

# High-Fidelity CFD-based Shape Optimization of Wind Turbine Blades

**Madsen, Mads Holst Aagaard**

*Link to article, DOI:*
10.11581/dtu:00000068

*Publication date:*
2020

*Document Version*
Publisher's PDF, also known as Version of record

# High-Fidelity CFD-based Shape Optimization of Wind Turbine Blades

## Development and utilization of discrete adjoint solvers

Department of Wind Energy PhD Report 2019

Mads Holst Aagaard Madsen

DTU Wind Energy PhD-0099(EN)
DOI number: https://doi.org/10.11581/dtu:00000068

December 2019

**Authors:** Mads Holst Aagaard Madsen

**Title:** High-Fidelity CFD-based Shape Optimization of Wind Turbine Blades. Development and utilization of discrete adjoint solvers.

**2019**

**Project Period:**
September 2016 – December 2019

**Education:**
PhD

**Supervisor:**
Senior Researcher, Frederik Zahle
**Co-supervisor:**
Professor, Niels Nørmark Sørensen
Assistant Professor, Søren Juhl Andersen

**Examiners:**
Senior Researcher, Niels Troldborg, DTU Wind Energy
Researcher, Ardeshir Hanifi, KTH Royal Institute of Technology
Reader, Jens-Dominik Müller, Queen Mary Univ. of London, School of Engineering and Materials Science

DTU Wind Energy is a department of the Technical University of Denmark with a unique integration of research, education, innovation and public/private sector consulting in the field of wind energy. Our activities develop new opportunities and technology for the global and Danish exploitation of wind energy. Research focuses on key technical-scientific fields, which are central for the development, innovation and use of wind energy and provides the basis for advanced education.

DTU Wind Energy has a staff of approximately 240 and a further 35 PhD-students, spread across 38 different nationalities. The variety of research, education, innovation, testing and consultancy is reflected in the employment profile which includes faculty with research and teaching responsibilities, researchers and technical academic staff, highly skilled technicians and administrative staff.

Our facilities are situated at DTU Risø Campus and at DTU Lyngby Campus. Furthermore the department is running the national test stations in Høvsøre and Østerild.

**Technical University of Denmark**
Department of Wind Energy
Frederiksborgvej 399
Building 118
4000 Roskilde
Denmark

www.vindenergi.dtu.dk

DTU Wind Energy
Department of Wind Energy

Ph.D. Thesis

# High-Fidelity CFD-based Shape Optimization of Wind Turbine Blades

## - development and utilization of discrete adjoint solvers

Mads Holst Aagaard Madsen

Risø campus, Roskilde, 2020

DTU

**Technical University of Denmark**
**DTU Wind Energy**
**Department of Wind Energy**

DTU Risø Campus
Frederiksborgvej 399 Building 125
4000 Roskilde, Denmark
Phone +45 46 77 50 85
mham@dtu.dk
www.vindenergi.dtu.dk/english

# Summary

This thesis presents a methodology to enable gradient-based high-fidelity shape optimization using computational fluid dynamics (CFD). Through several decades CFD solvers have matured as analysis tools. In the wind energy research community and industry they are often used to validate and improve lower-fidelity models which in turn are used in design. The next logical step is therefore to use CFD solvers in the context of optimization. This will push the envelope of performance further than we can with conventional methods by enabling the concurrent shaping of planform and cross-sectional shape. Subsequently, one can then include a high-fidelity structural solver to carry out aerostructural optimization. The aerodynamic and structural responses could then be closely tailored to lower loads and increase power production. This thesis details how to efficiently utilize a CFD solver in shape optimization, which is the necessary first step towards such a high-fidelity multidisciplinary optimization framework.

The methodology to enable CFD-based shape optimization presented in this thesis is but one of several possibilities. One alternative way to do so is to carry out gradient-free shape optimization. Here, the implementation cost is low. However, high-fidelity shape optimizations can easily involve hundreds of design variables in order to accurately model the rotor and for such a high dimension of the parameter space the gradient-free methods are likely to incur an excessive amount of CFD evaluations before convergence is achieved. To use a gradient-based method one could approximate the gradient with the finite difference method. Still, the method scales poorly with the number of design variables and the gradient precision is inaccurate. While one can remedy the gradient inaccuracy with the so-called complex-step method there is no immediate cure for the prohibitive scaling. Although the above mentioned methods have all been used in several shape optimization efforts we advocate for the use of the *adjoint* approach to enable high-fidelity shape optimization.

The adjoint approach allows for a gradient computation whose computation cost is independent of the number of design variables. Furthermore, the underlying mathematics are intriguing and stunningly counterintuitive. One is bound to appreciate the elegance of the method as soon as it is grasped, but there are certainly also drawbacks for the adjoint approach. Most importantly, the development of an adjoint method from scratch is extremely time consuming. Depending on the CFD solver at hand it may take years to complete. In the hope that the account may be of help to other researchers we will therefore describe this process in great detail.

In the process of refining the shape optimization methodology presented herein we set several goals to be attained. These were all achieved during the project and are listed below:

- As an initial step we sought to carry out a comprehensive literature review particularly focused on high-fidelity shape optimization efforts within wind energy research. The (now published) survey helped us map out the promising high-fidelity optimization community which has started to emerge in wind energy.
- The second goal was to develop a deformation library from scratch. The tool should be based on proven methods, have an easy-to-use Python interface and it should provide analytical gradients to support an efficient gradient evaluation. This tool was named FFDlib, and is be presented in Chapter 5.
- The most important goal of this project was to develop an adjoint solver to be able to compute gradients in an efficient manner. A large part of this thesis is dedicated to describe exactly how we did this. In fact, we plan to develop numerous types of adjoint solvers. In the present work we document the first three adjoint solver types. Importantly, the developmental efforts helped identify a road map which has been of immense help during development. Here, there are seven distinct steps to complete. Each step is meticulously designed to isolate the source of error in the emerging code base. This is crucial considering that we have amassed up to 95 thousand lines of Fortran90 code for our adjoint solver (not counting the flow solver or FFDlib).
- Also the in-house CFD solver was enhanced during the project. As we will describe in Chapter 6 one can obtain machine accurate gradients from the flow solver itself albeit with a method that will not scale well with the number of design variables. Still, this enhancement of the flow solver is paramount to ensure a thorough debugging of the adjoint solver.
- A fifth goal was to set up a unifying numerical framework which should be user-friendly. To this end, we provided FFDlib and the adjoint solver with external interfaces. The framework is still maturing but can already be used to carry out high-fidelity shape optimizations. We conclude the thesis with a series of optimization test cases to demonstrate its capabilities.
- Finally, we mention an important and ongoing effort we have prioritized, i.e., to establish a collaboration with experts[1] from the aerospace community. In short, we extended one of the most advanced numerical optimization frameworks, MACH[2], developed in the aerospace research community with forward- and reverse algorithmically differentiated routines for rotational viscous fluids and used it on a modern 10 MW reference wind turbine. This resulted in the most comprehensive high-fidelity aerodynamic shape optimization on a wind turbine to date which was recently published [71]. This study demonstrates that it is feasible to carry out large scale free-form shape optimization using CFD on wind turbines. An excerpt

[1]The Multidisciplinary Design Optimization Laboratory (MDOLab) in the Department of Aerospace Engineering at the University of Michigan, `http://mdolab.engin.umich.edu/`, (last access: 20 April 2020)

[2]MDO for aircraft configurations with high-fidelity (MACH), `https://github.com/mdolab/MACH-Aero`, (last access: 20 April 2020)

of the optimization results will be brought at the end of the thesis together with results from our own optimization framework.

As seen above, it is a laborious endeavor to implement an adjoint method for a CFD solver and use it in a context of optimization. Luckily, it is also a thrilling and exciting task opening up for endless scientific questions to be answered.

# Resumé

Denne afhandling præsenterer en metode til gradient-baseret formoptimering ved hjælp af *computational fluid dynamics* (CFD). CFD er gennem en længere årrække i stigende grad blevet brugt som et analyseværktøj. I vindenergisektoren bruges CFD-værktøjer til at validere og forbedre de designprogrammer, som er baseret på ingeniørmodeller. Det logiske næste skridt er derfor at bruge CFD direkte i en designsammenhæng. Dette vil kunne forbedre den aktuelle ydeevne for vindmøller, da værktøjer baseret på CFD tillader en samtidig optimering af vingens spanvise planform og af dens vingeprofiler. Brug af CFD i designsammenhæng vil være det første nødvendige skridt i retningen hen mod en præcis aerostrukturel modellering, hvor vindmøllen optimeres ved brug af CFD og en tilsvarende højpræcis modellering af de strukturelle forhold. Sådan en kombination af modeller med høj præcision inden for både fluid dynamik og struktur vil i meget høj grad kunne skræddersy en vinges design til at være et optimalt kompromis mellem reducerede kræfter ud langs vingen og maksimal ydeevne. Metoden beskrevet i nærværende afhandling omhandler det første nødvendige skridt i den retning.

Den foreslåede formoptimeringsmetode er blot en af flere muligheder for at inkludere CFD i et designforløb. Metoden afhænger af en såkaldt *adjoint* gradientberegning, som er yderst besværlig at implementere. Et alternativ kunne være er at bruge en optimeringsalgoritme, som ikke benytter gradientbaseret information. Disse optimeringsalgoritmer er nemmere at implementere, men de vil ofte resultere i for mange funktionsevalueringer, da optimeringsproblemerne typisk involverer hundredvis af designvariable. Man kunne også vælge at benytte den præcis samme gradientbaserede optimeringsalgoritme, men derimod udskifte selve beregningsmetoden for gradienterne. En nemmere måde at beregne disse på ville for eksempel være ved hjælp af *finite differences*. Denne metode er bestemt nemmere at implementere end metoden præsenteret i nærværende afhandling. Uheldigvis skalerer denne alternative metode dog lineært med antallet af designvariable, hvilket vil umuliggøre brugen ved hundredevis af designvariable. Et andet problem ved denne alternative gradientestimering er, at den uvægerligt vil være upræcis. Sidstnævnte kan afværges ved hjælp af den såkaldte *complex-step* metode, men den nævnte lineære skalering vil ikke kunne forbedres. Endnu et alternativ er at bruge surrogatmodellering af selve CFD modellen. Dette alternativ vil dog også være ineffektivt for hundredvis af designvariable. Da alle de oven for nævnte alternativer ikke vil fungere optimalt for et stort antal designvariable, foretrækker vi som nævnt at udvikle en metode til formoptimering baseret på en adjoint gradientbereging – også selvom dette vil kræve et omsiggribende implementeringsarbejde.

Der er mange fordele ved at bruge en adjoint gradientberegning. Først og fremmest er det en fordel, at omkostningerne ved at beregne en given gradient ikke afhænger af antallet af designvariable. Der er dog også andre fordele så som en udvidet indsigt

i sensitiviteten. Metoden muliggør nemlig en form for sensitivitetsanalyse, som kan assistere designeren i at parametrisere vingen på en optimal måde, således at områder med stor indvirkning på den samlede ydeevne for vindmøllen bliver bedre parametriseret end områder uden for nævneværdig indflydelse. En adjoint gradientberegningsmetode har dog også ulemper, hvor den største uden tvivl er en meget kompliceret implementering. En fuldbyrdet og højeffektiv adjoint implementering inden for CFD kan resultere i et årelangt udviklingsarbejde. Det er derfor håbet, at nærværende afhandlings detaljerede implementeringsbeskrivelse af en adjoint beregningsmetode vil være til hjælp for fremtidige forskere, som står overfor lignende udviklingsudfordringer.

Ved starten af ph.d.-projektet blev sat seks overordnede målsætninger, som sidenhen alle blev opfyldt. De seks målsætninger ses nedenfor:

- Først skulle der foretages et grundigt litteraturstudie, som fokuserede på formoptimering baseret på CFD inden for vindenergisektoren. Dette litteraturstudie, som sidenhen er blevet udgivet, tegner et billede af et spirende forskningsfelt, udviklet inden for det sidste årti.
- Den anden målsætning var at udvikle et deformationsværktøj. Det skulle have en brugervenlig interface i Python, og det skulle kunne give analytiske gradienter. Dette værktøj hedder FFDlib og bliver beskrevet i kapital 5.
- Det vigtigste delmål for ph.d.-projektet var at udvikle en adjoint beregningsmetode i strømningsprogrammet: EllipSys3D. Dette udviklingsarbejde fylder størstedelen af afhandlingen. Under udviklingen blev en brugbar *road map* identificeret, som består af 7 trin. Disse trin er tilrettelagt på en sådan måde, at udviklingsarbejdet lettes betydeligt.
- EllipSys3D-programmet baseret på CFD, som er udviklet på DTU, er også blevet forbedret i løbet af projektet. Denne proces er beskrevet i kapitel 6 og involverer den såkaldte *complex-step* metode. Denne forbedring viste sig særdeles værdifuld under det videre udviklingsarbejde.
- Et femte mål var at samle alle de individuelle programmer i et burgervenligt optimeringsværktøj. Afhandlingens sidste halvdel indeholder tre tests (Fig. 6.7, 7.12, and 11.11) af de udviklede optimeringsværktøjer.
- Endeligt var det en meget vigtig målsætning, at der blev etableret en kontakt til ledende eksperter[3] inden for feltet. Dette samarbejde muliggjorde at komme til at arbejde med et af de mest avancerede optimeringsværktøjer på verdensplan: MACH[4]. Efter at have implementeret forbedringer inden for algoritmisk differentiation kunne et designoptimeringsstudie gennemføres for en 10 MW havvindmølle. Dette studie er det grundigste af sin art (Tab. 3.2) og blev for nylig publiceret i en artikel [71]. Et uddrag kan findes i kapitel 11.

[3]The Multidisciplinary Design Optimization Laboratory (MDOLab) in the Department of Aerospace Engineering at the University of Michigan

[4]MDO for aircraft configurations with high-fidelity (MACH)

Det er, som det fremgår af ovenstående, et omfangsrigt arbejde at skulle implementere en adjoint gradientberegningsmetode inden for CFD for derefter at formoptimere vindmøllevinger. Samtidig er det heldigvis også et fantastisk spændende arbejde, som åbner op for talrige relevante anvendelsesområder.

# Preface

This thesis was prepared at the department of Wind Energy at the Technical University of Denmark in fulfillment of the requirements for acquiring a Ph. D. degree.

The research described in this thesis forms part of the *High-Fidelity CFD-based Shape Optimization of Wind Turbine Blades* project with supervisor *Frederik Zahle* and co-supervisors *Niels N. Sørensen* and *Søren J. Andersen*. The project was funded by the *Aerodynamic Design Section* at DTU Wind.

Risø campus, Roskilde, May 20, 2020

Mads Holst Aagaard
Madsen

# Acknowledgements

First and foremost I would like to express my sincere gratitude for the guidance and supervision from my main supervisor, senior researcher Frederik Zahle. His deep insight in numerical wind energy research helped shape the researcher I am today. The co-supervision from professor Niels N. Sørensen and his intimate knowledge of computational fluid dynamics and the EllipSys3D flow solver has similarly been crucial for the project. Also the repeated encouragements from co-supervisor Søren J. Andersen have been instrumental to overcome the many dead ends one is bound to meet throughout the course of a three year project. The successful outcome of this project would never have been possible without such a strong cast of supervisors. Needless to say, I look forward to our continued collaboration.

I also wish to mention my gratitude towards the AER section at DTU Wind Energy and head of section Flemming Rasmussen for funding the project. I was fortunate enough to share an office with PhD students Sinem Özçakmak, Christian Grinderslev, and Oliver Lylloff, whom I thank along with my other fellow PhD students at the department for creating the best atmosphere one could ask for to carry out such a challenging project. A special thanks to PhD student Antariksh Dicholkar for proof reading. Also PhD students at other DTU departments come to mind such as Cetin Dilgen and Sumer Dilgen whom I thank for long discussions on the nuances of their adjoint implementation.

The external research stay at the MDOLab at the University of Michigan proved to be a pivotal point during the project. I am truly grateful to professor Joaquim R. R. A. Martins for giving me the opportunity to come to the MDOLab and including me in all group activities. His continued emphasis on ever improving clarity when disseminating research results is something I strive for in all future work. A special mentioning is owed to research investigator Charles A. Mader for countless hours discussing the development of the adjoint approach. Indeed, most of what I know of adjoint methods is due to his tutoring. Many colleagues and friends also helped ensure a successful outcome of the external stay: PhD candidates Nicolas Bons and Anil Yildirim assisted with FFD setup and guidance in the ANK[5] usage, respectively. I also thank PhD candidate Eirikur Jonsson for assistance with installation of numerous tools and PhD candidate Shamsheer Chauhan for proof reading and many fruitful discussions. Indeed, I am truly grateful to the entire MDOLab group for welcoming my so wholeheartedly.

Finally, I thank my family. I am forever grateful for your unwavering support.

[5] Approximate Newton–Krylov method

# Contents

**Summary** i

**Resumé** v

**Preface** ix

**Acknowledgements** xi

**Contents** xiii

**1 Introduction** 1
1.1 Motivation . . . . . 1
1.2 A project in three parts . . . . . 3

**I High-fidelity design advances within wind energy research** 7

**2 The continuous and the discrete approach to adjoint shape optimization** 9
2.1 The total derivative equation . . . . . 10
2.2 The KKT system . . . . . 13
2.3 The continuous and the discrete approach . . . . . 21

**3 State-of-the-art high-fidelity shape optimization of wind turbine blades** 25
3.1 Literature review . . . . . 25

**II The development of a numerical design optimization framework** 47

**4 Components of a numerical design framework** 49
4.1 Literature on discrete adjoint solvers for the SIMPLE algorithm . . . . . 51

**5 FFDlib: A stand-alone, free-form deformation library** 67
5.1 Free-form deformation . . . . . 67
5.2 Integration in a numerical optimization framework . . . . . 69

**6 The EllipSys3D flow solver** 77
6.1 The Reynolds-averaged Navier–Stokes equations . . . . . 77
6.2 Solution procedure with curvilinear coordinates . . . . . 78

6.3 Complexification . . . . . 84
6.4 Integration in a numerical optimization framework . . . . . 86
6.5 Framework evaluation with complexified solver on an extruded airfoil . . 87

**7 eDA: The EllipSys discrete adjoint solver 91**
7.1 Derivation of the residuals in a SIMPLE algorithm . . . . . 92
7.2 An adjoint road map . . . . . 94
7.3 Extension to mesh sensitivities . . . . . 106
7.4 Framework evaluation with adjoint solver on an extruded airfoil . . . . . 106

**8 Algorithmic differentiation 113**
8.1 Algorithmic differentiation through Tapenade . . . . . 115
8.2 Gradient verification of an adjoint solver based on algorithmic differentiation 122

**9 Turbulence and transition modeling 127**
9.1 Residual subroutines for transport equations . . . . . 128
9.2 Turbulence . . . . . 128
9.3 Transition . . . . . 132

**10 MPI implementation 135**
10.1 PETSc MPI solver . . . . . 136
10.2 Gradient verification using MPI . . . . . 138

**III High-fidelity shape optimization of wind turbine blades 141**

**11 High-fidelity shape optimization of wind turbine blades 143**
11.1 3-D wing optimization . . . . . 143
11.2 Multipoint shape optimization of a 10 MW offshore wind turbine . . . . 163

**12 Conclusion 171**
12.1 Outlook . . . . . 176

**A Multipoint high-fidelity CFD-based aerodynamic shape optimization of a 10 MW wind turbine 177**
A.1 Nomenclature . . . . . 208
A.2 Abbreviations . . . . . 209

**Bibliography 211**

# CHAPTER 1

# Introduction

This project is dedicated to the development of a numerical high-fidelity aerodynamic shape optimization framework. It is a considerable task to develop such a framework often necessitating years of hard work. Looking to the progressive aerospace community one will find research centers refining their frameworks over several decades. Only a strong motivation can warrant such an extensive effort. We present this motivation in Sec. 1.1 and subsequently outline the various parts of the thesis in Sec. 1.2.

## 1.1 Motivation

What is high-fidelity? The term pervades shape optimization literature and is often used in various meanings. In the present work we use the term "high-fidelity" to refer to an accurate aerodynamic modeling based on computational fluid dynamics (CFD). As our CFD model we use the Reynolds-averaged Navier–Stokes (RANS) equations and we view these as the lower bound for high-fidelity fluid models. By geometrically resolving the wind turbine blades we capture the 3-D complexity of the flow resulting in an accurate prediction of the loading on the turbine. Importantly, this approach includes the intricate flow at the blade tip and at the root where the engineering blade element momentum (BEM) type models using 2-D airfoil characteristics as input have their shortcomings in accuracy and thus cannot really be used for detailed design. For these very reasons the wind energy research community and industry have long used CFD in a context of analysis. Here, the scope is often to validate and improve lower-fidelity models such as models using the BEM method. These engineering models are in turn used in design since they provide an extremely fast analysis by combining empirical corrections and pre-computed data. A CFD-based approach will on the other hand model the aerodynamic response directly from the geometric representation itself. Therefore, a logical next step is to use CFD in a context of optimization which will allow for a concurrent shaping of planform and cross-sectional shape to improve rotor performance. We have now not only given the answer to what high-fidelity is, but also why to use it in design.

High-fidelity models cannot, however, replace the use of their lower-fidelity counterparts. To do this, the incurred amount of extra computation time is far too large. Indeed, such a transition is hard to imagine ever taking place when contemplating the amount of load cases one must consider. To comply with the standards set forth by the International Electrotechnical Commission (IEC)[1], a blade designer must take numerous aspects into account, including: average annual wind speed at location, extreme wind gusts over a 50

[1] `https://www.iec.ch/`,(last access: 21 June 2019)

year span, and the turbulent intensity of the wind. To name but one obvious problem for high-fidelity models we point to the matter of turbulence. Given that state-of-the-art applications at the moment by an large are steady-state optimizations without a turbulent inflow the gap to handle such load cases is sizeable. However, a complementary use of high-fidelity and lower-fidelity models in the preliminary aerodynamic design phase seems very feasible and beneficial for the industry.

In order to realize CFD-based wind turbine design one will have to face several challenges. The first challenge is related to the model being of high-fidelity. This necessitates the use of large CFD meshes to ensure a detailed geometric modeling in any analysis. This fact, which is easily verified on a given case study [71, Tab. 4 and Fig. 5], naturally also extends to optimization where many successive CFD-solutions are computed [71, Tab. 6]. Indeed, the final shapes after a completed optimization may even suggest completely contradicting design trends when comparing very coarse and very fine meshes [71, Fig. 10]. We mention the mesh size since it in turn means that each function evaluation will be very expensive. Thus, we will have to choose an optimization strategy that does not use too many function evaluations.

Another important aspect of high-fidelity optimization is that the ability to model the response in rotor performance to a change on the 3-D shape of the blade inevitably will lead to several hundreds of design variables. We will therefore have to choose an optimization strategy that works well with a high number of design variables. Within wind energy research one can find several works on CFD-based aerodynamic shape optimization using gradient-free methods [101, 26, 128, 64], which will also be evident from the literature review in Chapter 3. These gradient-free methods will, however, become too expensive for a high number of design variables since we cannot afford too many function evaluations. We will therefore only explore gradient-based shape optimization in our work.

The question that remains is then: How to obtain the gradient? In a way, the entire thesis revolves around this question. Gradient estimation using finite differences is easily implemented, but the method has several drawbacks. Finite difference gradients are not accurate to machine precision which might hinder a tight convergence of the optimization problem. Besides, a step size study is necessary to minimize the inaccuracy. Furthermore, the method scales linearly with the number of design variables, which renders the method intractable when using many design variables. One can opt to use the complex-step method [66, 67] to obtain machine accurate gradients, but also this method scales poorly with the number of design variables. Therefore, an obvious alternative is to use the adjoint approach which also allows for machine accurate gradient estimation. The computation cost is for this approach independent of the number of design variables. This makes it a perfect match for high-fidelity shape optimization which as mentioned is characterized by a high number of design variables. However, to use the adjoint approach on a CFD solver involves the development of an *adjoint solver* which will result in a considerable amount of work.

Development of adjoint solvers has been seen for decades in the aerospace community [46, 15, 47, 99, 79, 48] where compressible CFD solvers are used. The segregated incompressible CFD solvers common to wind energy research are less intuitive to linearize

and the development of adjoint solvers has as a result lacked behind. We have during the project identified a road map for the development of a specific type of discrete adjoint solver for these CFD solvers, and we find that – albeit laborious – it provides a systematic step by step guide to a successful adjoint implementation. This road map has been of immense help and we will dedicate a large part of the thesis to document exactly how we use it, and why.

As mentioned above, we have chosen a specific type of discrete adjoint solver, but there are other alternatives[2], all of which have led to impressive scientific results. While we do prefer the strategy presented in this thesis, we will not make any bold statements as to which method is superior. We do, however, firmly believe that the adjoint approach (regardless of the chosen adjoint solver strategy) is the right way to carry out high-fidelity shape optimization. A literature review of the various adjoint solver strategies, and the reasons for our chosen strategy will follow our literature survey on shape optimization before we present the implementation itself.

To conclude our motivation to embark on developing an adjoint solver we point to the advent of algorithmic differentiation. In short, algorithmic differentiation tools can be used to differentiate computer code. This should be seen as opposed to differentiating the entire CFD solver by hand. While algorithmic differentiation has been about for decades it is fair to say, that it has matured drastically since the early beginnings. The motivating aspect is, that algorithmic differentiation can greatly reduce the time needed to develop and maintain an adjoint solver. Algorithmic differentiation should, however, be used with care in order to obtain competitive performance. In this thesis we present not one, but three adjoint solvers. One of these are developed leveraging algorithmic differentiation. We believe our usage of algorithmic differentiation as well as our choice of adjoint solver type to be very generic, and researchers should as a result be able to apply the same approach to their CFD solvers.

Development of an adjoint solver for large, industrial scale CFD codes is by all means a daunting task but it is our hope, and the overall aim of the detailed documentation, to encourage more researchers within wind energy to develop adjoint solvers which in turn allow for high-fidelity gradient-based shape optimization.

## 1.2 A project in three parts

At the very beginning of this project, the Aerodynamic Design Section (AER)[3] at DTU Wind Energy had a state-of-the-art in-house CFD solver, and wished to incorporate it into a numerical optimization framework. This was achieved over a span of a 3 year project period, where the work naturally fell into three distinct phases. As a consequence, this thesis is also split in three parts, which we briefly summarize below.

In Part I we had a focus on application. Here, we sought the very limit of what could

[2]One could for example use the so-called 'continuous' approach or another type of discrete adjoint solver known as the 'fixed-point' approach

[3]`https://www.vindenergi.dtu.dk/english/research/research-sections/aer`, (last access: 21 June 2019)

currently be achieved with the most advanced tools world-wide. We also carried out an extensive literature review to identify the state-of-the-art specifically within wind energy research. Part 2 was a step back to the beginnings, where we went over all developmental phases required to go from CFD analysis to CFD design with a functioning solver as our starting point. This includes the development of an adjoint solver. Finally, in Part III, we reverted back to focusing on application to test the newly developed optimization framework. The three parts are described in more detail below.

### 1.2.1 Part 1: High-fidelity design advances within wind energy research

The very first task was to gain an overview of the adjoint approach and the related literature. We wished to identify the state-of-the-art of high-fidelity shape optimization, particularly within wind energy, and base our choice for future developments on this information. To this end, we conducted a literature survey and established contacts to leading experts from the aerospace research community. This resulted in a collaboration, where we extended the MACH[4] framework [53] developed at the MDOLab[5] to encompass adjoint derivatives for rotating viscous fluids. The extension allowed us to conduct the most comprehensive high-fidelity shape optimization study of a wind turbine to date [71], thus establishing state-of-the-art within the wind energy research field. Part I of the present thesis is an account of the above described efforts.

The first chapter of Part I is an introductory chapter to the adjoint approach. The introduction explains terms pertinent to the adjoint approach and is meant as a preparation for the ensuing literature review in Chapters 3 and 4. Given, that the adjoint approach might come across as counterintuitive at first glance we strongly encourage more novice readers to spend ample time on this chapter. One can hardly hope to decipher the various contributions within the research field if basic concepts such as, the *discrete* approach, the *continuous* approach, or the *KKT system* are not well understood.

Following the initial introduction is a literature review in Chapter 3. The aerospace research field is particularly rich with applications dating back to the first seminal papers by Jameson [46, 47]. With time, the focus on flow control spilled over to sister sciences such as the automotive industry. Recently, the push for high-fidelity design optimization seems to have reached wind energy research [102, 57, 71]. The level of detail in the literature review will heighten as the scope narrows in. Thus, the more distant related literature from the aerospace industry will be lightly touched upon, whereas the available literature on high-fidelity shape optimization within wind energy will be presented in detail. Given that this latter part of the literature review already has been published [71, p. 163-169] we will simply bring an excerpt in Chapter 3 with an updated overview containing the most relevant works. Interested readers are referred to the appendix (Sec. A) for the paper itself.

[4] MDO for aircraft configurations with high-fidelity

[5] `http://mdolab.engin.umich.edu/`,(last access: 21 June 2019)

### 1.2.2 Part 2: The development of a numerical design optimization framework

Part II of the thesis describes the assembly of a numerical design optimization framework – both the development of the individual components and how to connect them into a common framework. Given that the most crucial component by far is the adjoint solver, we will have an emphasis on the steps needed to develop a functioning adjoint solver. Part II starts in Chapter 4 with an overview of the four components in our framework: an optimizer, a geometry deformation module, a flow solver, and of course, an adjoint solver. In this chapter one will also find a review of literature on various architectures for *discrete* adjoint solvers in Sec. 4.1 focusing on the exact type of adjoint solver architecture that we chose. The remaining chapters in Part II, Chapter 5 to 10, describe these components in the following way: Chapter 5 describes the development of an in-house deformation tool, called FFDlib, which is based on the well-known free-form deformation technique. Then follows a presentation of the in-house CFD solver in Chapter 6 before Chapters 7 to 10 describe the development and gradient verification of the adjoint solver.

### 1.2.3 Part 3: High-fidelity shape optimization of wind turbine blades

The third and final part of the thesis is also its shortest. Here, we present high-fidelity shape optimization results in Chapter 11. All results are generated on the basis of gradient-based optimizations. However, where we in some cases have utilized the developed adjoint solver to generate the gradient, we have in other cases used less efficient methods such as the finite difference method and the complex-step method to allow for a comparison between the methods.

To conclude Part III we present results from what we still believe to be the most comprehensive high-fidelity aerodynamic shape optimization of a wind turbine rotor. These results were published along with the above mentioned literature review in a recent paper [71]. The study we bring in Chapter 11 is a multipoint high-fidelity aerodynamic shape optimization of a 10 MW rotor. There are also other studies, such as a comparison across fidelities to BEM optimization results but we refer to the paper itself for these results.

At the end of Chapter 11, we point to future work and various possibilities to extend the framework (Sec. 11.1.4). Finally, an overall conclusion of the project is given in Chapter 12.

# Part I

# High-fidelity design advances within wind energy research

# CHAPTER 2
# The continuous and the discrete approach to adjoint shape optimization

The adjoint approach may come across as counterintuitive at first, but once the matter is grasped, one is bound to appreciate its elegance and simplicity. Countless introductions[1] [48, 31] to the approach have been written and yet beginners may find the need to plow through several of these to find an angle they can relate to. This conundrum has been expressed by P. Farrell in the following way:

> "Reading the literature, there are almost as many ways to approach the topic as there are practitioners!"[2]
>
> Patrick E. Farrell, associate professor, University of Oxford,
> on the topic of adjoints and their application

While it may be difficult to pinpoint precisely which example to base an introduction on, we contend that viewing it through the prism of *any* example is indeed rewarding.

This opening chapter is first and foremost meant as an introduction to the topic for novice readers. Therefore, the first part (Sec. 2.1) will be in plain English, void of terms pertaining to functional analysis. Later on (Sec. 2.2), we will elaborate slightly on considerations such as functional spaces, existence of optimal solutions and first-order optimality conditions. However, this will hardly satisfy a rigorous mathematician. This is a choice we have made to nurture the intuition of the adjoint approach – also for readers less experienced in the theory of numerical PDE-constrained optimization.

In order to avoid muddying the chapter too much we choose an example with the so-called 'oneshot' approach. While this does require the introduction of the full set of auxiliary equations of the KKT system[3] (forward-, adjoint-, and control-equations), it will also allow us to leave out the topic of numerical optimization methods altogether. Thus, no discussion of either gradient-free or gradient-based methods is needed in this chapter.

The notation throughout the thesis is owed to the researchers that most successfully explained the theory to us. With respect to functional analysis and theoretical background

[1] `http://www.dolfin-adjoint.org/en/latest/documentation/maths/`,(last access: 21 June 2019)

[2] `http://www.dolfin-adjoint.org/en/latest/documentation/maths/1-foreword.html`,(last access: 21 June 2019)

[3] Named after the three mathematicians William Karush, Harold W. Kuhn, and Albert W. Tucker

this means the notation used by P. Farrell and M. Gunzburger [35]. When it comes to the introduction of the discrete equations for the total derivative and the adjoint equation we have aligned the notation with that used at the MDOLab[4].

## 2.1 The total derivative equation

In optimization we are interested in minimizing some functional, $\mathrm{J}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})$, which depends on the flow variables[5], $\tilde{\mathbf{w}} = [\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{p}]^T$, and the design variables, $\mathbf{x}_{\mathrm{DV}}$. $\mathrm{J}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})$ could e.g. be the drag on an airfoil. The flow variables are implicitly determined by some partial differential equation (PDE) which we will have to solve in order to determine said variables. Thus, we can view the PDE as a constraint in our minimization problem. For now, we make no choice of PDE-constraint and cost functional, but we will assume that whatever they may be, they can be discretized and solved. This means, that variables will in this first section be shown in a bold notation to stress that they are now discrete vectors due to some discretization. Thus, $\tilde{\mathbf{w}}$, is a vector of length $[\mathrm{N}_{\mathrm{state}} \times 1]$ whereas $\mathbf{x}_{\mathrm{DV}}$ is a vector of length $[\mathrm{N}_{\mathrm{DV}} \times 1]$. The functional of interest, $\mathrm{J}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})$, is on the other hand taken to be a scalar. In actual applications we will be interested in at least one functional and several constraints. The extension to more than one scalar is thankfully straight forward meaning that the following does not suffer a loss of generality.

Returning to the functional, $\mathrm{J}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})$, we see that it both depends on some parameters, $\mathbf{x}_{\mathrm{DV}}$, which we will call design variables (DV) throughout this work, and on the solution, $\tilde{\mathbf{w}}$, to some PDE. Later on, when we study shape optimization of wind turbine blades, the solution, $\tilde{\mathbf{w}}$, will be the flow, pressure, and turbulence states of the RANS equations. However, no matter the choice of PDE we will always use the symbol, $\mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})$, for the PDE. When the residuals of the PDE, $\mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})$, are all zero, it means that our PDE is solved, and a correct solution or state, $\tilde{\mathbf{w}}$, has been found. Note that we assume we always can solve the PDE, $\mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})$, to obtain a corresponding solution, $\tilde{\mathbf{w}}$, for a given set of design variables, $\mathbf{x}_{\mathrm{DV}}$. This means, that the solution is implicitly determined from our design variables through the PDE: $\tilde{\mathbf{w}}(\mathbf{x}_{\mathrm{DV}})$. One can, in other words view our functional, $\mathrm{J}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})$, as a pure function of our design variables: $\mathrm{J}(\tilde{\mathbf{w}}(\mathbf{x}_{\mathrm{DV}}), \mathbf{x}_{\mathrm{DV}}) = \hat{\mathrm{J}}(\mathbf{x}_{\mathrm{DV}})$. The reduced functional which only depends on $\mathbf{x}_{\mathrm{DV}}$ will be denoted with a hat to discern the two functionals. In summary, we have introduced the symbols:

$$\mathbf{x}_{\mathrm{DV}} : \text{design variables}, \tag{2.1}$$
$$\tilde{\mathbf{w}} : \text{state solution}, \tag{2.2}$$
$$\mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}}) : \text{PDE relating design variables and states}, \tag{2.3}$$
$$\mathrm{J}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}}) : \text{functional of interest, and} \tag{2.4}$$
$$\hat{\mathrm{J}}(\mathbf{x}_{\mathrm{DV}}) : \text{reduced functional}. \tag{2.5}$$

[4] `http://mdolab.engin.umich.edu/`,(last access: 21 June 2019)

[5] Here, we only mention the velocity variables $u$, $v$, and $w$ as well as the pressure variable, $p$, but if turbulence and transition models are included we would have to add extra variables to $\tilde{\mathbf{w}}$

Note that an overview of all symbols introduced throughout this work can be found in the appendix (Sec. A.1).

Since we want to minimize the functional, it seems relevant to study how it changes. To this end, we use the chain rule on its derivative,

$$\underset{[1\times N_{DV}]}{\frac{d\hat{J}(\mathbf{x}_{DV})}{d\mathbf{x}_{DV}}} = \underset{[1\times N_{DV}]}{\frac{\partial J(\tilde{\mathbf{w}},\mathbf{x}_{DV})}{\partial \mathbf{x}_{DV}}} + \underset{[1\times N_{state}]}{\frac{\partial J(\tilde{\mathbf{w}},\mathbf{x}_{DV})}{\partial \tilde{\mathbf{w}}}} \quad \underset{[N_{state}\times N_{DV}]}{\frac{d\tilde{\mathbf{w}}}{d\mathbf{x}_{DV}}} \quad , \tag{2.6}$$

where we have added the sizes of each term. The total derivative, $d\hat{J}(\mathbf{x}_{DV})/d\mathbf{x}_{DV}$, is a vector if we only focus on one functional. This also holds for the partial derivative, $\partial J(\tilde{\mathbf{w}},\mathbf{x}_{DV})/\partial \mathbf{x}_{DV}$, which describes the immediate change in the functional when the design variables change. This term is sometimes zero, and can be omitted, whereas one must compute it at other times to obtain a correct gradient. To be specific, imagine our functional, $J(\tilde{\mathbf{w}},\mathbf{x}_{DV})$, to be the integrated drag over an airfoil surface, and that each CFD mesh point of said surface is a design variable, $\mathbf{x}_{DV}$. Then a change in $\mathbf{x}_{DV}$ would most certainly result in a direct change in $J(\tilde{\mathbf{w}},\mathbf{x}_{DV})$ and we would have to compute this partial derivative term. However, had our design variables instead been related to the inlet boundary conditions (BC) which are far removed from the airfoil surface in the CFD mesh, there would be no immediate effect, and the term would be zero. Turning to the two remaining terms we have the immediate change in our functional when the state variables change, $\partial J(\tilde{\mathbf{w}},\mathbf{x}_{DV})/\partial \tilde{\mathbf{w}}$, and finally, $d\tilde{\mathbf{w}}/d\mathbf{x}_{DV}$, which is known as the solution Jacobian[6]. Here, the former is also a vector and easy to compute, but unlike $\partial J(\tilde{\mathbf{w}},\mathbf{x}_{DV})/\partial \mathbf{x}_{DV}$ it is never entirely zero. The solution Jacobian, $d\tilde{\mathbf{w}}/d\mathbf{x}_{DV}$, is, on the other hand, a large total derivative matrix and much more difficult to compute. To analyze this term better, we turn to the PDE, $\mathbf{R}(\tilde{\mathbf{w}},\mathbf{x}_{DV})$, that determines the implicit relation between $\tilde{\mathbf{w}}$ and $\mathbf{x}_{DV}$.

