The last test case is chosen as the airfoil optimization in subsonic turbulent flow. For this configuration, a structured incompressible RANS solver is used. The subsonic flow case is primarily chosen to demonstrate the feasibility of the one-shot method on a sophisticated flow solver. In the following, we first present the performance results of the adjoint codes used in these three scenarios are presented in terms of memory and run-time. Then, we present and discuss shape the results of shape optimization studies.

7.1. The run-time and the memory requirements of the adjoint codes

In this section, run-time and memory measurements results obtained from the adjoint flow solvers are presented. Note that, the adjoint solvers are based on the consistent discrete adjoint approach and they are generated using the source transformation AD tool *Tapenade* for the present work. The underlying primal flow solvers used for these adjoint solvers are:

- structured compressible Euler/Navier-Stokes solver *struct2d* [**Bla01**].
- unstructured compressible Euler/Navier-Stokes solver *unstr2d* [**Bla01**].
- structured incompressible RANS solver *ELAN* [**Xue98**].

All performance measurements are done on a 2.4GHz Intel Core i5 processor with 4GB 1067 MHz DDR3 RAM. Due to oscillations that may happen in the run-time measurements, all the results are averaged and normalized with the total number of iterations used in the simulation. Therefore, presented values show the computational time required for one pseudo time iteration in each case. The measurements are performed using the Fortran `SYSTEM_CLOCK` function and code pieces that are not required in the adjoint part are not taken into account. Typical examples of these passive parts of the code are the IO and preprocessing subroutines (e.g., post-processing output, reading grid input, etc.). For the measurements of the memory demand, profiling tool VALGRIND has been employed, and all presented results indicate the peak memory usage.

In Table 1, measurements for the adjoint version of *struct2d* code are presented. The run-time and peak memory requirements for the primal and piggy-back solvers are presented using different grid levels. In Table 2, measurements results that are obtained for the same solver including viscous terms on finer grids are presented. In Tables 3 and 4, results that are obtained from the adjoint versions of *unstr2d* and *ELAN* are shown on different grid levels. In all of the three adjoint solvers, factor of memory and run-time increase proved to be grid independent. As far as the run-time is concerned, a factor between 6 and 8 is achieved. The factors of memory increase vary between approximately 4 and 7.

Case	# of CVs	run-time (s)	memory (MB)	run-time factor	memory factor
primal	92K	0.428	25.3	1.0	1.0
adjoint	92K	2.972	137.6	6.94	5.43
primal	21K	0.088	6.5	1.0	1.0
adjoint	21K	0.632	33.7	7.19	5.18
primal	6K	0.026	2.6	1.0	1.0
adjoint	6K	0.189	12.0	7.26	4.61

Table 1. *Run-time and memory measurements for the struct2d (compressible Euler).*

Case	# of CVs	run-time (s)	memory (MB)	run-time factor	memory factor
primalS	156K	1.328	60.4	1.0	1.0
adjoint	156K	8.963	374.0	6.74	6.19
primal	48K	0.384	19.5	1.0	1.0
adjoint	48K	2.583	119	6.76	6.10
primal	25K	0.191	10.6	1.0	1.0
adjoint	25K	1.229	63.0	6.43	5.94

Table 2. *Run-time and memory measurements for the struct2d (compressible Navier-Stokes).*

Case	# of CVs	run-time (s)	memory (MB)	run-time factor	memory factor
primal	67K	1.227	18.5	1.0	1.0
adjoint	67K	9385	120.6	5.07	6.34
primal	24K	0.379	8.3	1.0	1.0
adjoint	24K	2.026	45.3	5.34	5.45
primal	10K	0.129	3.7	1.0	1.0
adjoint	10K	0.719	17.8	5.57	4.81

Table 3. *Run-time and memory measurements for the unstruct2d (compressible Navier-Stokes).*

Case	# of CVs	run-time (s)	memory (MB)	run-time factor	memory factor
primal	122K	0.489	145	1.0	3.73
adjoint	122K	3.402	541	6.96	3.73
primal	43K	0.173	51	1.0	1.0
adjoint	43K	1.216	198.5	7.02	3.90

Table 4. *Run-time and memory measurements for the ELAN (incompressible RANS).*

7.2. Airfoil optimization in inviscid transonic flow

The first test case is the drag minimization of a RAE 2822 airfoil. As far as the flow conditions are concerned, inflow Mach number is $M_\infty = 0.73$ and angle of attack (AoA) is taken as $2°$. For this test case, governing state equations are the compressible Euler equations. The numerical solution for the state vector is obtained by the *TAUij* solver [**Hei06**], which is a structured $2D$ version of the DLR *TAU* code. The *TAUij* code is implemented in C and comprises approximately 6000 lines of code distributed over several files. For the discretization of the convective fluxes, the flux-vector splitting MAPS+ scheme of Rossow [**Ros00**] is used. As the temporal discretization, explicit pseudo time-stepping method using fourth order Runge-Kutta scheme is taken. To accelerate the convergence of the simulation, local time stepping, explicit residual smoothing and geometric multi-grid methods are used. As the boundary conditions, slip wall condition for the airfoil surface and far-field boundary condition for the outer boundary are applied.

As the shape parameterization, Hicks-Henne parameterization is chosen. In this type of parameterization, deformations that are applied to the initial airfoil shape are evaluated by the Hicks-Henne functions, which are scaled by certain parameters. The basic idea is to evaluate these basis functions, and to deform the camberline of the airfoil accordingly. The new airfoil shape is obtained by using

the deformed camberline and the initial thickness distribution. Note that, the only design parameters in the optimization study are the scaling parameters used to evaluate camberline deformation.

The computational grid used for this study is a $2D$ structured grid with 161×33 control volumes. In Fig. 1, computational grid and the pressure distribution around the RAE 2822 airfoil are shown. It can be observed that there exists a severe inviscid shock on the suction side of the airfoil at $x \approx 0.6$. Note that, for this flow configuration viscous effects are not significant and do not contribute much to the drag. Therefore, inviscid flow assumption made in deriving Euler equations is well justified. Since the drag force is mainly caused by the form drag, elimination of the inviscid shock is the main goal in the optimization.

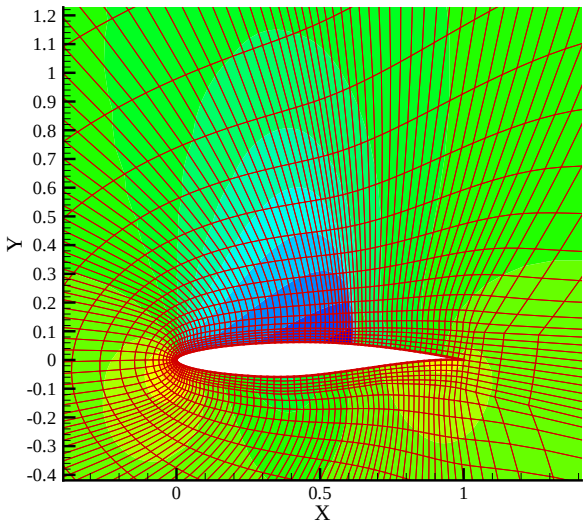

Figure 1. *The structured grid around the RAE 2822 airfoil and the initial pressure distribution,* $Ma = 0.73$ *and* $AoA{=}2^\circ$ *(reprinted from* [**GWOM12**] *with permission).*

As the optimization algorithm, we employ the single-step one-shot approach using augmented Lagrangian as described in Chapter 2. The objective function used in the optimization study is taken as the drag coefficient. The values of penalty coefficients α and β for the augmented Lagrangian are computed using Eq. (2.5.42) based on the estimation of the contraction rate of primal solver. The derivatives that are required for the one-shot method and the design space preconditioner are computed by employing the adjoint chain as outlined in Chapter 3. The adjoint versions of the flow solver *TAUij* and the mesh deformation tool *meshdefo* are generated by using the operator overloading based AD tool *ADOL-C*. Note that, by using AD, all the parts of the flow solver (e.g., multi-grid solver, implicit residual smoothing, etc.) are fully differentiated. In this way, the reduced gradient and adjoint vectors are accurately computed in each optimization cycle.

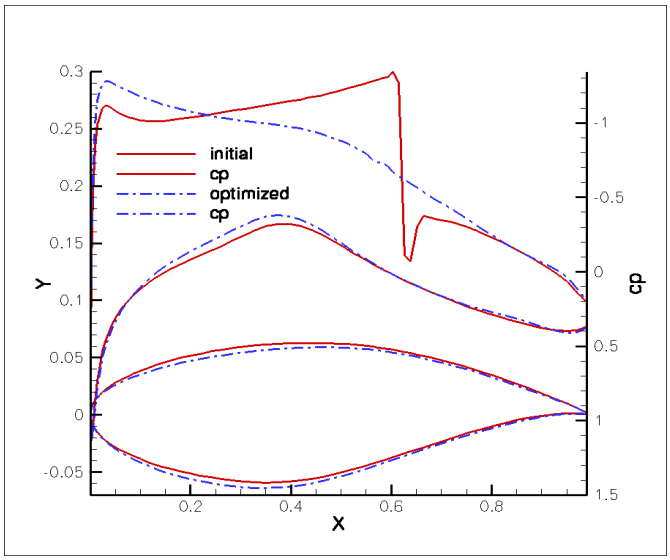

Figure 2. *The optimized airfoil with C_p distribution, $Ma = 0.73$ and AoA=2° (reprinted from* [**OG09**]*).*

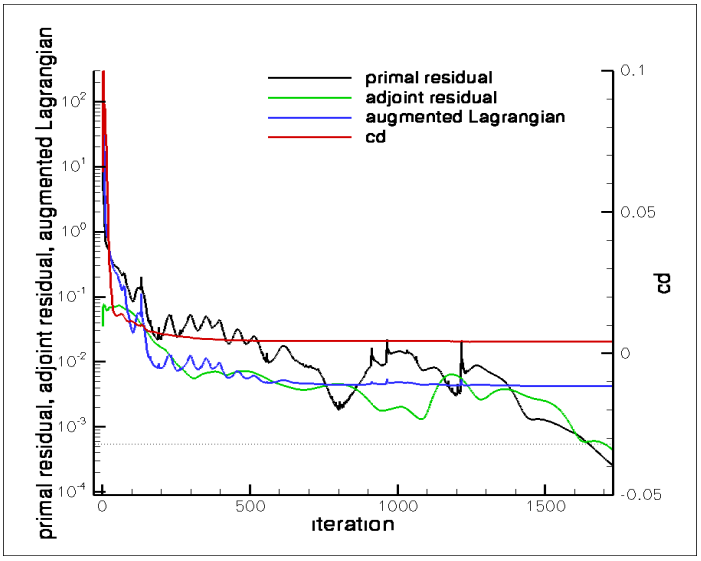

Figure 3. *Optimization history, $Ma = 0.73$ and AoA=2° (reprinted from* [**OG09**]*).*

The result of the optimization is presented in Fig. 2. In the figure, RAE 2822 and optimized airfoil geometries and their C_p distributions are plotted. It can be observed that, there is only a slight difference between the shapes. However, from C_p distribution curves, it can be observed that the strong inviscid shock disappears

in the optimized geometry. The drag coefficients of the optimized and initial airfoils are computed as 0.00442 and 0.00908 respectively. Hence, approximately 60% decrease in drag coefficient is achieved. In Fig. 3, the optimization histories of the augmented Lagrangian, drag coefficient as well as primal and adjoint residuals are plotted. The coupled iterations converge after approximately 1600 iterations. In Fig. 4, convergence history of the simulation done for the RAE 2822 airfoil is shown. It can be observed that the primal convergence is reached after approximately 400 pseudo time iterations. In conclusion, the retardation factor of the overall optimization process is measured as 4 in iteration counts. However, due to expensive evaluations of second order derivatives and the overhead related with the *ADOL-C* taping mechanism, the retardation factor in run-time is much higher.