The PDE is a set of equations, that we wish to solve, i.e., we wish to find the set of states, $\tilde{\mathbf{w}}$, that satisfies the condition:

$$\mathbf{R}(\tilde{\mathbf{w}},\mathbf{x}_{DV}) = \mathbf{0}. \tag{2.7}$$

By taking the derivative of Eq. (2.7) we obtain a new relation involving the solution Jacobian, which can be used in Eq. (2.6):

$$\frac{d\mathbf{R}(\tilde{\mathbf{w}},\mathbf{x}_{DV})}{d\mathbf{x}_{DV}} = \mathbf{0} \Leftrightarrow \tag{2.8}$$

$$\underset{[N_{state}\times N_{state}]}{\frac{\partial \mathbf{R}(\tilde{\mathbf{w}},\mathbf{x}_{DV})}{\partial \tilde{\mathbf{w}}}} \quad \underset{[N_{state}\times N_{DV}]}{\frac{d\tilde{\mathbf{w}}}{d\mathbf{x}_{DV}}} = -\underset{[N_{state}\times N_{DV}]}{\frac{\partial \mathbf{R}(\tilde{\mathbf{w}},\mathbf{x}_{DV})}{\partial \mathbf{x}_{DV}}}. \tag{2.9}$$

[6]Later on, we will talk of $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$ as the *state* Jacobian but the Jacobians should not be confused: $d\tilde{\mathbf{w}}/d\mathbf{x}_{DV}$ is the solution Jacobian and $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$ is the state Jacobian. $d\tilde{\mathbf{w}}/d\mathbf{x}_{DV}$ is in this work mainly of interest when discussing the underlying mathematics whereas $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$ will be ubiquitous when discussing the adjoint solver development in Part II

This new relation, where we added the sizes below, is called the tangent-linear system. The first term in Eq. (2.9) is the partial derivative of the PDE with respect to the states, $\partial \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})/\partial \tilde{\mathbf{w}}$. If we take $\mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})$ to be linear and write it on matrix form: $\mathbf{A}\tilde{\mathbf{w}} - \mathbf{b} = 0$ for some matrix, $\mathbf{A}$, and some vector, $\mathbf{b}$, then it is easy to see that $\partial \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})/\partial \tilde{\mathbf{w}} = \mathbf{A}$. In the case where the PDE is nonlinear it is less straightforward to visualize, but in any case it is the linearization of $\mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})$ at the specific solution, $\tilde{\mathbf{w}}$. The 'tangent-linear system' is therefore a fitting name for Eq. (2.9); it is a linearization of the original PDE-constraint, and it is a tangent to the original system in a specific point. Turning to the right hand side of Eq. (2.9) it is generally less easy to grasp without a specific example since it depends on the chosen design variables. We therefore choose a particular PDE in the next section[7].

Before we turn to an actual example PDE, there is yet another equation, that can be derived using the total derivative Eq. (2.6) and the tangent-linear system in Eq. (2.9). To do so, we first solve Eq. (2.9) for the solution Jacobian, which is the unknown in that equation[8]:

$$\frac{\mathrm{d}\tilde{\mathbf{w}}}{\mathrm{d}\mathbf{x}_{\mathrm{DV}}} = -\left[\frac{\partial \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\partial \tilde{\mathbf{w}}}\right]^{-1} \frac{\partial \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\partial \mathbf{x}_{\mathrm{DV}}}. \tag{2.10}$$

Now, the solution Jacobian can be inserted in the total derivative Eq. (2.6):

$$\frac{\mathrm{dJ}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\mathrm{d}\mathbf{x}_{\mathrm{DV}}} = \frac{\partial \mathrm{J}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\partial \mathbf{x}_{\mathrm{DV}}} - \underbrace{\frac{\partial \mathrm{J}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\partial \tilde{\mathbf{w}}} \left[\frac{\partial \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\partial \tilde{\mathbf{w}}}\right]^{-1}}_{=\boldsymbol{\Psi}^T} \frac{\partial \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\partial \mathbf{x}_{\mathrm{DV}}} \Leftrightarrow \tag{2.11}$$

$$\frac{\mathrm{dJ}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\mathrm{d}\mathbf{x}_{\mathrm{DV}}} = \frac{\partial \mathrm{J}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\partial \mathbf{x}_{\mathrm{DV}}} - \boldsymbol{\Psi}^T \frac{\partial \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\partial \mathbf{x}_{\mathrm{DV}}}. \tag{2.12}$$

As seen, we replaced a vector-matrix product with a transposed vector, $\boldsymbol{\Psi}^T$, but in order to calculate the total derivative from Eq. (2.12) we must now first determine $\boldsymbol{\Psi}^T$ through the relation:

$$\left[\frac{\partial \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\partial \tilde{\mathbf{w}}}\right]^T \boldsymbol{\Psi} = \left[\frac{\partial \mathrm{J}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\partial \tilde{\mathbf{w}}}\right]^T. \tag{2.13}$$

Eq. (2.13) is called the *adjoint* equation, and the vector we solve for, is known as the *adjoint* vector. The two partial derivative matrices are known from the previous equations, but they are now both transposed and intuition about the adjoint system is – at first glance – less obvious.

[7] We will also visualize $\partial \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})/\partial \mathbf{x}_{\mathrm{DV}}$ when we present our adjoint solver development in Chapter 7. The visualization can be found in Fig. 7.5

[8] Eq. (2.10) is known as the direct method. Here, one would insert the $\mathrm{d}\tilde{\mathbf{w}}/\mathrm{d}\mathbf{x}_{\mathrm{DV}}$ in Eq. (2.6) to obtain the gradient, `http://openmdao.org/twodocs/versions/latest/theory_manual/total_derivs/total_derivs_theory.html#direct-method`,(last access: 20 April 2020)

However, we can say something general about the two approaches, i.e., either solving Eq. (2.9) or Eq. (2.13), before finally evaluating Eq. (2.6) to find the total derivative. Starting with the tangent-linear system (Eq. (2.9)) we realize that none of the partial derivative matrices are made up of the functional, $\mathrm{J}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})$. This means, that by solving Eq. (2.9) just once, we can use the same solution Jacobian, $\mathrm{d}\tilde{\mathbf{w}}/\mathrm{d}\mathbf{x}_{\mathrm{DV}}$, to solve Eq. (2.6) for the derivative of numerous functionals. It is in other words the preferred approach if we are interested in the derivative of many different functionals, but only have few design variables.

The exact opposite of what we said for the tangent-linear system, can be said when it comes to the adjoint equation. Here, it is our design variables, $\mathbf{x}_{\mathrm{DV}}$, that do not enter the equation (2.13), so we can solve for $\boldsymbol{\Psi}$ just once, and use it repeatedly to compute the total derivative in Eq. (2.12) for any amount of design variables. We only have to solve the adjoint equation for each functional we are interested. The adjoint approach is in other words the best approach when we deal with a large number of design variables.

## 2.2 The KKT system

The introduction to the adjoint equation found in the previous Sec. (2.1) is by far the simplest possible introduction to the adjoint equation. A similar introduction strategy can be found in well known reviews (e.g., [31, Sec. 1-2]) that first introduce the adjoint approach in the context of linear algebra, and then focus on the continuous derivation. Using nothing but the chain rule, and a reduced view of the functional, $\mathrm{J}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}}) \rightarrow \hat{\mathrm{J}}(\mathbf{x}_{\mathrm{DV}})$, where we build the state equation into the functional, one can introduce the adjoint equation in a few steps. It is, however, not the full picture that is painted above. Besides the state- and the adjoint equation, there is another auxiliary equation called the *optimality condition*. Together, these three form a coupled set of equations known as the *KKT system*, which we will now introduce, since it is instructive to view the adjoint equation in the broader frame of reference of the KKT system.

To introduce the KKT system, we start by writing up the general, constrained optimization problem we are concerned with:

$$\begin{aligned} \min_{\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}}} \quad & \mathrm{J}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}}), \\ \text{subject to} \quad & \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}}) = \mathbf{0}. \end{aligned} \tag{2.14}$$

Our objective function, $\mathrm{J}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})$, is some function that depends on state variables, $\tilde{\mathbf{w}}$, and parameters, $\mathbf{x}_{\mathrm{DV}}$ which we call design variables.

We then build the PDE-constraint into the functional. However, we do this in a more rigorous manner by way of the Lagrange-multiplier method:

$$\mathcal{L}(\tilde{\mathbf{w}}, \boldsymbol{\lambda}, \mathbf{x}_{\mathrm{DV}}) = \mathrm{J}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}}) - \boldsymbol{\lambda}^T \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}}). \tag{2.15}$$

In the above, we have used the symbol , $\mathcal{L}(\tilde{\mathbf{w}}, \boldsymbol{\lambda}, \mathbf{x}_{\mathrm{DV}})$, for the augmented cost function, called the *Lagrangian*, which depends on an extra variable, $\boldsymbol{\lambda}$. This is owed to the

fact that we added the inner product, $\boldsymbol{\lambda}^T\mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})$, between the Lagrange multiplier variable, $\boldsymbol{\lambda}$, and the PDE-constraint, $\mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})$. When the PDE-constraint is satisfied this inner product term should of course be zero and little will have changed in our optimization problem. An equivalent problem should therefore be:

$$\min_{\tilde{\mathbf{w}}, \boldsymbol{\lambda}, \mathbf{x}_{\mathrm{DV}}} \quad \mathcal{L}(\tilde{\mathbf{w}}, \boldsymbol{\lambda}, \mathbf{x}_{\mathrm{DV}}). \tag{2.16}$$

As seen, we have now recast the originally constrained optimization problem (2.14) into an unconstrained optimization problem (2.16). However, this came at a considerable cost seeing that we now have a minimization problem with three independent variables.

To finally solve our optimization problem we set the derivative to zero in each of the three independent variable directions:

$$\frac{\mathrm{d}\mathcal{L}(\tilde{\mathbf{w}}, \boldsymbol{\lambda}, \mathbf{x}_{\mathrm{DV}})}{\mathrm{d}\boldsymbol{\lambda}} = \mathbf{0} : \mathrm{PDE} - \mathrm{constraint} \tag{2.17}$$

$$\frac{\mathrm{d}\mathcal{L}(\tilde{\mathbf{w}}, \boldsymbol{\lambda}, \mathbf{x}_{\mathrm{DV}})}{\mathrm{d}\tilde{\mathbf{w}}} = \mathbf{0} : \mathrm{adjoint\ equation} \tag{2.18}$$

$$\frac{\mathrm{d}\mathcal{L}(\tilde{\mathbf{w}}, \boldsymbol{\lambda}, \mathbf{x}_{\mathrm{DV}})}{\mathrm{d}\mathbf{x}_{\mathrm{DV}}} = \mathbf{0} : \mathrm{optimality\ condition} \tag{2.19}$$

These equations (2.17-2.19) are the KKT system. They describe the first-order necessary conditions for our optimization problem (2.16) to have a solution.

One advantage of using the entire KKT system is, that if we solve these three equations[9] at the same time, we can obtain a solution to our optimization problem without having to invoke an optimizer[10]. It is, however, more common to first solve the state equation, and then solve the adjoint equation to compute the gradient, since it can be very difficult to solve the three equations simultaneously. Therefore, researchers within high-fidelity shape optimization usually choose an iterative procedure where the optimization problem is solved using a gradient-based optimizer. Depending on the gradient precision and the optimizer it may be necessary to solve the state- and adjoint equation numerous times to allow the optimizer to take as many steps as needed to converge the problem. Still, it is often the preferred choice, since it is much easier to solve one equation at a time instead of solving all three equations simultaneously.

To be guaranteed that a given solution is indeed also a (local) extremum and not a saddle point one would also have to consider the second-order sufficient conditions which take the second-order derivative of the Lagrangian into account. On the right hand side in Eq. (2.17-2.19) one will find the name of the equation that each derivative will lead to. This is of course a postulate so far, but we can actually already make a sanity check on this postulate for Eq. (2.17) and Eq. (2.19) even though we have yet to choose an actual PDE-constraint. Looking at Eq. (2.15) it is evident that the derivative with respect to

[9] State equation 2.17, adjoint equation 2.18 and optimality condition 2.19

[10] This is, e.g., discussed in [35, Sec. 2.8]

$\boldsymbol{\lambda}$ in Eq. (2.17) is indeed the PDE-constraint, from Eq. (2.7). To check Eq. (2.19) we simply insert Eq. (2.15) into it:

$$\begin{aligned}
\frac{\mathrm{d}\,\mathcal{L}(\tilde{\mathbf{w}},\boldsymbol{\lambda},\mathbf{x}_{\mathrm{DV}})}{\mathrm{d}\mathbf{x}_{\mathrm{DV}}} &= \mathbf{0} \Leftrightarrow \\
\frac{\mathrm{d}\,\mathrm{J}(\tilde{\mathbf{w}},\mathbf{x}_{\mathrm{DV}}) - \boldsymbol{\lambda}^T\mathbf{R}(\tilde{\mathbf{w}},\mathbf{x}_{\mathrm{DV}})}{\mathrm{d}\mathbf{x}_{\mathrm{DV}}} &= \mathbf{0} \Leftrightarrow \\
\frac{\partial \mathrm{J}(\tilde{\mathbf{w}},\mathbf{x}_{\mathrm{DV}})}{\partial \mathbf{x}_{\mathrm{DV}}} + \frac{\partial \mathrm{J}(\tilde{\mathbf{w}},\mathbf{x}_{\mathrm{DV}})}{\partial \tilde{\mathbf{w}}}\frac{\mathrm{d}\tilde{\mathbf{w}}}{\mathrm{d}\mathbf{x}_{\mathrm{DV}}} - \boldsymbol{\lambda}^T\left[\frac{\partial \mathbf{R}(\tilde{\mathbf{w}},\mathbf{x}_{\mathrm{DV}})}{\partial \mathbf{x}_{\mathrm{DV}}} + \frac{\partial \mathbf{R}(\tilde{\mathbf{w}},\mathbf{x}_{\mathrm{DV}})}{\partial \tilde{\mathbf{w}}}\frac{\mathrm{d}\tilde{\mathbf{w}}}{\mathrm{d}\mathbf{x}_{\mathrm{DV}}}\right] &= \mathbf{0} \Leftrightarrow \\
\underbrace{\frac{\partial \mathrm{J}(\tilde{\mathbf{w}},\mathbf{x}_{\mathrm{DV}})}{\partial \mathbf{x}_{\mathrm{DV}}} - \boldsymbol{\lambda}^T\frac{\partial \mathbf{R}(\tilde{\mathbf{w}},\mathbf{x}_{\mathrm{DV}})}{\partial \mathbf{x}_{\mathrm{DV}}}}_{\text{total derivative Eq. (2.12)}} - \underbrace{\left[\boldsymbol{\lambda}^T\frac{\partial \mathbf{R}(\tilde{\mathbf{w}},\mathbf{x}_{\mathrm{DV}})}{\partial \tilde{\mathbf{w}}} - \frac{\partial \mathrm{J}(\tilde{\mathbf{w}},\mathbf{x}_{\mathrm{DV}})}{\partial \tilde{\mathbf{w}}}\right]}_{\text{adjoint Eq. (2.13)}}\frac{\mathrm{d}\tilde{\mathbf{w}}}{\mathrm{d}\mathbf{x}_{\mathrm{DV}}} &= \mathbf{0}. \qquad (2.20)
\end{aligned}$$

As seen, we have arrived at a term resembling the total derivative equation (2.12) from the previous Sec. 2.1. In addition, there is a large term involving the solution Jacobian, $\mathrm{d}\tilde{\mathbf{w}}/\mathrm{d}\mathbf{x}_{\mathrm{DV}}$. This Jacobian is as mentioned very cumbersome to compute. To avoid this, we set the part inside the bracket to zero, but this is of course nothing but the adjoint Eq. (2.13). Thus, by substituting the Lagrange multiplier, $\boldsymbol{\lambda}$, with the adjoint variables, $\boldsymbol{\Psi}$, we have arrived at not only the total derivative equation but also the adjoint equation:

$$\underbrace{\frac{\partial \mathrm{J}(\tilde{\mathbf{w}},\mathbf{x}_{\mathrm{DV}})}{\partial \mathbf{x}_{\mathrm{DV}}} - \boldsymbol{\Psi}^T\frac{\partial \mathbf{R}(\tilde{\mathbf{w}},\mathbf{x}_{\mathrm{DV}})}{\partial \mathbf{x}_{\mathrm{DV}}}}_{\text{total derivative Eq. (2.12)}} - \underbrace{\left[\boldsymbol{\Psi}^T\frac{\partial \mathbf{R}(\tilde{\mathbf{w}},\mathbf{x}_{\mathrm{DV}})}{\partial \tilde{\mathbf{w}}} - \frac{\partial \mathrm{J}(\tilde{\mathbf{w}},\mathbf{x}_{\mathrm{DV}})}{\partial \tilde{\mathbf{w}}}\right]}_{\text{adjoint Eq. (2.13)}}\frac{\mathrm{d}\tilde{\mathbf{w}}}{\mathrm{d}\mathbf{x}_{\mathrm{DV}}} = \mathbf{0}. \qquad (2.21)$$

### 2.2.1 Iterative solution of the KKT system

From Eq. (2.21) we learn that the optimality condition in Eq. (2.19) is fulfilled once the total derivative equation is zero and the adjoint equation is solved. This means that we also implicitly solve the optimality condition when we use the iterative approach with gradient-based methods instead of a oneshot method. Once a gradient-based optimizer has solved the optimization problem, the total derivative is zero and the adjoint equation is certainly also solved since we solve it at each optimization step to obtain the gradient. A gradient-based method is therefore simply a way of solving the KKT system in a decoupled manner, where we solve a sequence of smaller problems instead of solving all three equations in the KKT system simultaneously. By now, it is quite obvious, that the variable we used to incorporate the PDE-constraint into the functional was indeed the adjoint variable, which is also known as the costate variable or Lagrange multiplier. Hence the name; the Lagrange multiplier method.

### 2.2.2 The derivation of the adjoint equation using the continuous approach

We now turn to equation (2.18) in the KKT system to study the derivation of an adjoint equation. We already derived a general expression for this equation (see Eq. (2.13)) but

that was for the matrices resulting from a discretization. Now, we want to derive the actual equation, which can then in turn be discretized in any way we see fit. To this end, we choose an example problem with a transient heat equation as the PDE-constraint defined for the space-time domain $\Omega \times [0, T]$:

$$\min_{y,u} \quad \frac{1}{2}\int_0^T \int_\Omega (y(x,t) - \hat{y}(x,t))^2 \mathrm{d}\Omega \mathrm{d}t + \frac{\beta}{2}\int_0^T \int_\Omega u^2(x,t) \mathrm{d}\Omega \mathrm{d}t \tag{2.22}$$

$$\text{subject to} \quad \frac{\partial y(x,t)}{\partial t} - \Delta y(x,t) = u(x,t), \quad \text{for } (x,t) \in \Omega \times [0,T], \tag{2.23}$$

$$y(x,t) = g(x,t), \quad \text{for } (x,t) \text{ on } \partial\Omega \times [0,T], \tag{2.24}$$

$$y(x,t) = y_0(x), \quad \text{for } (x,t) \in \Omega \text{ at } t = 0. \tag{2.25}$$

Eq. (2.22-2.25) make up a continuous problem formulation. As seen, we have no bold symbols signifying vectors and matrices since we first want to derive the adjoint equation before deciding on a discretization. We do this to introduce the continuous approach to the adjoint method. In the above, Eq. (2.22) is the cost functional we wish to minimize, whereas Eq. (2.23-2.25) are state equation[11], boundary equation, and initial condition, respectively. The first term in the functional from Eq. (2.22) is the error we wish to minimize between the actual temperature field, $y(x,t)$, and a desired temperature field, $\hat{y}(x,t)$, measured by the $L^2(\Omega)$-norm[12]. The second term is a penalization term, made up of a penalization parameter, $\beta$, and our design variable, $u(x,t)$, which is a heat source. This term ensures that the use of our control, i.e., the heating source on $\Omega$, can be moderated. The state Eq. (2.23-2.25) is a classical transient heat equation where we have chosen Dirichlet boundary conditions seen in Eq. (2.24). Given, that the state equation advances forward in time, $t$, we must also state some initial conditions, Eq. (2.25), at $t = 0$. The optimization is now a matter of adjusting the heat source, $u$, in the space-time domain so that the room temperature, $y$, most closely matches some desired distribution, $\hat{y}$, that we would like. Applications of the above heat control problem are endless. To give but one example it could be within biomedicine where certain products must be stored at a certain temperature. Variants of this well known heating control problem have been dealt with extensively elsewhere. We refer interested readers to [21, p. 36], [35, p. 48] and [121, p. 125] for a more in depth analysis. We will use the example problem to nurture the overall intuition of the adjoint approach and to point to some critical matters related to implementation that must be handled delicately, when we transition to industrial scale CFD applications instead of small examples merely serving as proof of concept.

Before proceeding with the derivation of the adjoint equation, it can be useful to visualize the sparsity pattern of the discrete state equation for later comparison with the

[11] In the state equation we express the Laplace operator, $\nabla^2$, using the delta symbol, $\Delta$

[12] The $\hat{y}(x,t)$ should not be confused with the reduced functional, $\hat{J}$. Here, we simply use the hat to discern $\hat{y}(x,t)$ from $y(x,t)$. $\hat{y}(x,t)$ is some temperature distribution that we know for all time steps, and $y(x,t)$ is the unknown temperature distribution which we can find by solving our PDE-constraint in Eq. (2.23)

adjoint equation. This can of course only be done once an actual discretization has been chosen. We will use a simple finite element method (FEM) based on Galerkin elements and a first-order backwards Euler time discretization that can provide a piecewise linear approximation of the true solution. An in-depth presentation of the discretization will not be given here, as it is out of the scope of the present work. Interested readers can consult [70] for further information where the FEM discretization in question was used to develop a fluid solver[13]. Finally, we note that the chosen notation for the discretization has been aligned with [97, 92].

Returning to the visualization we arrive at the following discretized form of the state Eq. (2.23):

$$\mathcal{K}\mathbf{y} = \tau\mathcal{M}\mathbf{u} + \mathbf{d}, \tag{2.26}$$

where, $\tau$ is the discrete time step, the matrix, $\mathcal{K}$, contains both the typical FEM mass and stiffness matrices and the matrix, $\mathcal{M}$, contains only FEM mass matrices. We limit the discretization of the time domain to four time steps $[\tau_1, \tau_2, \tau_3, \tau_4]$ in order to better discern the structure visualized in the *transient* matrices, $\mathcal{K}$ and $\mathcal{M}$. Eq. (2.26) can be inspected visually in Fig. 2.1 where we have highlighted a state equation solve at time step, $\tau_2$, with a red rectangle. As mentioned, we use a backwards Euler discretization in the time domain. This can be verified in Fig. 2.1 by noting that $y_1$ (the previous temperature values) are used to compute the $y_2$ values the time step after. An important point to note in the visualization in Fig. 2.1 is that the overall structure of the sparsity pattern is a lower triangular matrix on the left hand side, and a source vector on the right hand side.

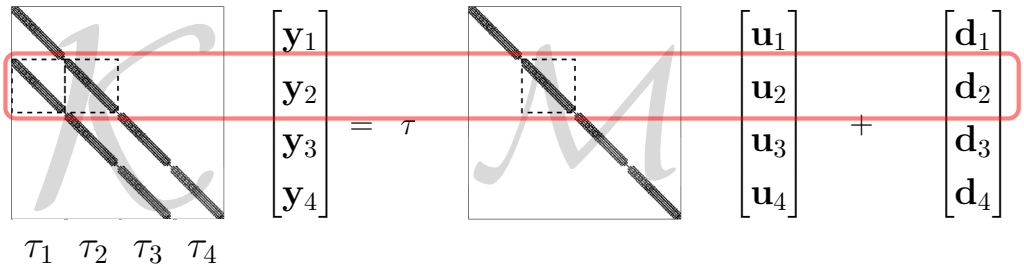



**Figure 2.1:** Discretized state Eq. (2.26) using a FEM discretization.

Finally, we are ready to follow the prescription from the previous section. Thus, we construct the Lagrangian and take the derivative in the direction of the state, $y$, to derive the adjoint equation. The resulting Lagrangian, $\mathcal{L}(y, u, \Psi)$, is found using Lagrange multipliers; $\Psi_{\mathrm{St}}, \Psi_{\mathrm{BC}}$, and $\Psi_{\mathrm{IC}}$, for state, boundary condition, and initial condition, respectively:

[13] link

$$\begin{aligned}\mathcal{L}(y,u,\Psi) =& \frac{1}{2}\int_0^T\int_\Omega (y-\hat{y})^2 \mathrm{d}\Omega\mathrm{d}t + \frac{\beta}{2}\int_0^T\int_\Omega u^2 \,\mathrm{d}\Omega\mathrm{d}t \\ &- \int_0^T\int_\Omega \Psi_{\mathrm{St}}\left[\frac{\partial y}{\partial t} - \Delta y - u\right]\mathrm{d}\Omega\mathrm{d}t \\ &- \int_0^T\int_{\partial\Omega} \Psi_{\mathrm{BC}}\left[y - g(x)\right]\mathrm{d}\Omega\,\mathrm{d}t - \int_\Omega \Psi_{\mathrm{IC}}\left[y|_{t=0} - y_0\right]\mathrm{d}\Omega. \end{aligned} \tag{2.27}$$

To derive the adjoint equation we must now set the derivative in the direction of the state, $y$, equal to zero. A preferred approach by many high-fidelity shape optimization practitioners[14] is here to turn to calculus of variations and set the first variation of the Lagragian, $\delta\mathcal{L}(y,u,\Psi)$, equal to zero for some direction. To set the first variation of the Lagrangian with respect to the state equal to zero amounts to [35, p. 17],

$$\delta_y\mathcal{L}(y,u,\Psi) \equiv \lim_{\epsilon\to 0}\frac{\mathcal{L}(y+\epsilon\tilde{y},u,\Psi) - \mathcal{L}(y,u,\Psi)}{\epsilon} = 0, \tag{2.28}$$

where $\tilde{y}$ is a small variation in the state, $y$. Inserting the Lagrangian from Eq. (2.27) we get,

$$\begin{aligned}\lim_{\epsilon\to 0}\frac{\mathcal{L}(y+\epsilon\tilde{y},u,\Psi) - \mathcal{L}(y,u,\Psi)}{\epsilon} =& \int_0^T\int_\Omega \left[(y-\hat{y})\right]\tilde{y}\,\mathrm{d}\Omega\mathrm{d}t + 0 \\ &- \int_0^T\int_\Omega \Psi_{\mathrm{St}}\left[\frac{\partial\tilde{y}}{\partial t} - \Delta\tilde{y} - 0\right]\mathrm{d}\Omega\mathrm{d}t \\ &- \int_0^T\int_{\partial\Omega}\Psi_{\mathrm{BC}}\left[\tilde{y} - 0\right]\mathrm{d}\Omega\mathrm{d}t - \int_\Omega \Psi_{\mathrm{IC}}\left[\tilde{y}|_{t=0} - 0\right]\mathrm{d}\Omega = 0. \end{aligned} \tag{2.29}$$

We can now insert Eq. (2.29) in Eq. (2.28) and use integration by parts in space and time, to remove as many derivatives from $\tilde{y}$ as possible:

$$\begin{aligned}\delta_y\mathcal{L}(y,u,\Psi) =& \int_0^T\int_\Omega \left[(y-\hat{y})\right]\tilde{y}\,\mathrm{d}\Omega\mathrm{d}t \\ &- \Bigg[\underbrace{\int_\Omega (\Psi_{\mathrm{St}}\,\tilde{y})|_{t=T}\mathrm{d}\Omega - \int_\Omega (\Psi_{\mathrm{St}}\,\tilde{y})|_{t=0}\mathrm{d}\Omega - \int_0^T\int_\Omega\left[\frac{\partial\Psi_{\mathrm{St}}}{\partial t}\tilde{y}\right]\mathrm{d}\Omega\mathrm{d}t}_{\text{due to temporal int. by parts}}\Bigg] \\ &- \Bigg[\underbrace{-\int_0^T\int_{\partial\Omega}\Psi_{\mathrm{St}}\,\frac{\partial\tilde{y}}{\partial n}\mathrm{d}\Omega\mathrm{d}t + \int_0^T\int_{\partial\Omega}\frac{\partial\Psi_{\mathrm{St}}}{\partial n}\,\tilde{y}\,\mathrm{d}\Omega\mathrm{d}t - \int_0^T\int_\Omega \tilde{y}\Delta\Psi_{\mathrm{St}}\,\mathrm{d}\Omega\mathrm{d}t}_{\text{due to spatial int. by parts (Green's second identity)}}\Bigg] \\ &- \int_0^T\int_{\partial\Omega}\Psi_{\mathrm{BC}}\,\tilde{y}\,\mathrm{d}\Omega\,\mathrm{d}t - \int_\Omega \Psi_{\mathrm{IC}}\,\tilde{y}|_{t=0}\mathrm{d}\Omega = 0. \end{aligned} \tag{2.30}$$

[14] See for example the seminal papers by Jameson [46, 47, 48]

As seen in the third line, one cannot always completely remove all derivatives from the state. The $n$ in the third line is the outward normal to $\partial\Omega$ and the related terms come from applying Green's second identity. We can now finally re-order the terms above,

$$\begin{aligned}\delta_y\mathcal{L}(y,u,\Psi) = &\int_0^T\int_\Omega \tilde{y}\left[\frac{\partial\Psi_{\mathrm{St}}}{\partial t} + \Delta\Psi_{\mathrm{St}} + (y-\hat{y})\right]\,\mathrm{d}\Omega\mathrm{d}t \\ &- \int_0^T\int_{\partial\Omega}\tilde{y}\left[\frac{\partial\Psi_{\mathrm{St}}}{\partial n} + \Psi_{\mathrm{BC}}\right]\,\mathrm{d}\Omega\mathrm{d}t + \int_0^T\int_{\partial\Omega}\frac{\partial\tilde{y}}{\partial n}\,\left[\Psi_{\mathrm{St}}\right]\,\mathrm{d}\Omega\mathrm{d}t \\ &- \int_\Omega \tilde{y}|_{t=T}[\Psi_{\mathrm{St}}|_{t=T}]\,\mathrm{d}\Omega - \int_\Omega \tilde{y}|_{t=0}[\Psi_{\mathrm{IC}} - \Psi_{\mathrm{St}}|_{t=0}]\,\mathrm{d}\Omega = 0, \end{aligned} \tag{2.31}$$

by collecting all terms in five distinct groups: First, a part related to the variation in the state in the space-time domain (row 1). Then, two parts covering integrals in time on the domain border, one with variations in the state and one with variations in its first-order derivative (row 2). Finally, we have two space integrals: One evaluated at the final time, $t = T$, and one at the initial time, $t = 0$ (row 3). The reason for this exact reordering is, that we now for each term have an arbitrary variation multiplied on a bracketed term. Since Eq. (2.31) should be equal to zero it follows, that each of the terms inside the brackets *must* be zero for this to be true for arbitrary variations. Thus, as mentioned in Eq. (2.18) we have arrived at the adjoint equation:

$$-\frac{\partial\Psi_{\mathrm{St}}(x,t)}{\partial t} - \Delta\Psi_{\mathrm{St}}(x,t) = (y(x,t)-\hat{y}), \quad \text{for } (x,t)\in\Omega\times[0,T], \tag{2.32}$$

$$\Psi_{\mathrm{St}}(x,t) = 0, \qquad \text{on } \partial\Omega\times[0,T], \tag{2.33}$$

$$\Psi_{\mathrm{St}}(x,t) = 0, \qquad \text{on } \Omega \text{ at } t = T, \tag{2.34}$$

for the PDE-constraint in Eq. (2.23-2.25). We note that $\Psi_{\mathrm{St}}$ indeed is a complete set of adjoint variables, i.e., for each $y$ we have a dual- or costate- or adjoint variable, $\Psi_{\mathrm{St}}$. We simply gave it the subscript 'St' for 'State equation' to distinguish it from other adjoint variables, $\Psi_{\mathrm{BC}}$ and $\Psi_{\mathrm{IC}}$, used to augment the Lagrangian with boundary conditions (Eq. (2.24)) and the initial condition (Eq. (2.24)), respectively. One can derive a few equations more from Eq. (2.31) but these relate to the determination of $\Psi_{\mathrm{BC}}$ and $\Psi_{\mathrm{IC}}$ and are not needed to compute the $\Psi_{\mathrm{St}}$ variables.

To complete the KKT system we finally write up the optimality condition from Eq. (2.19) by again considering the first variation of the Lagrangian, $\delta_u\mathcal{L}(y,u,\Psi)$, but this time with respect to the controls, $u$:

$$\begin{aligned}\delta_u\mathcal{L}(y,u,\Psi) = 0 \Leftrightarrow \\ \beta u(x,t) - \Psi_{\mathrm{St}}(x,t) = 0, \quad \text{for } (x,t)\in\Omega\times[0,T].\end{aligned} \tag{2.35}$$

Now that the entire KKT system has been derived with state Eq. (2.23-2.25), adjoint Eq. (2.32-2.34) and optimality condition (Eq. 2.35), we highlight a few points of interest:

- The adjoint state Eq. (2.32) (resulting from the first term in Eq. (2.31)) runs *backwards* in time.
- As a result hereof, it does not have an initial condition but a terminal condition in Eq. (2.34) which is derived from the fourth term in (2.31).
- The reversal of time can also be observed when comparing the discretized versions of the state and adjoint equations. To this end, we use the same FEM discretization we used for the state equation to discretize the adjoint equation,

$$\mathcal{K}^T\Psi = \tau\mathcal{M}_0\mathbf{y} + \tau\mathbf{z}_0, \tag{2.36}$$

The visualization of the sparsity pattern of the discretized adjoint equation is seen in Fig. 2.2 where we again have used only four time steps $[\tau_1, \tau_2, \tau_3, \tau_4]$.

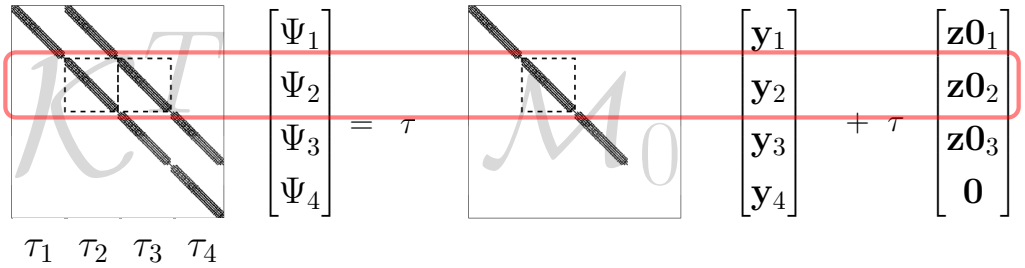



**Figure 2.2:** Visualization of the sparsity pattern of the discretized adjoint Eq. (2.36). The computation of the adjoint variables for the second time step, $\tau_2$, is highlighted in red. As seen, the $\Psi_3$ variable (pertaining to a later time step) is used.

Evidently, the operator[15] in Fig. 2.2 is purely upper triangular whereas the operator in the state Eq. (2.1) was purely lower triangular. The lower triangularity means, that only previous states are used to solve for the next state. The opposite is true in the adjoint equation, where only 'future' adjoint variables are used to compute present variables.

- Considering the time reversal of the adjoint Eq. (2.32), it is now evident that the KKT system is fully coupled (we cannot march in time). However, one can still opt to solve the system without an optimization algorithm by solving the entire coupled system for all variables, $y, \Psi_{\mathrm{St}}$, and $u$, at all time steps, simultaneously (in one shot). Hence, the name *oneshot* method. We will, however, choose a gradient-based optimization strategy as explained below.

We could of course have taken an example problem closer to the overall topic of our project. Extending the PDE complexity to Stokes could be one option. Indeed, the very first application of the adjoint method to fluid mechanics was this exact problem. We

[15] i.e., the left hand side matrix: $\mathcal{K}^T$

refer readers to the excellent seminal paper by Pironneau for further details [95]. Taking the unsteady Reynolds-averaged Navier–Stokes (URANS) equations as our example PDE would also have been very relevant. However, while it is certainly possible to derive URANS adjoint equations, the derivation is rather lengthy and therefore outside the scope of the present introduction. A good starting point for interested readers is given elsewhere [35, chapter 6]. Still, given that the PDE we chose is time-dependent, the reader should easily be able to appreciate the immense consequences the oneshot method would incur on the memory-front had the PDE-constraint been made up by an industrial scale URANS CFD solver treating $\mathcal{O}(10^7)$ variables *at each time step*!

The approach we will take instead of the oneshot method is that of a gradient-based optimization method. We do so, even though we only work with a steady-state RANS model throughout this work. In general, there are three overall steps we handle:

1. Solve the state equations.
2. Solve the adjoint equations.
3. Compute the gradient (Eq. 2.12) and pass the gradient to an optimizer, which takes a step in the design space.

This three-step loop will continue until some converge threshold for the optimization algorithm is met, and we have found an optimized shape.

Finally, a remark is in order to clarify that some variants of the oneshot method *will* in fact include the above mentioned three steps. Typically, these approaches would first complete a few CFD iterations for the state variable, then complete a few adjoint CFD iterations before finally taking a step in the design space to update the geometry (See, e.g., [124]). We have, however, in the above used the 'oneshot' term to refer to an "approach that does not involve an optimization iteration" [35, Sec. 2.8]. A further discussion of the 'oneshot' term is outside the scope of the present introduction.

## 2.3 The continuous and the discrete approach

In the previous section where we derived the KKT system using the continuous approach we chose the exact same FEM discretization for the state equation (Eq. 2.23-2.25), the adjoint equation (Eq. 2.32-2.34), and the optimality condition (Eq. 2.35). Each of the three equations could, however, be discretized in whichever way we would prefer. It was only out of convenience, that we chose the same linear FEM discretization. This freedom in choice of discretization can be seen in Fig 2.3 visualizing the continuous approach.

Turning to the discrete approach, we note that it overall offers less freedom in discretization - or at least, one cannot independently decide on the discretization of the adjoint equation. Here, we begin by discretizing the PDE-constraint and the cost functional. This discretization completely determines the discretization of the adjoint equation. A diagram of the discrete approach is therefore simpler as seen in Fig. 2.4.

These two approaches form a great divide among practitioners and each approach presents distinct challenges to overcome. Before solving an optimization problem the first thing to do is therefore to choose an approach: Either the continuous approach or

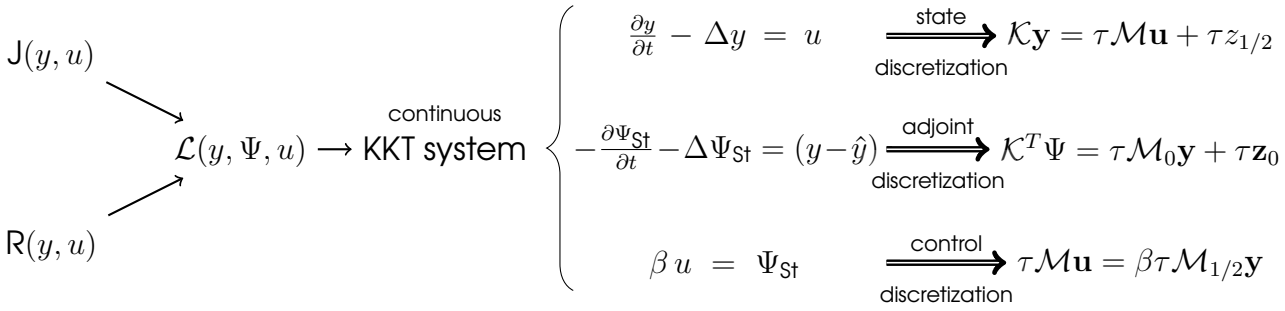



**Figure 2.3:** Illustration of the continuous approach where the cost functional, $J$, and the PDE-constraint, $R$, are combined in the Lagrangian, $\mathcal{L}$, which in turn is linearized. First at the very end, the equations are discretized.

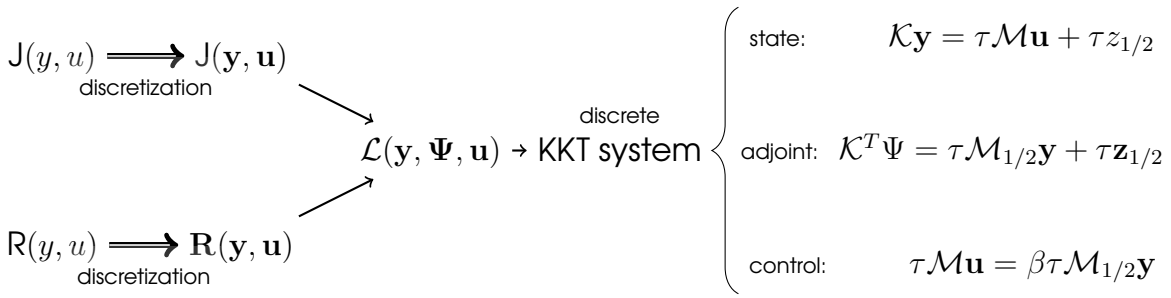



**Figure 2.4:** Illustration of the discrete approach where cost functional, $J$, and PDE-constraint, $R$, are discretized at the very beginning. The discrete $J$ and $\mathbf{R}$ are then combined to construct a discrete Lagrangian, $\mathcal{L}$. Finally, a linearization of the discrete Lagrangian leads to the equations for state-, adjoint-, and control variables.

the discrete approach. For completeness we should mention that with time also hybrid approaches (e.g., [118]) have emerged where one combines the discrete and the continuous approach to overcome said challenges.

The choice of approach (continuous or discrete) will often result in different KKT systems which can be seen in Fig. 2.5. The reader will notice a slight difference in the adjoint equation of the two KKT systems in Fig. 2.5. It is related to the terminal condition of the adjoint equation. It is in other words not a given that a linearization followed by a discretization (the continuous approach) yields the same result as a discretization followed by a linearization (the discrete approach). One can, however, make the two operations (linearization and discretization) commute in the above example if the discretization is chosen more carefully. This topic is studied in more depth elsewhere [97, 92].

To better grasp the memory related consequences of the oneshot approach we highlighted the operator linearization of the state equation's first time step for the continuous approach (marked with a black dashed outline). As seen, even for just four time steps

it is a fraction of the total KKT system, yet for industrial scale applications it is the storage of such a single linearization matrix that can hardly be achieved. Storing the entire KKT system is in this light truly unimaginable.

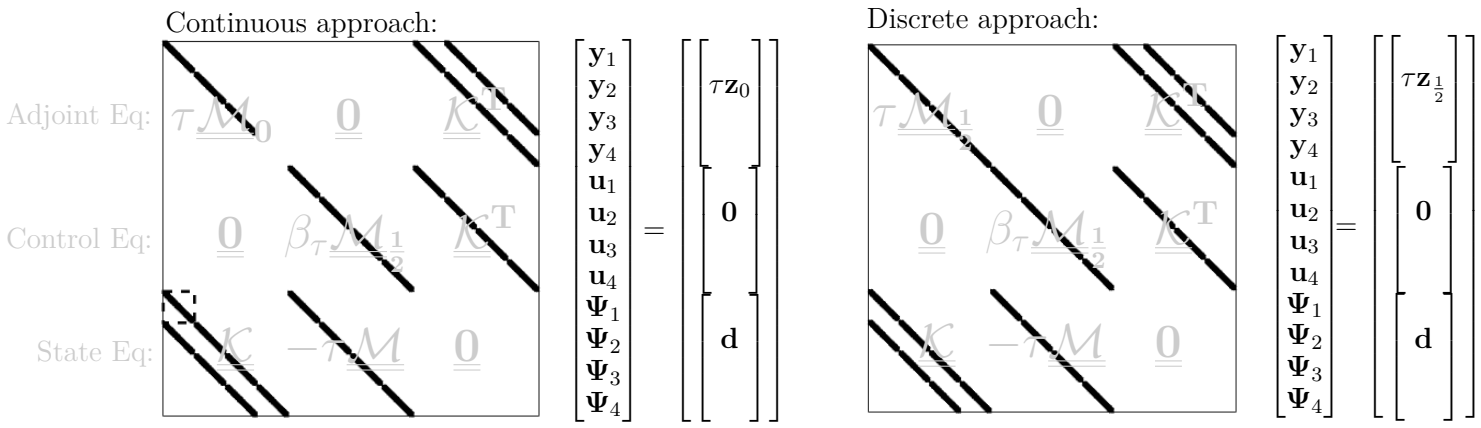



**Figure 2.5:** Visual comparison between KKT systems from the continuous approach (left) and the discrete approach (right).

Fig. 2.6 shows the process flow for the two approaches. As seen, the right most vertical process arrow is dashed since the two basic operations, 'linearization' and 'discretization', are not guaranteed to commute, i.e., to result in the exact same final discrete equation.

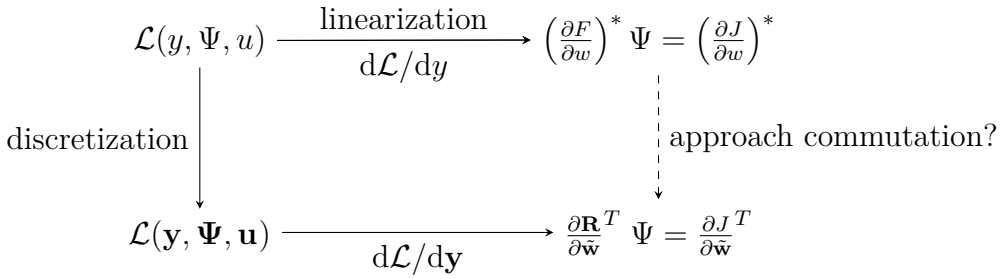



**Figure 2.6:** Visualization of the process flow for the continuous and the discrete approach. The continuous approach starts with a linearization followed by a discretization. The discrete approach starts with a discretization followed by a linearization. The adjoint operator, $(\ )^*$, is simply the matrix conjugate transpose for finite-dimensional cases [35, p. 16].

One can find expert reviews in favor of the discrete approach [31] and others in favor of the continuous approach [99]. One boon offered by the discrete approach that is often pointed to, is the complete consistency in terms of gradient precision [31]. Given that $J$ is discretized right at the very beginning of the discrete approach (Fig. 2.4) the discrete cost function gradient will be exact. In the continuous approach we will on the other hand obtain a discrete approximation to the continuous cost functional. This will not perfectly match the gradient based on the discrete approach and thus, a minor inconsistency is

introduced. This is of course not that relevant in the context of the oneshot method but when using gradient based approaches to converge towards a minimum it can incur reduced convergence. Indeed, one of the earliest comparisons [113] we have between the two approaches even report of optimization failures due to this inconsistency. Thus, it is certainly worth mentioning.

We have chosen the discrete approach since we greatly value the following features:

- a gradient consistency to machine accuracy,
- a possibility to use automatic differentiation software, and
- a manageable setup maintenance that is easily extended.

Our choice of the discrete approach means that the reader will find no lengthy derivation of adjoint terms in the rest of this thesis. Indeed, we do not even have to worry about the discretization of the PDE since an in-house CFD solver already has been developed. Instead, all focus will be on the final part of the discrete approach, i.e., how to efficiently linearize and transpose the RANS equations and how the resulting linear system can be solved in an efficient manner.

# CHAPTER 3
# State-of-the-art high-fidelity shape optimization of wind turbine blades

This chapter will conclude the first part of the project where we aim to i) give an overall introduction to the adjoint method (Chapter 2), ii) give an overview of the relevant literature, and iii) identify state-of-the-art rotor studies within wind energy.

## 3.1 Literature review

There are three levels of literature review relevant for the current project. First, an overall historical review (Sec. 3.1.1) where we mention some of the most known, seminal papers that brought the research field about. This is not done out of historical interest in itself. Rather, we use this review to identify overall trends across research communities that point in which direction the communities are heading, and types of further developments one should consider for future work.

Secondly, we narrow the scope in to high-fidelity shape optimization applied within wind energy research. Here, we examine the high-fidelity shape optimization efforts found within wind energy applications.

The third and final level of literature review (Sec. 4.1) is highly specific to the development of discrete adjoint solvers for the SIMPLE algorithm and we therefore bring it in part II of the thesis, which is concerned with the development of a discrete adjoint solver. We have yet to find this niche-literature exhaustively treated and will to this end give an overview in Chapter 4.

### 3.1.1 A historical perspective

We now introduce some of the early, seminal papers that helped form the research field in the final part of the previous century. Literature reviews on the broader field of adjoint methods have been done numerous times. Readers particularly interested in further reading for this historical part can advantageously consult: [46, 83, 94, 55].

#### 3.1.1.1 Early advances made with the continuous approach

Professor Anthony Jameson is one the pioneers within adjoint-based shape optimization. His early work in 1988 on formulating the design problem as a control problem [46] mark the beginning of a new era where the study of the adjoint method intensified –

especially within aerospace. The paper starts with a concise recap of the preceding period starting from Sir James Lighthill's 1945 paper [65] on 2-D aerodynamic design using conformal mapping and an incompressible fluid model. Jameson then shows via several examples that one can indeed formulate feasible aerodynamic design problems via the derivation of the adjoint equation. This is shown for a 3-D wing using the inviscid Euler equations. The final example is an inverse design case where a target pressure distribution is sought by deforming the wing surface. Jameson points out that Lighthill's work was extended by Mcfadden [74] to encompass compressible fluids and Jameson sees his own work [46, Sec. 2], as the generalization of these methods. He also compares his approach (of viewing the design problem as a control problem) to that by Bristeau et al. [17] who use FEM to solve a least square problem with a potential flow model as the PDE-constraint. Impressively enough, they present 3-D computations of an entire aircraft configuration [17, Sec. 6.4] and talk of 'industrial' interest. Here, it should be noted that one of the authors, i.e., Prof. O. Pironneau, already had published work where he derived the first-order necessary optimality conditions (the KKT system) for the minimum drag problem in Stokes flow back in 1973 [95]. This is – to the best of our knowledge – the first application of control theory within fluid mechanics. While Pironneau did indeed successfully derive the KKT system, he did not have a subroutine to handle the Stokes flow in an unbounded domain and ends his conclusion with an open invitation to fellow researchers to deal with this task.

Returning to the 1988 paper by Jameson we note, that save for potential flow, Jameson's paper also includes design studies using 2-D and 3-D Euler equations to model compressible flow with the purpose of exemplifying that one can indeed formulate aerodynamic design problems *feasibly* in the context of control theory. However, the work is (like [95]) entirely focused on the continuous derivation and formulation of the final problem. No numerical implementations are presented, which he identifies as future work. Whilst Jameson certainly provides ample numerical examples in later works (see, e.g., [47] for a CFD implementation), we point to the curiosity that interested readers already in McFadden's work have a rich opportunity to inspect the Fortran77 code used at that time (although readability is less than perfect) [74, pp. 105-164]. Another early numerical optimization is that by Hicks, Murman, and Vanderplaats [42], but the gradients are found with finite differences and not an adjoint method.

Especially the work by Pironneau [95] and Jameson [46] seem ubiquitous. Readers will be hard pressed to find a historical survey not mentioning these two papers.

In the period after Jameson's 1988 paper and until the turn of the century several research groups successfully implemented adjoint solvers where some groups favored the discrete approach and other groups favored the continuous approach. See [31] or [48] for further details on these early advances. We will loosely follow the path of Jameson due to his important contributions. Jameson solidified his work on the 2-D Euler equations together with Reuther [98]. They present two design implementations, one based on an analytic mapping of the vertices, and another where the vertices can be displaced into 'arbitrary meshes' using a projection method. We focus on the latter method, as it resembles our own framework more closely. As pointed out, the modelling of the inviscid shocks is a clear improvement to the early work relying on potential flow

models. This progression in fluid model complexity has continued to this day, where state-of-the-art optimization frameworks now handle the full transient Navier–Stokes model with turbulence and transition models *and* couples disciplines (e.g., aerostructural) in a true multidisciplinary framework. Interestingly, Reuther and Jameson replaced the mesh deformation method relying on successive mesh generations with a projection method where volume mesh points are deformed with an amount proportional to their arc length distance from the surface point. This new method is directly extendable to 3-D. Note, that the mesh necessarily is structured, so that each vertex lies on a uniquely defined mesh line that does not merge or split between its origin at the surface and its terminal point on the outer boundary. Using 50 design variables they compare gradient computation time between the adjoint method and a finite difference gradient. The latter, is in the paper called a 'brute force' method, but the method is as always to perturb design variables, one at a time, and compute new mesh and flow variables to obtain a corresponding change in the cost function. As one would expect, they find that the computation time using the adjoint method is reduced (with a factor of 6.8). However, most surprisingly, they state that the adjoint gradient is *less* accurate. They then proceed to seven test cases using the 'arbitrary mesh' method. Here, case five is quite interesting. They use a potential flow model to generate a target pressure distribution. Said distribution can of course not be exactly reproduced when using Euler equations. However, the optimizer comes quite close. The interesting part is that the shapes generating the closely matching pressure profiles differ considerably more than for the other test cases. This discrepancy is due to the model fidelity difference between the potential and Euler fluids. This finding stresses the importance of the ever-occurring transition towards higher fidelities. The presented optimizations range from 3-100 in design cycles.

As promised, the presented mesh perturbation technique is used in a complex 3-D aircraft configuration in a later work [99]. Here, two design cases are shown where a wing is redesigned for a business jet using a multiblock mesh with 750.000 cells. First, they check the adjoint gradients against finite differences. They state that the agreement is 'excellent' (no quantification), and that the remaining discrepancy can be removed by converging the problem further if it is not owed to the inherent inaccuracy of the continuous approach due to a discretization mismatch. However, they point to the fact that they can estimate the gradient computation time to be reduced by (at least) a factor of 27. A reduction factor of 27 is of course much better than the above reported 6.8, but now we also consider a 3-D case. In any case, it is evident from these few historical examples that the adjoint method offers a considerable speedup compared to the finite difference method.

In an attempt to continuously increase the model fidelity, Jameson, Martinelli, and Pierce extended previous work to encompass the compressible Navier–Stokes equations [48]. Here, it is noteworthy that the change from Euler to Navier–Stokes equations incurs at least an order of magnitude greater computational effort. The number of design cycles also rise from the range 10-20 for Euler equations [47] to 20-40 cycles for Navier–Stokes equations. This trend is in line with the number of design cycles we recently reported ranging from 30-120 [71, Fig. 17] for a RANS fluid model complemented

by the Spalart–Allmaras (SA) turbulence model [117]. Of course, the number of design cycles depend drastically on the optimizer settings and strategy, but the overall trend of increased number of design cycles for increased problem complexity is certainly evident. Interestingly, although the work is impressive, Jameson, Martinelli, and Pierce talk of it as an "intermediate step toward the eventual goal of full multidisciplinary optimal design" [48, p. 215]. Thus, as more complex problems are being mastered, the need for including more than a single discipline surfaces. To reduce the high computational cost they split the design process in two stages. First stage is based on the Euler equations and results in an intermediate design based on 60 design cycles. This reduces the amount of necessary design cycles to 10 in the second stage, which is based on the Navier–Stokes equations. This trick of mixing fidelities to increase the overall performance is yet another trend we can trace up to present-day in the form of multifidelity frameworks. Calculations were also performed in parallel to raise the mesh resolution to 1.8 million mesh points. This calculation (starting from a preliminary result) needed 20 design cycles to converge. Impressively enough, the results showed such promise that the method was evaluated for industrial use.

The study of Jameson's early works provided several indications of overall research directions:

i) A trend towards higher fidelities (potential → Euler → Navier–Stokes).

ii) As model complexity increases the need for a finer resolved meshes and high performance computing increases.

iii) A push for multifidelity and for multidisciplinary models arise as the single discipline (aero) is being mastered.

Given, that Jameson certainly is an advocate for the continuous approach it seems only fair to at least touch upon a few important works from researchers advocating the discrete approach.

#### 3.1.1.2 Early advances made with the discrete approach

Despite the increasing activity in aerodynamic design studies based on the adjoint method in the late 1990s, two experts in a contemporary review state:

> "Considering the importance of design to aeronautical engineering, and indeed to all of engineering, it is perhaps surprising that the development of adjoint CFD codes has not been more rapid in the decade since Jameson's first papers appeared."
>
> Giles and Pierce [31, pp. 393-394]

The explanation given by Giles and Pierce to the above is, that the complexity of both the continuous and the discrete approach is considerable. For the continuous approach (favored by Jameson) there is a considerable effort in mathematically deriving the adjoint equation and for the discrete approach the development of the adjoint CFD code can be

extremely complex. They also point out that the discrete approach has a salient feature, namely the use of algorithmic differentiation[1] [29, 34] to generate parts of the discrete CFD code. They specifically point to the possibility of handling the transpose-matrix-vector product, $[\partial \mathbf{R}/\partial \tilde{\mathbf{w}}]^T \mathbf{\Psi}$, by leveraging algorithmic differentiation. However, for the $\partial \mathbf{R}/\partial \mathbf{x}$ matrix they find that the complex-step method is much simpler. Finally, they also comment on the variety within the discrete adjoint approaches, i.e., some explicitly saves the state Jacobian, $\partial \mathbf{R}/\partial \tilde{\mathbf{w}}$, to disc, whilst others prefer to circumvent this memory overhead by forming the dot product to which it is used, $[\partial \mathbf{R}/\partial \tilde{\mathbf{w}}]^T \mathbf{\Psi}$, directly. As indicated in the remark by Giles and Pierce, there are numerous ways to implement the linearization in the discrete approach. Below, we will indeed mention in broad terms when researchers favor, e.g., explicit storage of the state Jacobian, $\partial \mathbf{R}/\partial \tilde{\mathbf{w}}$, or say, the fixed-point iteration method but the detailed explanation of these subtleties is postponed until the literature review particularly directed towards discrete adjoint solvers (Sec. 4.1).

One of the earliest discrete adjoint implementations was done in a comparison to the continuous approach by Shubin and Frank [113]. Using the Euler equations as a fluid model in a 1-D duct where the cross-sectional area can be deformed they solve an inverse problem targeting a predetermined velocity profile. They find that the implicitly erroneous continuous gradient sometimes deteriorates or completely hampers the convergence of the optimization problem. Also Prof. Baysal and his group made advances in the early '90s within shape optimization. In the same year, Baysal and Eleshaky report of a discrete adjoint solver for the 2-D compressible Euler equations. Here, gradients from both a direct (i.e., a tangent-linear) solver and an adjoint solver are compared to finite difference gradients. The analytical gradients are reported to be 'identical' whereas the finite difference gradients where less accurate. In 1993 they present an improved framework with a third-order accurate treatment of the Euler equations [13]. All partial matrices in the discrete adjoint solver are derived by hand. Using a naive parameterization where every single surface vertex is a design variable they can repeatedly optimize a ramp shape on a jet for maximum thrust. Every optimization has a different starting point but they show that they converge to the same optimized shape every time. Perhaps more interesting than the optimization results is their comment on memory consumption, which is a notorious difficulty for the discrete approach. Given, that they work on a structured mesh, they can meticulously identify all non-zero entries in the state Jacobian, $\partial \mathbf{R}/\partial \tilde{\mathbf{w}}$. In short, the band-structure can be completely determined based on the computational stencil. This allows for a reduced memory storage in sparse form where only the non-zero entries a saved. Many of these early discrete efforts were entirely derived by hand. However, use of forward [15] and reverse [79] mode algorithmic differentiation was reported as early as 1992 and 1997, respectively.

Also 3-D work was carried out using the discrete approach. James Newman et al. carried out a 3-D shape optimization maximizing the lift-to-drag ratio of an aircraft configuration using 10 design variables [45]. Here, special attention is on using exact state Jacobian matrix-vector products to reduce memory requirements. Interestingly, they obtain their grid sensitivities, $\partial \mathbf{X}_{\mathrm{VOL}}/\partial \mathbf{X}_{\mathrm{DV}}$, using the algorithmic differentiation

[1] Also known as automatic differentiation

tool ADIFOR [15]. Notice, that the opposite was preferred by Giles and Pierce [31] who advocate the use of complex-step for the grid sensitivities. Similar to the above mentioned efforts using a continuous approach, the gradient precision is only compared to central finite difference gradients, and as such cannot be verified to machine precision. However, the gradients matched to 3-4 significant digits depending on the case. Note, that they do not use an adjoint solver, but a 'forward' solver since they compute the tangent-linear equation instead of the adjoint equation. This is of course perfectly justifiable, and as explained in the introduction (Chapter 2) it comes down to whether one has more design variables than functions of interest. Furthermore, we point out that it is only the grid sensitivities they obtain with ADIFOR. The remaining partial derivative matrices are constructed by hand based on the discretization of the flow solver. Impressively, by avoiding the storage of the state Jacobian their discrete implementation uses about the same amount of memory as the CFD solver. However, the 3-D case was stopped after just 3 design cycles which makes conjectures on the final design irrelevant. They mention that their framework has been extended to incorporate a FEM code to allow for aerostructural analysis in future work. Again, we recognize the push for multidisciplinary design optimization (MDO) as frameworks mature.

Another 3-D shape optimization study is that by Elliott and Peraire [27]. It also relies on unstructured meshing but it is evident that the unstructured mesh approach yet cannot rival the capability of structured grids to accurately resolve the boundary layer. As pointed out by the authors, the exact same relaxation algorithm is employed to solve the adjoint equation as they use to solve the flow problem. This type of discrete adjoint solver is known as a 'fixed-point' adjoint solver which will be explained further in Chapter 4. Interestingly, Elliott and Peraire report that saving the state Jacobian to disk results in a speedup factor of 4. This operation was strained in serial, but for parallel applications, only local sub domains are saved for each CPU. Thus, distributing the memory cost makes it feasible. Using a 860.000 cell mesh they achieve an optimized shape in only 7 design cycles for the full aircraft configuration. However, due to the skewness in the mesh, they had to recalculate the grid after the fourth iteration. The optimized shape resulted in a recovery of the target pressure distribution as is desired for inverse problems. Based on the 3-D case, they deemed their implementation to be both accurate and efficient. Elliott and Peraire also remark, that the boundary conditions present no trouble for the discrete approach since the boundary conditions already are included in the discretization of the primal solver [27, p. 4]. Boundary conditions are on the other hand notoriously tricky for the continuous approach. A dissertation by Nadarajah a few years later also considers the discrete approach as a possible remedy for these boundary condition issues, albeit it is acknowledged that it may warrant more work than what is needed in the continuous approach:

> "The cost of deriving the discrete adjoint is greater, but it may provide a route to improving the boundary conditions for the continuous adjoint for viscous flows."
>
> Nadarajah [82, p. 228]

As a final example of the discrete adjoint achievements in the early period we point to the impressive aerodynamic optimization by Nielsen and Anderson using a RANS model coupled to an SA turbulence model on an unstructured grid [84]. We pick this work for several reasons. First of all they show results for both a compressible solver and an incompressible flow solver (artificial compressibility). A second reason is that they show a rather thorough gradient 'verification'. We do not entirely view this as a verification since they use finite differences and not the complex-step method, but it is rather thorough compared to contemporary works. Both for the compressible and incompressible adjoint solvers they show a good match with finite difference gradients obtained with a step size of $h = 1 \cdot 10^{-5}$. Specifically, they show agreement on 3-5 significant digits for meshes of 4901 vertices and 16.391 vertices in 2-D and 3-D, respectively. They also assess the frozen turbulence assumption and point out, that even the wrong sign of the gradient has been observed. Clearly, this assumption (where the turbulence is not treated in the adjoint model) has grave consequences. Given that they use a discrete approach it is noteworthy, that they resort to differentiation by hand compared to leveraging algorithmic differentiation techniques. Their FFD-based geometry parameterization [106] is similar in spirit to the method developed in our project. In the conclusion they point to multipoint optimization as an interesting means of attaining a more robust result that also performs well off-design. Their final 3-D case is a redesign of an ONERA M6 wing using 4 design variables on a mesh with 62.360 nodes. Using 10 design cycles the drag coefficient is reduced with 8 %. Although mesh and parameterization choice is too coarse, the overall picture of a maturing framework is evident.

To briefly recap the early period from Jameson's '88 paper and until the new millennium we have seen both for the continuous and the discrete approaches some trends, namely:

- a shift towards higher fidelities (potential, Euler, RANS),
- an increasing need for parallel computing to facilitate large scale optimization,
- an increased interest in including other disciplines (e.g., aerostructural),
- explorations of multifidelity frameworks to reduce computation time, and
- mentioning of multipoint optimizations to obtain more robust designs.

All the mentioned trends should be expected to repeat themselves in the related community of wind energy research. Indeed, our recent publication [71] is both high-fidelity, multipoint and parallelized to run on several hundred processors. It is, however, an aero-only shape optimization, but we did identify the incorporation of a high-fidelity structural model as future work, and we fully expect this trend to be observable in the near future within wind energy research.

These trends in the aerospace community only amplify in the period from 2000 to 2009 (the year of the first ever contribution to high-fidelity shape optimization within wind energy [102]). This is true for the aerospace community, but likewise for related research communities, such as the automotive, marine and biomedical research communities where the observed trends in the aerospace community spill over. Of course, one can find early

advances in these communities as well, but one can safely view the aerospace community as one of the absolute leaders within the art of adjoint shape optimization. This is easily seen when comparing to wind energy research where the first contribution emerged in 2009 – more than 30 years later than Jameson's '88 paper. Not to mention, that the research was in fact carried by aerospace researchers applying their tools to a wind energy application [102].

#### 3.1.1.3 Advancements in the algorithmic differentiation community at the turn of the century

The historical literature review in Sec. 3.1.1 showed that advances for the continuous approach started in 1988, and that the discrete approach followed suite not long after [113]. In the period from 2000 to 2009 we point to the advances made specifically within algorithmic differentiation communities that enabled a more systematic use of algorithmic differentiation on industrial scale CFD solvers[2]. In short, the maturity of algorithmic differentiation tools increased in these years. Heimbach, Hill, and Giering tackled an unprecedented transient optimization problem in 2002 within oceanographic research spanning 9 years with hourly time steps [40]. Here, a three-level checkpointing algorithm rendered the problem feasible. The primal and adjoint codes contained about 100 thousand code lines. Of other important contributions advancing the algorithmic differentiation tools we could point to Courty et al. [20] and Hascoët, Vázquez, and Dervieux [37]. Both works use the algorithmic differentiation tool Tapenade [36] and refine its checkpointing techniques while emphasizing a selective use of algorithmic differentiation to parts of the source code. They both develop fixed-point solvers to keep memory consumption at a minimum. By now, there are numerous algorithmic differentiation tools available for any desired programming language. Interested readers can consult the community portal[3] for automatic differentiation for an overview of available tools, applications, and related research groups.

In general, the selective approach when using algorithmic differentiation tools became very popular and it was shown that minor changes could render algorithmic differentiation produced code competitive to hand derived code [80, 30]. Here, we bring the attention to an important point also made elsewhere [49][5, p. 99] namely that the selective approach with manual assembly can be more easily applied to the compressible finite volume method formulation than to the segregated incompressible finite volume method formulation. Indeed, efforts showcasing selectively applied algorithmic differentiation tools to the segregated SIMPLE algorithm are very hard to find before the paper by Jones, Müller, and Christakopoulos [50]. From then on, works on the SIMPLE algorithm slowly seem to emerge. Some of these focused on the black-box approach [119] and others focused on less memory consuming methods with selective algorithmic differentiation usage which allow for industrial scale applications [49, 5]. We revisit this track of discrete

[2]Interested readers can also find an excellent literature review focused on the algorithmic differentiation assisted discrete adjoint solvers in a recent PhD dissertation [5, chapter 4].

[3]`http://www.autodiff.org/`,(last access: 11 July 2019)

adjoint efforts specifically tailored for the SIMPLE algorithm in Chapter 4 where we also outline our chosen developmental strategy.

## 3.1.2 Literature within wind energy

We now turn to the literature on high-fidelity CFD-based shape optimization within wind energy. While we focus on works utilizing the adjoint method below, we will also mention high-fidelity works using gradient-free methods for completeness. The below review is an updated excerpt of the literature review published in a recent paper [71] together with a comprehensive multipoint shape optimization study we carried out in corporation with the MDOLab at the University of Michigan. Readers can find the paper in the appendix (Sec. A).

### 3.1.2.1 High-fidelity shape optimization efforts within wind energy research without an adjoint method

As mentioned in the introduction (Chapter 1) we associate "high-fidelity" shape optimization with works where the aerodynamics are modeled using CFD. More precisely, we do not include works based on potential flow and Euler equations when reviewing advances within wind energy, but limit the discussion to focus on works based on RANS equations or even higher-fidelity models such as Detached Eddy Simulation (DES) or Large Eddy Simulation (LES). The 2-D examples on high-fidelity shape optimization but with no adjoint method mentioned in the review are:

Kwon, You, and Kwon (2012) [60] who present a lift-over-drag ratio optimization using 2-D RANS coupled with models for turbulence and transition. They use a gradient-based optimization algorithm[4] and approximate the gradients using finite differences. The results are evaluated in 3-D by re-modeling the rotor[5] using the 2-D optimization results. Using 9 design variables they achieve a 11% increase in torque.

Ribeiro, Awruch, and Gomes (2012) [101] use 9 design variables to carry out multiobjective optimization with a genetic algorithm. They use an incompressible RANS formulation combined with the SA turbulence model. By introducing a neural network as a surrogate model they achieve a speedup of $\sim 50\%$ while still obtaining similar results.

Zahle et al. (2014) [136] use 21 design variables and a gradient-based sequential linear programming optimization algorithm to design a new airfoil series. To reduce computation time they use a lower-fidelity panel code, XFOIL, in combination with the 2-D RANS flow solver, EllipSys2D. A subsequent wind tunnel testing is presented and the developed airfoils are shown to outperform equivalent FFD-W3 airfoils.

Liang and Li (2018) [64] use 2 design variables to carry out a multipoint shape optimization of a NACA0015 airfoil spanning several angles of attack. The objective

[4]Modified method of feasible directions

[5]NREL Phase VI turbine

function is to maximize the tangential force coefficient to improve a vertical axis wind turbine (VAWT) using a gradient-free method. A subsequent 3-D evaluation show that the VAWT based on the optimized airfoil has a $\sim 7\%$ increase in power coefficient compared to a VAWT based on the NACA0015 airfoils.

As the reader will notice, even in 2-D it is popular to reduce the computation cost by covering multi fidelities. Ribeiro, Awruch, and Gomes [101] do so by way of a surrogate approach whereas Zahle et al. [136] do so by using a panel code in combination with a RANS code. Another thing to notice is the rather few amount of design variables spanning from 2 design variables [64] to 21 design variables [136]. Notice that we find both examples of gradient-free and gradient-based methods in the 2-D efforts above, but they all struggle to handle more than a few dozen design variables even though we are not in 3-D yet.

We now focus on efforts mentioned in the published review carried out in 3-D but without an adjoint method.

Vucina, Marinic-Kragic, and Milas (2016 [128]) use a gradient-free genetic algorithm optimizer to improve the performance of the NREL Phase VI rotor. Several cases are presented and they use up to 25 design variables. The chosen mesh size (not stated) results in about 10 min computation time per CFD simulation. Objective functions such as a maximization of annual energy production are used. They do not state the actual optimization result but find that the presented framework has proven its functionality and robustness.

Elfarra, Sezer-Uzol, and Akmandor (2014) [26] use 24 RANS CFD evaluations to train an artificial neural network which in turn is used to carry out gradient-free optimization through a genetic algorithm. The multipoint winglet optimization is based on 2 design variables and three wind speeds resulting in a 9% increase in power.

Zahle et al. (2018) [135] are the final high-fidelity shape optimization effort we mention that does not use an adjoint method. Just as in the previous effort by Elfarra, Sezer-Uzol, and Akmandor [26] it is a winglet optimization, but this time a gradient-based method is used. They do, however, also reduce the computational effort by introducing a surrogate model. Using 12 design variables they increase the power production with 2.6% while satisfying constraints on loads. The flow solver used in this work, EllipSys3D, is also the flow solver we work with in the present work. It will be presented in more detail in Chapter 6.

Again, we sum up the mentioned efforts by noting that only a moderate amount of design variables seems feasible in 3-D ranging from 2 design variables [26] to 25 design variables [128]. Furthermore, we see that 3-D efforts just like 2-D efforts use the initial CFD results to train simplifying models in order to reduce computation time.

We now turn to adjoint-based efforts. Here, it will be evident that the number of design variables rise with about one order of magnitude to a few hundred design variables. This is, however, not due to a limitation in the adjoint method. As many 720 design variables are reported in one of the smaller 2-D efforts [108]. Such a high number of

design variables is typically due to the fact that every single surface mesh point is used as a design variable. This is, however, rarely advantageous since it is far too easy to obtain negative mesh volumes if the deformation procedure is not regulated. In other words, one can with the adjoint method (in theory) introduce as many design variables as needed but typically a few hundred will suffice for rotor design studies.

#### 3.1.2.2 High-fidelity shape optimization efforts within wind energy research using the adjoint method

To give an overview of all related works we found within high-fidelity wind energy research, we start this section by bringing a table with an overview of these efforts (Tab. 3.1). The table is an updated version of the overview found in [71, Tab. 1]. In Tab. 3.1 below we added recently published 2-D effort by Kaminsky and Ekici [52] and a very important rotor study by Nielsen and Diskin [86], which we failed to include in the original review. This negligence is hereby corrected.

**Table 3.1:** High-fidelity shape optimization efforts within wind energy.

| Year | Reference | Turbine[g)] | Adjoint | Dim. | Re[f)] | Mesh size[a)] | Design variables | Iterations[b)] |
|---|---|---|---|---|---|---|---|---|
| 2009 | Ritlop and Nadarajah [102] | – | Discrete | 2-D | $2.0 \cdot 10^6$ | $3.3 \cdot 10^4$ | 385 | 100–200 |
| 2011 | Khayatzadeh and Nadarajah [57] | – | Discrete | 2-D | $2.1 \cdot 10^6$ | $1.3 \cdot 10^5$ | 385 | – |
| 2014 | Schramm, Stoevesandt, and Peinke [108] | – | Continuous | 2-D | $3.0 \cdot 10^6$ | $5.5 \cdot 10^4$ | 720 | – |
| 2016 | Schramm, Stoevesandt, and Peinke [111] | – | Continuous | 2-D | $7.9 \cdot 10^6$ | – | 480 | – |
| 2016 | Barrett and Ning [10] | – | Continuous | 2-D | $1.0 \cdot 10^6$ | $1.4 \cdot 10^4$ | 10–22 [Table 2] | – |
| 2017 | Vorspel et al. [126] | – | Continuous | 2-D | $5.0 \cdot 10^4$ | $5.0 \cdot 10^4$ | 2–364 [Table 1] | – |
| 2018 | Schramm, Stoevesandt, and Peinke [110] | – | Continuous | 2-D | $2.0 \cdot 10^6$ | $2.1 \cdot 10^5$ | 20–50 | 0–30 [Fig. 5,7] |
| 2018 | Barrett and Ning [11] | – | Continuous | 2-D | $1.0 \cdot 10^6$ | $1.4 \cdot 10^4$ | 10–68 [Table 1] | – |
| 2019 | Kaminsky and Ekici [52] | – | Discrete | 2-D | $2.0 \cdot 10^6$ | $3.2 \cdot 10^4$ | 30 | 25 |
| 2012 | Nielsen and Diskin [85, 86] | NREL Phase VI | Discrete | 3-D | $1.0 \cdot 10^6$ | $1.4 \cdot 10^7$ [h)] | 76 | 8 [Fig. 8] |
| 2013 | Economon, Palacios, and Alonso [25] | – | Continuous | 2-D | $1.0 \cdot 10^3$ | $3.2 \cdot 10^4$ | 50 | 10 |
| | | NREL Phase VI | | 3-D | $1.0 \cdot 10^6$ | $7.9 \cdot 10^6$ | 84 | 3 |
| 2016 | Vorspel et al. [127] | – | Continuous | 2-D | $5.0 \cdot 10^4$ | – | 2 | 30 [Fig. 3] |
| | | – | | 3-D | $1.2 \cdot 10^6$ | $2.4 \cdot 10^6$ | – | <8 [Fig. 6] |
| 2017 | Dhert, Ashuri, and Martins [22] | NREL Phase VI | Discrete | 3-D | $1.0 \cdot 10^6$ | $2.6 \cdot 10^6$ [d)] | 1–252 | 9–23 |
| 2018 | Vorspel, Stoevesandt, and Peinke [125] | NREL Phase VI | Continuous | 3-D | $1.0 \cdot 10^6$ | $5.2 \cdot 10^6$ [e)] | 5–9 | <8 [Fig. 5] |
| 2019 | Tsiakas et al. [122] | MEXICO | Continuous | 3-D | $1.0 \cdot 10^6$ | $2.5 \cdot 10^6$ [c)] | 135 | 10 [Fig. 4] |
| 2019 | Madsen et al. [71] | IEA | Discrete | 3-D | $1.0 \cdot 10^7$ | $1.4 \cdot 10^7$ | 1–154 | 100–200 |

This table is an updated version of the table found in a recently published literature review [71, Tab. 1]. [a)] Number of cells in largest mesh used for optimization. [b)] Not all papers state the number of optimization iterations explicitly. In some cases, we report the number of iterations estimated from the cited figures. As mentioned in [126], this number depends on the optimization problem and optimizer settings, meaning that cross-setup comparison is difficult. [c)] Tsiakas et al. [122] only gives the number of mesh nodes. [d)] Reduced geometry where the root section was removed. [e)] Applied symmetric boundary conditions double the mesh size compared to others. [f)] In cases where a range of Reynolds numbers were used, we report the maximum values. [g)] We only found high-fidelity shape optimization for three turbine configurations in the literature: two smaller turbines—NREL Phase VI and MEXICO [107]—and the large, commercial-scale IEA 10 MW wind turbine. We find it reasonable to assume that the simulations for NREL Phase VI and MEXICO have a Reynolds number on the order of $\mathrm{Re} = 10^6$ [116, p. 152][107, p. 10], while we estimate the Reynolds number for the IEA turbine to be on the order of $\mathrm{Re} = 10^7$ [6, p. 15-16]. [h)] For Nielsen and Diskin we use number of mesh nodes since that is where the variables are stored. Impressively, the resolution is comparable to that seen in our recent study [71]. This is particularly noteworthy seeing that the effort by Nielsen and Diskin is the only transient study in the table.