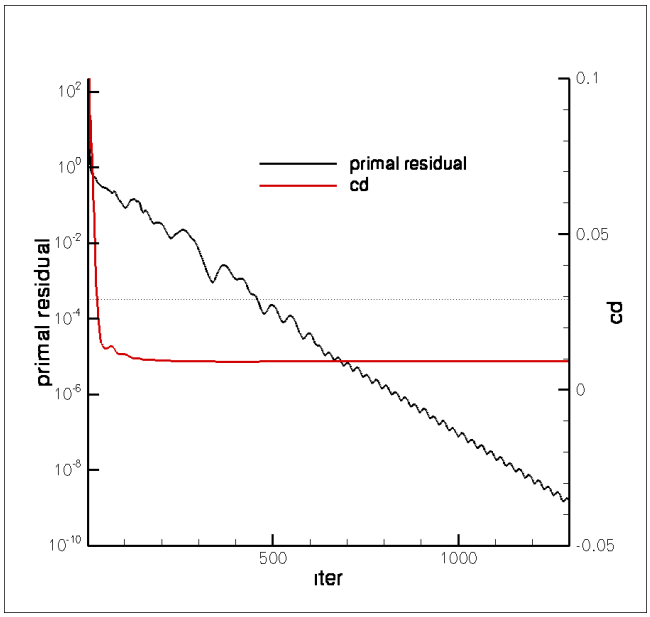

Figure 4. *Convergence history for the simulation of the RAE 2822 airfoil,* $Ma = 0.73$ *and* AoA=2° *(reprinted from* [**OG09**]*).*

As the second test case, drag minimization of the symmetric NACA 0012 airfoil is chosen. The Mach number for this configuration is 0.8 and angle of attack is 1.25°. Similar to the previous problem, this test case is also a benchmark case that is typically used to validate convective schemes in Euler solvers. The flow solver used for the simulation is the Blazek's code *struct2d*. Similar to the *TAUij* solver, the *struct2d* is a density based, cell-centered, structured Finite Volume solver, and is used for simulation of external and internal compressible flows. It has several convective schemes, out of which the JST scheme is used for the present test case.

In order to choose a suitable grid size for the optimization, a grid independency study is performed using three grid levels. In Table 5, the values of drag and lift coefficients (C_d and C_l) obtained for the each grid are presented. In each simulation, pseudo time iterations are performed until five decade residue fall is achieved. From the results, it can be observed that the middle-sized 192×108 grid is appropriate

to ensure independent results. Therefore, this grid is used in the optimization step. Note that, although computational grid is deformed after each optimization cycle, number of control volumes and grid topology are not allowed to vary.

number of CVs	dimension of state space	C_d	C_l
6144	25576	0.02356	0.3514
20736	82944	0.02348	0.3655
92160	368640	0.02342	0.3664

Table 5. *Grid independency study for the initial NACA 0012 geometry,* $Ma = 0.8$ *and* AoA=1.25°.

In Fig. 5, the 192 × 108 C-grid generated for the initial NACA 0012 airfoil is shown. Note that, an extra refinement in the trailing edge region is used to be able to capture shocks accurately. In Fig. 6, pressure contours around the initial geometry are plotted. As it can be seen from the figure, a strong shock occurs in the suction side of the airfoil, which is the major contribution to the drag. The aim of the optimization process is then to find the optimal airfoil shape, which has the minimum amount of drag while generating the same lift as the initial airfoil. Similar to the previous test case, reducing the inviscid shock on the suction side of the NACA 0012 airfoil is the main goal of the optimization.

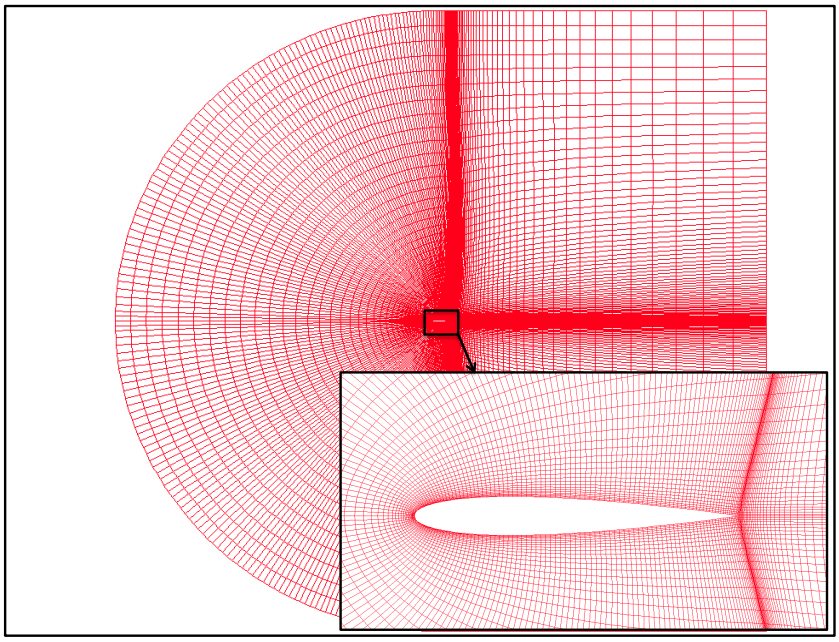

Figure 5. *The C-grid for the initial NACA 0012 airfoil.*

For the validation of the adjoint solver, which computes the adjoint vector and the reduced gradient, shape sensitivities generated by the adjoint and tangent linear solvers are compared with each other. Note that, only correct values of the adjoint vector lead to accurate shape sensitivities. For this reason, a comparison based on the shape sensitivities seems to be the most suitable choice for the validation. The exact shape sensitivities are generated by the tangent linear solver, which is generated by using the forward mode of AD. The validation of the tangent linear solver itself is done by using finite difference results. The shape sensitivities are generated by the adjoint solver in a piggy-back manner without any design updates, and coupled piggy-back iterations are performed until five decade residue fall is achieved. For the comparison, only sensitivities of the drag coefficient with respect to grid coordinates in y direction on the wall boundary are taken. In Fig. 7, comparison of results computed by the adjoint and tangent linear solvers is given. The distribution of the relative error is shown in Fig. 8. From the results, it can be observed that the shape sensitivities get large values in three distinct locations: trailing edge, leading edge and shock position. In locations that are close to leading and trailing edges, sensitivities show an extremely oscillating character. The sensitivities that are calculated by the adjoint method match perfectly with the exact tangent linear results in all of the grid points. The results clearly indicate that an excellent accuracy can be achieved using the consistent discrete adjoint method based on AD.

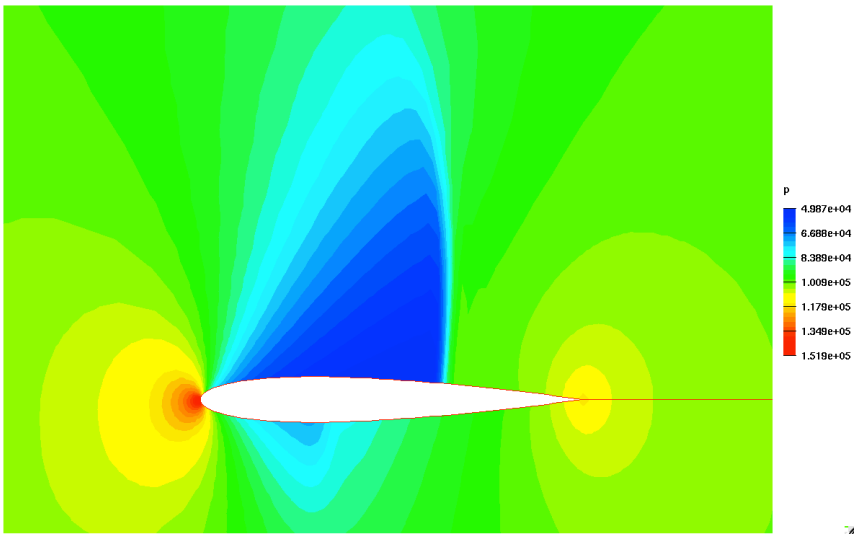

Figure 6. *The initial pressure field for the NACA 0012 airfoil,* $Ma = 0.8$ *and* AoA=1.25°.

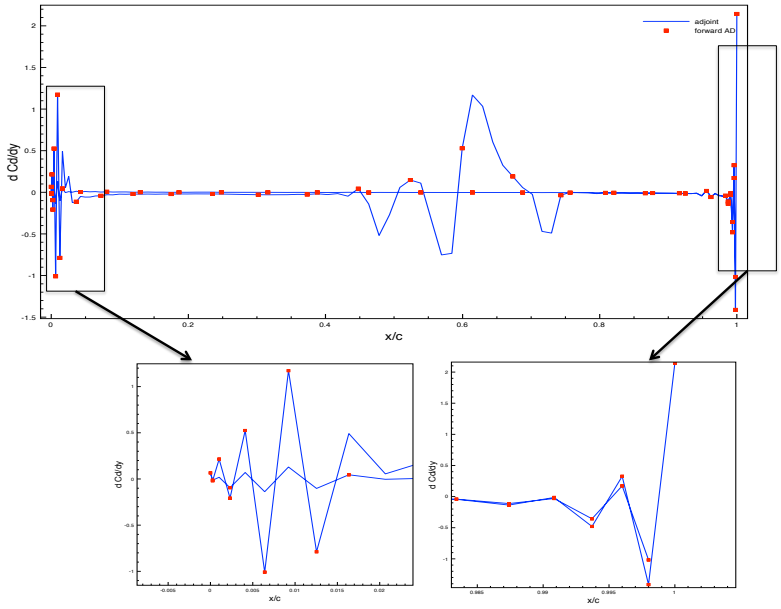

Figure 7. *The comparison of shape sensitivities obtained from adjoint and tangent linear solvers.*

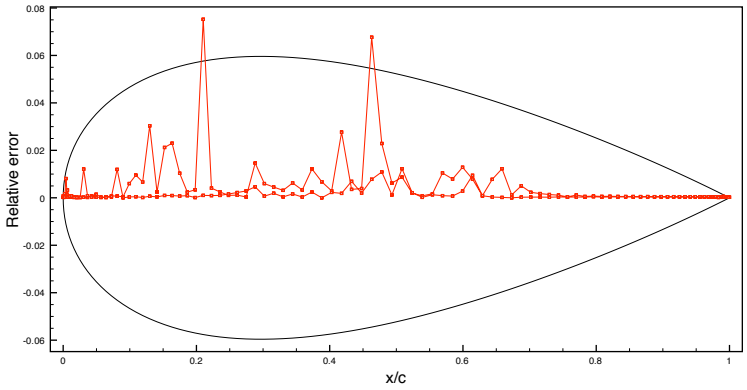

Figure 8. *The relative error of shape sensitivities based on the adjoint method.*

As the shape parameterization technique, free-node parameterization is used to provide maximum degrees of freedom to the optimization process. In this approach, each grid point on the wall boundary is allowed to move. In the present work, we allow variations only in the y direction, in this way the chordline of the initial airfoil does not change. This geometrical constraint ensured that, the airfoil geometry does not elongate or shrink in chordwise direction. Note that, in contrast to Hicks-Henne parameterization, the thickness distribution is not kept constant. Furthermore, the free-node parameterization does not guarantee that the positions of leading and

trailing edges stay fixed. This is ensured by an additional constraint, which sets the deformations at these locations to zero. Correspondingly, angle of attack chosen for the initial configuration does not vary during the optimization process. Since free-node parameterization may lead to wavy shapes, Sobolev smoothing technique is employed to obtain smooth shapes. Moreover, an additional constraint on the lift coefficient C_l is used to avoid that airfoil geometry shrinks to line. To achieve this, the lift constraint is treated implicitly with the quadratic penalty method [**NW99**].