There is quite a lot of information in Tab. 3.1 which we will now try to summarize. As seen, we found nine works presenting solely 2-D studies. These are above the horizontal black line mid table. Below the line we listed the seven 3-D efforts we know of. Both groups are listed in chronological order. We first comment on the 2-D efforts, then on the 3-D efforts.

Ritlop and Nadarajah (2009) [102] are the first within wind energy research to present high-fidelity shape optimization using an adjoint solver. They use a discrete adjoint solver originally developed by Nadarajah [82]. Coming from the aerospace research community they have to extend their numerical design framework to the low-Mach number regime by implementing a preconditioner. They also implement the SA turbulence model[6]. After upgrading the framework they successfully optimize lift-to-drag on the S809 airfoil and mention the implementation of the $k-\omega$ shear-stress transport (SST) turbulence model [76] as well as a transition model in order to increase the fidelity of their framework.

Khayatzadeh and Nadarajah (2011) [57] can then only a couple of years later present a successful implementation of the framework additions which were identified as future work by Ritlop and Nadarajah. Thus, Khayatzadeh and Nadarajah combine the $k-\omega$ SST turbulence model with the $\gamma-\tilde{Re}_{\theta t}$ transition model by Langtry and Menter [75, 63] and demonstrate that they can postpone the onset of transition on the S809 airfoil by extending the natural laminar flow region.

Schramm, Stoevesandt, and Peinke (2014) [108] present their first contribution (out of several [108, 109, 111, 110]) to 2-D RANS shape optimization three years later. In this initial paper, they optimize the lift-to-drag ratio of the DU 91-W2-250 airfoil. They also demonstrate the usefulness of constraining the cross-sectional area. Here, the continuous adjoint approach is favored. They use the C++ flow solver open field and manipulation (OpenFOAM) [129] where a continuous adjoint solver already has been implemented [89, 88] for internal flows. By changing the boundary conditions they can use said adjoint solver for external flows. As mentioned previously a total of 720 design variables are used in this work since every single surface point can be used as a design variable. This work is based on the frozen turbulence assumption, which they later on [110] evaluate more closely.

Schramm, Stoevesandt, and Peinke (2016) [111] then turn to optimize an airfoil with a leading edge slat two years later. In this work, they also introduce a new mesh deformation approach based on the work by Reuther and Jameson [98] mentioned in Sec. 3.1.1. They first investigate the gradient precision by comparing to reference gradients obtained with finite differences. The gradient verification is carried out in the laminar regime ($Re = 2000$) on a NACA0012 airfoil. They observe a mean relative difference in gradients of 9% and 2% for drag and lift objectives, respectively. Before the final optimization a comparison to wind tunnel measurements is given which shows a good agreement outside the stall region. The presented leading edge

[6] In Nadarajah's Ph. D. dissertation [82] the Baldwin-Lomax turbulence model was used

slat optimization leads to a 2% drag reduction and is carried out at a Reynolds number of $6 \cdot 10^5$.

Barrett and Ning (2016) [10] use a CFD-based adjoint method in a comparison across fidelities also including a panel code as the lower fidelity bound and wind tunnel data as the upper fidelity bound. The flow solver they use is known as Stanford University Unstructured ($SU^2$) [90]. It is written in C++ and it has a built in adjoint solver as well as an easy-to-use Python interface. As the authors state in a later work, the comparison shows that "significant changes in optimal blade design occur" [11, p. 3] when using airfoil analysis methods of different fidelities.

Vorspel et al. (2017) [126] present a benchmark study where they investigate the efficiency of various optimization algorithms. Here, a gradient-free (Nelder-Mead) and two gradient-based methods (Quasi-Newton[7] and steepest-descent) are compared. Notably, when discussing gradient-based methods the gradients can either be computed with finite differences or with the adjoint method. They use the continuous adjoint approach within OpenFOAM which we mentioned above. It is important to clarify, that their criteria for efficiency both takes computation time as well as ease of use into account. This line of thought agrees with the recent study by Kenway et al. [55][8]. The overall finding in this benchmark is, that for a large number of design variables one should use a gradient-based optimization algorithm where the gradient is computed with the adjoint method. However, the adjoint method demands a high level of user expertise, initial implementation, and maintenance.

Schramm, Stoevesandt, and Peinke (2018) [110] follow their two previous studies [108, 111] up with an investigation of the frozen turbulence assumption. By including the SA turbulence model in the adjoint formulation they find an improved representation of the finite difference reference gradients compared to results with the frozen turbulence assumption. They then present several unconstrained single-point design cases focusing on coefficients for lift and drag, respectively. By running optimizations with a varying amount of control points they are able to identify a recommended minimum amount of control points one should use for the presented parameterization method. They conclude based on the shown cases that the adjoint turbulence is not always necessary but in other cases it is a must. Therefore, it is in general recommended to include the turbulence model in the adjoint formulation.

Barrett and Ning (2018) [11] can in this work extend the previously presented effort [10] to investigate the difference between sequential design and integrated design. In the sequential approach the aerostructural optimization is held fixed during a planform optimization whereas the optimization of cross-sectional shape and blade planform occurs concurrently in the integrated approach. In fact, they investigate two versions of the integrated approach: a precomputational approach and a true free-form approach. In the integrated precomputational approach, one can only

[7] It is the Broyden-Fletcher-Goldfarb-Shanno algorithm they use

[8] In this paper, the overall criteria is to be 'effective' – not efficient. However, the point is the same, namely that one must take both computation time and implementation cost into account

change the airfoil shape in limited ways during optimization in order to access and use precomputed airfoil data during the optimization. This is not allowed in the integrated free-form approach. Here, however, the possible airfoil shapes are indeed free to take any shape the parameterization allows for. Again, they use different fidelities (a panel code and a RANS CFD code[9]) to obtain a more nuanced result. They find that the restricted integrated approach using precomputation indeed yields more than 80% of the benefits although only incurring a modest extra amount of computational cost. The integrated approach using free-form is of course superior, but it only brings a modest additional benefit compared to the required development time.

Kaminsky and Ekici (2019) [52] is the final 2-D effort we mention. This work does not figure in our literature review since it only recently was published. The work is a study in how to reduce computation time when using a discrete adjoint solver which is algorithmically differentiated in the reverse mode. The specific type of discrete adjoint solver used herein is known as a fixed-point iterative method. We will explain this term more thoroughly in Chapter 4. By mapping the relationship between sensitivity solution and residuals they create what is known as a reduced order model which helps accelerate the convergence by approximating the converged solution for the zero residual. They evaluate the acceleration technique on an inverse design study involving the RAE 2822 airfoil as starting point and the NREL S809 airfoil as the targeted design and observe a 57% reduction in computational costs for viscous problems. The work is based on a compressible formulation of the RANS equations.

Looking at the 2-D efforts as a whole, they all seem to have a rather high amount of design variables when comparing to Sec. 3.1.2.1 where we presented works without an adjoint method. Indeed, some 2-D efforts in Tab. 3.1 have several hundred design variables. As mentioned, this is due to the fact that all mesh points are used as design variables. The numerical frameworks in the 3-D efforts from Tab. 3.1 we describe below are naturally fully capable of such a parameterization, but it is often found advantageous to introduce a geometry module which provides a more regularized deformation using fewer design variables. This is also seen in the later 2-D efforts (compare e.g., [108] with [110] where splines are used in the latter effort as deformation control points). As we will shortly see, the FFD methodology [112] is a very popular deformation strategy.

Finally, we have arrived at the seven 3-D efforts. Here, particular focus is on the six rotor design studies.

Nielsen and Diskin (2012) [85, 86] is the earliest rotor study we found. Out of the six rotor studies in Tab. 3.1 it is the only transient study yet it is the only effort to include both nacelle and tower in the computation which results in a very high mesh resolution. Using the impressive FUN3D[10] solver they are able to verify their discrete adjoint gradients with the complex-step method since the entire code

[9] They cannot use wind tunnel data here due to the free-form deformation

[10] `https://fun3d.larc.nasa.gov/`,(last access: 11 July 2019)

base has been complexified as well as differentiated by hand. Up to 13 significant digits are achieved [85, Tab. 3]. The framework allows for both compressible and incompressible formulations where the incompressibility is formulated using artificial compressibility. Like three other works [25, 22, 125] in Tab. 3.1 they study the NREL VI rotor which has a span of 10 m. They simulate 720 time steps where each time step correspond to 1° which adds up to two full rotor revolutions. In the optimization they maximize torque for a single wind speed. The objective function only considers torque values from the second rotor revolution where transients are less pronounced. Using 76 design variables they achieve a 22% increase in torque coefficient. The optimization was carried out using 2880 CPUs which is roughly ten times the CPUs we used in [71] per discipline[11]. The optimized design has an increase in thickness over a large part of the blade as well as increased camber towards the trailing edge. We find the reported increase in thickness rather surprising given that the optimization is purely aerodynamic. Indeed, in our own studies which are also purely aerodynamic we find, that the thickness is reduced as much as possible while still satisfying our thickness constraints. Turning to the reported increase in camber it agrees with the results from a later steady-state study [22] on the exact same rotor. Finally, they specifically mention constraints on bending moment and thrust and state that they can easily be incorporated.

Economon, Palacios, and Alonso (2013) [25] is the second effort to study the NREL VI rotor. In this steady-state study they use the compressible $SU^2$ RANS solver and a continuous adjoint method for all computations. In their 2-D airfoil study they use Hicks-Henne bump functions to perturb the mesh whereas they use an FFD-based approach for their 3-D rotor study with a total of 84 design variables. Noticeably, their FFD approach is the conventional surface mesh deformation. They then propagate the deformation out in the volume mesh using linear elasticity. To avoid the complications at root and trailing edge some of the control points are held fixed and the FFD box only covers part of the blade. They also optimize the torque coefficient and obtain a 4% increase in three design cycles. The results is noticeably lower than the 22% by Nielsen and Diskin. Likewise, a later NREL VI rotor study [22] achieves an increase of 22.4% in a comparable single-point optimization. This discrepancy may be explained by the amount of design cycles. Economon, Palacios, and Alonso only report of three design cycles which is considerably fewer design cycles than in the other two studies where 8 [85] and 9 [22] design cycles are reported. One should of course be cautious when comparing design cycles since the number drastically depends on gradient precision and optimization algorithm but it is possible that the optimization problem by Economon, Palacios, and Alonso should have been further converged. Another possible explanation is that they use the frozen turbulence assumption which also is one of the areas they identify as future work. Finally, they mention multipoint optimization as a way to obtain a more realistic objective function.

[11] We used 216 CPUs in the single-point steady-state case and $3 \times 216$ CPUs in the multipoint case where three wind speeds were considered

Vorspel et al. (2016) [127] present in this effort a pre-study to the ensuing rotor study [125] a few years later. They use the continuous adjoint setup in OpenFOAM also used in [126, 108] to carry out a 2-D case and a 3-D case with 2 design variables (camber and thickness). The 3-D case is a wing based on the NACA0012 airfoil where no complicating factors such as root separation and vortices are present. They compare a bend-twist case to a pure twist movement and find no discernible difference. They do, however, expect this to change for a rotor blade configuration.

Dhert, Ashuri, and Martins (2017) [22] is one of two multipoint rotor studies in Tab. 3.1. Using a discrete adjoint approach they optimize the NREL VI rotor with a 2.6 million cell mesh resolution. Up to 252 design variables (twist and shape) are used to deform the blades. Due to convergence issues at the root they had to cut out the inner most part of the blade. They obtain a 22.1% increase in torque coefficient in their final multipoint study and report of increased camber which, as mentioned, agrees with other studies.

This study by Dhert, Ashuri, and Martins [22] was carried out using an earlier version of the very same framework at MDOLab which was used in our study [71]. There have been many improvements made to the refactored framework at the MDOLab as mentioned in [71, Sec. 2.3]. The improvements we had to implement are listed in Chapter 11 where we present the results from our multipoint rotor study. Other improvements[12] are explained further in Chapter 4. A table listing differences between the rotor study by Dhert, Ashuri, and Martins and our study is given in [71, Tab. 2]. Furthermore, the present chapter concludes with Tab. 3.2 where all six rotor studies are compared.

Vorspel, Stoevesandt, and Peinke (2018) [125] carry out an unconstrained optimization on the NREL VI rotor using the continuous adjoint setup in OpenFOAM which has been used in several other efforts [127, 126, 108]. They use the steepest descent algorithm to converge the problem. Unlike the other NREL VI rotor studies [85, 25, 22] focusing on torque maximization Vorspel, Stoevesandt, and Peinke present a thrust minimization problem where up to nine twist design variables are used. This allows them to test a newly developed FSI inspired projection method which connects the many cell gradients to a reduced design space. Due to vortices at tip and root resulting in convergence issues they limit the movable part of the blade to 20-50%. The applicability of the developed tool is demonstrated by converging 12 different test cases. Like the study by Economon, Palacios, and Alonso they use the frozen turbulence assumption and identify the inclusion of turbulence models in the adjoint formulation as future work.

Tsiakas et al. (2019) [122] is the only rotor study we found for the MEXICO reference wind turbine rotor. They maximize power using a continuous adjoint approach where the turbulence model (SA) has been included. Interestingly, they use the less typical mesh deformation approach where both surface mesh and volume mesh

[12] such as the transition from forward to reverse algorithmic differentiation to enable the memory saving transpose-matrix-vector matrix-free operations

is deformed by the same component. This is the same approach we will take when developing our numerical framework in Part II. Tsiakas et al. use the non-uniform rational B splines (NURBS) by wrapping all rotor blades in boxes. These boxes then embed both surface and volume mesh points. The chosen box resolution results in 135 shape design variables that can move in the streamwise direction. Impressively, both the flow and adjoint solver use graphics processing units (GPU)s to accelerate the computations. They state that the transition from CPU to GPU can offer a speedup of up to 50. The final optimization result is a 3% increase in torque. The result is viewed as a minor increase, which they explain by the somewhat limited design freedom in the NURBS setup.

Madsen et al. (2019) [71] is a presentation of what we believe to be the most comprehensive shape optimization study for a rotor. We do acknowledge, that the transient treatment by Nielsen and Diskin [86] above is extremely impressive. Indeed, it is hard to imagine other numerical frameworks rival that at NASA Langley Research Center. At the same time, when considering the presented rotor studies in [85] and [71] we find that the more robust multidisciplinary approach as well as the extra constraints on load and bending moment are crucial for an industrially relevant design as detailed in [71]. The following bullet points highlight some of the assets of our paper:

1. Enforcement of geometric constraints to ensure structural feasibility.
2. Normal operation rotor load constraints limiting thrust and flapwise bending moment.
3. More precision and stability in the convergence of flow and adjoint solvers.
4. Turbulence model included in adjoint formulation.
5. A comprehensive set of design variables.
6. Modeling and deformation of the entire blade shape.

The paper presents a series of design optimization cases increasing in complexity, namely: pitch optimization (1 design variable), planform optimization (14 design variables), and shape optimization (154 design variables). The planform optimization problem is solved both with high-fidelity CFD-based methods and with low-fidelity BEM-based methods allowing us to compare optimization results across fidelities. Considering the various optimization problems with increasing complexity, it is evident from the results ([71, Fig. 8, 11, 14, and 17]) that more flow and adjoint solutions are needed as the problem complexity increases. This finding resonates with earlier efforts [110, Fig. 7, 14, and 16]. Thus, it is extremely important for numerical optimization frameworks to ensure robustness since realistic design tasks may include more than hundred successive flow and adjoint solutions. Leading up to the various design studies is a mesh convergence study including meshes up to 48 million cells. We use this convergence study to compare the compressible flow solver, ADflow, used in [71] with the incompressible flow solver, EllipSys3D, which

we use in the present work. For the paper itself we refer readers to the appendix (Sec. A). However, we do bring an excerpt of the results in Chapter 11 where we present the final multipoint rotor study (Sec. 11.2).

Having commented on all the works in Tab. 3.1 we now bring a comparison of the six mentioned rotor studies in Tab. 3.2 which summarizes the above discussion. As a general comment to Tab. 3.2 it is evident that particularly the thrust and load constraints are not yet common practice. Inded, Tab. 3.2 shows that the research on high-fidelity rotor studies has only just begun.

Before a few concluding remarks listed just after Tab. 3.2 we cannot help but mention the recent effort by Anderson et al. [3] on an adjoint-based high-fidelity *structural* optimization of a wind turbine blade for load stress minimization. This goes against the rules we stated at the beginning of the literature review, given that we only would focus on aerodynamic shape optimization efforts. However, we relent since the effort helps point to the ultimate goal mentioned in the Summary, i.e., a comprehensive aerostructural high-fidelity shape optimization of a wind turbine rotor. Such a study has to the best of our knowledge yet to be carried out. As pointed out in the Summary one could for such a study closely tailor the aerodynamic and structural responses to lower loads and increase power production. Replacing the geometrical constraints we imposed in our aerodynamic design study [71] with a structural discipline including an adjoint solver would be a major improvement, which we fully expect would bring new insights to the field.

Returning to the work by Anderson et al. [3] they study the 13 m blade from the SWiFT wind turbine rotor just as in the earlier effort [2] by the same authors we mentioned in our paper [71]. They couple a RANS flow solver (NSU3D) to a structural finite element solver (AStrO) and carry out a structural optimization using a load distribution generated by the RANS solver. Both high-fidelity solvers are developed in-house and the AStrO adjoint solver allows for an impressive 16310 design variables to be used in the optimizations. It is a discrete adjoint method and they optimize composite fiber angles in the internal blade structure to minimize a stress objective function. They observe a reduced amount of driving stress for fatigue (18-60%).

Anderson et al. [3] view their work as an interim step to the declared goal of an aerostructural optimization where the aerodynamic shape and the structural sizing simultaneously are optimized. These types of optimizations can be found in aerospace research [54] but we have, as mentioned, yet to find one within wind turbine rotor studies.

**Table 3.2:** Overview of aerodynamic optimization works of wind turbine rotors using the adjoint method.

| **Reference** | Multi | Turbulence | Deformation (✓= full blade) | Geometry (✓= full blade) | Load constraints | | Geometric constraints | Design variables | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | Thrust | Moment | | Twist | Chord | Shape |
| Nielsen and Diskin [85] | | ✓ | ✓ | ✓ | | | ✓ | ✓ | [a)] | ✓ |
| Economon, Palacios, and Alonso [25] | | | | ✓ | | | ✓ | | | ✓ |
| Dhert, Ashuri, and Martins [22] | ✓ | ✓ | | | | | ✓ | ✓ | | ✓ |
| Vorspel, Stoevesandt, and Peinke [125] | | | | ✓ | | | | ✓ | | |
| Tsiakas et al. [122] | | ✓ | ✓ | ✓ | | | | | | ✓ |
| Madsen et al. [71] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

This table is an updated version of the table found in a recently published literature review [71, Tab. 3].
**Multi**: Multipoint optimization; **Turbulence**: Whether the turbulence model is included in the adjoint solver; **Deformation**: Whether the entire blade was allowed to deform; **Geometry**: Whether the entire blade was modeled; **Geometric constraints**: Whether any geometric constraints were imposed; [a)] They have 'twist', 'thickness' and 'camber' variables which we have marked as 'twist' and 'shape' to align with the chosen columns

We conclude our literature review on high-fidelity CFD-based shape optimization within wind energy with the following remarks:

1: As seen in Tab. 3.1 it is a young but promising research field with only six applications to an actual rotor geometry.

2: Of these six efforts, three are based on numerical frameworks developed in the aerospace community [25, 22, 71] and another is carried out by extending the continuous approach within OpenFOAM leaving (the pseudo-compressible) [122] as the only effort to be developed from scratch.

3: Our paper shows [71, Fig. 5] that incompressible SIMPLE flow solvers dedicated for wind energy applications may offer a considerable performance advantage on rotor simulations compared to, e.g., compressible aerospace CFD solvers, since these are tailored for aeronautical applications.

4: The only SIMPLE algorithm found among the six efforts is the OpenFOAM solver.

5: A push should be made to encourage researchers from the wind energy community to extend their large incompressible CFD codes dedicated for wind energy with an adjoint method (be it the continuous or the discrete method).

# Part II

# The development of a numerical design optimization framework

# CHAPTER 4
# Components of a numerical design framework

This project's starting point was a very capable CFD solver called EllipSys3D [77, 78, 114]. In order to transition from using EllipSys in an analysis context to using it in design optimizations we had to build an optimization framework up around the CFD solver. This numerical optimization framework can be seen in Fig. 4.1 using an extended design structure matrix (XDSM) diagram [62].

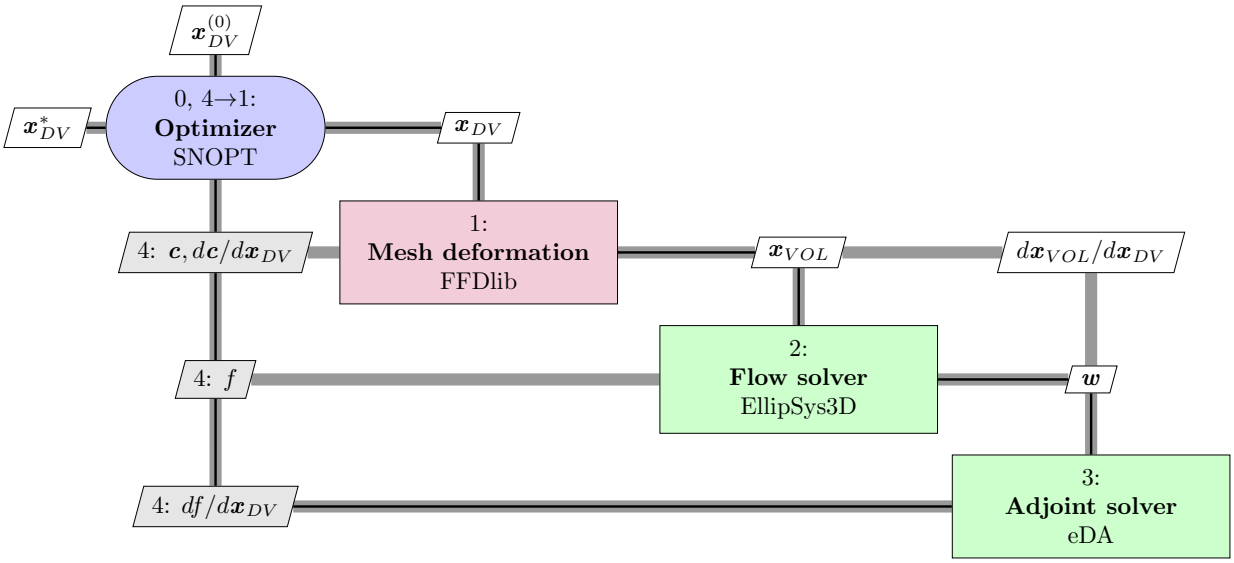



**Figure 4.1:** Components in the numerical high-fidelity shape optimization framework at DTU Wind Energy. The four components are (0,4) an optimizer, (1) a mesh deformation module, (2) a CFD solver, and (3) a discrete adjoint solver. Gray lines show data flow. Black lines show process flow.

As seen, there are three new components: an optimizer, a deformation module, and an adjoint solver. We decided to use the Sparse Nonlinear OPTimizer (SNOPT) [32] as our optimizer which we use through the open-source MDO framework: OpenMDAO [33]. In OpenMDAO we use pyOptSparse[1] [93] as our driver, which in turn provides

[1] `https://github.com/mdolab/pyoptsparse`,(last access: 27 June 2019). pyOptSparse stands for: PYthon OPTimization (Sparse) Framework

the Python interface to SNOPT. The underlying optimization algorithm in SNOPT is a sequential quadratic programming (SQP) algorithm. The other two new components in Fig. 4.1 (the deformation module and the adjoint solver) have been developed during this project. The mesh deformation module is called FFDlib and perturbs both surface and volume mesh points to provide an updated mesh, $\mathbf{x}_{VOL}$. Furthermore, it provides analytical gradients of the mesh with respect to the design variables, $\mathrm{d}\mathbf{x}_{VOL}/\mathrm{d}\mathbf{x}_{DV}$. We describe this component further in Chapter 5. The EllipSys discrete adjoint solver (eDA) is by far the most arduous component to develop from scratch. After introducing the EllipSys flow solver in Chapter 6 we will dedicate Chapters 7 to 10 to key aspects in the development of our discrete adjoint solver, eDA, thus concluding Part II of the thesis.

In this thesis a term such as 'industrial scale' is often used. Using this term we wish to stress, that we are dealing with CFD solvers used by the industry and that the solver is applied to cases involving meshes with dozens – or even hundreds – of millions of cells. As a consequence, the underlying code base for such a CFD solver may easily involve several hundred thousand lines of code. Of course, it is not a goal in itself to have a large code base. We mention it since it stresses the importance of choosing methods that result in an implementation which is somewhat easily maintained. Before proceeding, it may be instructive to get a rough estimate of the code base for a framework such as the one shown in Fig. 4.1. To this end we bring Tab. 4.1 where the SNOPT optimizer has been omitted since it is a third-party component. We also note that the CFD solver itself was not developed within the present project which leaves complexification, adjoint solver development, and geometry module development, as the main developmental tasks we handled within the project.

**Table 4.1:** Overview of the number of code lines for components in the framework seen in Fig. 4.1.

| Component | # of lines of code | |
|---|---|---|
| | Python | Fortran |
| | $\cdot 10^3$ | $\cdot 10^3$ |
| CFD solver | 9 | 120 |
| Complexified CFD solver | - | 100 |
| Adjoint solver | 1 | 95 |
| Geometry module | 3 | 5 |

It may be difficult to know what to expect when inspecting Tab. 4.1. Therefore, we mention two examples from the literature from the previous chapter. The first example is the vast FUN3D solver maintained at NASA Langley Research Center which was used in the rotor study be Nielsen and Diskin [85]. The built in discrete adjoint solver in FUN3D [84] is said to have "several hundred thousand lines of exact hand-differentiated" [58, p. 13] code. Note, that this is just for the adjoint solver. It is fairly safe to assume that the code base for the FUN3D adjoint solver can be seen as an upper bound in most cases. The second example we bring is the DAFoam adjoint solver [38, 39] which comprises

more than 40 thousand lines of source code[2]. We will present the DAFoam solver in much more detail further down. For now, we are simply interested in the size of the code base, which as mentioned is above 40 thousand lines of C++ code. Looking to Tab. 4.1 we see that our adjoint is placed somewhere between these two implementations in size. We now proceed to a specific literature study directed for a narrow group of adjoint solvers. However, it is important to keep Tab. 4.1 in mind, especially when discussing maintenance of these adjoint solvers.

The below literature review will first and foremost show, that adjoint solver development is a time consuming task. Before diving into the literature we briefly set the scene by returning to the FUN3D adjoint solver. Certainly, it is one of the most prominent fixed-point adjoint solvers world-wide and incredibly enough, it is derived by hand. To fully grasp the immense work involved we consider the quote:

> " This effort represents the only capability of its kind and relies on several hundred thousand lines of exact hand-differentiated linearizations of the preprocessor, flow solver, and mesh movement codes with respect to both the dependent variables and the grid coordinates ... The adjoint results are in excellent agreement with those obtained using a complex-variable approach ... This accuracy can easily be compromised by a single error anywhere in the source code."
>
> Kleb et al. [58, p. 13]

It is not difficult to imagine why researchers pursue alternative routes to high-fidelity shape optimization (e.g., surrogate modeling) when 'a single error anywhere' can ruin the adjoint solver performance. Needless to say, one should leverage algorithmic differentiation to construct adjoint solvers whenever possible in order to reduce the immense work load. While bug-free hand differentiated code does provide the best attainable performance, one can often arrive at competitive algorithmically differentiated code with careful implementation [80].

## 4.1 Literature on discrete adjoint solvers for the SIMPLE algorithm

The rest of this chapter presents the third part of the literature review which divulges into the niche field of discrete adjoint solvers. Here, we will continuously bend the discussion towards discrete adjoint solvers tailored for the segregated SIMPLE algorithm, as it is the CFD solver type of the present work. There are several ways to construct a discrete adjoint solver, given that there are several ways to solve the discrete adjoint equation:

$$\left[\frac{\partial \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\partial \tilde{\mathbf{w}}}\right]^T \boldsymbol{\Psi} = \left[\frac{\partial \mathrm{J}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\partial \tilde{\mathbf{w}}}\right]^T . \tag{4.1}$$

[2] `http://drpinghe.com/index.php/hercules/`,(last access: 27 June 2019)

Below, we briefly describe two common approaches namely, the fixed-point approach and the Krylov approach, and proceed to explain why we chose the latter. Here, the names of the two approaches is a reference to their solution procedure when solving the adjoint equation.

### 4.1.1 Fixed-point iteration discrete adjoint solvers

The following is a simplified presentation of the fixed-point adjoint solvers conveying a basic understanding of the approach. Further information is needed in order to actually implement this approach - particularly for the SIMPLE algorithm. Readers are referred to the effort by Akbarzadeh, Wang, and Müller [1] for further information.

The fixed-point iteration approach takes its starting point in the flow solver at hand. A CFD flow solver typically drives a residual to zero,

$$\mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}) = 0, \tag{4.2}$$

by iteratively determining next iteration's flow value, $\tilde{\mathbf{w}}^{k+1}$, based on known values at previous iterations, $\tilde{\mathbf{w}}^{k}$:

$$\tilde{\mathbf{w}}^{k+1} = \mathbf{M}(\tilde{\mathbf{w}}^{k}, \mathbf{x})\mathbf{R}(\tilde{\mathbf{w}}^{k}, \mathbf{x}). \tag{4.3}$$

Above, the $\mathbf{M}$ matrix depends on the iterative method being used[3]. By taking the derivative of equation 4.3 we can create a fixed-point tangent-linear solver from the flow solver,

$$\frac{\mathrm{d}\tilde{\mathbf{w}}^{k+1}}{\mathrm{d}\mathbf{x}} = \mathbf{M}(\tilde{\mathbf{w}}^{k}, \mathbf{x}) \quad \underbrace{\left[\frac{\partial \mathbf{R}}{\partial \tilde{\mathbf{w}}}\frac{\mathrm{d}\tilde{\mathbf{w}}^{k}}{\mathrm{d}\mathbf{x}} + \frac{\partial \mathbf{R}}{\partial \mathbf{x}}\right]}_{\text{tangent-linear residual: } \mathrm{d}\mathbf{R}/\mathrm{d}\mathbf{x}} . \tag{4.4}$$

In Eq. (4.4) we have used the identity for the tangent-linear residual from Eq. (2.9). Where we could obtain flow states, $\tilde{\mathbf{w}}$, by converging Eq. (4.2) using the flow solver, we can now using the same iterative method, $\mathbf{M}$, converge Eq. (4.4) using a tangent-linear solver to obtain the total derivative of flow states with respect to a design variable, $\mathrm{d}\tilde{\mathbf{w}}/\mathrm{d}\mathbf{x}$. It is in other words $\mathrm{d}\tilde{\mathbf{w}}/\mathrm{d}\mathbf{x}$ which is the unknown in Eq. (4.4). At convergence, $\mathrm{d}\tilde{\mathbf{w}}^{k+1}/\mathrm{d}\mathbf{x} \approx \mathrm{d}\tilde{\mathbf{w}}^{k}/\mathrm{d}\mathbf{x}$ and we would have driven the tangent-linear residual, $\mathrm{d}\mathbf{R}/\mathrm{d}\mathbf{x}$, to zero. This *linear* forward problem will converge with the same convergence rate as the nonlinear flow problem [30].

[3] It can be beneficial to consider a few examples to fully grasp the meaning of the method matrix, $\mathbf{M}$. Assuming the flow residuals have the following generalized form: $\mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}) = \mathbf{b} - \mathbf{A}\tilde{\mathbf{w}} = \mathbf{0}$ the right hand side in Eq. 4.3 would for the Gauss-Seidel method be, $\mathbf{M}(\tilde{\mathbf{w}}, \mathbf{x})\mathbf{R}(\tilde{\mathbf{w}}^{k}, \mathbf{x}) = \mathbf{L}_{*}^{-1}(\mathbf{b} - \mathbf{U}\tilde{\mathbf{w}}^{k})$ where $\mathbf{L}_{*}$ and $\mathbf{U}$ are the lower and (strictly) upper triangular decompositions of $\mathbf{A}$. For the Newton-Raphson method we would have: $\mathbf{M}(\tilde{\mathbf{w}}, \mathbf{x})\mathbf{R}(\tilde{\mathbf{w}}^{k}, \mathbf{x}) = \tilde{\mathbf{w}}^{k} - (\partial\mathbf{R}/\partial\tilde{\mathbf{w}})^{-1}\mathbf{R}(\tilde{\mathbf{w}}^{k}, \mathbf{x})$.

Notice, that for the fixed-point iterative method we do not necessarily have to construct an explicit residual subroutine, $\mathbf{R}$. Rather, we use the flow solver code as is. Assuming `w` is the flow variables, and `Ae,Aw,An,As,At` and `Ab` are the influence coefficients for the east, west, north, south, top, and bottom cell faces, the flow solver simply computes the various NS terms through a series of subroutine calls:

```
   ⋮
call diffusion_coefficients(w,Ae,Aw,An,As,At,Ab)
call convection_coefficients(w,Ae,Aw,An,As,At,Ab)
   ⋮
```

As a result, the total state Jacobian, $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$, is never explicitly constructed. Instead, one simply differentiates the flow solver subroutines either by hand or leveraging algorithmic differentiation, and follow exactly the same call graph to obtain the corresponding tangent-linear solver[4]:

```
   ⋮
call diffusion_coefficients_d(w,wd,Ae,Aed,Aw,Awd,An,And,As,Asd,At,Atd,Ab,Abd)
call convection_coefficients_d(w,wd,Ae,Aed,Aw,Awd,An,And,As,Asd,At,Atd,Ab,Abd)
   ⋮
```

In the above fixed-point tangent-linear sample code we borrowed the algorithmic differentiation notation where derived variables have a `d` suffix. Thus, `Aed,Awd,And,Asd,Atd` and `Abd` are *derivatives* of the influence coefficients for the east, west, north, south, top, and bottom cell faces. Likewise, the dotted flow state, `wd`, is the *derivative* of the flow variable, `w`. To better relate the sample code to the equation above we note that `wd` from the sample code is the unknown, $\mathrm{d}\tilde{\mathbf{w}}/\mathrm{d}\mathbf{x}$, in Eq. (4.4).

In the same vein, a fixed-point adjoint solver can be generated from the flow solver. Using Eq. (2.13) we now express an adjoint residual, $\mathbf{R}_{\text{adj}}$, and instead of Eq. (4.4) the relevant expression for a fixed-point adjoint solver becomes,

$$\boldsymbol{\Psi}^{k+1} = \mathbf{M}(\tilde{\mathbf{w}}^*, \mathbf{x})^T \underbrace{\left[\frac{\partial \mathbf{R}}{\partial \tilde{\mathbf{w}}}^T \boldsymbol{\Psi}^k - \frac{\partial \mathrm{J}}{\partial \tilde{\mathbf{w}}}^T\right]}_{\text{adjoint residual: } \mathbf{R}_{\text{adj}}}, \tag{4.5}$$

where the method operator is now transposed, $\mathbf{M}^T$.

Still, this fixed-point adjoint solver can be constructed completely from the flow solver using the same method. The transpose on the method operator, $\mathbf{M}^T$, and the two partial derivative matrices in the adjoint equation, $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}^T$ and $\partial\mathrm{J}/\partial\tilde{\mathbf{w}}^T$, affects the structure of the code. As an example consider the case where the flow residuals have the following generalized form: $\mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}) = 0 \Leftrightarrow \mathbf{b} - \mathbf{A}_2\mathbf{A}_1\tilde{\mathbf{w}} = \mathbf{0}$. Here, the state Jacobian can evidently be decomposed into two matrices: $\partial\mathbf{R}/\partial\tilde{\mathbf{w}} = \mathbf{A}_2\mathbf{A}_1$. The two matrices,

[4] Obviously, in a NS solver the diffusive influence coefficients will not depend on $\tilde{\mathbf{w}}$ when cross-diffusive calls are treated explicitly. Only the convective term is nonlinear in $\tilde{\mathbf{w}}$ and would give non-zero dotted variables. However, a correctly differentiated code would simply return zero for dotted variables, and no harm would be done.

$\mathbf{A}_1$ and $\mathbf{A}_2$, could account for, e.g., transient effects and flux discretization, respectively. Taking the transpose would reverse the order of operations, $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}^T = \mathbf{A}_1^T\mathbf{A}_2^T$, and the resulting adjoint Fortran code would therefore `call A2()` before a `call A1()` statement would occur. As a simple transposition of $\mathbf{M}$ in Eq. (4.5) will not change the eigenvalues (on which the convergence properties depend [19]) one can expect the same convergence for the adjoint solver and for the flow solver.

In summary, we have shown how to use a flow solver (Eq. 4.3) to construct a fixed-point tangent-linear solver (Eq. 4.4) and a fixed-point adjoint solver (Eq. 4.5). Furthermore, we have explained that the convergence rate for these three solvers (flow solver, tangent-linear solver, and adjoint solver) will be the same if the primal is contractive since the eigenvalues are the same in the three methods. Finally, we stress the point that the above is a simplified description of the fixed-point approach. Eq. (4.4) and Eq. (4.5) will not be so simple in the case of a SIMPLE algorithm as is explained elsewhere [1].

For an adjoint fixed-point iterative solver, the reverse differentiation of the flow solver subroutines is inherently more cumbersome and one must also take care to reverse the order of the subroutine calls as explained elsewhere [30]. However, it can be done either by hand or (preferably) using an algorithmic differentiation tool. Either way, we would never construct the entire state Jacobian $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$ much like the flow solver never would assemble its NS influence coefficient matrix. In this regard, some fixed-point solvers are *matrix-free* methods and incur little memory overhead. A discussion of the matrix-freeness and order reversal of subroutine calls can be found in [28].

The fixed-point approach also has other benefits, such as a large amount of code re-usability and a guaranteed terminal convergence rate equal to that of the flow solver as mentioned above. This feature is known as duality preservation [94, 5, 28]. Interestingly, the performance of the fixed-point approach can be drastically enhanced by a selective algorithmic differentiation usage [30]. Finally, there is no denying that there is a clear description for the fixed-point approach: You linearize the equations and then you transpose them (resulting in the reversal of subroutine calls). Indeed, the fixed-point approach is perhaps overall, the most typical way of developing a discrete adjoint solver, and numerous examples can be found in related literature. For the SIMPLE algorithm we would recommend interested readers to consult [50, 49, 5, 1, 120].

### 4.1.2 Krylov discrete adjoint solvers

> "This is not quite the end of the story however, as there are regularly situations in practice where the original iterative method for the flow equations stalls, implying divergence for the corresponding adjoint iteration. This may be remedied by reintroducing a Krylov method ... "
>
> Peter and Dwight [94, p. 382]

Indeed, as the quote by Peter and Dwight alludes to, there have been reports of convergence issues for the discrete fixed-point adjoint solvers [16, 24, 5, 132].

Peter and Dwight point to a) approximations made to $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}^T$ or b) convergence of the nonlinear flow problem stalling in limit cycles oscillations (LCO), as explanations

for such convergence issues. As stressed by Peter and Dwight, this behavior does not necessarily reflect an error in the flow solver. It could simply be owed to the fact that the flow for the geometry in question is inherently instationary and thus not well represented by the steady-state RANS model. Indeed, these limit cycles oscillations are often seen for wind turbine rotor simulations and we would as a result be bound to tackle these issues for the adjoint solver at some point. An example of limit cycles oscillations can be seen in Fig. 4.2. Here, the EllipSys3D flow solver was used in a RANS simulation of the 10 MW International Energy Agency (IEA) Wind Task 37[5] rotor case study using a 113 M cell mesh[6].

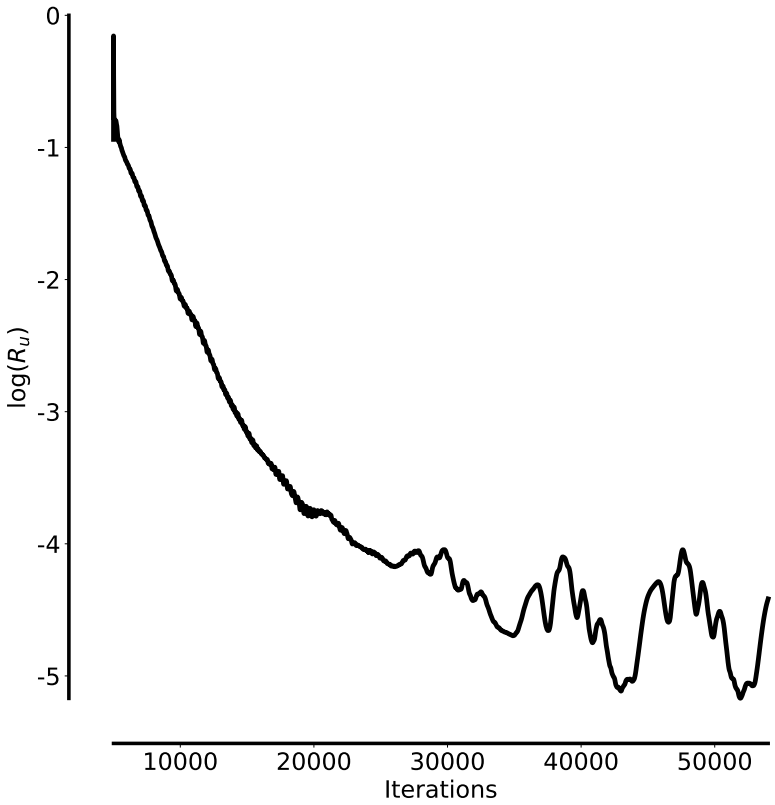



**Figure 4.2:** Visualization of limit cycle oscillations. The $u$ velocity residual from EllipSys during a steady-state RANS simulation on a $113 \cdot 10^6$ cell mesh of a three bladed rotor. The limit cycles oscillations occur around iteration $40 \cdot 10^3$. The joining of the three blades at the nacelle forms a massive blunt object causing the instationarities.

In search of more stable discrete adjoint approaches, another popular method for developing a discrete adjoint solver, i.e., the 'Krylov approach', has emerged. Here, we adopt the naming convention from a recent paper [55] which we also refer to for

[5] `https://community.ieawind.org/tasks/taskdirectory`,(last access: 10 December 2019)
[6] The rotor mesh is further explained in [71]

further details. As pointed out elsewhere [104, p. 769] the only necessary code feature for this approach is a subroutine that calculates the residual based on the converged flow states. However, given the structure of industrial scale CFD solvers with numerous separate subroutine calls for, diffusion, convection, etc., one will rarely have such a global subroutine at hand. One must therefore construct the residual subroutine from the source code itself. With the residual subroutine, one can then compute the two partial matrices, $\partial \mathbf{R}/\partial \tilde{\mathbf{w}}^T$ and $\partial \mathbf{J}/\partial \tilde{\mathbf{w}}^T$, from Eq. (4.1). Finally, the adjoint equation is solved using the GMRES method [105] which minimizes the residual over a Krylov subspace, $K_n$;

$$K_n = \text{span}\{\mathbf{R}_0, \left[\left[\frac{\partial \mathbf{R}}{\partial \tilde{\mathbf{w}}}\right]^T\right]^1 \mathbf{R}_0, ..., \left[\left[\frac{\partial \mathbf{R}}{\partial \tilde{\mathbf{w}}}\right]^T\right]^{n-1} \mathbf{R}_0\}, \tag{4.6}$$

with $\mathbf{R}_0$ being the initial residual. Hence the name; Krylov approach. Assuming the discrete adjoint equation is solved using some general parallel linear solver package (e.g., PETSc) one is of course free to choose any kind of method to solve the system with, but in the literature the GMRES method seems by far to be the favored choice.

As pointed out by Kenway et al. [55] the GMRES method uses information from all iterations whereas the fixed-point method only uses the previous iteration to compute the current iteration. As a result, the fixed-point convergence will always be linear, whereas the GMRES method should converge faster [55, p. 24]. Furthermore, a variant of the GMRES method is Jacobian-free [59] and can converge the adjoint Eq. (4.1) *without* the need for the state Jacobian, $\partial \mathbf{R}/\partial \tilde{\mathbf{w}}^T$, to be stored. This potential reduction in memory consumption is a major advantage, considering the favorable convergence rate. Finally, should the same solution strategy be applied to the flow problem one should observe a duality preserving behavior with similar convergence rates for the flow solver and the adjoint solver. This fact is also mentioned by Giles, Ghate, and Duta [30] and elsewhere [94]. However, it is also pointed out by Giles, Ghate, and Duta that the tangent-linear and adjoint Krylov type solvers are unlikely to produce exact same gradient value at the same iteration count – something that a fixed-point iterative adjoint solver can offer [30]. We have yet to find this investigated. It should be stated, that from a developmental point of view, this *exact* duality preservation is a very powerful feature of the fixed-point method when debugging as only a few iterations need to be executed before a check can be made.

We have, however, found reports of fixed-point discrete adjoint solvers using GMRES as an inner smoother resulting in a point-implicit method with low memory requirements and very stable convergence [132]. Finally, we mention that also the convergence enhancement for the Krylov methods is an active area of research. By recycling the Krylov subspace both a memory reduction and a speedup of factors 2 and 3, respectively, have been reported [131]. We have also experienced that the GMRES convergence occasionally stalls. However, it does not diverge as is observed for fixed-point iterations.

Due to the promise of a favourable convergence while still being able to use memory efficient matrix-free routines we opted to develop a Krylov discrete adjoint solver for EllipSys. However, constructing the explicit residual subroutines needed for the Krylov

approach is *particularly* strenuous for SIMPLE algorithms, which are typical in segregated incompressible CFD solvers like EllipSys. One might fear that the segregated algorithm structure simply is ill suited for the Krylov approach where one global residual subroutine must be made for each primitive variable. However, also fixed-point advocates report of the segregated SIMPLE algorithm as particularly cumbersome for adjoint development compared to compressible solvers [5, p. 99]. We therefore contend, that the particularly strenuous implementation for pressure residuals is owed to the SIMPLE algorithm and not the Krylov approach when developing the adjoint solver. We cannot of course give a final verdict on which approach to prefer between the fixed-point approach and the Krylov approach since we have yet to try out the former on an industrial scale CFD solver. Both approaches have been shown to produce extremely impressive adjoint solvers (e.g., [81] and [55]). However, we do admit, that we prefer the Krylov approach since we find it rather intuitive, albeit this comes at the cost of having to construct the full residual subroutine. Furthermore, we loose the ability to use the exact duality preservation while debugging. Still, the developmental process can be done in an intuitive manner, which we thoroughly describe in Chapter 7.

We have found the available literature extremely sparse with respect to adjoint solvers for SIMPLE algorithms based on the Krylov approach – both when it comes to a general literature survey and when it comes to the actual implementation. One can find a general overview of some of the most prominent adjoint solvers (both fixed-point and Krylov types) in [55, Tab. 1]. However, not all known SIMPLE discrete adjoint solvers are mentioned, since focus is laid elsewhere. We therefore give an in-depth overview in the present section of all the works we found using the Krylov approach on a SIMPLE algorithm to construct an adjoint solver. The overview is seen in Tab. 4.2.

As seen, in Tab. 4.2 there was (to our knowledge) only one reference available at the beginning of this project, namely the work by Roth and Ulbrich [104]. We find this work to be the most impressive in Tab. 4.2 given, that it is an *unsteady* LES adjoint solver. This work was crucial for the success of the present adjoint solver development. Importantly, Roth and Ulbrich explain how a correct linearization of the residual routines necessitates the introduction of new independent variables, namely the face fluxes $c1$, $c2$, and $c3$. This is due to the convective term in the NS equations. Thus, in all works in Tab. 4.2 the extended discrete NS residual vector, $\mathbf{R}$, and the extended discrete state vector, $\tilde{\mathbf{w}}$, become; $\mathbf{R} = [\mathbf{R}_u, \mathbf{R}_v, \mathbf{R}_w, \mathbf{R}_p, \mathbf{R}_{c1}, \mathbf{R}_{c2}, \mathbf{R}_{c3}]^T$ and $\tilde{\mathbf{w}} = [\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{p}, \mathbf{c1}, \mathbf{c2}, \mathbf{c3}]^T$. A better word for this approach might therefore in the context of SIMPLE algorithms be the 'extended Jacobian approach'. That is, however, not a fitting name given that the approach in early stages was championed by researchers from aerospace [69, 72] who used compressible solvers without the need for the laborious extension of independent variables. We therefore stick to the term 'Krylov approach' despite examples in the literature of fixed-point adjoint solvers with GMRES smoothers [81] which blur the divide between the fixed-point approach and the Krylov approach.

To better grasp the memory-related impact of introducing new independent variables we visualize the extension presented by Roth and Ulbrich:

**Table 4.2:** Krylov discrete adjoint solvers developed for the SIMPLE algorithm.

| Ref | Year | $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}^T$ | Source | AD | | Precision | Turbulence |
|---|---|---|---|---|---|---|---|
| | | | | mode[a)] | method[b)] | (digits) | |
| [104][c)] | 2013 | assembled | F77/F95 | forward | OO | - | Smagorinsky[d)] |
| [23] | 2018 | assembled | C++ | reverse | OO | - | SA and $k-\omega$[e)] |
| [55][f)] | 2019 | matrix-free | C++ | reverse | OO | 10 | several[g)] |
| eDA[h)] | 2019 | assembled | F90/F95 | forward | ST | 16 | SST[i)] |

In this table we abbreviate algorithmic differentiation with AD

a) AD mode can either be the forward mode or the reverse mode

b) AD method can either be operator overloading (OO) or source transformation (ST)

c) Unsteady LES adjoint solver. All other references are steady-state RANS solvers

d) Also the Germano turbulence model is implemented

e) The Wilcox $k-\omega$ turbulence model [130]

f) Early developments for this discrete adjoint solver (known as DAFoam) figure in [38, 39]

g) There are four turbulence models available [39, Tab. 1]: SA, $k-\omega$ shear-stress transport (SST) turbulence model [76], $\gamma-\tilde{Re}_{\theta t}$ transition model by Langtry and Menter [75, 63], and $k-\epsilon$

h) eDA is the EllipSys discrete adjoint solver, i.e. the present work

i) We implemented the $k-\omega$ SST turbulence model and the $\gamma-\tilde{Re}_{\theta t}$ transition model

$$\frac{\partial\mathbf{R}}{\partial\tilde{\mathbf{w}}} = \begin{bmatrix} \frac{\partial\mathbf{R}_u}{\partial\mathbf{u}} & \frac{\partial\mathbf{R}_u}{\partial\mathbf{v}} & \frac{\partial\mathbf{R}_u}{\partial\mathbf{w}} & \frac{\partial\mathbf{R}_u}{\partial\mathbf{p}} & \frac{\partial\mathbf{R}_u}{\partial\mathbf{c1}} & \frac{\partial\mathbf{R}_u}{\partial\mathbf{c2}} & \frac{\partial\mathbf{R}_u}{\partial\mathbf{c3}} \\ \frac{\partial\mathbf{R}_v}{\partial\mathbf{u}} & \frac{\partial\mathbf{R}_v}{\partial\mathbf{v}} & \frac{\partial\mathbf{R}_v}{\partial\mathbf{w}} & \frac{\partial\mathbf{R}_v}{\partial\mathbf{p}} & \frac{\partial\mathbf{R}_v}{\partial\mathbf{c1}} & \frac{\partial\mathbf{R}_v}{\partial\mathbf{c2}} & \frac{\partial\mathbf{R}_v}{\partial\mathbf{c3}} \\ \frac{\partial\mathbf{R}_w}{\partial\mathbf{u}} & \frac{\partial\mathbf{R}_w}{\partial\mathbf{v}} & \frac{\partial\mathbf{R}_w}{\partial\mathbf{w}} & \frac{\partial\mathbf{R}_w}{\partial\mathbf{p}} & \frac{\partial\mathbf{R}_w}{\partial\mathbf{c1}} & \frac{\partial\mathbf{R}_w}{\partial\mathbf{c2}} & \frac{\partial\mathbf{R}_w}{\partial\mathbf{c3}} \\ \frac{\partial\mathbf{R}_p}{\partial\mathbf{u}} & \frac{\partial\mathbf{R}_p}{\partial\mathbf{v}} & \frac{\partial\mathbf{R}_p}{\partial\mathbf{w}} & \frac{\partial\mathbf{R}_p}{\partial\mathbf{p}} & \frac{\partial\mathbf{R}_p}{\partial\mathbf{c1}} & \frac{\partial\mathbf{R}_p}{\partial\mathbf{c2}} & \frac{\partial\mathbf{R}_p}{\partial\mathbf{c3}} \\ \frac{\partial\mathbf{R}_{c1}}{\partial\mathbf{u}} & \frac{\partial\mathbf{R}_{c1}}{\partial\mathbf{v}} & \frac{\partial\mathbf{R}_{c1}}{\partial\mathbf{w}} & \frac{\partial\mathbf{R}_{c1}}{\partial\mathbf{p}} & \frac{\partial\mathbf{R}_{c1}}{\partial\mathbf{c1}} & \frac{\partial\mathbf{R}_{c1}}{\partial\mathbf{c2}} & \frac{\partial\mathbf{R}_{c1}}{\partial\mathbf{c3}} \\ \frac{\partial\mathbf{R}_{c2}}{\partial\mathbf{u}} & \frac{\partial\mathbf{R}_{c2}}{\partial\mathbf{v}} & \frac{\partial\mathbf{R}_{c2}}{\partial\mathbf{w}} & \frac{\partial\mathbf{R}_{c2}}{\partial\mathbf{p}} & \frac{\partial\mathbf{R}_{c2}}{\partial\mathbf{c1}} & \frac{\partial\mathbf{R}_{c2}}{\partial\mathbf{c2}} & \frac{\partial\mathbf{R}_{c2}}{\partial\mathbf{c3}} \\ \frac{\partial\mathbf{R}_{c3}}{\partial\mathbf{u}} & \frac{\partial\mathbf{R}_{c3}}{\partial\mathbf{v}} & \frac{\partial\mathbf{R}_{c3}}{\partial\mathbf{w}} & \frac{\partial\mathbf{R}_{c3}}{\partial\mathbf{p}} & \frac{\partial\mathbf{R}_{c3}}{\partial\mathbf{c1}} & \frac{\partial\mathbf{R}_{c3}}{\partial\mathbf{c2}} & \frac{\partial\mathbf{R}_{c3}}{\partial\mathbf{c3}} \end{bmatrix}. \tag{4.7}$$

Above, the gray shaded area is the extension of the original NS state Jacobian. This extended Jacobian is then further augmented with new rows/columns to accommodate the inclusion of turbulence and transition models.

Note that Roth and Ulbrich takes the extension a step further by introducing a residual equation for outflow scaling. This mitigates the slowdown (a factor of 2 for 8 CPUs) due to the non-local scaling effects the CPUs handling the outflow boundary condition will experience. Also, the viscosity is changed to an independent variable [104, Fig. 1a-c] which further speeds up the procedure. The reason is, that the turbulent viscosity depends on both flow and turbulence variables, resulting in a huge fill in for $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$.

At this point we invite the reader to consider the matrix in Eq. (4.7) a brief moment. For industrial scale optimizations with $\mathcal{O}(10^7)$ variables of each type this matrix is enormous. Remembering the discussion on the oneshot method from the introduction in Chapter 2 (see Fig. 2.5) where we stored the much larger KKT system for a small example problem, it is truly hard to imagine storing an industrial scale KKT system matrix to disc.

Notice, that the introduction of new independent variables is not needed for fixed-point iterations which certainly is a salient feature for the fixed-point method. Another way of discerning the two methods is that fixed-point methods very often rely on reverse mode algorithmic differentiation. The Krylov approach can however be done with forward mode algorithmic differentiation (see, e.g., [104] and the present work), but in order to achieve the transpose-matrix-vector matrix-free operations the reverse mode is of course needed [55].

Returning to the paper by Roth and Ulbrich [104], they point out, that a strictly correct implementation of the Krylov approach adjoint solver would introduce a residual for the pressure correction, $\mathbf{R}_{pc}$, and not the pressure itself, $\mathbf{R}_p$. However, they tested both implementations and found that it could be omitted given that the pressure correction is zero at convergence. They use forward mode algorithmic differentiation with operator overloading to construct $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$. This procedure is sped up by a sparsity-exploiting method [123]. Said sparsity method is akin to another Jacobian computation acceleration known as 'graph coloring' [55, Sec. 4.3]. Roth and Ulbrich report of a relative gradient error of $1 \cdot 10^{-10}$ to $1 \cdot 10^{-8}$ depending on the case. Unfortunately, the reference gradient is not machine accurate but computed with central finite differences. They describe their Krylov approach as having 'comparable efficiency' to the fixed-point method and show a runtime ratio[7] between 1 to 3 for steady-state problems. For transient cases the runtime ratio is between 3 to 8. Like all works in Tab. 4.2 they use the GMRES algorithm at some point when solving the discrete adjoint equation. GMRES is by Roth and Ulbrich used as a smoother in a multigrid scheme. To precondition GMRES they use a SIMPLE-type preconditioner for velocity and pressure fields whereas the remaining system (fluxes, temperature, turbulence, etc.) is preconditioned blockwise using the incomplete lower upper (ILU) factorization. Instead of using a parallel linear solver package such as PETSc (which is used in the other works in Tab. 4.2) they construct an efficient linear solver by using the multigrid structure of the flow solver. The resulting expression [104, Eq. 4] has a remarkable resemblance to a fixed-point iteration hinting that the two approaches ('fixed-point' and 'Krylov') might be cut from the same cloth in the end. An equivalence

[7] i.e., the time for the adjoint solver divided with the time for the flow solver

also remarked by the authors. We certainly find their solution procedure promising albeit more laborious than simply using PETSc in some form. However, given that our flow solver also makes heavy use of grid sequencing we will investigate the procedure in the near future. Impressively, although they are the only work in Tab. 4.2 to model unsteady problems they report no problems with respect to memory consumption. Thus, no checkpointing is used in their transient test case with $3.9 \cdot 10^5$ cells, but as they point out, this can be done if the need should arise. The explanation is a meticulous bookkeeping of the nonzero entries in the state Jacobian where all storage on a given CPU core is in sparse format.

The next work figuring in Tab. 4.2 is a recent publication within topology optimization [23]. They also use operator overloading but here the use of the reverse algorithmic differentiation mode is favored. Furthermore, they use the 'single-cell residual' [55, Sec. 4.3.1] approach known from previous works in the aerospace community [72, 68]. This is another way of accelerating the Jacobian computation[8]. Typically, flow solvers do not have subroutines for computing the residual in a single cell (i.e., a single control volume). This subroutine must therefore be constructed after which one applies the reverse mode algorithmic differentiation on it. The state Jacobian can now be assembled in an efficient manner by looping over all cells and extracting all non-zero entries of the Jacobian. As a result, the Jacobian can automatically be saved in optimal sparse format *and* no computations are done in vain, i.e., we never compute a residual change which turns out to be irrelevant unlike in the forward mode. Like we do in the present work, they directly assemble the transpose of the Jacobian to avoid this operation on the assembled system.

A common misconception is that the reverse mode is faster than the forward mode when computing the Jacobian. This might be true for single-cell approaches as explained just above, but in general reverse mode algorithmic differentiation is not faster than forward mode algorithmic differentiation given that the matrix is square. However, the reverse mode does allow for matrix-free operations for the adjoint equation resulting in huge memory savings which certainly is an advantage.

As pointed out by Kenway et al. [55], the single-cell approach is rarely seen given that the construction of the single-cell routine (a formidable task in itself) introduces a large amount code duplication. This complicates maintenance and makes the adjoint solver susceptible to discrepancies between the adjoint solver and the original flow solver, as it is further developed. Nonetheless, Dilgen et al. [23] present a well-functioning steady-state RANS adjoint solver. The RANS equations are complemented by either the Wilcox $k-\omega$ turbulence model [130] or the SA turbulence model. We would like to point out, that this work is the only one we found which explicitly outlines their derived pressure residual in an actual equation (see [23, Eq. 28]). Something that is *very* useful for other developers as inspiration albeit the SIMPLE implementation might differ from solver to solver. The pressure residual we will present (Eq. 7.6) is very similar, and indeed, after conferring with the authors it seems Eq. (7.6) actually is, what is implemented in their code.

[8] See [68, p. 866-867] for a concise explanation of exactly why this single-cell approach will speed of the computation of a square Jacobian

The solution process in the work by Dilgen et al. is based on the GMRES algorithm and also they use a SIMPLE-like preconditioning. However, no mentioning of multigrid techniques as in [104]. They compare the obtained gradients to those computed with central finite difference gradients and report 3-6 digits correspondence for a step size of $10^{-6}$. Furthermore, they quantify the effect of the frozen turbulence assumption and given that it sometimes even results in wrong signs altogether, one should avoid this assumption at all costs. Finally, they then document the capability to conduct topology optimization by presenting a series of 2-D and 3-D tests cases. The largest test case has a Reynolds number of 3500 and entails $1.3 \cdot 10^6$ cells.

The third work found in Tab. 4.2 is the open-source[9] adjoint solver, DAFoam, which is derived from the OpenFOAM flow solver. DAFoam has been continuously developed in recent years and various stages of the development have been documented in three papers [38, 39, 55]. In [38] there are reports of aerodynamic design cases with up to 10 million cells running on 1024 CPU cores underlining the industrial scale capabilities. Here, all partial derivative matrices are computed using the finite difference method. This adjoint solver architecture is very similar to the first adjoint solver we developed, called EllipSys discrete adjoint using finite differencing. Henceforth abbreviated as eDA*fd*. By carefully choosing the step size they ensure a reasonable gradient precision. Their developed framework is split in two major layers. One layer consists of flow solver, adjoint solver, and a graph coloring solver (to speed up Jacobian computations). The other layer contains geometric deformation routines and other components needed to conduct optimization. These two layers interact via input and output files. It is the discrete adjoint solver, DAFoam, and the graph coloring solver, that are the novelty in [38]. The other components such as mesh deformation routines have been used in various previous MDOLab publications.

Graph coloring or Jacobian coloring will be mentioned throughout this thesis, so a short explanation is in order. Coloring is a method to accelerate the computation of the partial derivatives, $\partial\mathbf{R}/\partial\mathbf{x}$, $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$, $\partial\mathbf{J}/\partial\mathbf{x}$, and $\partial\mathbf{J}/\partial\tilde{\mathbf{w}}$. Here, it is particularly the computation of the two huge partial derivative matrices $\partial\mathbf{R}/\partial\mathbf{x}$ and $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$ which must be accelerated. Taking $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$ as an example the basic idea is to perturb several states from the extended state vector, $\tilde{\mathbf{w}}$, simultaneously. Then, the resulting derivative in the residuals are computed using, e.g., finite differences, $(\mathbf{R}^* - \mathbf{R})/\epsilon$, where $\epsilon$ is a small step size. The important part is now, that the computation above will have more non-zero entries when we perturb the extended state vector, $\tilde{\mathbf{w}}$, in more than one element. We have in other words computed more of the needed entries for $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$ than we would have if we only had perturbed the extended state vector, $\tilde{\mathbf{w}}$, in one element. Readers interested in further literature on coloring can also consult the work by Nielsen and Kleb [87] which has an illustrative description [87, p. 5] of their coloring implementation. This effort is also mentioned by He et al. in [38].

Unfortunately, no exact expression for the derived pressure residual is given in any of the DAFoam papers. However, in this case we would benefit little from it given that

[9] `https://github.com/mdolab/dafoam`,(last access: 27 June 2019)

the underlying solver uses a pressure equation[10] whereas we have a pressure *correction* equation (Eq. 6.15). Again, an acceleration of the Jacobian computation is needed, to which end they employ a sparsity exploiting graph coloring solver reminiscent of the approach by [104]. The presented heuristic graph coloring solver is fully parallel and tested on Jacobians with $\mathcal{O}(10^7)$ rows using $\mathcal{O}(10^3)$ CPU cores. The solution procedure for the adjoint equation is a GMRES solver, where the additive Schwartz method (AMS) is used as a global preconditioner and all local sub-blocks are preconditioned with ILU. We have also found this combination to be effective and use similar settings in our PETSc-based solution procedure to solve the adjoint equation. He et al. [38] report of an enhanced convergence by scaling the partial derivative matrices. The paper culminates in a series of performance evaluations and aerodynamic optimization test cases showing the maturity of the developed framework. Here, the graph coloring acceleration enables computations on 10 million cell meshes with a corresponding Jacobian row size of $8.4 \cdot 10^7$. The reported number of colors ($\mathcal{O}(10^3)$) is reasonably constant across various Jacobian sizes which points to a well-functioning algorithm. However, the number is rather high compared to numbers from the related literature ($\mathcal{O}(10^2)$). They specifically point to the inclusion of turbulence in the adjoint formulation as a contributing factor. Still, the graph coloring implementation certainly renders industrial scale cases feasible. With respect to speed, they report runtimes between flow and adjoint solver on the order of unity. Furthermore, we note that the reported memory usage is rather high for the chosen solution procedure. Where the flow solver uses 2.9 GB and 25.2 GB for mesh sizes of $1 \cdot 10^6$ and $10 \cdot 10^6$ cells, respectively, the corresponding adjoint solver memory usage is 101.8 GB and 886.1 GB. In the authors' own words, the chosen adjoint implementation is 'memory bound' [38, p. 295]. In comparison, neither [104] nor [23] report of any memory problems. They also test the gradient precision albeit using finite difference reference gradients. Here, up to 6 significant digits are achieved. Finally, they identify the implementation of algorithmic differentiation derivatives as future work and point to the matrix-free solution procedure to alleviate the prohibitively high memory usage.

The second paper [39] on DAFoam is a push towards a framework which does not depend on the underlying solver type. The school of thought here is, that the authors would like to enable all of the 80+ primal solvers available in OpenFOAM. This would allow researchers to relatively quickly set up an adjoint framework for disciplines within hydrodynamics, heat transfer, or multiphase flow to name but three. They report of five flow solvers and four turbulence models [39, Tab. 1] being included in the generic framework. Still, the partial derivative matrices are computed with finite differences in [39] as outlined in [38]. In [39], the runtime ratio is now slightly higher (1.7 to 2.2). They again conclude the paper with a performance evaluation and three aerodynamic design optimizations. In the performance evaluation their adjoint solver shows a better scaling than that of the OpenFOAM flow solver. For 1536 CPU cores the parallel efficiency on a 10 M cell mesh is 65% and 86% for flow solver and adjoint solver, respectively. With respect to memory consumption the ratio between the adjoint solver and the flow solver

[10] `https://www.openfoam.com/documentation/guides/latest/doc/guide-applications-solvers-pressure-velocity-intro.html`

shows a 1146.2 GB / 73.2 GB $\approx$ 16 times higher memory usage. Although very high, this is actually much better than the previously reported [38] memory consumption where ratios around 35 could be seen. There is no explanation for this improvement and again they point to matrix-free operations as a means of reducing the memory usage.

The precision is also revisited in [39]. Here, they show 3 significant digits when comparing to finite difference reference gradients. Of their three test cases we find the second particularly interesting. Here, the $\gamma - \tilde{R}e_{\theta t}$ transition model, which we also implemented, is included in the adjoint solver. They obtain a 6.0% drag reduction by increasing the laminar region.

The final reference where DAFoam figures in is the recent publication by Kenway et al. [55]. The work revolves around the declared goal of identifying effective approaches, i.e., not just to be efficient, accurate, low on implementation cost or low on maintenance, but all of the above at the same time. Indeed, this thorough paper where they use two adjoint solvers developed at the MDOLab in various investigations is the most comprehensive adjoint solver evaluation we have found. We will however, mainly focus on the DAFoam results, as DAFoam is based on the SIMPLE algorithm. They report [55, Tab. 2] a memory consumption with a ratio of 23.7 for the standard Jacobian computation using finite differences in DAFoam which is not as good as [39] but still better than [38]. It seems the ratio is somewhat case dependent. As a new developmental milestone, they have implemented reverse algorithmic differentiation for DAFoam when running in serial. This allows them to test the memory consumption which results in a massive 53.3 in memory consumption ratio which is *higher* than the finite difference implementation (highest ratio was 35). Certainly, not what was hoped for in [39]. The explanation given, is that operator overloading in combination with the object-oriented code structure in OpenFOAM with "tens of levels of subroutines" [55, p. 33] incurs many intermediately stored variables. This can be avoided by more specific algorithmic differentiation directives stating which variables are active and which are passive. Backing up this claim is the performance of ADflow which exhibits memory consumption ratios of 1.2 and 1.8 for matrix-free and finite difference methods, respectively. Here, the matrix-free implementation uses only 67% of the memory used during a finite difference Jacobian computation. In general, the ADflow performance is very dominant. Its structured mesh approach allows for a much more compact coloring scheme than that used in the unstructured DAFoam solver (162 colors and 935 colors, respectively). This discrepancy results in a 4.6 times faster derivative computation. However, also other factors such as the increased fill in due to the Rhie-Chow interpolation factor in. These factors depend on the physics and are unavoidable in any incompressible SIMPLE-type algorithm using the Rhie/Chow interpolation.

A nice upgrade to previous works [38, 39] is that they now use machine precise reference gradients. DAFoam's reference gradient is generated from another adjoint solver [119] and ADflow uses the complex-step method via a complexified build to generate its reference gradient. We laud the authors for making the effort to use machine precise reference gradients. It really should be the standard since finite difference gradients are inherently inaccurate. We will also use the complex-step method via a complexified build to provide machine precise reference gradients as explained in Chapter 6 and 7.

In [55], the DAFoam solver shows 9 to 10 significant digits [55, Tab. 4] in correspondence between the reference gradient and the algorithmic differentiation implementation using the operator overloading approach. When using DAFoam with the finite difference Jacobian implementation they obtain 2 to 4 significant digits. Overall, we find the work by Kenway et al. [55] to be of immense importance as a future reference. They also provide platform-independent benchmarks as a means of comparing adjoint solvers. To this end, we plan to benchmark our developed discrete adjoint solver, eDA, in the immediate future.

Based on the extremely thorough adjoint solver tests Kenway et al. conclude that:

> "Overall, the Jacobian-free approach with source code transformation AD is the best option."
>
> Kenway et al. [55, p. 41]
> on the optimal discrete adjoint solver architecture

Finally, we briefly comment on the final effort figuring in Tab. 4.2, which is the present work where we developed the EllipSys3D discrete adjoint solver (eDA). Given that the remainder of this dissertation is dedicated to the development, verification and application of eDA we will be brief and only point to a few similarities/differences to the other three adjoint solvers in Tab. 4.2. As will be further explained, we have planned a very comprehensive build for eDA comprising four adjoint solver versions based on four different methods of computing the partial derivative matrices. These four methods are, i) finite differences, ii) complex-step, iii) forward algorithmic differentiation, and iv) reverse algorithmic differentiation with matrix-free operations. Of these four implementations we have at present finalized three implementations, namely: i)→iii). An overview of the four implementations can be seen in Tab. 4.3.

**Table 4.3:** Overview of adjoint solver architectures within eDA.

| | **Type** | **AD** | | **Precision** | $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}^T$ | **Implemented** |
|---|---|---|---|---|---|---|
| | | mode[a)] | method[b)] | (digits) | | |
| eDA*fd* | FD | - | - | - | assembled | ✓ |
| eDA*cs* | CS | - | - | 15 | assembled | ✓ |
| eDA*adf* | AD | forward | ST | 15 | assembled | ✓ |
| eDA*adb* | AD | reverse | ST | - | matrix-free[c)] | |

In this table we abbreviate finite difference with (FD), complex-step with (CS), and algorithmic differentiation with AD

[a)] AD mode can either be the forward mode or the reverse mode

[b)] AD method can either be operator overloading (OO) or source transformation (ST)

[c)] The only motivation for us to use reverse mode is the matrix-free approach. One can of course assemble the Jacobian using reverse mode, but to experience a speedup over eDA*adf* it would require single-cell residual subroutines, which we do not use

eDA*fd* is very similar to DAFoam in that both use finite differences to compute the derivatives and the solution procedure is similar. eDA*cs* is however not directly relatable to any of the other works in Tab. 4.2. Indeed, we know of no other SIMPLE adjoint solver like it, but we have found a counterpart [87] where the incompressibility is modelled through an artificial compressibility formulation [84][11]. However, the overall code structure is very similar to eDA*fd* and we certainly find the extra developmental effort worthwhile given that the gradients become machine accurate. The third eDA implementation, eDA*adf*, is most easily related to the work by Roth and Ulbrich [104] in that both make use of forward mode algorithmic differentiation in Fortran. However, the solution procedure is much more refined in [104].

The most dire need in the present stage of the eDA implementation is further acceleration of the Jacobian computation. In the literature review above, we have seen two ways to accelerate the Jacobian computation. The perhaps most efficient method is to form a single-cell subroutine [72, 68, 23] which effectively minimizes the amount of computations per column by applying reverse algorithmic differentiation to these special subroutines. Another less code-duplicative way is to leverage some form of sparsity exploitation [104, 38] to compute several columns in $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$ at the same time. One can of course also choose to invest further developmental efforts to enable matrix-free operations using the reverse mode algorithmic differentiation. This would obviate the need to further accelerate the Jacobian computation via coloring [55, p. 41]. However, one would still need the coloring abilities to accelerate the PC computation so at present, the next goal is to improve our coloring. Currently, we have implemented an analytical coloring scheme, that allows us to compute the inner parts of all blocks simultaneously. While this coloring scheme does account for an increasing percentage of the total mesh as the block size increases, it does not ensure a constant number of colors for increasing mesh sizes. We therefore plan to implement a complementing coloring scheme to handle the interface between mesh blocks.

**Summary:** In this opening chapter of Part II we started out by introducing the overall design of our numerical framework (Fig. 4.1). We then reviewed the available literature within discrete adjoint solvers using the Krylov approach for the SIMPLE algorithm (Sec. 4.1). Based on this review, we decided to implement a Krylov type discrete adjoint solver which we compared (Tab. 4.2) to three recent implementations. We are now ready to present the three components of our numerical optimization framework. Next chapter will describe the geometry module, FFDlib. Then follows a description of flow and adjoint solvers in the ensuing chapters.

[11] Another reference that comes to mind is a dissertation using an Euler fluid model where the partial derivatives are computed using complex-step [51]

# CHAPTER 5

# FFDlib: A stand-alone, free-form deformation library

At the onset of this project we developed a deformation module, called FFDlib, from scratch. FFDlib is a stand-alone deformation library, and has been used in publications on industrial scale fluid structure interaction (FSI) [43]. However, in the present work we will use FFDlib as an integrated part of a numerical optimization framework as described in Chapter 4 (see Fig. 4.1). An illustration of FFDlib used to produce flaps on wind turbine blades for FSI investigations is given in Fig. 5.1.