As the optimization algorithm, multi-step one-shot approach with $s = 10$ is taken. In Fig. 9, convergence histories for the drag, lift, Lagrangian as well as primal and dual residuals are plotted. The Fig. 10 shows the pressure distribution around the optimized airfoil. In Fig. 11, the pressure coefficient distributions of the initial and the optimized airfoils are given. It can be observed that the inviscid shock is significantly reduced by the optimization process. The drag and the lift coefficients of the optimized airfoil geometry is computed as 0.00904 and 0.361, which correspond to a 60% percent decrease in drag coefficient while maintaining a constant lift. The retardation factor of the overall optimization process is measured as approximately 4 in iteration counts and 36 in run-time.

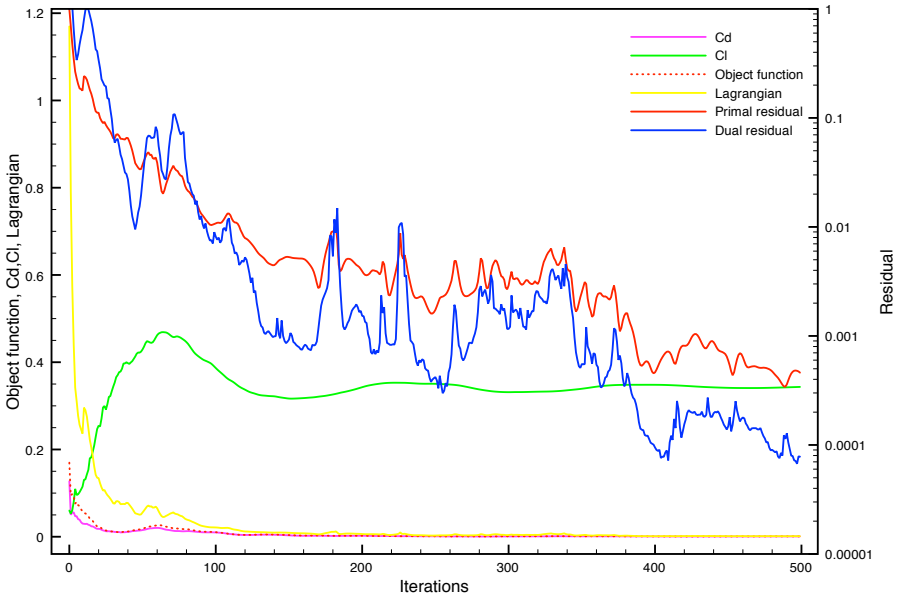

Figure 9. *Optimization history of the drag minimization of NACA 0012 airfoil,* $Ma = 0.8$ *and* AoA=1.25°.

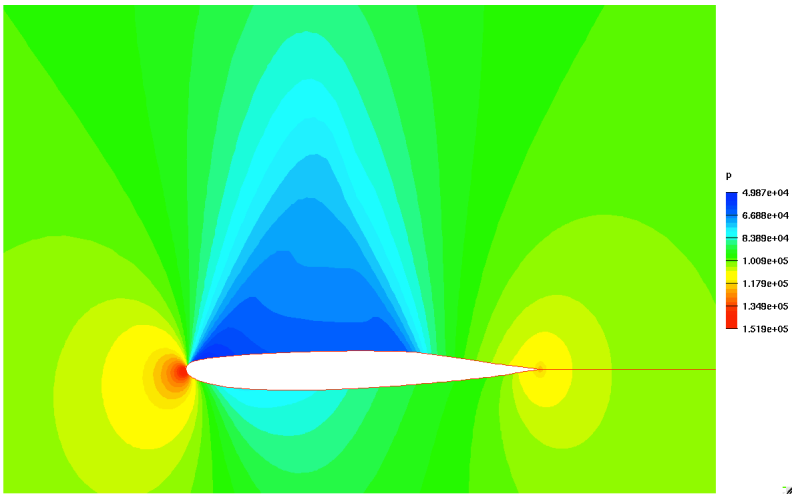

Figure 10. *Pressure field for the optimized airfoil,* $Ma = 0.8$*,* AoA=1.25°.

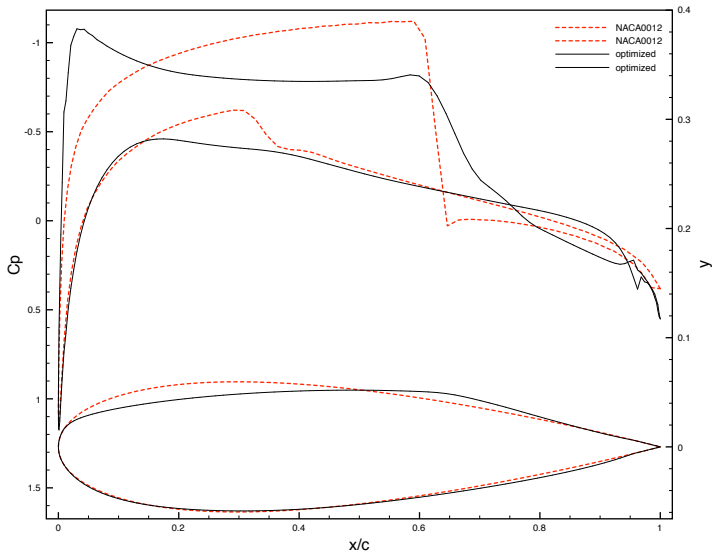

Figure 11. *Cp distribution of the initial and the optimized airfoils,* $Ma = 0.8$ *and* AoA=1.25°.

Finally, to assess the performance of multi-step one-shot method, a comparison between the nested optimization approach and the multi-step one-shot method is made. The nested optimization is the classical BFGS method with line searches. The performance results of both methods compared to a single primal simulation are presented in Table 6. The nested approach takes totally 11 adjoint and 65 primal solver evaluations with 192788 primal/adjoint iterations. The overall optimization takes approximately 28 hours on a 2.4GHz Intel machine. Therefore, retardation factor of the nested approach is 73.8 in iteration counts and 953 in

Case	iteration counts	ret. factor	run-time (s)	ret. factor
primal simulation	2613	1	107	1
nested opt.	192788	73.8	102016	953
one-shot opt.	10140	3.9	3867	36.1

Table 6. *Iteration count and run-time measurements for the primal simulation, nested optimization and one-shot optimization,* $Ma = 0.8$ *and* AoA=1.25°.

run-time. As it can be seen from the results in Table 6, the one-shot method is significantly faster than the nested approach and has a retardation factor 3.9 in iteration counts and 36.1 in run-time. In Fig. 12, the run-time that is taken by different optimization approaches can be seen. The blue and red curves represent the optimization histories of the nested approach with three and five decade residual fall stopping criteria respectively. Note that, both these optimization are performed using Sobolev smoothing. Optimization with three decade residue fall converges to a different design since the state equation can be inadequately resolved in each cycle. The other curves belong to the optimizations, which are performed using various multi-step one-shot schemes with different number of iterations taken for the primal and adjoint steps. For practical reasons, the lower bound given in Eq. (2.6.5) is not used, and the number of primal and adjoint iterations is specified heuristically. We observe that, $20-20$ scheme (20 primal and 20 adjoint iterations in one optimization cycle) finds the same result as of $10-10$ scheme, however with a time delay. On the other hand, $10-5$ scheme (10 primal and 5 adjoint iterations) also gives a very similar result, and it is even faster than the $10-10$ scheme. The $10-2$ and $10-1$ schemes converge to different designs. From these results, we can conclude that the idea of having less number of adjoint iterations than the primal iterations works if the adjoint solution leads to a reduced gradient that shows the right tendency. If too few iterations are taken for the adjoint part, the reduced gradient vector may be erroneous and the optimization may lead to designs that do not satisfy optimality. The airfoil shapes obtained from different optimization studies are presented in Fig. 13.

In an other study, the influence of Sobolev smoothing on the convergence behavior of the optimization is investigated. In Fig. 14, optimization histories of the nested approach with and without Sobolev smoothing are compared. The curves in the figure are plotted using a larger time horizon, and the blue curve is same as that in Fig. 12. The red curve belongs to the optimization performed without Sobolev smoothing. In both cases, minimum five decade residue fall is used as the stopping criteria such that primal and adjoint equation are accurately resolved after each design update. It can be observed that, convergence rate of the optimization without Sobolev smoothing is significantly slower than the one with Sobolev smoothing. The reason for this is the fact that the BFGS updates, especially in initial iterations of optimization, cannot properly approximate the Hessian, thus leading to irregular shapes. Once the airfoil shape becomes irregular, the flow solver can no more resolve the physics correctly. In this case, the state and adjoint solutions are associated with a large modeling error. As a result, it can be argued that Sobolev

smoothing not only ensures smoother shapes but also plays an important role in achieving faster convergence.

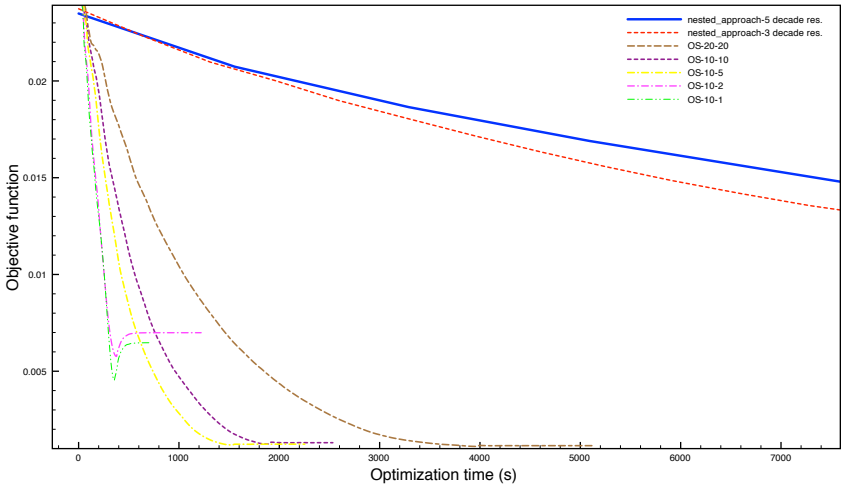

Figure 12. *Run-time comparison between the one-shot and the nested approach,* $Ma = 0.8$ *and* AoA=1.25°.