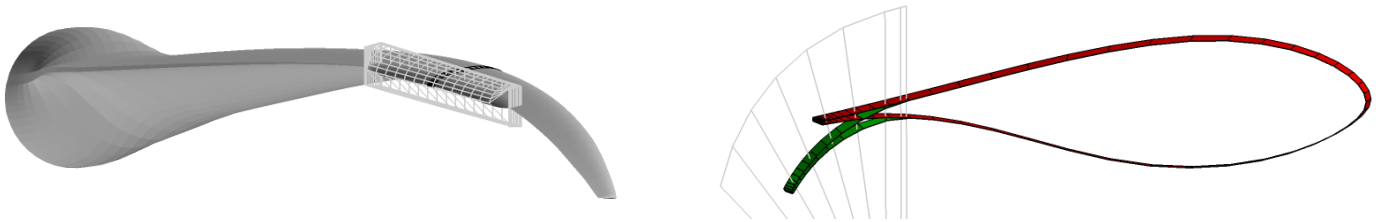


**Figure 5.1:** A flap FFD box from FFDlib (left) is seen on the outer part of a 96.2 m wind turbine blade. Black mesh lines mark a cut on the blade. The right hand side shows a zoom of a cut marked with black mesh lines on the blade to the left. The zoom shows the original geometry (red) and the deformed geometry (green) due to the movement of the FFD box control points (gray lines).

## 5.1 Free-form deformation

The underlying deformation method is Free-form deformation (FFD) [112] which originates from the field of computer graphics. Its basic principle is to wrap the geometry in a rubber-like box defined by a number of control points (CPs). Once these control points are moved, the deformation will propagate inwards to all embedded material and the geometry will be deformed accordingly. The method provides numerous salient features, of which we highlight:

- exact numerical representation of arbitrary shapes,
- independent of discrete CFD mesh, which can have arbitrary topology,
- analytical gradients,

- means to deform in a volume preserving way,
- $C^i$-continuity control for a smooth deformation transition to undeformed material.

The list is however much longer, for which we refer readers to [112, 106, 56].

The mathematical space defined inside the FFD box is a trivariate Bernstein polynomial expression [112, Eq. 2],

$$\mathbf{X}_{FFD}(s,t,u) = \sum_{i=0}^{l} \binom{l}{i} (1-s)^{l-i} s^i \left[ \sum_{j=0}^{m} \binom{m}{j} (1-t)^{m-j} t^j \left[ \sum_{k=0}^{n} \binom{n}{k} (1-u)^{n-k} u^k \cdot \mathbf{CP}_{ijk} \right] \right], \tag{5.1}$$

where $\mathbf{CP}_{ijk}$ are the $(l \cdot m \cdot n)$ control points defining the FFD box and $\mathbf{X}_{FFD}(s,t,u)$ is the resulting mesh point. The $(s,t,u)$ tuple is a set of *natural* coordinates between 0 and 1 which are valid inside the FFD box. One can view the $(s,t,u)$ coordinates as weights to correctly combine the control points to obtain a given point, $\mathbf{X}_{FFD}(s,t,u)$, in physical coordinates, $(x,y,z)$.

From Eq. (5.1) it is now evident how the analytical gradients are obtained. Since the system of equations is linear, by differentiating the expression with respect to the control points, we are left with the Bernstein coefficients, which in other words represent the analytical gradients. The obvious question based on the introduction of Eq. (5.1) is; how to map a tuple of physical coordinates, $(x,y,z)$, i.e., a mesh point, to a tuple of *natural* coordinates, $(s,t,u)$? To this end, an inverse Newton search must be carried out. As explained elsewhere [18, Eq. 7], one must solve the equation,

$$\begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} = \mathbf{0} \Leftrightarrow \mathbf{X}_{FFD}(s,t,u) - \mathbf{P} = \mathbf{0}, \tag{5.2}$$

where $\mathbf{P}$ is the mesh point we wish to find the corresponding $(s,t,u)$ tuple for. Defining Eq. (5.2) as a function, $\mathbf{f}(s,t,u)$, we wish to drive to zero, the iterative Newton search for this process in 3-D becomes:

$$\begin{bmatrix} s \\ t \\ u \end{bmatrix}^{n+1} = \begin{bmatrix} s \\ t \\ u \end{bmatrix}^{n} - \begin{bmatrix} \frac{\partial f_1}{\partial s} & \frac{\partial f_1}{\partial t} & \frac{\partial f_1}{\partial u} \\ \frac{\partial f_2}{\partial s} & \frac{\partial f_2}{\partial t} & \frac{\partial f_2}{\partial u} \\ \frac{\partial f_3}{\partial s} & \frac{\partial f_3}{\partial t} & \frac{\partial f_3}{\partial u} \end{bmatrix}^{-1} \cdot \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix}. \tag{5.3}$$

The process in Eq. (5.3) is repeated for all mesh points until the entire geometry we wish to deform has been mapped to natural coordinates.

## 5.2 Integration in a numerical optimization framework

The practical implementation of FFDlib is split up in two parts: i) a Fortran library containing subroutines which are computationally demanding to ensure high performance (4.5k+ code lines) and ii) an application programming interface (API) in Python that provides easy-to-use functions to quickly set up a system of FFD boxes (3k+ code lines). The Python side of FFDlib can access the low-level Fortran subroutines through the 'Fortran to Python interface generator' (f2py)[1].

With the developed FFDlib tool we can now assemble the first part of the numerical optimization framework from the previous chapter (Fig. 4.1). The result is seen in Fig. 5.2.

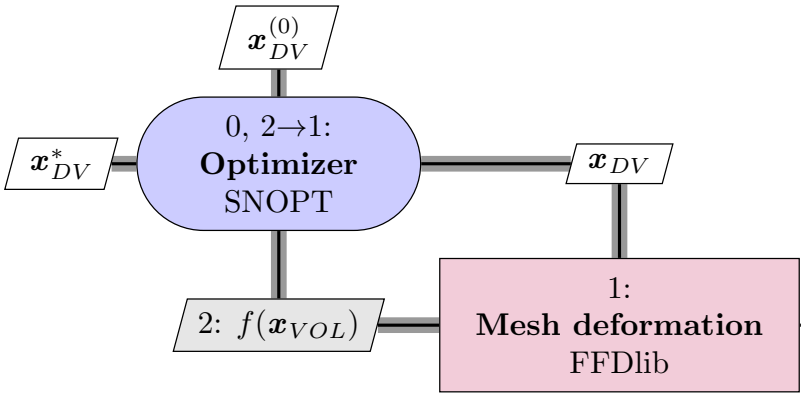



**Figure 5.2:** An optimization framework lacking both a flow solver and an adjoint solver. Gray lines show data flow. Black lines show process flow. At this point, the only optimizations we can carry out are purely geometrical, which explains why the cost function, $f(\mathbf{x}_{VOL})$, only depends on the mesh, $\mathbf{x}_{VOL}$.

### 5.2.1 Complexification

The main problem with the framework in Fig. 5.2 is of course that to conduct any interesting optimizations we must add at least a flow solver. This is dealt with in the next chapter. Then, we develop the adjoint solver and add that as well to the framework. Once both flow solver and adjoint solver have been added there are two ways we can carry out high-fidelity shape optimizations. Either we can use the flow solver for both flow computation and gradient computation, or we can use the adjoint solver for the gradient computation. FFDlib is ready to be used in conjunction with the adjoint solver, but when the flow solver will be used for gradient computation we plan to use the complex-step method. We therefore have to complexify FFDlib.

[1] `https://docs.scipy.org/doc/numpy/f2py/`,(last access: 27 June 2019)

Given that the complex-step method is used extensively, either for gradient computation or debugging we briefly outline the method. Our starting point will be the real-valued function, $f(x)$, for which we write up a Taylor series expansion about the point, $x_0$. Using an imaginary step, $ih$, this results in:

$$f(x_0 + ih) = f(x_0) + ihf'(x_0) - h^2 f''(x_0)/(2!) - h^3 f''(x_0)/(3!) + ... \tag{5.4}$$

We now truncate it above the second order. Then we take the imaginary part on both sides and finally we divide with our step, $h$. The result is:

$$Im[f(x_0 + ih)] = 0 + hf'(x_0) + \mathcal{O}(h^2). \tag{5.5}$$

Isolating the derivative, $f'(x_0)$, we arrive at the complex-step method's second-order accurate expression:

$$f'(x_0) = \mathrm{Im}[f(x_0 + ih)]/h + \mathcal{O}(h^2). \tag{5.6}$$

As seen in Eq. (5.6) we should ensure that FFDlib can handle complex variables. To complexify FFDlib one must handle both the high-level API in Python and the low-level routines in Fortran. Thankfully, it is straight forward to use complex variables on the Python side of FFDlib. For the Fortran side we ended up hardcoding `c_`-versions of the relevant subroutines. This required changes to $\mathcal{O}(10^2)$ lines in the duplicated subroutines. An example from the FFDlib Fortran source code where a subroutine has been manually complexified is seen in Fig. 5.3. As seen, we only have to change two lines in the present example. In the very first line, we change the subroutine name, `FFD_3D` → `c_FFD_3D`, and in the lowest visible line, we change `REAL*8` to `COMPELX*16`.

Importantly, the extension of FFDlib to complex variables provides us with an opportunity to verify the analytical gradients. By using the analytically provided gradient from FFDlib as the 'true' reference gradient, $f'_{ref}$, we can compute the relative error, $\epsilon$, according to:

$$\epsilon = \frac{f' - f'_{ref}}{|f'_{ref}|}. \tag{5.7}$$

Next thing we need is to choose a setup. To this end, we use the high-level Python API to quickly set up an `FFDobj`, which is the main object a design engineer would interact with.

As seen in the listing (5.1) we use the `FFDobj` type 'WT3' meaning that we are about to embed a three-bladed rotor geometry. The other Python class we instantiate in listing (5.1) is the `Boxobj`, which in turn is embedded into the `FFDobj`. The result from listing (5.1) can be inspected in Fig. 5.4.

The top level object, `FFDobj`, can own numerous `Boxobj` instantiations. In the present case, the boxes are linked so that the three blades undergo the exact same deformation.

```
SUBROUTINE FFD_3D(s,ps,pt,pu,l,m,n&
&     ,dim,pijk_def,s_coeff,t_coeff&
&     ,u_coeff,X_ffd)
! ---------------------------------
IMPLICIT NONE
! ---------------------------------
!Fortran-python inteface directives
!f2py intent(in) s,ps,pt,pu,l,m,n
!f2py intent(in) dim,pijk_def
!f2py intent(in) s_coeff
!f2py intent(in) t_coeff,u_coeff
!f2py intent(inout) X_ffd
!f2py depend(s) ps,pt,pu,X_ffd
!f2py depend(s) s_coeff
!f2py depend(s) t_coeff,u_coeff
!f2py depend(dim) pijk_def,X_ffd
!f2py depend(l,m,n) pijk_def
!f2py depend(l,m,n) s_coeff,t_coeff
!f2py depend(l,m,n) u_coeff
! ---------------------------------
! [in]
INTEGER,INTENT(IN) :: s
REAL*8, INTENT(IN) :: ps(s),pt(s),&
&     pu(s) ! pre-allocated arrays
INTEGER,INTENT(IN) :: dim
INTEGER,INTENT(IN) :: l,m,n
REAL*8, INTENT(IN) :: pijk_def(l &
&     *m*n,dim)
```

```
SUBROUTINE c_FFD_3D(s,ps,pt,pu,l, &
&     m,n,dim,pijk_def,s_coeff &
&     ,t_coeff,u_coeff,X_ffd)
! ---------------------------------
IMPLICIT NONE
! ---------------------------------
!Fortran-python inteface directives
!f2py intent(in) s,ps,pt,pu,l,m,n
!f2py intent(in) dim,pijk_def
!f2py intent(in) s_coeff
!f2py intent(in) t_coeff,u_coeff
!f2py intent(inout) X_ffd
!f2py depend(s) ps,pt,pu,X_ffd
!f2py depend(s) s_coeff
!f2py depend(s) t_coeff,u_coeff
!f2py depend(dim) pijk_def,X_ffd
!f2py depend(l,m,n) pijk_def
!f2py depend(l,m,n) s_coeff,t_coeff
!f2py depend(l,m,n) u_coeff
! ---------------------------------
! [in]
INTEGER,INTENT(IN)    :: s
REAL*8, INTENT(IN) :: ps(s),pt(s),&
&     pu(s) ! pre-allocated arrays
INTEGER,INTENT(IN) :: dim
INTEGER,INTENT(IN) :: l,m,n
COMPLEX*16,INTENT(IN):: pijk_def(l&
&     *m*n,dim)
```

**Figure 5.3:** Excerpt from the low-level Fortran source code of FFDlib exemplifying how minor manual changes can yield a complexified deformation library. Indeed, it is only the top and bottom lines that change.

```
# setup the FFDlib: we need a main FFDobj() and a Boxobj()
FFDobj = FFDobj(input_dict={'name':'MyFFDobj','type':'WT3', \
                            'FlowDir':np.array([0.,0.,1.]),'vb':0})
# here we add the box directly to the FFDobj as soon as we make it
FFDobj.add_FFDbox(Boxobj(input_dict={'dim':3,'l':8,'m':7,'n':2, \
                                     '_output':True,'vb':0, \
                                     'chord_est':Chord_,'eps':1.6e-0, \
                                     'blade_dir':2,'chord_dir':1}))
# we still need to give the FFDobj some points
FFDobj.add_pts(x,y,z) # here we insert the points into FFDobj
```

**Listing 5.1:** Code-snippet exemplifying FFDlib's user-friendly Python API.

**Figure 5.4:** A typical setup for three-bladed rotors using FFDlib for high-fidelity shape optimization. An overall class object (`FFDobj`) owns three class instantiations of a box class (`Boxobj`). The three boxes will deform simultaneously when a design engineer (or optimizer) interacts with `FFDobj`.

To test the analytical gradients below, we choose the box pointing vertically upwards in Fig. 5.4. These linked boxes are the standard box type for a full rotor setup, but there are also other box types, e.g., a flap box (see Fig. 5.1), which can be embedded in any of the outer boxes to form a nested hierarchy.

We are now ready to perform the gradient verification. To this end, we define a scalar function,

$$f(\mathbf{X}_{FFD}) = \sum_{i=1}^{3}(x_i - r_i)^4, \tag{5.8}$$

which is a measure of the distance between some test point, $\mathbf{X}_{FFD} = [x_1, x_2, x_3]^T$, generated with an FFD box and some reference point, $\mathbf{r} = [r_1, r_2, r_3]^T$, also inside the FFD box. We take the fourth power of the distance measure between the points to increase nonlinearity in the scalar function, $f(\mathbf{X}_{FFD})$. To compute an analytical gradient of $f(\mathbf{X}_{FFD})$ using FFDlib we differentiate Eq. (5.8),

$$f'(\mathbf{X}_{FFD}) = 4\sum_{i=1}^{3}(x_i - r_i)^3 \cdot \dot{x}_i$$
$$= 4[(x_1 - r_1)^3, (x_2 - r_2)^3, (x_3 - r_3)^3] \cdot [\dot{x}_1, \dot{x}_2, \dot{x}_3]^T. \quad (5.9)$$

The first vector above can be computed from the test point and the reference point. The second vector, $[\dot{x}_1, \dot{x}_2, \dot{x}_3]^T$, with the analytical gradients can be computed with FFDlib in a few lines:

```
# reset array with embedded pts (here, we embed the test point)
FFDobj.ResetPts(x=test_pt[0],y=test_pt[1],z=test_pt[2])
# obtain the FFDlib Jacobian dict
Jacobian = FFDobj.compute_derivatives()
# use basic derivative Jacobian for embedded pt(s)
xdot = np.dot(Jacobian['pts','CPs'],step_direction)
```

**Listing 5.2:** Listing showing necessary calls to FFDlib to compute the analytical gradients needed in Eq. (5.9).

In the above listing 5.2, the `xdot` is $[\dot{x}_1, \dot{x}_2, \dot{x}_3]$ from Eq. (5.9) whereas `Jacobian['pts','CPs']` in the present case is a $[3 \times (l \cdot m \cdot n \cdot 3)]$ matrix with the Bernstein coefficients which make up the basic analytical gradients in FFDlib. Finally, we should mention that `step_direction` is a Numpy array of size $[(l \cdot m \cdot n \cdot 3) \times 1]$ determining the direction we push all the control points in to provoke a change in $f(\mathbf{X}_{FFD})$. For the present gradient test we simply choose a direction at random using Numpy's `np.random.rand` function. Using the very same step direction we estimate $f'(\mathbf{X}_{FFD})$ with finite differences and the complex-step method. Finally, we can use the definition of the relative error in Eq. (5.7) to compute $\epsilon$ for forward finite difference gradients ('FWD'), central finite difference gradients ('CD'), and complex-step gradients ('CS') to see how precise the estimated gradients are compared to our FFDlib reference, $f'(\mathbf{X}_{FFD})$. The result is seen in Fig. 5.5.

The visualization of the relative gradient error in Fig. 5.5 is a prototypical sensitivity plot. For too large steps both 'FWD' and 'CD' suffer from truncation[2] errors whereas too small steps will incur increasing cancellation errors. The complex-step gradient is however completely immune to the cancellation error given that there is no cancellation for this method (Eq. 5.6). Thus, as we reduce the step size the complex-step gradient approaches the analytical gradient from FFDlib.

What if we had no analytical gradient? This is for example the case in the next chapter, when we add a flow solver (without an adjoint solver) to the framework. Could we then somehow still assess the implementation? This is indeed possible. From the Taylor expansion for forward finite differences, central finite differences and the complex-step method we know, that forward finite differences should show a first-order convergence rate, $p = 1$, whereas the two other methods should exhibit a second-order convergence

[2]The name 'truncation' error refers to the fact the we are truncating a Taylor series

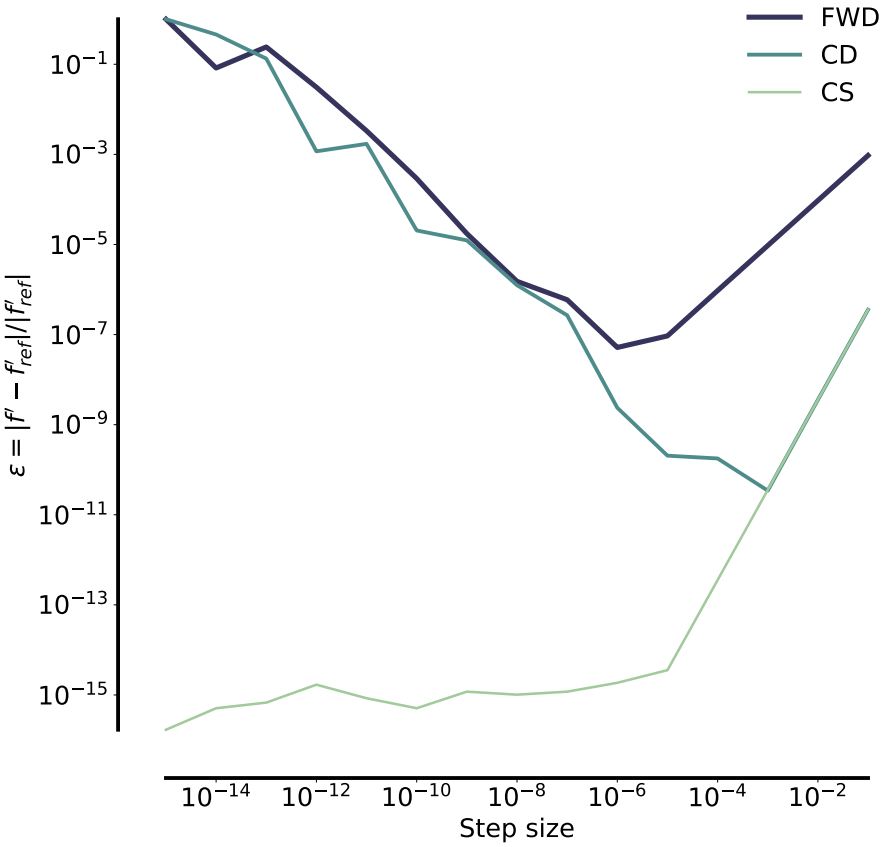



**Figure 5.5:** A gradient verification where forward finite difference (called 'FWD'), central finite difference (called 'CD'), and complex-step gradients (called 'CS') are compared to analytical gradients ($f'_{ref}$) from FFDlib (see Eq. (5.9)). The y-axis shows the relative error, $\epsilon$, from Eq. 5.7. As expected, both CD and CS show second-order convergence whereas FD shows first-order convergence (Tab. 5.1). As seen, the agreement between CS and $f'_{ref}$ approaches machine accuracy ($1. \cdot 10^{-16}$) as step size diminishes.

rate, $p = 2$. To verify the convergence rates for the relative error we show the computed results for the first four step sizes in Tab. 5.1. As seen, the findings are as predicted, and we are ready to implement a flow solver in the framework.

**Table 5.1:** Convergence rates the for step sizes: $[10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}]$ in Fig. 5.5.

| **Step size** | **Convergence rate**[a]: $p$ | | |
|---|---|---|---|
| | FD | CD | CS |
| $10^{-1}$ | 1.00 | 2.00 | 2.00 |
| $10^{-2}$ | 1.00 | 2.02 | 2.00 |
| $10^{-3}$ | 1.00 | −0.71 | 2.00 |
| $10^{-4}$ | 1.01 | −0.06 | 2.00 |

[a] The rate, $p$, is estimated with the formula: $p \approx \log(\epsilon_i/\epsilon_j)/\log(h_i/h_j)$, where $h_i$ and $h_j$ are step sizes for two subsequent relative error results, $\epsilon$, in Fig. 5.5

# CHAPTER 6
# The EllipSys3D flow solver

In the present chapter we introduce the flow solver used in the project. However, we only introduce aspects of the solver which are relevant with respect to the development of a discrete adjoint solver. A more comprehensive presentation is given elsewhere, and we shall point to references as needed.

The in-house EllipSys3D [77, 78, 114] code is developed at DTU Wind Energy and solves the incompressible RANS equations in general curvilinear coordinates using a multiblock finite volume discretization. All primitive variables, $u, v, w$, and $p$ are stored at the cell centers. This collocated mesh strategy is also used for any auxiliary variables for turbulence and transition. To couple velocity and pressure an algorithm based on the semi-implicit method for pressure-linked equations (SIMPLE) [91][1] is used and checkerboard patterns are avoided using the Rhie/Chow interpolation [100]. The code is parallelized using multiblock domain decomposition and the Message Passing Interface (MPI) library. The three main references for the code platform are [77, 78] and [114]. They describe the base solver and the multiblock platform it is built on.

Over the years it has been continuously developed leading to extensions such as overset grids [134, 133], compressibility [14], fluid-structure interaction (FSI) [41] as well as numerous turbulence and transition models.

It is by all means a comprehensive code base entailing separate, optimized solver versions for 1-D [61], 2-D, and of course 3-D. We will in the following exclusively focus on the 3-D solver, and we will only focus on the steady-state part. The flow solver in itself (i.e., not counting the herein developed adjoint solver, eDA) comprises 120K+ lines of Fortran90/95 code and we will give a brief summary of the solution procedure below.

## 6.1 The Reynolds-averaged Navier–Stokes equations

When using the Einstein summation convention, we use $u_i$ and $x_i$ in the meaning,

$$\frac{\partial u_i}{\partial x_i} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}. \tag{6.1}$$

The steady-state RANS equations reads,

[1] also the Pressure Implicit with Splitting of Operators (PISO) algorithm is implemented

$$\frac{\partial \rho u_j}{\partial x_j} = 0, \tag{6.2}$$

$$\underbrace{\frac{\partial \rho u_i u_j}{\partial x_j}}_{\text{convection}} - \underbrace{\frac{\partial \left[(\mu + \mu_t)\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right]}{\partial x_j}}_{\text{diffusion}} + \underbrace{\frac{\partial p}{\partial x_i}}_{\text{pressure}} = \underbrace{S_{\text{vol}}}_{\text{source}} \quad \text{for i } = 1, 2, 3, \tag{6.3}$$

where $\rho$ is density and $\mu_t$ is the turbulent eddy viscosity which has to be modelled using auxiliary equations for turbulence and transition. We will in the present work use the $k-\omega$ SST turbulence model possibly coupled to the $\gamma - \tilde{Re}_{\theta t}$ transition model by Langtry and Menter as previously explained (Tab. 4.2). The related transport equations are introduced later (Chapter 9) when describing the inclusion of turbulence and transition into the adjoint solver. Above, Eq. (6.2) is the continuity equation and Eq. (6.3) really is three momentum equations written in compact form using the Einstein notation.

## 6.2 Solution procedure with curvilinear coordinates

In the solution procedure we first solve for the velocity using a red/black Gauss–Seidel point solver. At this point $u, v, w$ are updated and now satisfy the momentum equations. Then a pressure correction equation is solved in an accelerated manner using multigrid techniques after which $u, v, w$ and $p$ as well as face fluxes $c_1, c_2, c_3$ are updated using the pressure correction, $p^c$. To enhance performance an outer grid sequence spanning up to 7 levels is used. The above described solution procedure can be boiled down to 2 steps for a given iteration:

[**Predictor**:] We solve momentum equations and update velocities: $u, v, w \rightarrow u', v', w'$. The updated values, $u', v', w'$, will generally not fulfill the continuity equation.

[**Corrector**:] We solve an equation for the pressure correction, $p^c$. With this correction we update all variables: $u', v', w', p, c_1, c_2, c_3 \rightarrow u'', v'', w'', p'', c_1'', c_2'', c_3''$

A very important aspect of the flow solver is that it solves these equations in curvilinear coordinates to allow for solution of flows involving complex geometries. In a similar vein to FFDlib's use of both natural $(s, t, u)$ and physical $(x, y, z)$ coordinates we will in EllipSys3D now introduce a new natural curvilinear coordinate tuple, $(\xi, \eta, \zeta)$. This coordinate transformation has a drastic effect on the RANS equations which will expand considerably in number of terms. To explain this, we consider a single partial derivative in the x-direction, $\partial/\partial x$, which (given that $x$ now is a function of the curvilinear coordinates $x(\xi, \eta, \zeta)$) expands to,

$$\frac{\partial}{\partial x} = \frac{\partial}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial}{\partial \eta}\frac{\partial \eta}{\partial x} + \frac{\partial}{\partial \zeta}\frac{\partial \zeta}{\partial x} = \frac{\partial}{\partial \xi}\xi_x + \frac{\partial}{\partial \eta}\eta_x + \frac{\partial}{\partial \zeta}\zeta_x, \tag{6.4}$$

where $\xi_x$, $\eta_x$, and $\zeta_x$ are metric terms that can be derived from the mesh (see [114, Eq. 24]). The point here is, that one partial operator in physical coordinates splits into three partial operators in the curvilinear space.

The introduction of curvilinear coordinates into the adjoint solver is relevant when considering which approach to take, i.e., should you take a continuous or a discrete approach? We find, that the continuous approach is rather laborious when using these coordinate transformations. To consider the implications of the term expansion we explicitly write out the $u$ momentum equation (Eq. 6.3) without use of the Einstein notation and using $\mu_{eff} = \mu + \mu_t$ we arrive at the expression,

$$\underbrace{\frac{\partial \rho u u}{\partial x} + \frac{\partial \rho u v}{\partial y} + \frac{\partial \rho u w}{\partial z}}_{\text{convection}} - \underbrace{\frac{\partial \left[\mu_{eff}\left(2\frac{\partial u}{\partial x}\right)\right]}{\partial x} + \frac{\partial \left[\mu_{eff}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right]}{\partial y} + \frac{\partial \left[\mu_{eff}\left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}\right)\right]}{\partial z}}_{\text{diffusion}} + \underbrace{\frac{\partial p}{\partial x}}_{\text{pressure}} = \underbrace{S_{\text{vol}}}_{\text{source}}, \tag{6.5}$$

where we have a total of 10 terms to be added or subtracted. After applying the curvilinear coordinate transformation from Eq. (6.4) to all operators in Eq. 6.5 one arrives at a rather lengthy expression which can be reduced to about 25 terms (see [114, Eq. 37]). Keep in mind, that this is just one momentum equation we took as an example. All remaining equations also including the auxiliary transport equations would undergo a similar expansion of terms. For a standard rotor simulation with the $k-\omega$ SST turbulence model and the $\gamma - \tilde{Re}_{\theta t}$ transition model this would include at least four extra transport equations. Furthermore it also affects the BC implementation. The process of deriving all the corresponding adjoint terms in the continuous approach would be considerable. While we admit that it is still feasible to use the continuous approach in the present case, we contend that re-using the code base in the discrete approach gives a head start. In the present case, this head start comes in the form of 25 years of debugging which minimizes the risk for bugs.

Given that the face fluxes, $c_1, c_2,$ and $c_3$, are not visible in the RANS Eq. (6.2-6.3) a brief comment is in order. They will emerge if we apply the curvilinear coordinate transformation to the convective term. To do so, we use the result that the metric terms, e.g., $\xi_x$, can be expressed as a differential area, $\alpha_{\xi_x}$, divided with the Jacobian of the transformation, $J$ [114, Eq. 22]:

$$\xi_x = \frac{1}{J}\alpha_{\xi_x}. \tag{6.6}$$

One can think of the differential areas, $\alpha_{\xi_x}$, as a $\xi$-face from the curvilinear space projected on to the x-plane in the Cartesian system. Given that there are three Cartesian coordinates and three curvilinear coordinates we have nine metric expressions similar to Eq. (6.6). Finally, we can write out the convective term from Eq. (6.5),

$$\begin{aligned}\frac{\partial \rho uu}{\partial x}+\frac{\partial \rho uv}{\partial y}+\frac{\partial \rho uw}{\partial z} =&\frac{\partial \rho uu}{\partial \xi}\xi_x+\frac{\partial \rho uu}{\partial \eta}\eta_x+\frac{\partial \rho uu}{\partial \zeta}\zeta_x+\frac{\partial \rho uv}{\partial \xi}\xi_y+\frac{\partial \rho uv}{\partial \eta}\eta_y+\frac{\partial \rho uv}{\partial \zeta}\zeta_y\\ &+\frac{\partial \rho uw}{\partial \xi}\xi_z+\frac{\partial \rho uw}{\partial \eta}\eta_z+\frac{\partial \rho uw}{\partial \zeta}\zeta_z\\ =&\frac{1}{J}\frac{\partial(\rho u\alpha_{\xi_x}+\rho v\alpha_{\xi_y}+\rho w\alpha_{\xi_z})u}{\partial \xi}+\frac{1}{J}\frac{\partial(\rho u\alpha_{\eta_x}+\rho v\alpha_{\eta_y}+\rho w\alpha_{\eta_z})u}{\partial \eta}\\ &+\frac{1}{J}\frac{\partial(\rho u\alpha_{\zeta_x}+\rho v\alpha_{\zeta_y}+\rho w\alpha_{\zeta_z})u}{\partial \zeta}\\ =&\frac{1}{J}\left[\frac{\partial c_1 u}{\partial \xi}+\frac{\partial c_2 u}{\partial \eta}+\frac{\partial c_3 u}{\partial \zeta}\right], \end{aligned} \tag{6.7}$$

where we used Eq. (6.4) and (6.6) to arrive at the final expression where the face fluxes, $c_1, c_2,$ and $c_3$, are seen.

We have now presented the principle behind the curvilinear transformation of the RANS equations. We will, however, not redundantly derive the entire curvilinear RANS equations as this has been done elsewhere. In short, one can derive the curvilinear momentum equations by transforming the operators as we did in Eq. (6.4) and derive the resulting terms for diffusion, pressure, and source, as we did for the convection term in Eq. (6.7). It is slightly more involved to derive an equation related to the pressure as no equation of state exists to begin with. However, an equation for the pressure correction, $p^c$, can be derived from the continuity equation, Eq. (6.2). Given that neither $p$ nor $p^c$ appear in Eq. (6.2) a short explanation is in order:

The continuity Eq. (6.2) really is a sum of fluxes,

$$\frac{\partial \rho u}{\partial x}+\frac{\partial \rho v}{\partial y}+\frac{\partial \rho w}{\partial z}=0, \tag{6.8}$$

and given that the expression is very similar to the convection term in Eq. (6.5) we can re-use the previous derivation and easily write the curvilinear counterpart,

$$\frac{\partial c_1''}{\partial \xi}+\frac{\partial c_2''}{\partial \eta}+\frac{\partial c_3''}{\partial \zeta}=0, \tag{6.9}$$

where we have used the $c''$ notation from the description of the corrector step to stress that the above expression is indeed only zero if the fluxes fulfill the continuity equation. As we use the finite volume approach, we integrate the expression of a finite volume cell and leverage the divergence theorem to obtain,

$$\begin{aligned}&\int_b^t\int_s^n\int_w^e \frac{\partial c_1''}{\partial \xi}+\frac{\partial c_2''}{\partial \eta}+\frac{\partial c_3''}{\partial \zeta}\,\mathrm{d}\xi\mathrm{d}\eta\mathrm{d}\zeta=0 \Leftrightarrow\\ &\quad c_1''|_e-c_1''|_w+c_2''|_n-c_2''|_s+c_3''|_t-c_3''|_b=0, \end{aligned} \tag{6.10}$$

where lower case letters, $e, w, n, s, t, b$, refer to faces on a six sided finite volume cell. We are now very close to realizing how Eq. (6.2) can result in an equation for the pressure correction, $p^c$. The final step is to write the double primed variables, $c''$, as a sum of the fluxes after the predictor step, $c'$, and the flux correction, $c^c$, to fulfill the continuity equation:

$$c'' = c' + c^c. \tag{6.11}$$

Above, $c'$ is the Rhie/Chow flux which is function of the primed variables, $c'(u', v', w', p', c'_1, c'_2, c'_3)$. Importantly, the correction, $c^c$, is in EllipSys only a function of $p^c$ since we simply drop the other terms (see [114, p. 41]). We will therefore explicitly write $c^c(p^c)$ and not $c^c(u^c, v^c, w^c, p^c, c^c_1, c^c_2, c^c_3)$ to stress this reduced dependency moving forward. The Rhie/-Chow fluxes will as mentioned not fulfill the continuity equation and form a source term[2],

$$c'_1|_e - c'_1|_w + c'_2|_n - c'_2|_s + c'_3|_t - c'_3|_b = S_{\text{mass}}, \tag{6.12}$$

in the resulting equation for the pressure correction, $p^c$, when we insert Eq. (6.11) in Eq. (6.10):

$$c^c_1(p^c)|_e - c^c_1(p^c)|_w + c^c_2(p^c)|_n - c^c_2(p^c)|_s + c^c_3(p^c)|_t - c^c_3(p^c)|_b = S_{\text{mass}}. \tag{6.13}$$

As seen, omitting the terms for $c^c$ not pertaining to $p^c$ has resulted in an equation for the pressure correction.

The entire curvilinear derivation of the above RANS equations is rather lengthy and takes up the majority of a PhD dissertation. We therefore refer interested readers to Chapters 1-8 in [114].

### 6.2.1 Discretized equations

Given a set of RANS equations be it in physical or curvilinear coordinates they can be discretized using the finite volume method. The discretization of the momentum Eq. (6.3) yields,

$$A_P u_P + \sum_{nb} A_{nb} u_{nb} = S_F + S_C + S_P \quad , \text{ for } nb = [W, E, S, N, B, T], \tag{6.14}$$

where we now use upper case letters, $W, E, S, N, B, T$, to indicate neighboring cell centers, whereas the lower case equivalents refer to faces, as described above. As seen in Eq. (6.14) we have split the source term in the three relevant parts for our steady-state simulations. These parts are;

[2] Once the flow solver is converged, the fluxes will (within some minor tolerance) fulfill the equation

$S_F$: A false source term due to part of the diffusive terms being treated explicitly, i.e., using values from previous iteration. The relevant terms are known as the cross-diffusive terms of the curvilinear RANS equations.

$S_C$: A part of the convective term in the RANS equations that is treated explicitly. More precisely, EllipSys has numerous differencing schemes of which we will use a first-order and a third-order upwinding scheme. Regardless of the chosen scheme we will always solely treat the nearest neighbor contributions (which is a seven cell stencil) *implicitly* whereas all other terms are treated explicitly through $S_C$ (this is later visualized in Fig. 6.1).

$S_P$: The entire pressure term in Eq. (6.3).

Turning to the pressure correction, a finite volume method discretization of Eq. (6.13) leads to,

$$A_P p_p^c + \sum_{nb} A_{nb} p_{nb}^c = S_{\text{mass}} \quad , \text{ for } nb = [W, E, S, N, B, T]. \tag{6.15}$$

Besides the above equation we will also need the update step,

$$p'' = p' + \alpha_{\text{relax}} p^c, \tag{6.16}$$

where $\alpha_{\text{relax}}$ is an underrelaxation constant. When developing the adjoint solver in the next chapter we will use Eq. (6.14) to construct the velocity residual subroutines and we will use Eq. (6.15-6.16) to construct the pressure residual subroutine. In order to construct the flux residual routines we also introduce the EllipSys update expressions for the fluxes. Taking the east face as an example the relevant expression is [114, Eq. 82],

$$c''|_e = c'|_e + A_E(p_E^c - p_P^c). \tag{6.17}$$

### 6.2.2 Computational stencils

We briefly introduce four important computational stencils before proceeding to the complexification of the flow solver. The first two stencils we introduce are used in the momentum solution process. Here, we either use the first-order upwind differencing scheme (UDS) or the third-order Quadratic Upstream Interpolation for Convection Kinematics (QUICK) scheme. Both UDS and QUICK are visualized in Fig. 6.1.

As will be evident when describing the development of the adjoint solver the differencing schemes not only affect the flow solver but certainly also the adjoint solver. An obvious reason for this connection is that the adjoint system depends on properly converged flow states which can be stencil dependent. Another important aspect is that size of the stencil directly affects the fill in of the state Jacobian. This can be detrimental both with respect to memory and with respect to stiffness of the linear system.