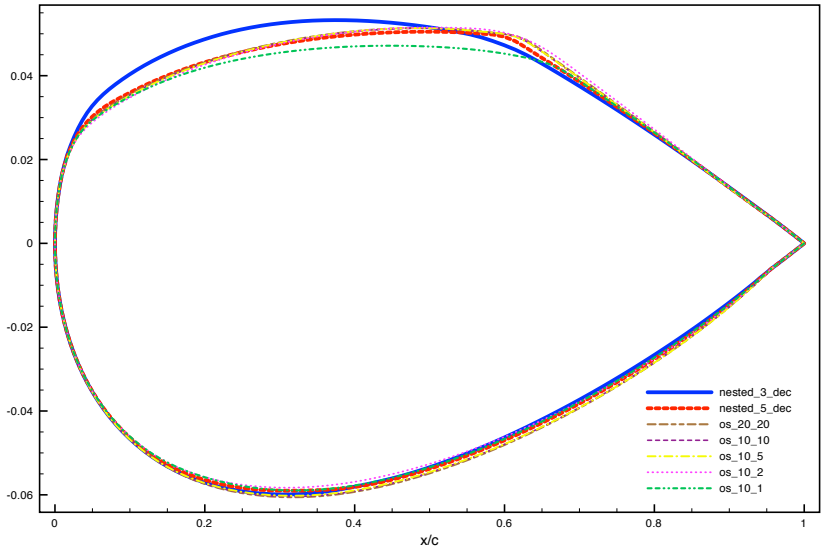

Figure 13. *Optimized airfoil geometries.*

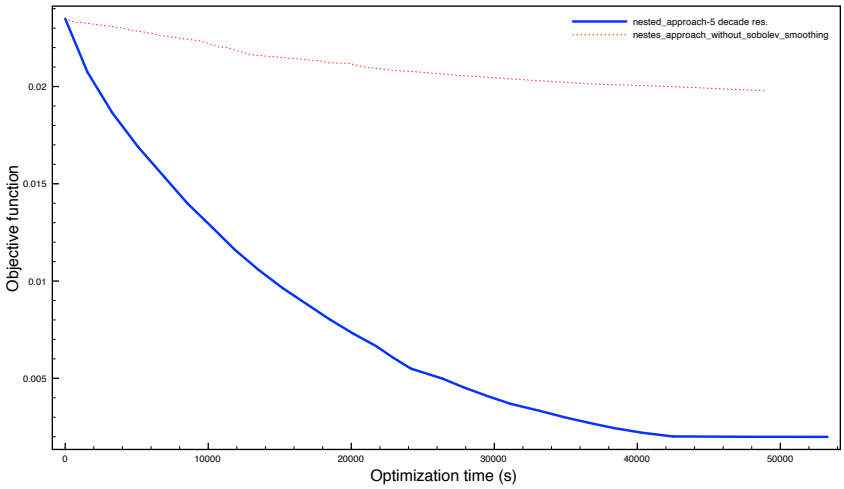

Figure 14. *Run-time comparison between the nested method with and without Sobolev smoothing,* $Ma = 0.8$ *and* AoA=1.25°.

7.3. Airfoil optimization in viscous laminar supersonic flow

As the second optimization scenario, drag minimization of the NACA 0012 airfoil in supersonic flow conditions is chosen. The Mach number is taken as 2.0, and the angle of attack is 1.25°. The flow solver used for simulation is the unstructured version of the Blazek's *struct2d* code. As the convective scheme, the Roe's scheme enhanced with Venkatakrishnan flux limiter is used. The discretization of the viscous terms is done using the central scheme. The Reynolds number for this test case is 50000. For the given flow conditions, drag and lift coefficients of the NACA 0012 airfoil are computed as 0.09349 and 0.04588 respectively. The result of the grid independency study, which is performed using three different grid levels are presented in Table 7. The grid independent result for this configuration is achieved with a grid having 24211 triangular control volumes. Although the initial grid is deformed during the optimization, number of control volumes, grid connectivities and grid topology are not allowed to vary. In Fig. 15, the unstructured grid used in the optimization study is shown.

number of CVs	dimension of state space	C_d	C_l
9737	38948	0.09314	0.04573
24211	98844	0.09349	0.04588
67369	269476	0.09341	0.04584

Table 7. *Grid independency study for the NACA 0012 airfoil,* $Ma = 2.0, Re = 50000$ *and* AoA=1.25°.

As far as the shape parameterization is concerned, we use the free-node parameterization. The validation of the shape sensitivities that are generated by the unstructured adjoint solver is done by comparing them with the exact sensitivities that are obtained from the tangent linear version. In Fig. 16, the comparison results are presented. It can be observed that the shape sensitivities are in very good agreement with the exact results. Moreover, it is worth to notice that they show a somewhat smoother behavior than the sensitivities of transonic case.

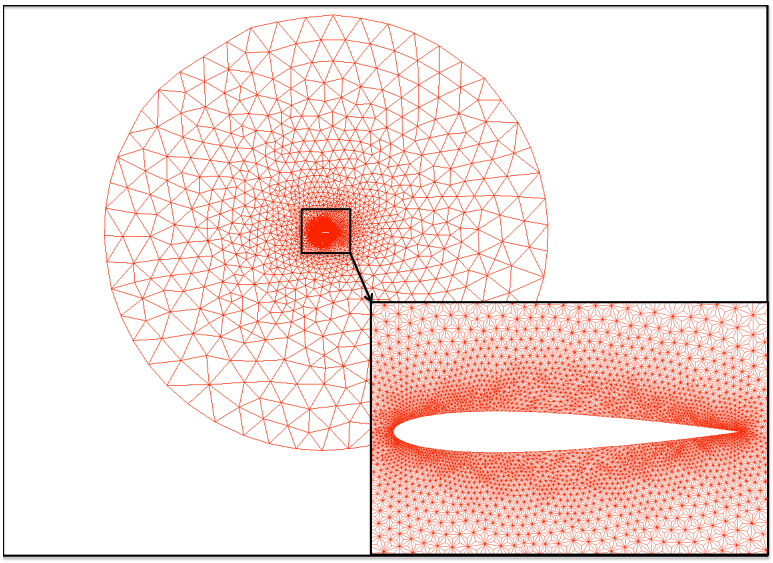

Figure 15. *The initial unstructured grid for NACA 0012 airfoil.*

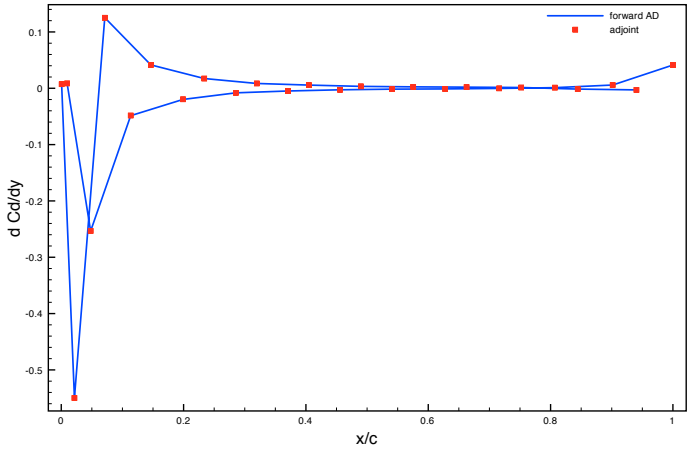

Figure 16. *Comparison between the adjoint and tangent linear shape sensitivities for the flow around NACA 0012 airfoil,* $Ma = 2.0, Re = 50000$ *and* AoA=1.25°.

For the flows around airfoils in supersonic flow regimes, it is well known that a sharp leading edge prevents the formation of a detached bow shock in front of the airfoil. This shape is in contrast to subsonic airfoils such as the NACA 0012 airfoil. These have often rounded leading edges to reduce the flow separation on the suction side in a wide range of angle of attack. In supersonic flow, however, a rounded edge behaves like a blunt body, leading to a bow shock close to the stagnation point. The bow shock formation for the NACA 0012 geometry is shown in Fig. 17. The elimination of the bow shock through the sharp profiling of the airfoil is the main goal of the present optimization.

As the optimization method, multi-step one-shot method with $s = 10$ is used. Similar to the transonic case, lift constraint is treated by the quadratic penalty method. The optimization history can be seen in Fig. 18. Since convergence behavior of the primal solver in the supersonic case is very fast due to strong hyperbolic character of the flow, optimization process also converges quite fast. After approximately 500 optimization cycles, drag coefficient is reduced to 0.00410, which corresponds to 90% decrease in drag coefficient. The lift coefficient of the optimized airfoil is 0.05012, which means a 8% increase in lift. As expected, optimized geometry is a thin airfoil with sharp leading and trailing edges as shown in Fig. 19. The pressure distribution around the optimized airfoil is presented in Fig 20. It can be observed that the strong bow shock that occurs in front of the initial airfoil disappears in the optimized shape, and transformed to an attached shock. Further, maximum thickness of the initial airfoil is significantly reduced. The retardation factor for the optimization in this test case is observed to be around 3 in iteration counts and 25 in run-time.

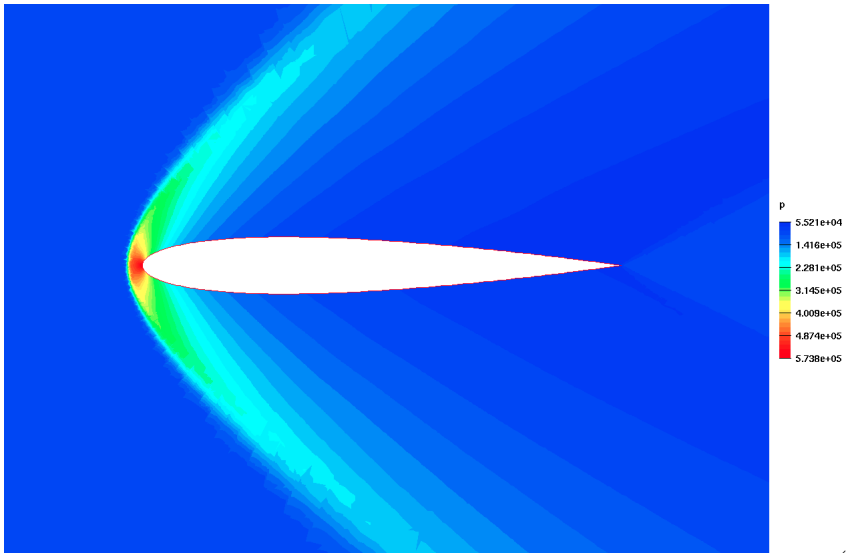

Figure 17. *The initial pressure field for the NACA 0012 airfoil,* $Ma = 2.0, Re = 50000$ *and* AoA=1.25°.

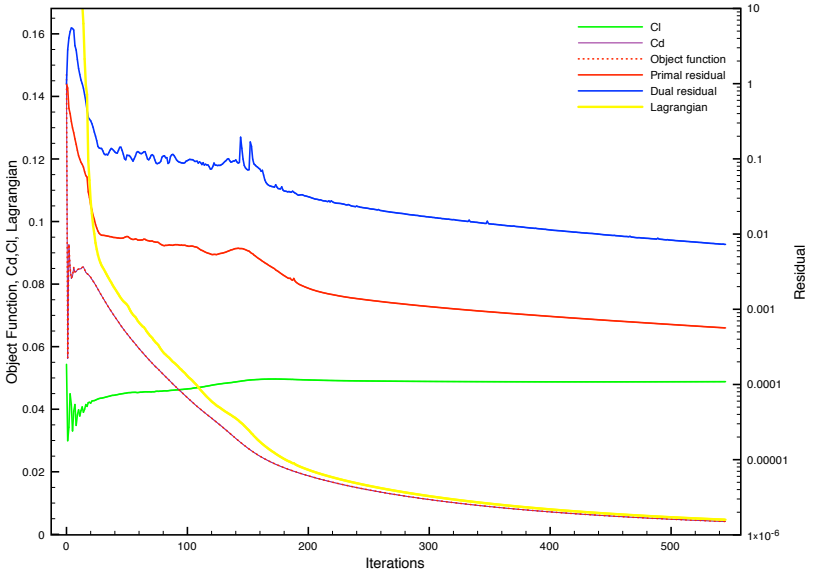

Figure 18. *Optimization history of the drag minimization of NACA 0012 airfoil at supersonic flow conditions.*

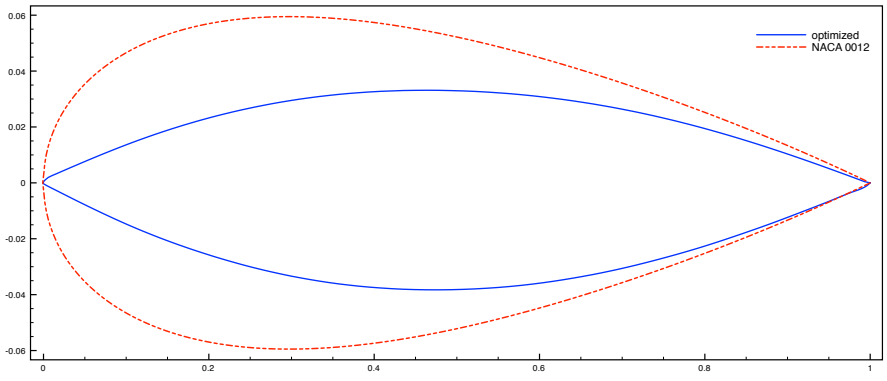

Figure 19. *Initial NACA 0012 and optimized airfoil geometries*

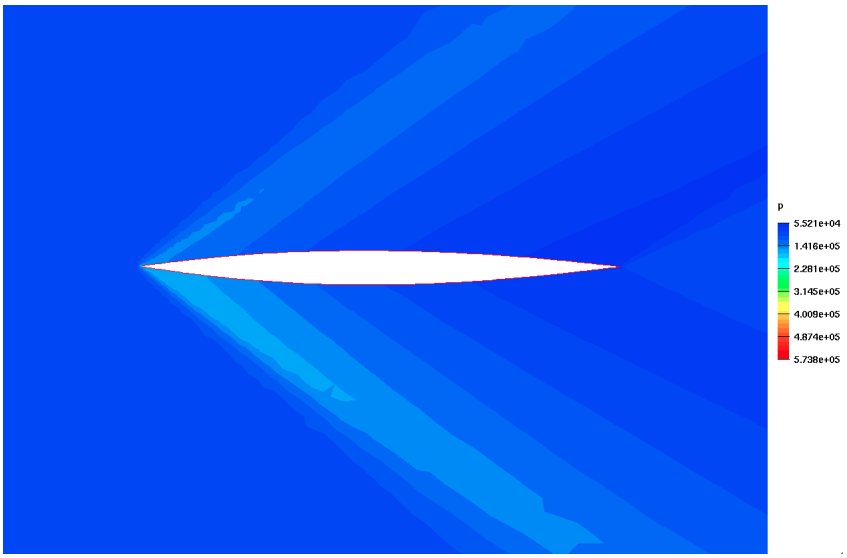

Figure 20. *The pressure field for the optimized airfoil,* $Ma = 2.0, Re =$ 50000 *and* AoA=1.25°.

7.4. Airfoil optimization in incompressible turbulent flow

As the third scenario, drag minimization of a NACA 4412 airfoil in subsonic turbulent flow ($M_\infty << 0.3$) using free-node parameterization is chosen. The simulations are performed using the incompressible flow assumption, which is well justified in subsonic flow regimes. Therefore, Navier-Stokes system is solved without the energy equation since the density is taken as constant. Note that, for incompressible flows, only similarity parameter is the Reynolds number, and therefore Mach number does not play any role. For the test case, the Reynolds number is taken as 10^6. The angle of attack is chosen as 5°. As far as the flow solver is concerned, we use the pressure-based RANS solver ELAN [**Xue98**]. The ELAN code is fully implicit and is of second order accuracy both in space and time. To account for turbulence effects, various sophisticated RANS and LES turbulence models are incorporated into the solver. In the present work, the Wilcox $k - \omega$ model with wall functions is used to account for the turbulence effects. As the velocity-pressure coupling scheme for the mean flow equations, SIMPLE algorithm with Rhie-Chow interpolation is used.

The computational grid used in the study is a structured 421×102 grid, which is shown in Fig. 21. Note that, for high Reynolds numbers, the boundary layer becomes very thin, and therefore grid refinement at the wall region is used to resolve the sharp gradients of flow quantities. In Fig. 22, pressure distribution around the initial NACA 4412 airfoil obtained from the initial simulation is shown. The lift and drag coefficients of the initial configuration, which are computed from the grid independent flow solution, are $C_d = 0.0185$ and $C_l = 0.884$.

The discrete adjoint solver used in the optimization is generated by differentiating one outer iteration of the pressure-velocity coupling scheme using the AD tool *Tapenade*. Based on the adjoint solution, shape sensitivities are then generated by the adjoint chain involving the adjoint mesh of the deformation tool. It

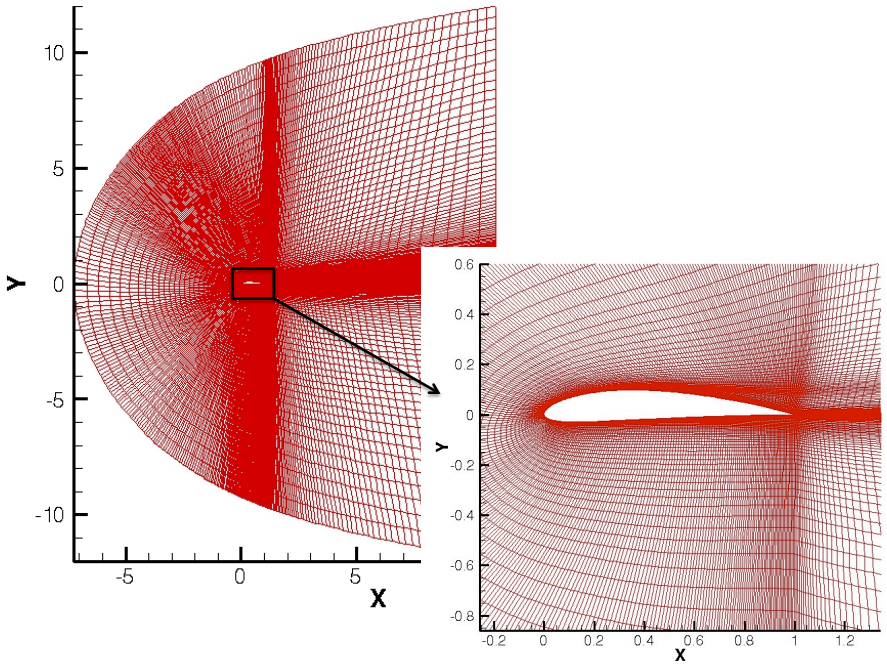

Figure 21. *Structured grid for the NACA 4412 airfoil with refinement at the wall region.*

should be noted that, in contrast to the majority of the discrete adjoint solvers, constant μ_T approach is not used in the differentiation and the $k-\omega$ model is also fully differentiated. The validation of the adjoint solver is done by comparing the adjoint shape sensitivities with the exact results obtained from the tangent linear solver. For convenience, results from every 20th surface grid point are compared. The comparison results are presented in Fig. 23. The relative error of the adjoint sensitivities are presented in Fig. 24. It can be observed that, the adjoint sensitivities are in excellent agreement with the forward AD values at some grid points. At some other points, however, the relative error goes up to 20%. The reason for this is the poor convergence of the primal solver due to stiffness caused by the turbulent transport equations. Note that, piggy-back approach that is used to compute adjoint results requires, at least theoretically, zero residual at each cell. However, this is impossible to achieve in reality since high residual levels, especially for turbulent quantities, are likely to occur at some local spots in the flow domain. As a result, adjoint sensitivity results may show some deviation from the exact values at some grid points.

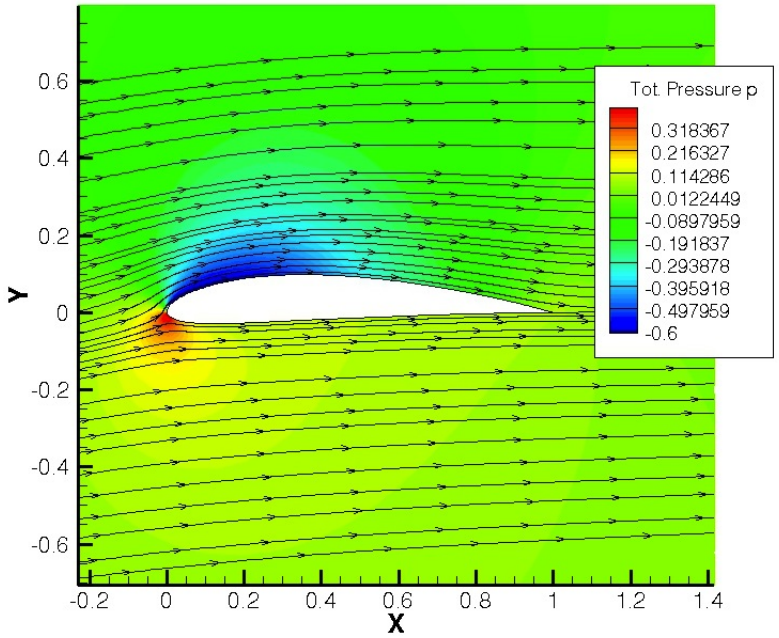

Figure 22. *Pressure distribution around the NACA 4412 airfoil,* $Re = 10^6$ *and* AoA=5°.

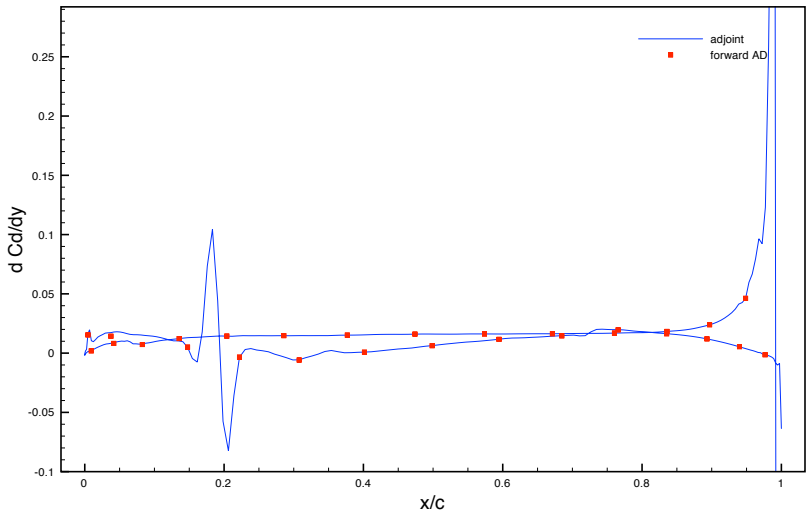

Figure 23. *Comparison between the adjoint and forward AD shape sensitivities for the flow around NACA 4412 airfoil.*

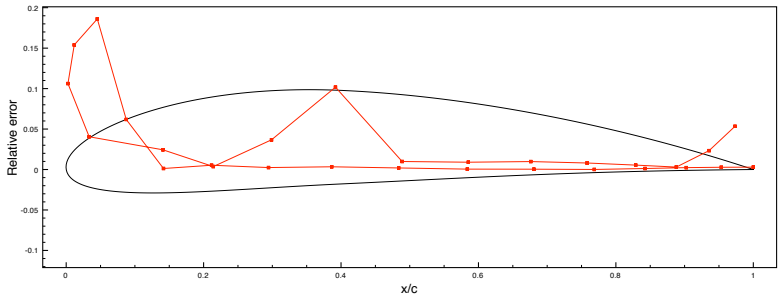

Figure 24. *Relative error of the adjoint shape sensitivities for the flow around NACA 4412 airfoil.*

In the first part, we present the results of single-step one-shot optimization. The lift constraint in this case is treated by the penalty multiplier method [**GK02**], in which the objective function is modified as $C_d + \lambda h$. The penalty function h is given as the difference between the target lift coefficient and actual lift coefficient ($h = C_{l,target} - C_l$). The multiplier for the penalty term is denoted by λ and is updated in each optimization iteration by the update $\lambda_{k+1} = \lambda_k(1 + \mu h)$. The constant $\mu > 0$ is to be adjusted according to the optimization problem. For the initial value of λ, a suitable value is given by $\lambda_0 = \|\nabla C_d\|/\|\nabla h\|$. Whenever the lift constraint is not fulfilled, i.e., $h > 0$, the penalty term is weighted more. Otherwise, it decreases in each iteration. In order to accelerate the procedure, alternatively λ can be set to zero whenever the target lift is achieved, i.e., $h \leq 0$, such that the optimizer tries to go in the direction of minimum drag.