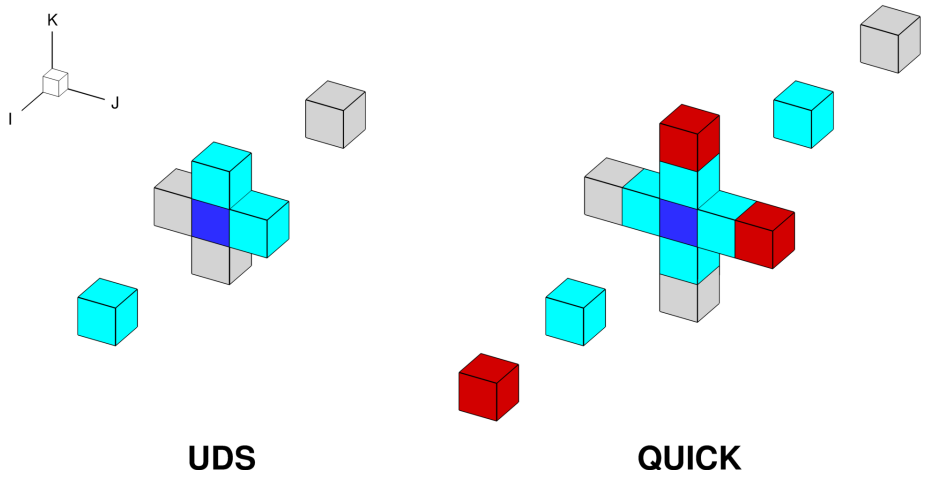



**Figure 6.1:** Visualization of computational stencils for the first-order upwind discretization scheme (UDS) and the third-order discretization scheme (QUICK). Assuming flow direction is $(-I, -J, -K)$ the color legend reads: Center cell (blue), cells with implicit and possibly also explicit influence (cyan), cells exclusively contributing with explicitly treated terms (red), and deactivated cells for the $(-I, -J, -K)$ flow direction (gray).

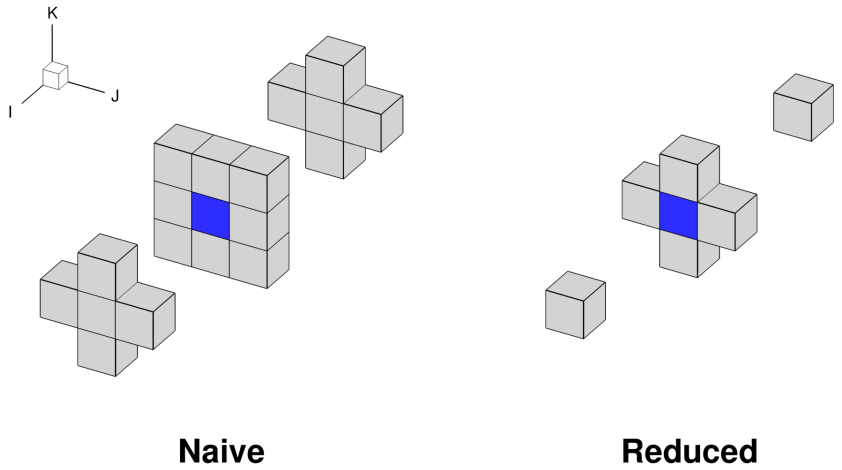



**Figure 6.2:** Visualization of two possible stencils to use when solving the pressure correction Eq. (6.15). The naive 19-cell stencil would result in a 19-banded coefficient matrix. Instead, EllipSys uses the reduced 7-cell stencil by dropping the remaining terms. This also affects the adjoint solver in that the Jacobian has a reduced fill in.

The final two stencils we present are two ways of solving the pressure correction Eq. (6.15). Following the full derivation [114, chapter 5] will lead to a 19-cell stencil. Instead of using this large molecule a 7-cell stencil is used and the remaining terms are dropped. The original 19-cell stencil and the reduced 7-cell stencil are shown in Fig. 6.2. Again, the drastic reduction has a positive effect on the memory requirements and the stiffness of the linear system. This choice of pressure correction solution method varies from solver to solver. As a result, readers will find the fill in for the pressure residual in the state Jacobian rather varying from case to case. Point in case, we mention DAFoam's flow solver stencil, which is described as a so-called level-3 stencil [38, Tab. 1], comprising 44 cells. This is much larger than our reduced stencil which has only 7 cells. Interested readers can inspect the resulting $\mathbf{R}_p$ fill in effect by comparing [38, Fig. 4(a)] to the Jacobians we will show later (e.g., in our Fig. 7.7).

## 6.3 Complexification

As alluded to in the previous chapter we strongly advocate for the use of complexification of the entire flow solver to render all reference gradients machine accurate. Developers opting for the fixed-point approach could alternatively use the exact duality preservation during development, but for the Krylov approach we find the complexification to be an absolute must in order to carry out the adjoint solver development. We cannot overstate the usefulness of the complex-step (CS) ability during the adjoint solver development. To stress that the complexification indeed should be considered a best practice, we mention the efforts by NASA Langley Research Center where prominent adjoint solvers such as FUN3D are developed. In a contemporary paper [58] they mention the automated CS approach by Martins, Sturdza, and Alonso [73] and state, that they also use a similar approach. Notably, the FUN3D code is a fixed-point adjoint solver so they could have used the exact duality. Still, they advocate for complexification to constantly test adjoint solvers. We might also add, that it is much less involved to conduct industrial scale optimizations without worrying about choosing the right step size for FD gradients. This means that if the number of design variables is moderate one can carry out the high-fidelity CFD-based optimization with the complexified flow solver instead of an adjoint solver.

It is unfortunately less straightforward to complexify a large flow solver compared to complexifying the FFDlib tool from the previous chapter. We have developed a setup where a CS-build of the EllipSys flow solver simply is one of many options in the makefile. An excerpt of the EllipSys code base after the extension to complex flow variables can be seen in Fig. 6.3.

Estimating derivatives using complex values was reported more than fifty years ago [66, 67]. It took several decades before the first Navier–Stokes solver was complexified [4]. To be specific, we us a complexification tool available from the MDOLab web page[3] [73] which has eased the implementation costs considerably when working with code bases of more that one hundred thousand lines. In Fig. 6.3 the two relevant files are `complexify.f` and

[3] http://mdolab.engin.umich.edu/content/complex-step-guide-fortran,(last access: 05 August 2019)

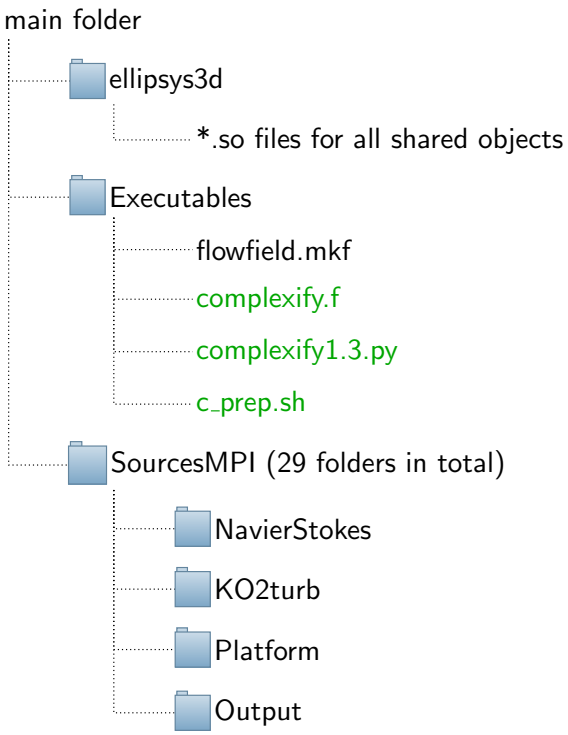



**Figure 6.3:** Visualization of the code folder structure after complexification has been implemented. Only three new files had to be added (green). However, running a complex flow solver build will automatically generate 234 complexified files tallying 100k code lines which are finally adjusted by `c_prep.sh` to be able to compile the code without errors.

`complexify1.3.py`. When prompting the makefile, `flowfield.mkf`, to build EllipSys in complex mode an internal call to the complexification tool is made. This will generate new complexified source files, where the main difference is that all real declarations are changed to complex declarations. Before compilation a correction script, `c_prep.sh`, is called to correct any parsing errors from the Python-based complexification step, before they are finally compiled. This `c_prep.sh` with $\mathcal{O}(10^3)$ lines has to be manually generated which is not too difficult albeit rather laborious. However, we stress that this is a one time effort. Once implemented, it is easily extended if needed.

Now that the flow solver is complexified, we will turn to the numerical framework and describe how we incorporate the flow solver in the overall framework. This will allow use to set up a small test optimization, verify the CS gradients, and of course run the optimization.

## 6.4 Integration in a numerical optimization framework

We chose to base our surrounding numerical framework on the Python programming language given that it is very user-friendly and has immense popularity across research communities. In an optimization context we will be exchanging data with the flow solver hundreds to thousands of times within a parallel computing environment, and to ensure that the data exchange is efficient, we require a direct memory access to the flow solver, which is written in the Fortran programming language. This is achieved through the `iso_c_binding` module[4]. Although we chose to use Python, the flow solver can in principle be inserted in any type of numerical setup. For interfacing to the flow solver from our Python optimization framework, we can use the general purpose Python framework PyEllipSys, that provides direct memory interfaces to the `iso_c_binding` interfaces defined in EllipSys3D compiled as a shared object. Specific interfaces have also been defined to the adjoint solver implemented in this work, that will be introduced in the following chapters. We developed a suite of test case tutorials which can be run on standard laptops to help users familiarize themselves with setting up shape optimizations using our framework. We will now use one of these test cases to test the numerical framework (Fig. 6.4) where a flow solver has been added. Specifically, we i) introduce the test case setup, ii) verify the CS gradients, and iii) run an optimization to inspect how the optimizer estimates gradients using complex perturbation steps, but optimizes the shape in the real-valued domain.

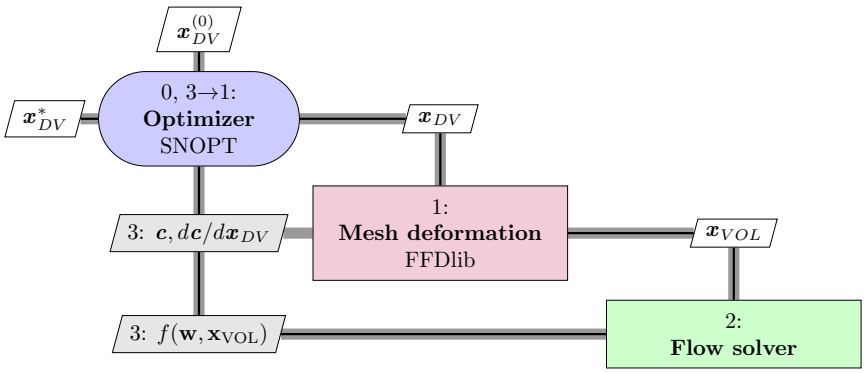



**Figure 6.4:** A high-fidelity optimization framework without an adjoint solver. Gray lines show data flow. Black lines show process flow. By adding a flow solver we can now shape optimize rotors in high-fidelity and our cost functions may depend on both flow and geometry, $f(\mathbf{w}, \mathbf{x}_{VOL})$.

[4] `http://fortranwiki.org/fortran/show/iso_c_binding`,(last access: 05 August 2019)

## 6.5 Framework evaluation with complexified solver on an extruded airfoil

The test case we will use for gradient verification as well as for shape optimization is the extruded airfoil seen in Fig. 6.5.

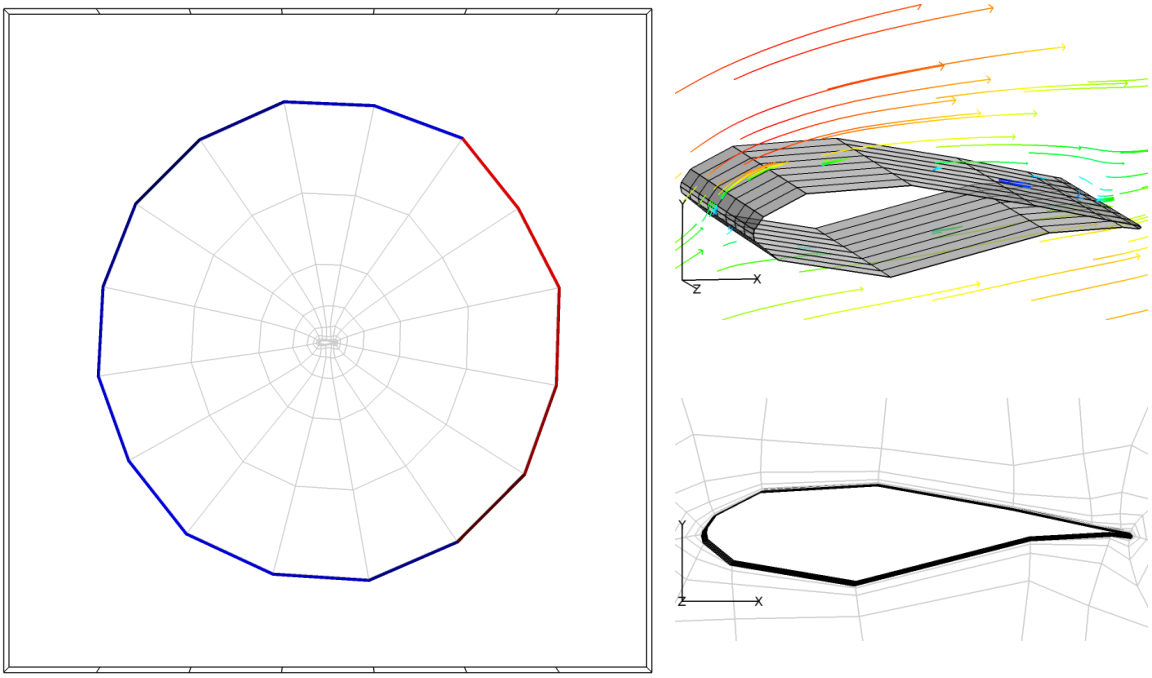


**Figure 6.5:** An overview of the extruded airfoil mesh used in the optimization (left). The chord length of the airfoil (gray) is unity and the mesh extends 16 chords to the farfield where inflow BCs (blue) and outflow BCs (red) can be distinguished. The mesh has a total of $4 \cdot 8^3 = 2048$ cells. The outer black box is the FFD box which deforms both surface and volume mesh. The right hand side shows two insets: Upper inset shows the airfoil (gray) and colored streamlines visualize the flow field. Lower inset shows the coarse mesh cells.

Due to the coarseness of the mesh it is possible to run an entire 3-D shape optimization in a few minutes on a standard laptop. We will consider a simple rotation of the airfoil to minimize the drag as our optimization test case. Formally, this is written as:

$$\begin{aligned} \min_{\theta} \quad & \mathcal{D}(\mathbf{w}, \theta), \\ \text{subject to} \quad & \mathbf{R}(\mathbf{w}, \theta) = \mathbf{0}, \end{aligned} \tag{6.18}$$

where $\mathcal{D}(\mathbf{w}, \theta)$ is the integrated drag over the airfoil and $\theta$ is the angle we rotate the mesh with about the $z$-axis. The rotation is carried out by FFDlib using the FFD box (black outer lines, Fig 6.5) where we embed both surface mesh and volume mesh. To not obtain a completely trivial solution to the optimization problem we use an angle of 20 degrees, $\theta = 20$, to impose the inflow BC, $[u, v, w]^T = [cos(\theta), sin(\theta), 0]^T$. This can also be observed in the right hand side in Fig. 6.5 in the upper inset where the flow inclination is visualized. The inflow is imposed on the blue part of the farfield in Fig. 6.5 whereas the red part is a scaled outflow BC to maintain an incompressible fluid. Finally, we note that the fluid model, $\mathbf{R}(\mathbf{w}, \theta)$, in Eq. (6.18) will be the RANS equations complemented with the SST turbulence model.

### 6.5.1 Verification of gradients using a complexified flow solver

When only a standard CFD solver and its complexified counterpart are at hand one cannot verify the CS gradient to machine precision. However, as mentioned in the previous chapter we can assess the convergence rate of the relative error. Without an adjoint solver we do not have an obvious reference gradient, so we use the gradient from smallest CS step in the sensitivity study ($h_{\text{small}} = 10^{-16}$). We now assess the $\mathrm{d}\mathcal{D}(\mathbf{w}, \theta)/\mathrm{d}\theta$ gradient precision by computing the relative error, $\epsilon$, we previously defined (Eq. 5.7) as seen in Fig. 6.6.

While Fig. 6.6 certainly is not as clear cut a text book example as the sensitivity plot from before (Fig. 5.5), it does still exhibit the important trend that the relative error of the CS gradient approaches machine zero as the step size diminishes. For large CFD solvers it is quite common to have sensitivity plots more resembling Fig. 6.6 than Fig. 5.5. Another similarity to Fig. 5.5 is, that we see the first order forward difference (FD) and the second order central difference (CD) gradients suffer from truncation errors for large step sizes, and that too small step sizes lead to cancellation errors. To check the convergence rate for the CS implementation we compute it for a few selected step sizes and find that the convergence rate is not far from the promised second-order rate, $p = 2$ (Tab. 6.1).

### 6.5.2 Shape optimization using the complex-step method

The optimization test case presented no problems executing and is summarized in Fig. 6.7. The upper graph shows the optimization history for the scaled drag cost function (gray, right) and for the design variable (black, left). As we expected for a cambered airfoil we find that the optimizer rotates the mesh slightly more than 20 degrees (nose down) to find the zero lift position resulting in the optimal drag reduction. The optimizer is in other words mirroring the rotation we imposed in the inflow BC at the start of the optimization. We also note, that the final couple of steps taken by the optimizer yields little improvement, suggesting that we could have terminated the optimization earlier.

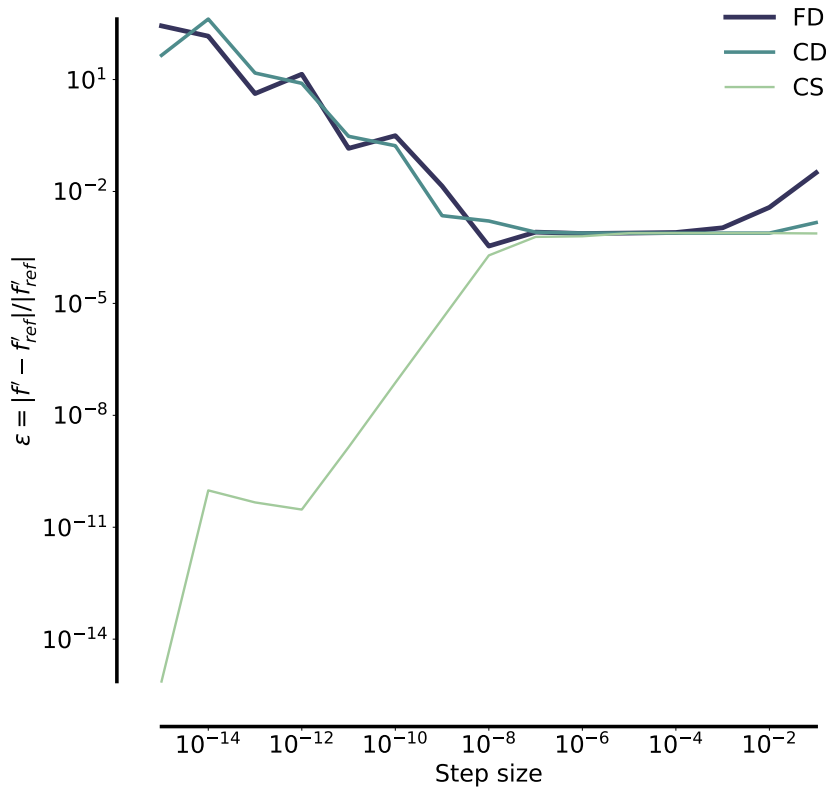



**Figure 6.6:** Step size dependency for gradient of a complexified solver.

**Table 6.1:** Convergence rates used to assess the complexification of the EllipSys flow solver.

| **Step size** | **Convergence rate**[a]: $p$ | | |
|---|---|---|---|
| $h$ | FD[b] | CD[b] | CS |
| $10^{-09}$ | – | – | 1.71 |
| $10^{-10}$ | – | – | 1.71 |
| $10^{-11}$ | – | – | 1.73 |
| $10^{-12}$ | – | – | 1.67 |

[a] The rate, $p$, is estimated with the formula: $p \approx \log(\epsilon_i/\epsilon_j)/\log(h_i/h_j)$, where $h_i$ and $h_j$ are step sizes for two subsequent relative error results, $\epsilon$, in Fig. 6.6

[b] These methods are both subjective to cancellation errors for the chosen step sizes as seen in Fig. 6.6

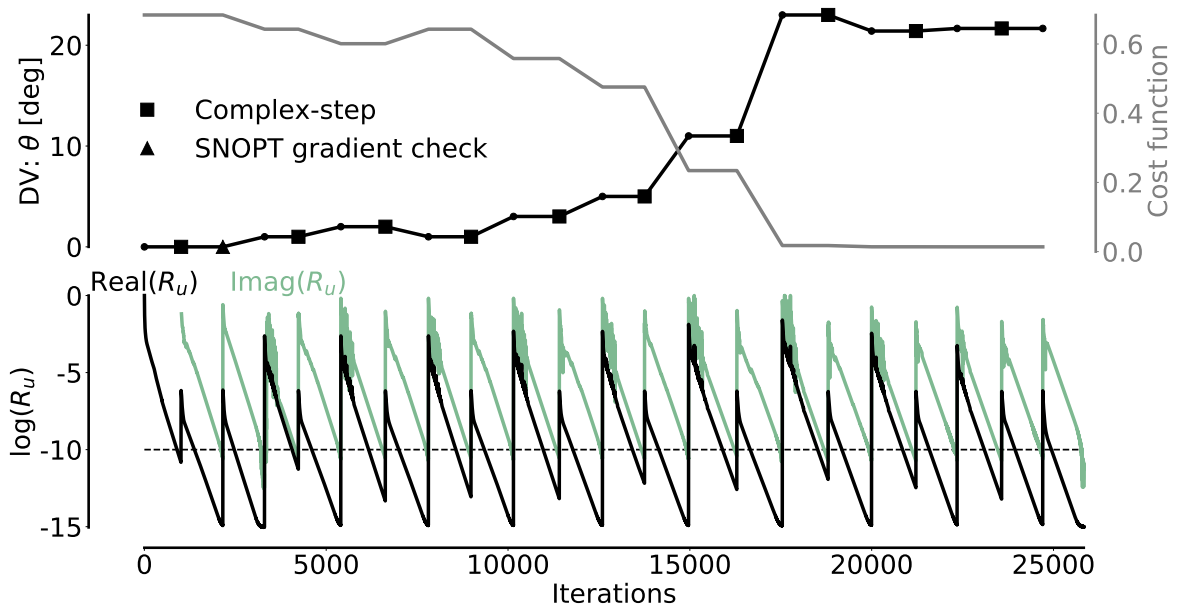



**Figure 6.7:** A summary of the optimization test case. The $x$-axis shows CFD iterations.

.

The lower graph shows the convergence history of the flow solver. Here, we chose the $\mathbf{R}_u$ as an example, but we could have chosen any of the flow, pressure, or turbulent variables. As seen, there are to parts of the residual: A real part (purple), and a imaginary part (green). One can also see a thin black dashed line which is the threshold below which EllipSys must converge all variables (real and imaginary) at each step.

The real $\mathbf{R}_u$ starts as soon as the optimizer prompts the flow solver for $f(\mathbf{w}, \mathbf{x}_{\text{VOL}})$ (see Fig. 6.4), i.e., when the optimization starts, the optimizer needs to compute the flowfield for the start mesh where $\theta = 0$. The imaginary part of $\mathbf{R}_u$ is however first seen when the optimizer makes a complex perturbation of $\theta$ to determine the gradient. These complex perturbations are marked with black squares (in the upper plot). For every complex perturbation the (now complex) angle is sent to FFDlib which sends a mesh with a complex perturbation, $\mathbf{x}_{\text{VOL}}$, to EllipSys which in turn computes a complex-valued flowfield. It is noteworthy, that the real-valued $\mathbf{R}_u$ hardly rises when we take a CS step. Only during the line searches do we see the real part rise to about the same level as the imaginary residual part.

As seen, the complex-step gradients are fully functional and we can proceed to use them as part of the verification of the adjoint solver. Likewise we will use our complex-step gradients in the final case studies presented in Chapter 11 to compare with the performance of the adjoint solver.

# CHAPTER 7
# eDA: The EllipSys discrete adjoint solver

This chapter presents the development of the core adjoint solver in great detail. It is an arduous task to develop an adjoint solver from scratch. Particularly CFD solvers based on the SIMPLE algorithm may present a formidable challenge. To substantiate this claim we bring two quotes from experienced developers of discrete adjoint solvers for flow solvers with the SIMPLE algorithm:

> "...implementing the discrete adjoint method for a partial differential equation (PDE) based primal solver is a time-consuming task, requiring a similar amount of development time and effort as the primal solver."
>
> He et al. [39, p. 1],
> on the development cost of a discrete adjoint solver

> "Despite its mind-boggling potential, the widespread utilization of the adjoint method in the field of CFD has been hindered by its innate implementation difficulties"
>
> Auvinen [5, p. 16],
> developmental lead on a fixed-point AD-based adjoint for a SIMPLE algorithm

To encourage other researchers in this endeavor we present a road map we find to be particularly helpful. Indeed, we believe that the presented systematic road map is key to achieving a successful implementation.

The developed discrete adjoint solver is based on the Krylov approach, which we introduced in Sec. 4.1.2. It is particularly for this adjoint solver type, which is based on a residual subroutine written by hand, that we find the road map useful. However, developments of other types of discrete adjoint solvers can certainly also benefit from the presented road map. A focal point throughout the road map is to nurture the intuition and visually inspect the various parts of the adjoint equation as well as the total derivative equation. Below, one will, e.g., find visualizations (Fig. 7.5 and 7.1) of the partial derivative fields created with the residual subroutine. This ensures not just a systematic approach, but also an intuitive and efficient way to develop and debug code. In a sense, it is the most important chapter of the entire thesis as everything else hinges upon the ability to produce machine accurate gradients for the NS equations.

The chapter opens with the derivation of the seven residual subroutines using a flow solver based on the SIMPLE algorithm (Sec. 7.1). Then follows a 'road map' section (7.2) with a description of a systematic way to develop a discrete adjoint solver from scratch. Here, we present two discrete adjoint solver architectures: One is based on FD partial derivative matrices (eDA*fd*) and another is based on CS partial derivative matrices (eDA*cs*). They both have advantages and disadvantages, some of which we highlight along the way. Once all steps of the road map are complete we have successfully developed two functioning adjoint solvers. We back this up (right hand side table in Fig. 7.9) by verifying gradients using a machine accurate CS gradient as reference. This allows us to present up to 6 and 16 significant digits of correspondence in gradient precision for eDA*fd* and eDA*cs*, respectively. The ensuing section (7.3) describes how to transition from the developed aerodynamic design variables to shape design variables. Also here we document the adjoint solver's gradient precision resulting in up to 5 and 16 significant digits for eDA*fd* and eDA*cs*, respectively. Overall, we find that the extra effort needed to implement the CS-based adjoint solver certainly is worth it, since it leads to machine accurate gradients, which is something the FD-based adjoint solvers simply cannot provide.

The final section (7.4) describes how to insert the developed adjoint solver into a numerical optimization framework. To test the framework we quickly set up a small multipoint aerodynamic optimization where gradient-based optimization is achieved through analytical gradients provided by the adjoint solver. The multipoint optimization also includes constraints and results in the expected outcome, thus suggesting a correct implementation.

## 7.1 Derivation of the residuals in a SIMPLE algorithm

The three velocity residual subroutines are by far the most straightforward ones to derive. Using the expression for the discretized momentum equations (6.14) we get,

$$\mathbf{R}_u = (S_F + S_C + S_P) - A_P u_P + \sum_{nb} A_{nb} u_{nb} \quad , \text{ for } nb = [W, E, S, N, B, T], \tag{7.1}$$

$$\mathbf{R}_v = (S_F + S_C + S_P) - A_P v_P + \sum_{nb} A_{nb} v_{nb} \quad , \text{ for } nb = [W, E, S, N, B, T], \tag{7.2}$$

$$\mathbf{R}_w = (S_F + S_C + S_P) - A_P w_P + \sum_{nb} A_{nb} w_{nb} \quad , \text{ for } nb = [W, E, S, N, B, T], \tag{7.3}$$

where we refer to Sec. 6.2.1 for further details on the discretization. Eq. (7.1-7.3) should present no problems writing residual routines for by hand. We comment on this process further in the next section (7.2) as it is the very first step of our road map. An example of the resulting Fortran code for Eq. (7.3) is seen in Fig 7.1.

An expression for a pressure residual subroutine can be derived from its correction. This was already stated in [104] although no actual expression was found. However, by using the pressure update expression from Eq. (6.16) we arrive at an initial expression:

```
1          do n=1,BPP;do k=2,b1;do j=2,b1; do i=2,b1
2 !      ResW = Source - Ap*Wp + Sum[Anb * Wnb]
3               ResW(i,j,k,n)= +( &
4       &            tmp_swp(i,j,k,n)-tmp_apwi(i,j,k,n) &
5       &            *w(i,j,k,n) &
6       &            -tmp_awm(i,j,k,n)*w(i-1,j,k,n) &
7       &            -tmp_aem(i,j,k,n)*w(i+1,j,k,n) &
8       &            -tmp_asm(i,j,k,n)*w(i,j-1,k,n) &
9       &            -tmp_anm(i,j,k,n)*w(i,j+1,k,n) &
10      &            -tmp_abm(i,j,k,n)*w(i,j,k-1,n) &
11      &            -tmp_atm(i,j,k,n)*w(i,j,k+1,n) &
12      &            )
13         end do; end do;end do;end do
```

**Figure 7.1:** Illustration of how Eq. (7.3) may be coded.

$$\mathbf{R}_p = p'' - (p' + \alpha_{\text{relax}} p^c), \tag{7.4}$$

which can be rephrased to,

$$\mathbf{R}_p = p'' - (p' + \alpha_{\text{relax}}(S_{\text{mass}} - \sum_{nb} A_{nb} p^c_{nb})/A_P) \quad , \text{ for } nb = [W, E, S, N, B, T], \tag{7.5}$$

by inserting $p^c_p$, which can be isolated from Eq. (6.15) for the discretized pressure correction. Remembering that both $p^c_p \to 0$ and $p'' \approx p'$ at convergence we arrive at the final expression for the pressure residual subroutine:

$$\mathbf{R}_p = -\alpha_{\text{relax}}(S_{\text{mass}})/A_P. \tag{7.6}$$

Here, $\alpha_{\text{relax}}$ is an underrelaxation constant, $A_p$ is an influence coefficient, and $S_{\text{mass}}$ is the mass source made up of the Rhie/Chow fluxes (see Eq. (6.12)). Given that the Rhie/Chow fluxes enter in the expression we can already at present state that the state Jacobian's fill in should be considerably higher for pressure residuals than for velocity residuals. The expression seen in Eq. (7.6) is quite close to the only other expression for $\mathbf{R}_p$ for SIMPLE adjoint solvers we have found reported elsewhere (see [23, Eq. 28]). We thank the authors for helpful discussions on this matter.

Finally, we must derive residual routines for the fluxes. Again, we have found this best covered by Dilgen et al. [23]. To derive the flux residual expression we use the flux update from Eq. (6.11) and the fact that the flux correction $c^c \to 0$ at convergence to obtain:

$$\mathbf{R}_{c_i} = c''_i - c'_i \quad , \text{ for } i = [1, 2, 3]. \tag{7.7}$$

Again, the $c_i'$ is the Rhie/Chow flux and will result in a rather extensive fill in for the related rows in the state Jacobian. The $c_i''$ is the 'true' flux, that fulfills the continuity equation. Readers should notice the subtle difference when comparing Eq. (7.7) to the single-cell residual approach used by Dilgen et al. [23]. In the single-cell approach the sum is on all six faces for a given finite volume cell. Here, however, we care not for the individual cells and simply sum over the three main curvilinear directions (the $c_1 = c_\xi$, $c_2 = c_\eta$, and $c_3 = c_\zeta$ introduced in Eq. (6.7) back in Chapter 6). This difference is also seen when writing up the extended state vector for the two approaches. As mentioned in Chapter 4 we extend the state vector to, $\tilde{\mathbf{w}} = [\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{p}, \mathbf{c1}, \mathbf{c2}, \mathbf{c3}]^T$, for the standard Krylov approach. However, for the single-cell Krylov approach the extended state vector becomes, $\tilde{\mathbf{w}} = [\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{p}, \mathbf{f_1}, \mathbf{f_2}, \mathbf{f_3}, \mathbf{f_4}, \mathbf{f_5}, \mathbf{f_6}]^T$, as seen in, e.g., [23, Eq. 25].

Now, that we have expressions for all seven residual subroutines, Eq. (7.1-7.3) $\rightarrow$ $\mathbf{R}_u, \mathbf{R}_v, \mathbf{R}_w$, Eq. (7.6) $\rightarrow$ $\mathbf{R}_p$, and Eq. (7.7) $\rightarrow$ $\mathbf{R}_{c_1}, \mathbf{R}_{c_2}, \mathbf{R}_{c_3}$, we are ready to proceed to the road map, which takes us through all necessary steps in the implementation of a discrete adjoint solver.

## 7.2 An adjoint road map

Our suggested road map for a systematic developmental procedure of a discrete adjoint solver has seven steps as seen in Fig. 7.2.

$$\mathbf{R} \rightarrow \frac{\partial \mathbf{R}}{\partial \mathbf{x}} \rightarrow \frac{\partial \mathbf{R}}{\partial \tilde{\mathbf{w}}} \rightarrow \frac{\mathrm{d}\tilde{\mathbf{w}}}{\mathrm{d}\mathbf{x}} \rightarrow \frac{\partial \mathbf{J}}{\partial \tilde{\mathbf{w}}} \rightarrow \boldsymbol{\Psi} \rightarrow \frac{\partial \mathbf{J}}{\partial \mathbf{x}}$$

**Figure 7.2:** Road map for adjoint solver development.

There is a *very* specific order in which we recommend completing the above steps. Before going over the steps we recall the total derivative Eq. (2.11),

$$\frac{\mathrm{dJ}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\mathrm{d}\mathbf{x}_{\mathrm{DV}}} = \frac{\partial \mathrm{J}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\partial \mathbf{x}_{\mathrm{DV}}} - \underbrace{\frac{\partial \mathrm{J}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\partial \tilde{\mathbf{w}}} \overbrace{\left[\frac{\partial \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\partial \tilde{\mathbf{w}}}\right]^{-1}}^{\mathrm{d}\tilde{\mathbf{w}}/\mathrm{d}\mathbf{x}_{\mathrm{DV}}}}_{=\boldsymbol{\Psi}^T} \frac{\partial \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\partial \mathbf{x}_{\mathrm{DV}}}, \tag{7.8}$$

the tangent-linear Eq. (2.9),

$$\frac{\partial \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\partial \tilde{\mathbf{w}}} \frac{\mathrm{d}\tilde{\mathbf{w}}}{\mathrm{d}\mathbf{x}_{\mathrm{DV}}} = -\frac{\partial \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\partial \mathbf{x}_{\mathrm{DV}}}, \tag{7.9}$$

and the adjoint Eq. (4.1),

$$\left[\frac{\partial \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\partial \tilde{\mathbf{w}}}\right]^T \boldsymbol{\Psi} = \left[\frac{\partial \mathrm{J}(\tilde{\mathbf{w}}, \mathbf{x}_{\mathrm{DV}})}{\partial \tilde{\mathbf{w}}}\right]^T, \tag{7.10}$$

since they are good to have fresh in memory for this chapter. For the total derivative Eq. (7.8) we also visualized where the tangent-linear solution ($\mathrm{d}\tilde{\mathbf{w}}/\mathrm{d}\mathbf{x}$) or the adjoint solution ($\mathbf{\Psi}$) would be substituted into the equation.

Returning to the presented road map in Fig. 7.2 the first step ($\mathbf{R}$) is to implement and verify the seven residual subroutines. Then, we implement subroutines to compute the two partial matrices needed to solve the tangent-linear system from Eq. 2.9 (step 2 to 4). In step 5 and 6 we implement subroutines to compute the $\partial\mathbf{J}/\partial\tilde{\mathbf{w}}$ matrix and solve the discrete adjoint system from Eq. (4.1) to obtain the adjoint variable, $\mathbf{\Psi}$. In step 7 we implement subroutines to compute the $\partial\mathbf{J}/\partial\tilde{\mathbf{x}}$ matrix. This term may be zero depending on the chosen design variable. However, for shape optimization $\partial\mathbf{J}/\partial\tilde{\mathbf{x}}$ will most certainly be needed, and we describe this process in Sec. 7.3. The reason, that we advocate for the implementation of a tangent-linear solver before making the adjoint solver is that the tangent-linear solution vector, $\mathrm{d}\tilde{\mathbf{w}}/\mathrm{d}\mathbf{x}$ has an obvious physical interpretation. This is also true for the adjoint solution vector, $\mathbf{\Psi}$, but it may be more difficult to interpret. Indeed, since we already complexified our flow solver we can leverage the CS method to obtain a machine accurate reference for $\mathrm{d}\tilde{\mathbf{w}}/\mathrm{d}\mathbf{x}$ by perturbing $\mathbf{x}$ in the complex plane (as explained further below the design variable, $\mathbf{x}$, is initially taken to be the inflow BC, but later we handle shape design variables). The key to the above road map is that each step can be verified before proceeding to the next step. This helps isolating errors and reduces the time spend debugging.

Before diving in to the various steps of the road map, we present an overview (Fig. 7.3) of the main additions (green) we had to make to our code base in order to develop the adjoint solver.

As seen in Fig. 7.3, the seven residual subroutines each have their own file (`adj_Res_u.f` → `adj_Res_c3.f`) whereas we gather all partial derivative subroutines in the `adj_partials.f` file. We also had to implement some form of solver to solve either the tangent-linear system or the adjoint system. Here, the `PETSc_Solver.F` contains all the relevant interface subroutines needed for PETSc (which is a C-based application). We comment on various options for this solution procedure in Chapter 4 and refer readers to this chapter for further details. While on the topic of interfaces we mention that `adj_ext_interface.f` is the external interface containing `iso_c_binding`s needed to implement the adjoint solver in our numerical optimization framework via PyEllipSys. Finally, there is the `eDAcs_prep.sh` script, which is a correction script necessary to compile regular and complexified code into one unified object (i.e. the eDA*cs* adjoint solver). This script can be seen as an extension to the `c_prep.sh` script we developed to complexify the flow solver (see Chapter 6). With the use of the road map we arrive at two functional adjoint solvers, with which we can verify gradients and perform basic optimizations for validation purposes.