In Fig. 25, the optimization histories of the objective function drag coefficient C_d, the constraint lift coefficient C_l, the augmented Lagrangian L^a, the multiplier λ of the penalty function as well as the penalty function h (difference between the actual and target lift) are presented. The resulting optimized airfoil geometry is shown in Fig. 26. It can be seen from the figure that the optimized airfoil gets thinner with an increase in camber favoring a decrease in the drag and increase in the lift. Since the positions of the leading and trailing edges are not kept constant, the angle of attack is also slightly modified. As a result, overall 5% drag decrease is achieved accompanied with 4% lift increase. Note that, although there exists no geometric constraint in the optimization, the lift constraint prevents the airfoil from getting too thin. As far as the efficiency of the optimization is concerned, the retardation factor is measured as 3 in iteration counts.

In the second part, the multi-step one-shot method with $s = 20$ is applied to the same problem. The only difference in the shape parameterization is that the positions of the leading and trailing edges are kept constant. Therefore, the angle of attack is not allowed to vary. In Fig. 27, the optimization history is shown. The comparison of the initial and optimized shapes are illustrated in Fig. 28. The optimized airfoil has a C_d value of 0.0170 and C_l value of 0.873. Consequently, the reduction in drag coefficient is around 9% associated with a slight decrease in lift by 1%. The retardation ratio of the optimization is measured as approximately 3 in iteration counts and 30 in run-time.

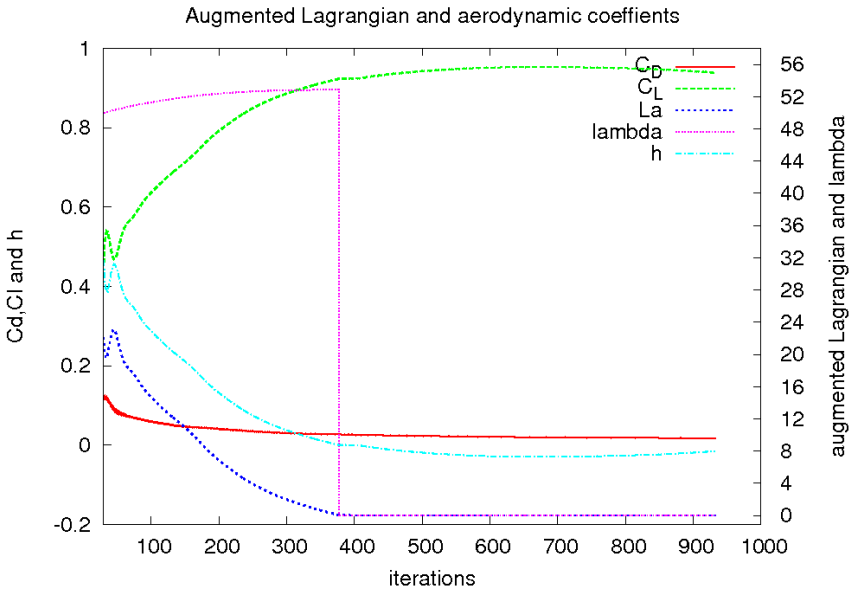

Figure 25. *Optimization histories of single-step one-shot optimization of NACA 4412 airfoil in turbulent flow, Re* $= 10^6$ *and AoA*$=5^\circ$ *(reprinted from* [**OG10**]*).*

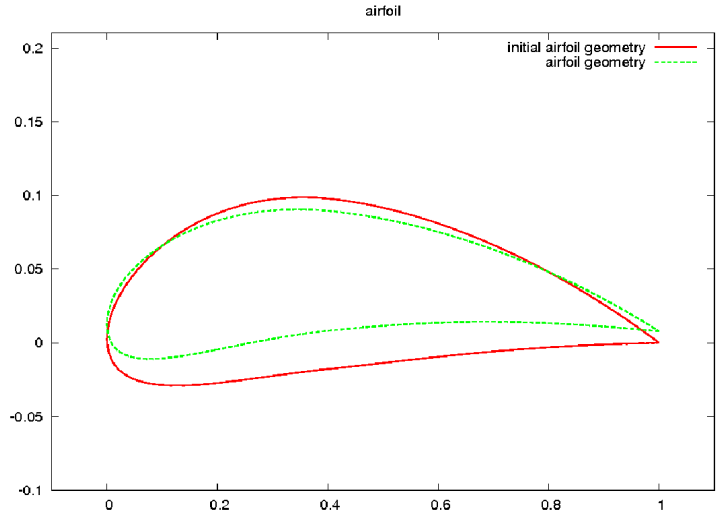

Figure 26. *Initial and optimized airfoil geometries for single-step one-shot optimization, Re* $= 10^6$ *and AoA*$=5^\circ$ *(reprinted from* [**OG10**]*).*

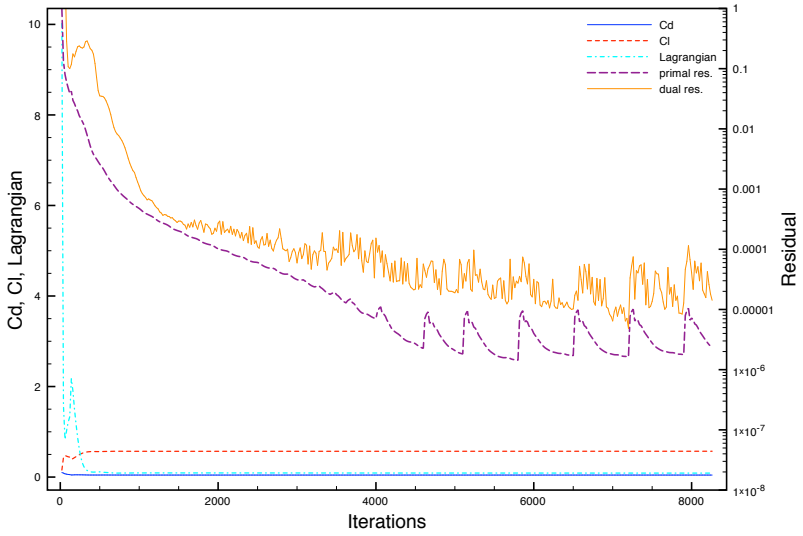

Figure 27. *Optimization history of the drag minimization of NACA 4412 airfoil,* $Re = 10^6$ *and* AoA=5°.

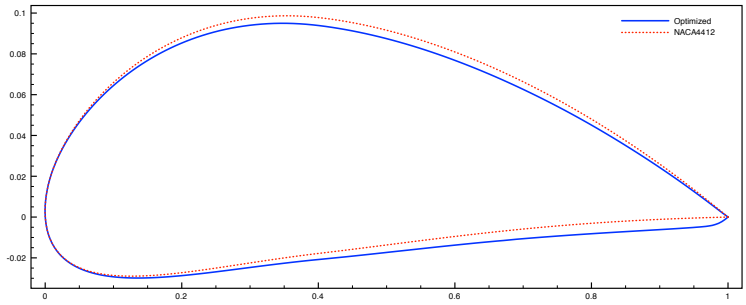

Figure 28. *NACA 4412 and the optimized airfoil geometries.*

CHAPTER 8

Conclusion and outlook

In this work, a one-shot framework based on the consistent discrete adjoint method for aerodynamic shape optimization problems was developed. The one-shot method, which is derived from the first order optimality conditions of a PDE optimization problem, was covered in detail. The preconditioning techniques and constructive conditions to ensure the contractivity of the coupled one-shot were presented. Theoretical results to quantify the retardation rate, which is a slow-down measure of the one-shot methods, was presented and the concept of bounded retardation was demonstrated on test problems, in which the state equation was solved with Newton and Jacobi methods. Related to the implementation of one-shot method for aerodynamic shape optimization, elements of aerodynamic design and optimization chains were introduced and implementation aspects were discussed. Several shape parameterization techniques as well as mesh deformation technique that are used in aerodynamic shape optimization were covered in detail. A review of sensitivity evaluation methods that are frequently employed in aerodynamic shape optimization was provided. In particular, advantages of the adjoint method to achieve a bounded retardation rate was emphasized. The application of Automatic Differentiation (AD) techniques to ease the development of discrete adjoint codes was discussed in detail. Various advanced techniques of AD, which help to improve the performance of adjoint solvers were introduced. Finally, the one-shot methodology coupled with the AD based discrete adjoint method was applied to three different scenarios of airfoil optimization. The results of the optimization studies as well as performance of the one-shot method for each test case were presented and discussed in detail. Further, a comparison of the one-shot method with a nested quasi-Newton method was performed to demonstrate the efficiency of the one-shot method.

As far as further developments of the one-shot methods are concerned, one important issue is the extension for unsteady problems. Even though it is possible to perform optimizations for steady-state flows (e.g., attached laminar flow past a streamlined body), unsteady flows cannot be treated with the current methodology. As typical examples of unsteady flows, massively separated flows (e.g. flow past a blunt body) and pulsating flows (e.g., blood flow) can be given. Furthermore, in many cases turbulent flows can be only resolved with unsteady methods (e.g., large eddy or detached eddy simulations). As a result, unsteady flows are much of interest, and extending capabilities of the one-shot method for unsteady problems is important, especially for practical engineering applications.

Another issue is the extension of the one-shot method in multidisciplinary design optimization (MDO) context. This involves problems, in which the fluid flow is coupled with other physical phenomena. Typical examples are aeroelasticity, reaction flows, aeroacoustics and conjugate heat transfer. Since there is an increasing

trend for simulations in a multidisciplinary environment, extending the capabilities of the one-shot method to be able to perform in MDO context is crucial.

Bibliography

[AJD03] M. Antreoli, A. Janka, and J. Désidéri. Free-form-deformation parameterization for multilevel 3D shape optimization in aerodynamics. Technical Report 5019, INRIA Sophia-Antipolis, 2003.

[AV99] W. Anderson and V. Venkatakrishnan. Aerodynamic design optimization on unstructured grids with a continuous adjoint formulation. *Computers and Fluids*, 28:443–480, 1999.

[BE91] O. Baysal and M. Eleshaky. Aerodynamic design sensitivity analysis methods for the compressible Euler equations. *J. Fluids Eng.*, 113(4):681–688, 1991.

[BE92] O. Baysal and M. Eleshaky. Aerodynamic design optimization using sensitivity analysis and Computational Fluid Dynamics. *AIAA Journal*, 30(3):718–725, 1992.

[BL78] B. Baldwin and H. Lomax. Thin layer approximation and algebraic model for separated turbulent flows, 1978. AIAA Paper 78-257.

[Bla01] J. Blazek. *Computational Fluid Dynamics: Principles and Applications*. Referex Engineering. Elsevier, 2001.

[BLG14] T. Bosse, L. Lehmann, and A. Griewank. Adaptive sequencing of primal, dual, and design steps in simulation based optimization. *Computational Optimization and Applications*, 57(3):731–760, 2014.

[BM77] W. Briley and H. McDonald. Solution of the multidimensional compressible Navier-Stokes equations by a generalized implicit method. *Journal of Computational Physics*, 24(4):372 – 397, 1977.

[BN03] C. Burg and J. Newman. Computationally efficient, numerically exact design space derivatives via the complex Taylor's series expansion method. *Computers and Fluids*, 32(3):373–383, March 2003.

[CGPS73] L. Caretto, A. Gosman, S. Patankar, and D. Spalding. Two calculation procedures for steady, three-dimensional flows with recirculation. In *Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics*, volume 19 of *Lecture Notes in Physics*, pages 60–68. Springer Berlin / Heidelberg, 1973.

[Chr94] B. Christianson. Reverse accumulation and attractive fixed points. *Optimization Methods and Software*, 3:311–326, 1994.

[CKH03] F. Courty, A. Dervieux B. Koobus, and L. Hascoët. Reverse automatic differentiation for optimum design: from adjoint state assembly to gradient computation. *Optimization Methods and Software*, 18(5):615–627, 2003.

[CKvT07] G. Carpentieri, B. Koren, and M. van Tooren. Adjoint-based aerodynamic shape optimization on unstructured meshes. *Journal of Computational Physics*, 224(1):267 – 287, 2007.

[CLPZ06] C. Castro, C. Lozano, F. Palacios, and E. Zuazua. A systematic continuous adjoint approach to viscous aerodynamic design on unstructured grids. *AIAA Paper 2006-0051*, 2006.

[CTÖG10] A. Carnarius, F. Thiele, E. Özkaya, and N. Gauger. Adjoint approaches for optimal flow control. *AIAA Paper 2010-5088*, 2010.

[Dur08] F. Durst. *Fluid Mechanics: An Introduction to the Theory of Fluid Flows*. Springer, 2008.

[EP97] J. Elliot and J. Peraire. Practical 3D aerodynamic design and optimization using unstructured meshes. *AIAA Journal*, 35(9):1479–1485, 1997.

[Far96] G. Farin. *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Code*. Academic Press, Inc., Orlando, FL, USA, 4th edition, 1996.

[FKN09] C. Frey, H. Kersken, and D. Nürnberger. The discrete adjoint of a turbomachinery RANS solver. In *Proceedings of ASME TurboExpo 2009*, pages 345–354, June 2009.

[FP01] J. Ferziger and M. Peric. *Computational Methods for Fluid Dynamics*. Springer, 2001.

[Gau03] N. Gauger. *Das Adjungiertenverfahren in der aerodynamischen Formoptimierung*. PhD thesis, DLR Braunschweig, 2003.

[Gau08] Nicolas R. Gauger. Efficient deterministic approaches for aerodynamic shape optimization. In Optimization and Computational Fluid Dynamics, editors, *D. Thevenin and G. Janiga*, pages 111–145. Springer Berlin Heidelberg, 2008.

[GB02] N. Gauger and J. Brezillon. Aerodynamic shape optimization using the adjoint method. *J. Aero. Soc. India*, 54(3):247–254, 2002.

[GDM01] M. Giles, M. Duta, and J. Müller. Adjoint code developments using the exact discrete approach. *AIAA Paper 2001-2596*, 2001.

[GDMP03] M. Giles, M. Duta, J. Müller, and N. Pierce. Algorithm developments for discrete adjoint methods. *AIAA Journal*, 41(2):198–205, 2003.

[GF02] A. Griewank and C. Faure. Reduced functions, gradients and Hessians from fixed point iteration for state equations. *Numerical Algorithms*, 30:113–139, 2002.

[GGD08] M. Giles, D. Ghate, and M. Duta. Using automatic differentiation for adjoint CFD code development. *Comp. Fl. Dyn. Journal*, 16(4):434–443, 2008.

[GGR08] N. Gauger, A. Griewank, and J. Riehme. Extension of fixed point PDE solvers for optimal design by one-shot method - with first applications to aerodynamic shape optimization. *European Journal of Computational Mechanics*, 17:87–102, 2008.

[GHO13] A. Griewank, A. Hamdi, and E. Özkaya. Quantifying retardation in simulation based optimization. In *Optimization, simulation and control*, volume 76, pages 79–96. Springer New York, 2013.

[Gil02] M. Giles. Automatic differentiation of algorithms. chapter On the Iterative Solution of Adjoint Equations, pages 145–151. Springer-Verlag New York, Inc., New York, NY, USA, 2002.

[GJU96] A. Griewank, D. Juedes, and J. Utke. ADOL-C: A package for the automatic differentiation of algorithms written in C/C++. *ACM Trans. Math. Softw.*, 22:131–167, 1996.

[GK98] R. Giering and T. Kaminski. Recipes for adjoint code construction. *ACM Trans. Math. Software*, 24:437–474, 1998.

[GK02] K. Geiger and C. Kanzow. *Theorie und Numerik restringierter Optimierungsaufgaben*. Springer, 2002.

[GN08] T. Gerhold and J. Neumann. The parallel mesh deformation of the DLR TAU-code. *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, 96:162–169, 2008.

[GO91] G. Golub and J. Ortega. *Scientific Computing And Differential Equations: An Introduction To Numerical Methods*. Academic Press, 1991.

[Gri00] A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, 2000.

[Gri06] A. Griewank. Projected Hessians for preconditioning in one-step one-shot design optimization. In Large-Scale Nonlinear Optimization, editor, *G. Pillo and M. Roma*, volume 83 of *Nonconvex Optimization and Its Applications*, pages 151–171. Springer US, 2006.

[GW00] A. Griewank and A. Walther. Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Trans. Math. Softw.*, 26:19–45, March 2000.

[GWMW07] N. Gauger, A. Walther, C. Moldenhauer, and M. Widhalm. Automatic differentiation of an entire design chain for aerodynamic shape optimization. *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, 96:454–461, 2007.

[GWOM12] N. Gauger, A. Walther, E. Özkaya, and C. Moldenhauer. Efficient aerodynamic shape optimization by structure exploitation. *Optimization and Engineering*, 13:563–578, 2012.

[HAP05] L. Hascoët and M. Araya-Polo. The adjoint data-flow analyses: Formalization, properties, and applications. In *Automatic Differentiation: Applications, Theory,*

and Tools, Lecture Notes in Computational Science and Engineering. Springer, 2005. Selected papers from AD2004 Chicago, July 2005.

[Has09] L. Hascoët. Reversal strategies for adjoint algorithms. In *From Semantics to Computer Science. Essays in memory of Gilles Kahn*, pages 487–503. Cambridge University Press, 2009.

[HD08] L. Hascoët and B. Dauvergne. Adjoints of large simulation codes through automatic differentiation. *REMN Revue Européenne de Mécanique Numérique / European Journal of Computational Mechanics*, 17:63–86, 2008.

[Hei06] R. Heinrich. Implementation and usage of structured algorithms within an unstructured CFD-code. *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, 92, 2006.

[HFH01] L. Hascoët, S. Fidanova, and C. Held. Adjoining independent computations. In *Automatic Differentiation of Algorithms, from Simulation to Optimization*, Computer and Information Science, pages 299–304. Springer, 2001. selected papers from the AD2000 conference, Nice, France.

[HG09] A. Hamdi and A. Griewank. Properties of an augmented Lagrangian for design optimization. *Optimization Methods and Software*, 25:645–64, 2009.

[HG11] A. Hamdi and A. Griewank. Reduced quasi-Newton method for simultaneous design and optimization. *Computational Optimization and Applications*, 49:521–548, 2011.

[HH78] R. Hicks and P. Henne. Wing design by numerical optimization. *Journal of Aircraft*, 15:407–412, 1978.

[Hir07] C. Hirsch. *Numerical Computation of Internal and External Flows: Fundamentals of Computational Fluid Dynamics*. Number Bd. 1 in Butterworth Heinemann. Elsevier/Butterworth-Heinemann, 2007.

[HJ12] S. Hazra and A. Jameson. One-shot pseudo-time method for aerodynamic shape optimization using the Navier-Stokes equations. *Int. J. Numer. Meth. Fluids*, 68:564–581, 2012.

[HLvL83] A. Harten, P. Lax, and B. van Leer. On upstream differencing and Godunov-type schemes for hyperbolic conservation laws. *SIAM Review*, 25:35–61, 1983.

[HSBG05] S. Hazra, V. Schulz, J. Brezillon, and N. Gauger. Aerodynamic shape optimization using simultaneous pseudo-timestepping. *Journal of Computational Physics*, 204:46–64, 2005.

[HUPU09] M. Hinze, S. Ulbrich, R. Pinnau, and M. Ulbrich. *Optimization with PDE constraints*, volume 23 of *Mathematical modeling*. Springer, Dordrecht, NL, 2009.

[IKT96] A. Iollo, G. Kuruvila, and S. Taasan. Pseudo-time method for optimal shape design using Euler equations. *AIAA Journal*, 9:18071813, 1996.

[Jam88] A. Jameson. Aerodynamic design via control theory. *J. Sci. Comput.*, 3(3):233–261, 1988.

[Jam04] A. Jameson. Efficient aerodynamic shape optimization. *AIAA Paper 2004-4369*, 2004.

[JB83] A. Jameson and T. Baker. Solution of the Euler equations for complex configurations. *AIAA Paper 83-1929*, 1983.

[JMP98] A. Jameson, L. Martinelli, and N. Pierce. Optimum aerodynamic design using the Navier-Stokes equations. *Theor. Comput. Fluid Dyn.*, 10(1-4):213–237, 1998.

[Jov04] J. Jovanovic. *The Statistical Dynamics of Turbulence*. Springer, 1st. edition, 2004.

[JST81] A. Jameson, W. Schmidt, and E. Turkel. Numerical solutions of the Euler equations by finite volume methods using Runge-Kutta time-stepping schemes. *AIAA Paper 81-1259*, 1981.

[JY87] A. Jameson and S. Yoon. Lower-upper implicit schemes with multiple grids for the Euler equations. *AIAA Journal*, 25(7):929–935, 1987.

[Kel99] C. Kelley. *Iterative Methods for Optimization*. SIAM, 1999.

[KKR01] C. Kim, C. Kim, and O. Rho. Sensitivity analysis for the Navier-Stokes equations with two equations turbulence models. *AIAA J.*, 39(5):838–845, 2001.

[KKR02] C. Kim, C. Kim, and O. Rho. Effects of constant eddy viscosity assumption on gradient-based design optimization. *AIAA Paper 2002-0262*, 2002.

[KMID05] G. Kalitzin, G. Medic, G. Iaccarino, and P. Durbin. Near-wall behavior of RANS turbulence models and implications for wall functions. *Journal of Computational Physics*, 204:265–291, 2005.

[KRG14] L. Kaland, J. De Los Reyes, and N. Gauger. One-shot methods in function space for PDE–constrained optimal control problems. *Optimization Methods Software*, 29(2):376–405, March 2014.

[KTS95] G. Kuruvila, S. Ta'asan, and M. Salas. Airfoil design and optimization by the one-shot method. *AIAA Paper 95-0478*, 1995.

[Leo79] B. Leonard. A stable and accurate convective modelling procedure based on quadratic upstream interpolation. *Computer Methods in Applied Mechanics and Engineering*, 19(1):59 – 98, 1979.

[LL93] F. Lien and M. Leschziner. Computational modeling of 3D turbulent flow in S-diffusor and transient ducts. In *Engineering Turbulence Modeling and Experiments*, pages 217–228. Elsevier, 1993.

[LLN11] J. Lotz, K. Leppkes, and U. Naumann. dco/c++ - derivative code by overloading in C++. Aachener Informatik-Berichte (AIB), 2011.

[Mav98] D. Mavriplis. On convergence acceleration techniques for unstructured meshes. *AIAA Paper 98-2966*, 1998.

[Mav06] D. Mavriplis. Multigrid solution of the discrete adjoint for optimization problems on unstructured meshes. *AIAA J.*, 44(1):42–50, 2006.

[Men94] F. Menter. Two-equation eddy-viscosity turbulence models for engineering applications. *AIAA Journal*, 32(8):1598–1605, 1994.

[MMAvdW08] C. Mader, J. Martins, J. Alonso, and E. van der Weide. ADjoint: An approach for the rapid development of discrete adjoint solvers. *AIAA J.*, 46(4):863–873, 2008.

[Moh97] B. Mohammadi. Optimal shape design, reverse mode of automatic differentiation and turbulence. *AIAA Paper 97-0099*, 1997.

[MP02] B. Mohammadi and O. Pironneau. *Applied Shape Optimization for Fluids*. Oxford Univ. Press, 2002.

[MT09] A. Madrane and E. Tadmor. Entropy stability of Roe-type upwind finite volume methods on unstructured grids. In *Proceedings of the 12th International Conference*, volume 67 of *Hyperbolic Problems: Theory, Numerics, Applications*, pages 775–784. AMS Proc. Symp. Applied Math., University of Maryland, 2009.

[NA99] E. Nielsen and W. Anderson. Aerodynamic design optimization on unstructured meshes using the Navier-Stokes equations. *AIAA Journal*, 37(11):957–964, 1999.

[NA02] E. Nielsen and W. Anderson. Recent improvements in aerodynamic design optimization on unstructured meshes. *AIAA Journal*, 40(6):1155–1163, 2002.

[Nau12] U. Naumann. *The Art of Differentiating Computer Programs. An Introduction to Algorithmic Differentiation.* Software, Environments, and Tools. SIAM, Philadelphia, PA, 2012.

[NDY09] E. Nielsen, B. Diskin, and N. Yamaleev. Discrete adjoint-based design optimization of unsteady turbulent flows on dynamic unstructured grids. *AIAA Paper 2009-3802*, 2009.

[NJ00] S. Nadarajah and A. Jameson. A comparison of the continuous and discrete adjoint approach to automatic aerodynamic optimization. *AIAA Paper 2000-0667*, 2000.

[NJ01] S. Nadarajah and A. Jameson. Studies of continuous and discrete adjoint approaches to viscous automatic aerodynamic shape optimization. *AIAA Paper 2001-2530*, 2001.

[NLPD04] E. Nielsen, J. Lu, M. Park, and D. Darmofal. An implicit, exact dual adjoint solution method for turbulent flows on unstructured grids. *Computers and Fluids*, 33:1131–1155, 2004.

[NM65] J. Nelder and R. Mead. A simplex method for function minimisation. *Comput.J.*, 4:308–313, 1965.

[NOG+11] A. Nemili, E. Özkaya, N. Gauger, A. Carnarius, and F. Thiele. Optimal control of unsteady flows using discrete adjoints. *AIAA Paper 2011-3720*, 2011.

[NW99] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operational Research, 1999.

[Oba95] S. Obayashi. Genetic algorithm for aerodynamic inverse optimization problems. *1st IEE/IEEE Int. Conf. GA's in Engineering Systems: Innovations and Applications*, pages 7 –12, 1995.

[OG09] E. Özkaya and N. Gauger. Single-step one-shot aerodynamic shape optimization. *Int. Ser. of Num. Mathematics*, 158:191–204, 2009.

[OG10] E. Özkaya and N. Gauger. Automatic transition from simulation to one-shot shape optimization with Navier-Stokes equations. *GAMM-Mitt.*, 33:133–147, 2010.

[ONG12] E. Özkaya, A. Nemili, and N. Gauger. Application of automatic differentiation to an incompressible URANS solver. In *Recent Advances in Algorithmic Differentiation*, volume 87 of *Lecture Notes in Computational Science and Engineering*, pages 35–45. Springer Berlin Heidelberg, 2012.

[OON97] A. Oyama, S. Obayashi, and K. Nakahashi. Transonic wing optimization using genetic algorithm. *AIAA Paper 97-1854*, 1997.

[Pat80] S. Patankar. *Numerical heat transfer and fluid flow*. Series in computational methods in mechanics and thermal sciences. Hemisphere Publ., 1980.

[PD10] J. Peter and R. Dwight. Numerical sensitivity analysis for aerodynamic optimization: A survey of approaches and applications. *Computers and Fluids*, 39:373–391, 2010.

[Per02] J. Periaux. *Genetic Algorithms in Aeronautics and Turbomachinery*. Wiley and Sons, 2002.

[Pet06] J. Peter. Discrete adjoint method in elsa (part I) : Method/theory. In *Proceedings of 7th ONERA- DLR Aerospace Symposium*, 2006.

[PHZ96] B. Prananta, M. Hounjet, and R. Zwaan. Thin layer Navier Stokes and its application for aeroelastic analysis of an airfoil in transonic flow. In *Proceedings of the International Forum on Aeroelasticity and Structural Dynamics*, 1996.

[Pir74] O. Pironneau. On optimum design in fluid mechanics. *J. Fluid Mech.*, 64:97–110, 1974.

[Pop00] S. Pope. *Turbulent Flows*. Cambridge University Press, 2000.

[RC83] C. Rhie and W. Chow. Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA Journal*, 21(11):1525–1532, 1983.

[RJ95] J. Reuther and A. Jameson. Aerodynamic shape optimization of wing and wing-body configurations using control theory. *AIAA Paper 95-0213*, 1995.

[RJF+96] J. Reuther, A. Jameson, J. Farmer, L. Martinelli, and D. Saunders. Aerodynamic shape optimization of complex aircraft configurations via an adjoint formulation. *AIAA Paper 96-0094*, 1996.

[Roe81] P. Roe. Approximate Riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, 43(2):357 – 372, 1981.

[Ros00] C. Rossow. A flux splitting scheme for compressible and incompressible flows. *Journal of Computational Physics*, 164:104–122, 2000.

[RT96] T. Rung and F. Thiele. Computational modeling of complex boundary-layer flows. In *Proc. 9th Intern. Symposium on Transport Phenomena in Thermal-Fluid Engineering*, pages 321–326. 1996.

[RZ10] M. Rumpfkeil and D. Zingg. The optimal control of unsteady flows with a discrete adjoint method. *Optimization and Engineering*, 11:5–22, 2010.

[SA92] P. Spalart and S. Allmaras. A one-equation turbulence model for aerodynamic flows. *AIAA-Paper 92-0439*, 1992.

[Sam01] J. Samareh. Survey of shape parametrization techniques for high-fidelity multidisciplinary shape optimization. *AIAA J.*, pages 877–884, 2001.

[Sam04] J. Samareh. Aerodynamic shape optimization based on free-form deformation. *AIAA Paper 2004-4630*, 2004.

[SC67] A. Smith and T. Cebeci. Numerical solution of the turbulent boundary layer equations, 1967. Douglas aircraft division report DAC 33735.

[SG09] V. Schulz and I. Gherman. One-shot methods for aerodynamic shape optimization. In *MEGADESIGN and MegaOpt-German Initiatives for Aerodynamic Simulation and Optimization in Aircraft Design*, volume 107 of *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, pages 207–220. Springer Berlin Heidelberg, 2009.

[SISG11] S. Schmidt, C. Ilic, V. Schulz, and N. Gauger. Airfoil design for compressible inviscid flow based on shape calculus. *Optimization and Engineering*, 12:349–369, 2011.

[Son89] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 10(1):36–52, 1989.

[SP86] T. Sederberg and S. Parry. Free-form deformation of solid geometric models. *Computer Graphics*, 20(4):151–160, 1986.

[SS86] Y. Saad and M. Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7(3):856–869, July 1986.

[SS10] S. Schmidt and V. Schulz. Shape derivatives for general objective functions and the incompressible Navier-Stokes equations. *Control and Cybernetics*, 39:677–713, 2010.

[SSIG11] S. Schmidt, V. Schulz, C. Ilic, and N. Gauger. Three dimensional large scale aerodynamic shape optimization based on shape calculus. *AIAA Paper 2011-3718*, 2011.

[ST92] R. Swanson and E. Turkel. On central difference and upwind schemes. *J. Comp. Phys.*, 101:292–306, 1992.

[ST98] W. Squire and G. Trapp. Using complex variables to estimate derivatives of real functions. *SIAM Rev.*, 40:110–112, March 1998.

[Sto68] H. Stone. Iterative solution of implicit approximations of multidimensional partial differential equations. *SIAM Journal of Numerical Analysis*, 5:530–538, 1968.

[SW10] P. Stumm and A. Walther. New algorithms for optimal online checkpointing. *SIAM J. Sci. Comput.*, 32(2):836–854, 2010.

[SWGH08] S. Schlenkrich, A. Walther, N. Gauger, and R. Heinrich. Differentiating fixed point iterations with ADOL-C: Gradient calculation for fluid dynamics. In H.G. Bock, E. Kostina, H.X. Phu, and R. Rannacher, editors, *Modeling, Simulation and Optimization of Complex Processes - Proceedings of the Third International Conference on High Performance Scientific Computing 2006*, pages 499–508, 2008.

[Ta'91] S. Ta'asan. One-shot methods for optimal control of distributed parameter systems I:Finite dimensional control. *ICASE Report 91-2*, 1991.

[Ta'95] S. Ta'asan. Pseudo-time methods for constrained optimization problems governed by PDE. *ICASE Report 95-32*, 1995.

[TKH91] A. Taylor, V. Korivi, and G. Hou. Sensitivity analysis applied to the Euler equations: a feasibility study with emphasis on variation of geometric shape. *AIAA Paper 91-0173*, 1991.

[TL72] H. Tennekes and J. Lumley. *A First Course in Turbulence*. MIT Press, 1972.

[TSVW91] E. Turkel, R. Swanson, V. Vatsa, and J. White. Multigrid for hypersonic viscous two- and three-dimensional flows. *AIAA Paper 91-1572*, 1991.

[vAvLR82] G. van Albada, B. van Leer, and W. Roberts. A comparative study of computational methods in cosmic gas dynamics. *Astron. Astrophysics*, 108:7684, 1982.

[vDR84] J. van Doormaal and G. Raithby. Enhancements of the SIMPLE method for predicting incompressible fluid flows. *Numerical Heat Transfer*, 7(2):147–163, 1984.

[vdV92] H. van der Vorst. BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13(2):631–644, March 1992.

[Ven95] 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.

[vL79] B. van Leer. Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method. *Journal of Computational Physics*, 32(1):101 – 136, 1979.

[VM95] H. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Longman Group Ltd., 1995.

[VM96] V. Venkatakrishnan and D. Mavriplis. Implicit method for the computation of unsteady flows on unstructured grids. *J. Computational Physics*, 127:380–397, 1996.

[Wer92] J. Werner. *Numerische Mathematik 1: Lineare und nichtlineare Gleichungssysteme, Interpolation, numerische Integration*. Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, 1992.

[Wil88] D. Wilcox. Re-assessment of the scale-determining equation for advanced turbulence models. *AIAA Journal*, 26(11):1299–1310, 1988.

[Wil04] D. Wilcox. *Turbulence modeling for CFD*. DCW Industries, 2nd ed. edition, 2004.

[Xue98] L. Xue. *Entwicklung eines effizienten parallelen Lösungsalgorithmus zur dreidimensionalen Simulation komplexer turbulenter Strömungen*. Ph.d thesis, Institute of Fluid Mechanics and Engineering Acoustics, Technical University of Berlin, 1998.

[YM05] Z. Yang and D.J. Mavriplis. Unstructured dynamic meshes with higher order time integration schemes for the unsteady Navier–Stokes equations. In *AIAA 2005-1222, 41th AIAA Aerospace Sciences Meeting and Exhibit*, January 10-13,NV, 2005.

[ZPGO09] A. Zymaris, D. Papadimitriou, K. Giannakoglou, and C. Othmer. Continuous adjoint approach to the Spalart-Allmaras turbulence model for incompressible flows. *Computers and Fluids*, 38:1528–1538, 2009.

[ZPGO10] A. Zymaris, D. Papadimitriou, K. Giannakoglou, and C. Othmer. Adjoint wall functions: A new concept for use in aerodynamic shape optimization. *J. Comput. Phys.*, 229(13):5228–5245, July 2010.