### 7.2.1 R-step

The very first step in the road map is to develop the global residual subroutines that take in a *converged* extended state vector, $\tilde{\mathbf{w}} = [\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{p}, \mathbf{c1}, \mathbf{c2}, \mathbf{c3}]^T$, and compute a residual vector, $\mathbf{R}$. To be specific, we choose the $w$ variable as an example. The

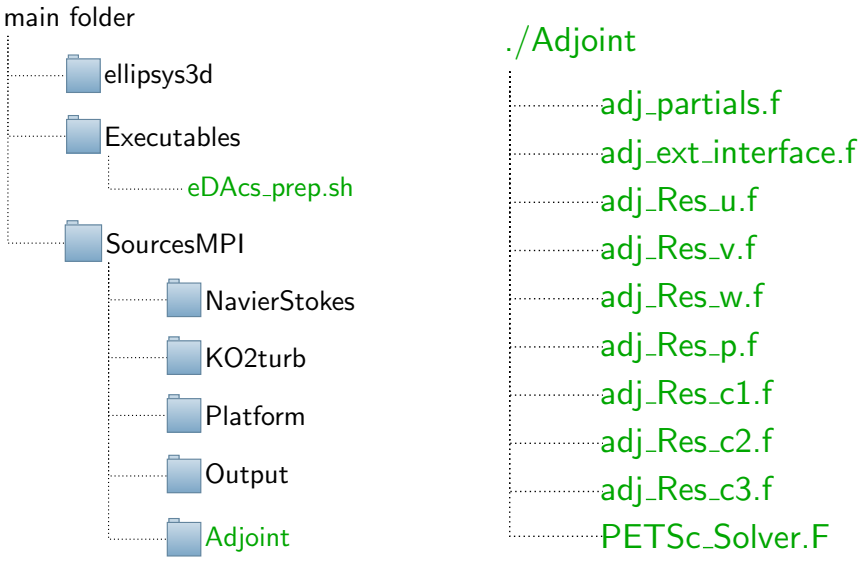



**Figure 7.3:** Visualization of the code folder structure after an adjoint solver has been implemented. Files in green were added in this process. The left hand side shows an extension of Fig. 6.3 showing the EllipSys code structure. The right hand side shows contents of the added `./Adjoint` folder.

constructed $\mathbf{R}_w(\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{p}, \mathbf{c1}, \mathbf{c2}, \mathbf{c3})$ subroutine assembles all influence coefficients, $A_P$ and $A_{nb}$, from Eq. (7.3) as well as the source terms, $S_F + S_C + S_P$, to finally compute $\mathbf{R}_w$. Now, the `tmp_` prefix for, e.g., `tmp_awm` seen in the code snippet from Fig. 7.1 makes more sense: It is a temporarily (`tmp_`) computed influence coefficient (`a` for $A_{nb}$) for the west neighbor (`w`) in the momentum equations (`m`). Hence, the coded name: `tmp_`+`a`+`w`+`m` = `tmp_awm`. To be sure, the $A_{nb=W}$ influence coefficient is already available in the flow solver, but we are not only interested in its value later on. We will also be interested in its change (either a finite difference or complex-step perturbation). Therefore, we need the ability to compute every single coefficient since a change in any of these coefficients might very well result in a change in the residual value.

Once all seven residual subroutines have been implemented there are two important tasks in this first road map step. First, a meticulous verification procedure should be made. This is slightly more involved for the SIMPLE algorithm than for compressible solvers due to the updates of $u$, $v$, and $w$ variables after the predictor step. We chose to run simulations with our adjoint residual subroutines inserted in the flow solver and check that the difference between, e.g., the flow solver $u$-residual and $\mathbf{R}_u$ was less than $10^{-16}$ at every single iteration. For transient simulations with a second-order backwards Euler discretization this necessitates storing six copies of all variables to check a given time step. These six copies are due to the three time steps $t = n$, $t = n-1$, and $t = n-2$, where storage is needed both before the predictor step and before the corrector step.

Although a laborious process, it is conceptually straightforward.

The second task in this first road map step is more subtle. Now, that we are 100% sure that we mirror our flow solver, we will likely need to diverge from the flow solver behavior. There are countless of examples to give here, such as a possible BC enforcement in the flow solver midway through the residual procedure to enhance convergence. This will likely kill off some of the BC perturbations we send through our adjoint residual routines and steps must be taken to ensure this does not happen. Another example is due to the stencils that have a direction, such as UDS (see the left hand side in Fig. 6.1). For the FD-based adjoint solver, eDA*fd*, this presents a challenge since a perturbation albeit using a small step size of $h = 10^{-6}$ might change the direction of the stencil. This will result in a wrong fill in pattern for the state Jacobian and should not be allowed. Remember, the state Jacobian matrix, $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$, actually is an operator that is a *linearization about a particular converged solution*, $\tilde{\mathbf{w}}$. Therefore, everything including the direction of upwinded stencils, must be exactly like they were in the converged state of the flow solver.

A final thing to do in this first road map step is to choose a minimal mesh which is used to develop the adjoint solver. The mesh should be as small as possible so that the entire state Jacobian can be written out and analyzed if needed. This often means taking your stencil with highest order, and fitting a mesh around it. Furthermore, it is helpful that the mesh resembles a setup, that can become industrially relevant through simple upscaling. In the present case, we choose the minimal extruded single-block airfoil mesh with $8^3 = 512$ cells seen in the left hand side of Fig. 7.4. Notice, it has both inflow (blue) and outflow (red) BC zones, meaning that the mesh through simple upscaling could be used to study, e.g., transition on an airfoil section assuming that the adjoint solver can handle the relevant physics. When deciding on a mesh, it is *crucial* that the setup can be converged to machine precision as we show in the right hand side of Fig. 7.4. This is indeed solved to machine accuracy which for our solver is $10^{-16}$. In EllipSys the $\mathbf{R}_u$ is summed over all cells (in the adjoint solver we can of course compute the residual value in each cell). In a mesh with $8^3 = 512$ cells we typically want to converge the problem below an average cell-wise residual error of $10^{-16}$ so in Fig. 7.4 we aim at the threshold $\text{Log}_{10}(512 \cdot 10^{-16}) = -13.3$ which we are certainly below. We have kept this convergence format throughout the present work so that developers from the broader user base in the industry may easily reproduce all plots. If the shown setup is too complex which we have seen for some non-curvilinear CFD solvers, then the developer must choose a simpler setup such as a square pipe.

### 7.2.2 $\frac{\partial\mathbf{R}}{\partial\mathbf{x}}$-step

Turning to the second road map step it is time to compute the first partial derivative matrix, $\partial\mathbf{R}/\partial\mathbf{x}$. To do so, one must decide on $\mathbf{x}$, the design variable. Here, we choose the inflow BC for $u$ which is aligned with the chord axis for the idealized airfoil (gray) in Fig. 7.4. The key here is to choose a design variable that can easily be interpreted. This step results in at least two subroutines being inserted in the code base. First, we need to define a `subroutine Calc_dRdX(u,v,w,p,c1,c2,c3,dRdX)`. Secondly, a perturbation

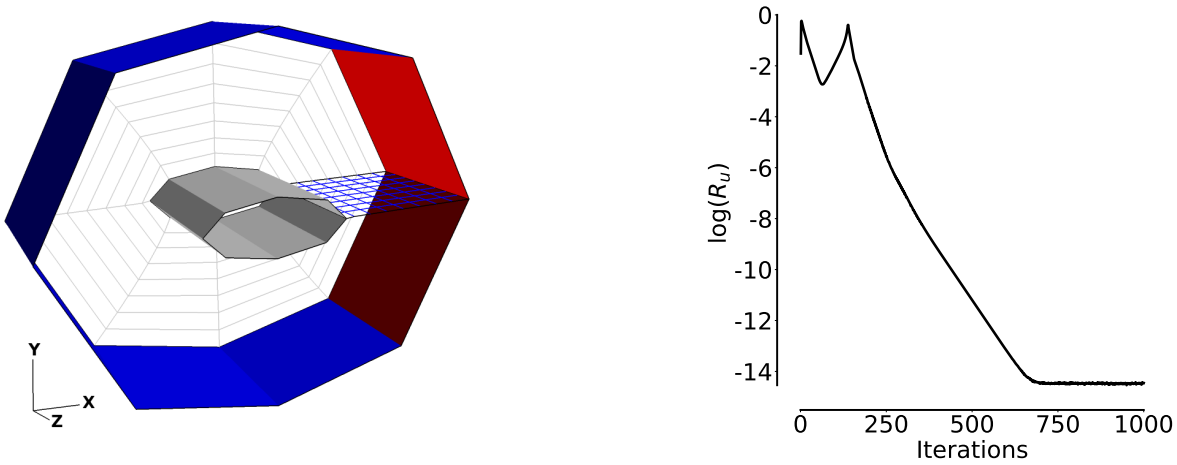



**Figure 7.4:** The left hand side shows a minimal one-block mesh to develop the adjoint solver with airfoil (gray) and inflow (blue) and outflow (red) BC zones. The blue grid shows cell sizes and mark the interface where the block wraps around to form the airfoil mesh. The right hand side shows a $u$-residual exemplifying that EllipSys can converge the mesh to machine accuracy.

mechanism for the BC layer perturbation is needed, and here it might be necessary to change the original flow solver implementation. This was the case for EllipSys, where we in the new `subroutine adj_SetBoundaryFlow` can add small perturbations instead of simply overwriting the values.

Once the residual subroutine and perturbation subroutine are in place, the $\partial\mathbf{R}/\partial\mathbf{x}$ matrix can be computed. We will now explicitly state how entries in partial derivative matrices are computed. Although we are still at the $\partial\mathbf{R}/\partial\mathbf{x}$-step in the road map we will use the $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$ matrix in this example as it is a slightly more tricky partial derivative matrix to compute. A given element in the Jacobian can for eDA*fd* be computed as,

$$\left(\frac{\partial\mathbf{R_u}}{\partial\mathbf{p}}\right)_{i,j} \approx \frac{\mathbf{R}_{u+\epsilon\mathbf{e}_i} - \mathbf{R}_u}{\epsilon}, \tag{7.11}$$

where $\epsilon$ is a small step size and $\mathbf{e}_i$ is unit vector with zeroes everywhere except a one in the i'th element. As seen, we chose the change in $\mathbf{R}_u$ with respect to a perturbation in $\mathbf{p}$ as an example. Notice that Eq. 7.11 is a sub-matrix of the total $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$ matrix. The placement of the sub-Jacobian in the $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$ Jacobian was explained in Chapter 4 (Eq. 4.7) and it will be further discussed below (Fig. 7.6). The relevant expression for the eDA*cs* version based on the complex-step method (Eq. 5.6) becomes,

$$\left(\frac{\partial\mathbf{R_u}}{\partial\mathbf{p}}\right)_{i,j} \approx \frac{\mathbf{R}_{u+\epsilon_2\mathbf{e}_i}}{\epsilon}, \tag{7.12}$$

where $\epsilon_2 = \sqrt{-1} \cdot \epsilon$ (the $i$ symbol was used for selecting element $(i, j)$).

Once the two subroutines[1] are in place at least a visual inspection should be made to ensure a correct implementation. We will as an example inspect $\mathbf{R}_u$ but checks should be made to all seven residuals in the extended residual vector; $\mathbf{R} = [\mathbf{R}_u, \mathbf{R}_v, \mathbf{R}_w, \mathbf{R}_p, \mathbf{R}_{c1}, \mathbf{R}_{c2}, \mathbf{R}_{c3}]^T$. We have of course also had to employ other correction methods from time to time such as the basic analytic cross-check where the exact perturbation value is computed by hand but for such a low-level procedure we would recommend completely orthogonal meshes such as a channel or a square duct. However, these procedures make for poor reading material, which explains why we have opted for the visual checks in the following.

For $\partial\mathbf{R}_u/\partial\mathbf{x}$, where $\mathbf{x}$ is the $u$-inflow at the boundary, we would intuitively expect only the outermost layer of cells to be non-zero when plotting the $\partial\mathbf{R}_u/\partial\mathbf{x}$-field. This can easily be verified as seen in Fig. 7.5 where we have superimposed the thresholded $\partial\mathbf{R}_u/\partial\mathbf{x}$-field on the now transparent mesh setup from Fig. 7.4. As seen, we only have non-zero values in the outer most cell layer. The $\partial\mathbf{R}/\partial\mathbf{x}$ matrix is a nice starting point since it has only one value per cell (for each of the seven residual subroutines) and in general is much easier to succeed with than the state Jacobian, $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$, which is the topic of the next section.

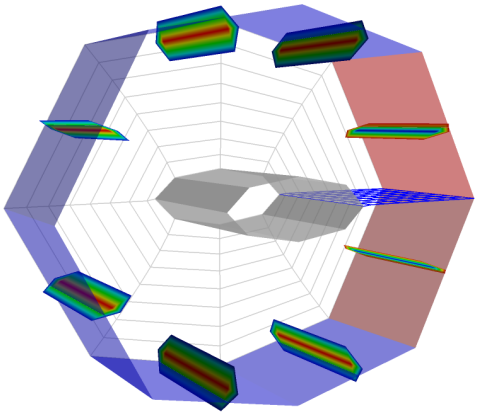


**Figure 7.5:** A visualization of the $\partial\mathbf{R}_u/\partial\mathbf{x}$ field superimposed on the (now transparent) setup from Fig. 7.4. The $\partial\mathbf{R}_u/\partial\mathbf{x}$ field has been thresholded to remove all values which were zero. As expected, we only have non-zero $\partial\mathbf{R}_u/\partial\mathbf{x}$ values at the inflow and outflow BC zones. For inflow BC zones (transparent blue) the $\partial\mathbf{R}_u/\partial\mathbf{x}$ effect is positive (red lines) whereas the $\partial\mathbf{R}_u/\partial\mathbf{x}$ effect is negative (blue lines) at the outflow BC zones (transparent red).

[1] `Calc_dRdX(u,v,w,p,c1,c2,c3,dRdX)` and `adj_SetBoundaryFlow`

### 7.2.3 $\frac{\partial \mathbf{R}}{\partial \tilde{\mathbf{w}}}$-step

The state Jacobian, $\partial \mathbf{R}/\partial \tilde{\mathbf{w}}$, is certainly the most tricky of the four partial derivatives in the total derivative Eq. (7.8). In this step we must at least write two subroutines. We need the `Calc_dRdW(u,v,w,p,c1,c2,c3,dRdW)` which calculates the state Jacobian, but we also need a `adj_PerturbFlow` subroutine which not only perturbs the flow, but also makes auxiliary calls to update the ghost layer lest we were perturbing a cell close to the BC zones.

We cannot hope to cover all topics for this vast matrix, but we will highlight some of the main takeaways. While the illustration in the previous section was straightforward given that the $\partial \mathbf{R}_u/\partial \mathbf{x}$ field had one value per cell, it is more cumbersome to make a similar 3-D visualization for $\partial \mathbf{R}/\partial \tilde{\mathbf{w}}$ as it is *much* larger. It is, however, instructive to visualize $\partial \mathbf{R}/\partial \tilde{\mathbf{w}}$ as a simple 2-D matrix to inspect the sparsity patterns. To this end, we visualize $\partial \mathbf{R}/\partial \tilde{\mathbf{w}}$ in Fig. 7.6 for the mesh seen in Fig. 7.4. To the left in Fig. 7.6 we bring an overview of the 49 sub-matrices in $\partial \mathbf{R}/\partial \tilde{\mathbf{w}}$ whereas we show the actual sparsity pattern to the right in Fig. 7.6. The matrix is generated with the first-order accurate UDS scheme which results in less fill in than the third-order QUICK scheme.

There is quite a lot of information in Fig. 7.6. As a few points of interest we mention:

- There are $7^2 = 49$ sub-matrices for the core NS adjoint solver. These are separated by thin black lines in Fig. 7.6. A given column of a sub-matrix can be used to generate a 3-D visualization as we did in the previous step. Here, one should expect to see effects from a perturbation that mirror the stencil employed during the development. This is a useful first step in the debugging phase.
- Two gray areas are also visible. These areas mark flux states that are classified as 'outer fluxes' in the adjoint solver. To interpret the meaning of these inner and outer fluxes it is important to understand that EllipSys might rotate any of its blocks in a way which is found most suitable for ensuing computations. In the mesh seen in Fig. 7.4 the $I$ direction is wrapped around the airfoil and starts from the blue grid and proceeds 8 steps around the airfoil. The $J$ direction goes from airfoil to farfield BCs, and finally the $K$ direction is the spanwise direction. Returning to Fig. 7.6 we see that only direction $I$ and $J$ result in outer fluxes since no area is seen for the $\partial/\partial \mathbf{c}_3$ column. The reason is that the cyclic BC in EllipSys results in the spanwise faces being treated as inner faces during an iteration. Furthermore we note that it seems from Fig. 7.6 that there are twice as many outer faces in the $J$ direction compared to the amount in the $I$ direction (i.e. the gray area in the $\partial/\partial \mathbf{c}_1$ column is half the size of the gray area in the $\partial/\partial \mathbf{c}_2$ column). This is indeed the case, and the reason is, that we count both wall and farfield BCs as outer faces, whereas the face on the blue grid in Fig. 7.4 is only counted once. This is a rather subtle point. In the flow solver both faces exist, but in the adjoint solver, the two faces in the grid represent the very same interface between cells. As a result, we only dedicate one column in the $\partial \mathbf{R}/\partial \tilde{\mathbf{w}}$ matrix to each face in the mesh no matter how many times it is represented in the flow solver.
- To better inspect the sparsity pattern in one of the 49 sub-matrices we zoom in on

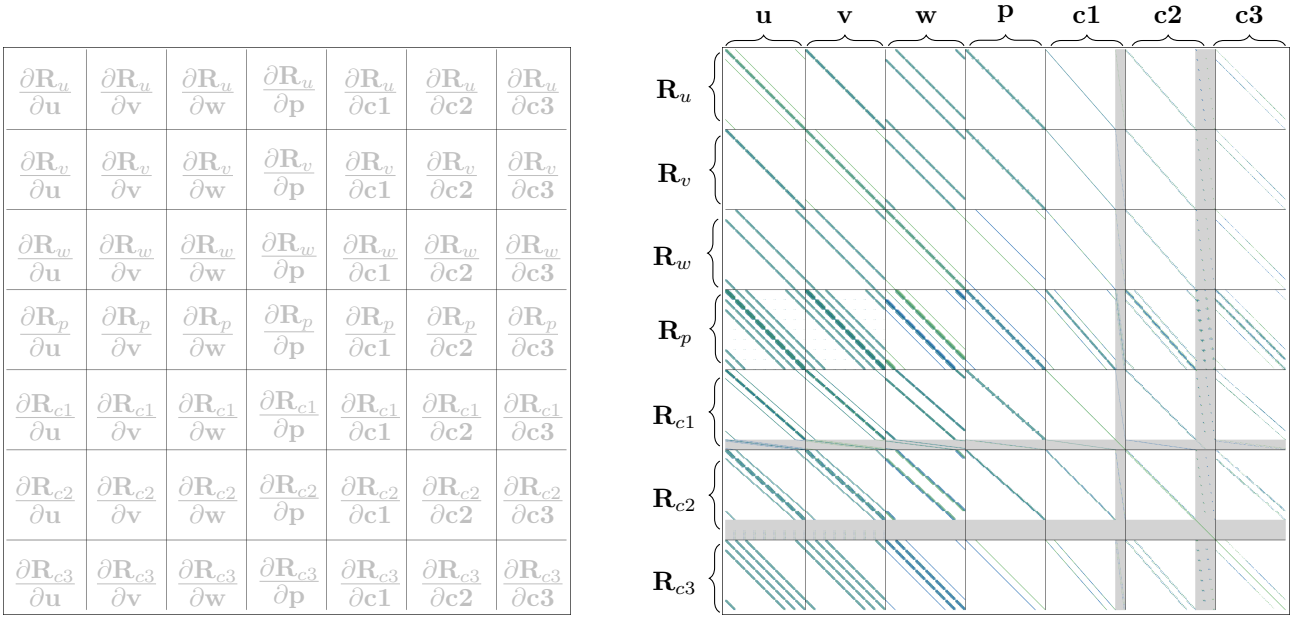



**Figure 7.6:** The left hand side shows an overview of the 49 sub-matrices in the state Jacobian, $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$, for the core NS adjoint solver without turbulence or transition. No actual sparsity data is seen here. The right hand side shows the sparsity pattern of the state Jacobian where positive (green) and negative (blue) entries can be discerned. Here, we notice the regularity of the sparsity pattern due to the structured grid. Some fluxes ($c1$ and $c2$) are for the mesh seen in Fig. 7.4 classified with inner and outer faces (the outer faces are marked by a gray area). See Sec. 7.2.3 for further details. See Fig. 7.7 to better inspect the sparsity pattern of one of the 49 sub-matrices.

the $\partial\mathbf{R}_p/\partial\mathbf{w}$-matrix in Fig. 7.7. One can clearly recognize a particular structure in the pattern since the flow solver uses a structured mesh. This fill in is the absolute minimum to expect given that we used the first-order UDS scheme to make this plot. Still, it is evident the there is a considerable amount of storage savings to be had by storing only non-zero values.

The debugging phase for the $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$ subroutine is quite extensive. Beside the visual inspections mentioned above we found it extremely helpful to develop two different architectures at the same time; both the FD-based eDA*fd* adjoint solver and the CS-based eDA*cs* adjoint solver. In particular, the ability to subtract the large $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$ fields produced by the two adjoint solvers will quickly pinpoint where the problems occur. Here, no absolute difference larger than $10^{-2}$ should be expected. We concede, that the development of the eDA*cs* adjoint solver is slightly more laborious, but it is immensely useful to be capable of producing machine accurate gradients later on. Besides, the eDA*cs* solver is also extremely useful when debugging algorithmically differentiated adjoint

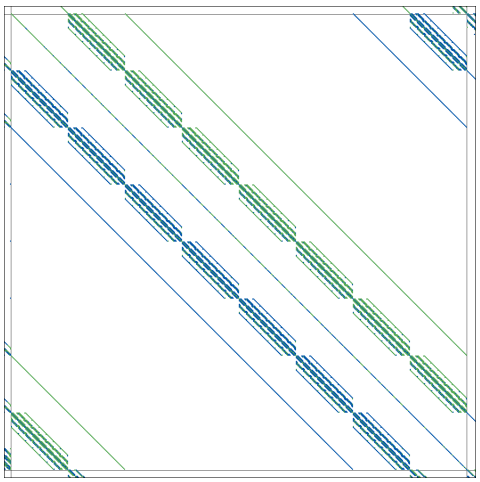


**Figure 7.7:** A visualization of one of the 49 sub-matrices from the state Jacobian seen in Fig. 7.6. This is the $\partial \mathbf{R}_p / \partial \mathbf{w}$ matrix. Green values are positive and blue values are negative.

routines. Here, errors in $\partial \mathbf{R}/\partial \tilde{\mathbf{w}}$ are much easier to spot since no errors above $\approx 10^{-15}$ should be accepted (assuming the eDA*cs* implementation is correct).

Once the maximal value in the error field in $\partial \mathbf{R}/\partial \tilde{\mathbf{w}}$ between the FD-based and the CS-based adjoint solver is below $\approx 10^{-2}$ it cannot get much better since the finite differences are inherently inaccurate. At this point it is time for the solution of the tangent-linear system, which is the next step in the road map.

### 7.2.4 $\frac{\mathrm{d}\tilde{\mathbf{w}}}{\mathrm{d}\mathbf{x}}$-step

At this step in the road map we have written at least four subroutines which enable us to compute the $\partial \mathbf{R}/\partial \mathbf{x}$ matrix[2] and the $\partial \mathbf{R}/\partial \tilde{\mathbf{w}}$ matrix[3]. Together, they constitute the tangent-linear system introduced in Chapter 2 which we bring below in Eq. (7.13) for convenience:

$$\frac{\partial \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x})}{\partial \tilde{\mathbf{w}}} \frac{\mathrm{d}\tilde{\mathbf{w}}}{\mathrm{d}\mathbf{x}} = -\frac{\partial \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x})}{\partial \mathbf{x}}. \tag{7.13}$$

Ideally, this $\mathrm{d}\tilde{\mathbf{w}}/\mathrm{d}\mathbf{x}$-step in the road map is rather brief and involves solving Eq. (7.13) and comparing the obtained solution ($\mathrm{d}\tilde{\mathbf{w}}/\mathrm{d}\mathbf{x}$) to some reference ($\mathrm{d}\tilde{\mathbf{w}}/\mathrm{d}\mathbf{x}_{ref}$).

The $\mathrm{d}\tilde{\mathbf{w}}/\mathrm{d}\mathbf{x}_{ref}$ is easily computed with the complexified EllipSys flow solver by perturbing the $u$ inflow BC in the complex plane using a step size of $10^{-20}$ as advocated elsewhere[4].

[2] using `Calc_dRdX(u,v,w,p,c1,c2,c3,dRdX)` and `adj_SetBoundaryFlow`

[3] using `Calc_dRdW(u,v,w,p,c1,c2,c3,dRdW)` and `adj_PerturbFlow`

[4] `http://mdolab.engin.umich.edu/content/guide-complex-step-derivative-approximation-0`

With respect to the solution procedure, we find the parallel linear solver package, PETSc [9, 8, 7], to be extremely user-friendly while still providing a platform that can be used for large parallel industrial scale cases. The PETSc solver settings are at this stage not too important since any combination should work for these small problems. We use the GMRES algorithm [105] with a simple Jacobi-preconditioning during development, but we refrain from further comments on the solution process until Chapter 10 where we introduce MPI capabilities into the adjoint solver and commit to a final setup of the linear solver.

Solving the tangent-linear system to machine precision through PETSc one should obtain a solution vector $\mathrm{d}\tilde{\mathbf{w}}/\mathrm{d}\mathbf{x}$ which matches the reference $\mathrm{d}\tilde{\mathbf{w}}/\mathrm{d}\mathbf{x}_{ref}$ to $\approx 10^{-6}$ and $\approx 10^{-16}$ for eDA$fd$ and eDA$cs$, respectively.

### 7.2.5 $\frac{\partial \mathbf{J}}{\partial \tilde{\mathbf{w}}}$-step

At this stage in the road map we are now close to a functioning adjoint solver. We only need to implement the partial derivative vector[5] $\partial\mathbf{J}/\partial\tilde{\mathbf{w}}$. The only subroutine we need to make here, is `Calc_dJdW(u,v,w,p,c1,c2,c3,dJdW)`, since we can reuse the `adj_PerturbFlow` subroutine from the $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$-step. First of all we have to decide on some cost function for, $\mathbf{J}$. Here, we choose $\mathbf{J}$ to be the force exerted on the airfoil in the $x$-direction in Fig. 7.8. Since our symmetric airfoil has its chord axis in this direction it means that $\mathbf{J}$ (for now) is equal to the drag. We will later on test the adjoint solver in an optimization where the inflow direction may vary. Once we arrive at that point, the drag will have two components (one in $x$ and one in $y$) and we will have to introduce two cost functions, to describe the drag, but for now we can say that $\mathbf{J}$ is the drag.

Reminiscent of our approach to the $\partial\mathbf{R}/\partial\mathbf{x}$ matrix, we can quite easily visualize parts of the $\partial\mathbf{J}/\partial\tilde{\mathbf{w}}$-values in 3-D by superimposing the values on the setup. Choosing the change in $\mathbf{J}$ due to the $u$ variables, $\partial\mathbf{J}/\partial\mathbf{u}$, results in Fig. 7.8 and as expected, we find, that only the $u$ states adjacent to the airfoil surface results in a change in the drag when perturbed. Again, we only show the non-zero values to not clutter the plot too much.

Visual check as the one seen in Fig. 7.8 should be made to all seven sub-fields of $\partial\mathbf{J}/\partial\tilde{\mathbf{w}}$. Typically, this partial derivative field is much less problematic than the two involving the residual computation and should present no difficulty for the reader. Thus, we proceed to the next step of the road map.

### 7.2.6 $\Psi$-step

Having already set up the linear solver in the fourth step of the road map, we should ideally be able to rather quickly set up the adjoint solver. The task here amounts to feeding the transposed $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$ and $\partial\mathbf{J}/\partial\tilde{\mathbf{w}}$ partials to the PETSc solver, thus solving the adjoint equation,

[5] Assuming we only focus on one cost function, $\mathbf{J}$

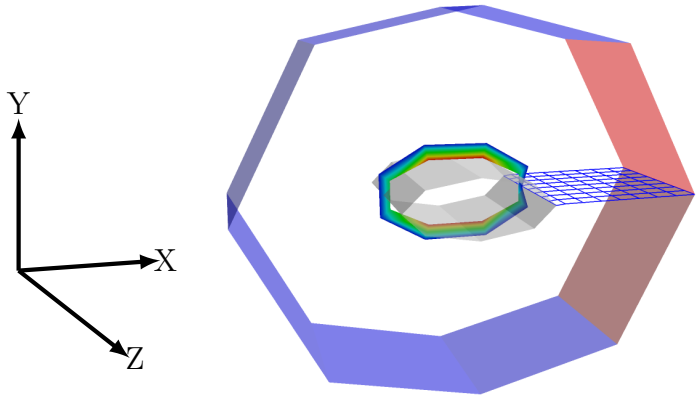



**Figure 7.8:** A visualization of the $\partial \mathbf{J}/\partial \mathbf{u}$ sub-vector which is the first part of seven in the partial derivative $\partial \mathbf{J}/\partial \tilde{\mathbf{w}}$ where $\tilde{\mathbf{w}} = [\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{p}, \mathbf{c1}, \mathbf{c2}, \mathbf{c3}]^T$ is the extended state vector. As expected, the only $u$ states that affect the drag ($\mathbf{J}$) are the states adjacent to the airfoil surface. We only show the non-zero values of $\partial \mathbf{J}/\partial \mathbf{u}$ to not clutter the plot too much.

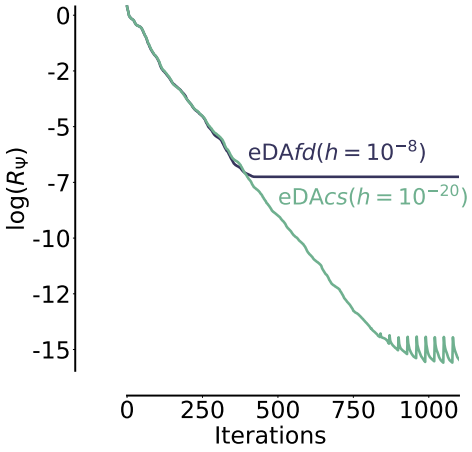
$$\left[\frac{\partial \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x})}{\partial \tilde{\mathbf{w}}}\right]^T \boldsymbol{\Psi} = \left[\frac{\partial \mathrm{J}(\tilde{\mathbf{w}}, \mathbf{x})}{\partial \tilde{\mathbf{w}}}\right]^T, \tag{7.14}$$

which allows us to finally obtain the adjoint variables, $\boldsymbol{\Psi}$.

While it is quite easy to set up this road map step, we do recommend a rather meticulous verification phase to assess the behavior of the developed adjoint solvers. One way of doing this is visualized in Fig. 7.9 where the convergence behavior is seen to the left and the resulting gradient precision for the two adjoint solvers is seen to the right.

To briefly comment on the verification phase visualized in Fig. 7.9 we start by noting, that the PETSc convergence in the left hand side of the figure is indeed to be expected. The eDA*fd* produces an inherently noisy discrete adjoint equation marred with small errors, and as a result we cannot converge it all the way to machine precision. This is, however, possible for the eDA*cs* which due to the CS method provides machine accurate partials for the discrete adjoint equation, which in turn can be converged to machine accuracy.

The right hand side in Fig. 7.9 is a quantitative gradient evaluation where computed gradients are compared to a reference gradient, $\mathrm{d}F_x/\mathrm{d}\mathbf{u}_{BC} = 8.324356097717334 \cdot 10^{-1}$, which we have computed with the complexified flow solver. The analytical gradients from the adjoint solvers are computed using the total derivative equation,




| **Step size**, $h$ | **Significant digits**: |
|---|---|
| | eDA$fd$ $(\cdot 10^{-1})$ |
| $10^{-2}$ | 8.651335561732930 |
| $10^{-4}$ | 8.328715604947636 |
| $10^{-6}$ | 8.324399803580036 |
| $10^{-8}$ | 8.324355107475286 |
| $10^{-10}$ | 8.324245302505610 |
| $10^{-20}$ | 0.000000000000000 |
| | eDA$cs$ $(\cdot 10^{-1})$ |
| $10^{-2}$ | 8.238895578075072 |
| $10^{-4}$ | 8.324348038604391 |
| $10^{-6}$ | 8.324356096911429 |
| $10^{-8}$ | 8.324356097717258 |
| $10^{-10}$ | 8.324356097717334 |
| $10^{-20}$ | 8.324356097717343 |

[a] The reference gradient has been computed using the complexified EllipSys flow solver resulting in: $\mathrm{d}F_x/\mathrm{d}\mathbf{u}_{BC} = 8.324356097717334 \cdot 10^{-1}$.

**Figure 7.9:** The left hand side shows the PETSc $\Psi$-residual while solving the linear adjoint equation. The table to the right shows the amount of significant digits (black) obtained for a given step size, $h$, in the adjoint solvers eDA$fd$ and eDA$cs$.

$$\begin{aligned} \frac{\mathrm{dJ}(\tilde{\mathbf{w}},\mathbf{x})}{\mathrm{d}\mathbf{x}} &= \frac{\partial \mathrm{J}(\tilde{\mathbf{w}},\mathbf{x})}{\partial \mathbf{x}} - \boldsymbol{\Psi}^T \frac{\partial \mathbf{R}(\tilde{\mathbf{w}},\mathbf{x})}{\partial \mathbf{x}} \\ \Leftrightarrow \mathbf{0} \qquad &- \boldsymbol{\Psi}^T \frac{\partial \mathbf{R}(\tilde{\mathbf{w}},\mathbf{x})}{\partial \mathbf{x}} = \frac{\mathrm{d}F_x}{\mathrm{d}\mathbf{u}_{BC}}, \end{aligned} \tag{7.15}$$

we originally introduced in Chapter 2 (Eq. (2.12)). In Eq. (7.15) we have in the second line stressed the point that the $\partial \mathrm{J}(\tilde{\mathbf{w}},\mathbf{x})/\partial \mathbf{x}$ is assumed to be zero, and we have inserted the chosen cost function, $\mathbf{J} = F_x$, and design variable, $\mathbf{x} = \mathbf{u}_{BC}$.

Returning to the table in the right hand side of Fig. 7.9 we comment that also this outcome is to be expected for the presented adjoint solver architectures. Since eDA$fd$ is based on the FD method, one would do well to remember to carry out a step size study before running industrial scale optimizations. For further details on scaling a FD-based adjoint solver we refer to [38]. With respect to the CS-based adjoint solver, we observe the salient feature of providing accurate gradients for extremely small step sizes. At this point we draw the attention to a curiosity we highlighted in the bottom two rows of the table. It seems eDA$cs$ oscillates between the 14'th and the 16'th digit. We have in this regard observed that the oscillatory behavior[6] of the PETSc residual very close to

[6] See the green curve to the left in Fig. 7.9 which starts to oscillate around iteration 1000

convergence can influence eDA*cs*'s precision. Overall, the performance of the developed adjoint solvers are certainly satisfactory, and proves that the suggested road map may indeed lead to successful adjoint solver implementations.

## 7.3 Extension to mesh sensitivities

We now cover the final step in the road map (Fig. 7.2) where the $\partial \mathbf{J}/\partial \mathbf{x}$ partial derivative is added to the total derivative equation. This fourth partial derivative matrix is needed when we want to use shape design variables.

Instead of the above described situation in Eq. (7.15) where $\mathbf{x} = \mathbf{u}_{BC}$ we now add the $\partial \mathbf{J}/\partial \mathbf{x}$ and we must also multiply the previous total derivative equation with a new partial derivative matrix containing mesh derivatives,

$$\frac{\mathrm{dJ}}{\mathrm{d}\mathbf{x}_{\mathrm{DV}}} = \left[ \frac{\partial \mathrm{J}}{\partial \mathbf{x}_{\mathrm{mesh}}} - \mathbf{\Psi}^T \frac{\partial \mathbf{R}}{\partial \mathbf{x}_{\mathrm{mesh}}} \right] \frac{\partial \mathbf{x}_{\mathrm{mesh}}}{\partial \mathbf{x}_{\mathrm{DV}}}. \tag{7.16}$$

As seen, we now use $\mathbf{x}_{\mathrm{DV}} = \mathbf{x}_{\mathrm{shape}}$ instead of BC design variables whereas the cost function stays the same. Notice, that nothing changes in the adjoint equation which produces $\mathbf{\Psi}$. It is only the partial matrices involving the design variables which are affected. The extension to mesh sensitivities has a rather drastic effect on the $\partial \mathbf{R}/\partial \mathbf{x}$ partial which for inflow BC design variables was a vector but now is a rather large matrix. However, conceptually, the extension is straightforward and amounts to isolating the routines in the flow solver that calculate the metric terms. It is also necessary to make an `adj_PerturbMesh` subroutine akin to the `adj_PerturbFlow` subroutine we introduced earlier. The final question is of course how to acquire the $\partial \mathbf{x}_{\mathrm{mesh}}/\partial \mathbf{x}_{\mathrm{DV}}$. This is completely parametrization method dependent. As outlined in Chapter 5 we advocate for the use of the FFD method, and one of the main reasons for this is, that it provides explicit analytic gradients needed to construct the $\partial \mathbf{x}_{\mathrm{mesh}}/\partial \mathbf{x}_{\mathrm{DV}}$ matrix.

After duly debugging, the mesh sensitivities should naturally also be verified against a gradient from the complexified flow solver. While we have indeed verified the FFD gradients (Fig. 5.5) we will for this exercise omit the developed FFD tool to make sure that any reported inaccuracies are due to the adjoint solvers. We therefore manually define a design variable that moves all mesh points at the mid chord position of the airfoil in some direction. The result of the gradient comparison for various step sizes can be seen in Tab. 7.1.

## 7.4 Framework evaluation with adjoint solver on an extruded airfoil

The developed adjoint solver has been given a general interface using the `iso_c_binding` module to align with the EllipSys flow solver interface. After providing the adjoint solver with the interface we can finally access it from Python, thus completing our framework. The result (originally shown in Chapter 4, Fig. 4.1) is seen below in Fig. 7.10.

**Table 7.1:** Results in the form of significant digits (black) from the gradient verification step using shape design variables. The reference gradient has been computed using the complexified EllipSys flow solver resulting in: $\mathrm{d}F_x/\mathrm{d}\mathbf{x}_{\mathrm{mesh}} = 3.444002592873105 \cdot 10^{-2}$.

| **Step size**, $h$ | **Significant digits**: | |
|---|---|---|
| | eDA$fd$ $(\cdot 10^{-2})$ | eDA$cs$ $(\cdot 10^{-2})$ |
| $10^{-2}$ | 6.526642484672462 | 3.410887765561024 |
| $10^{-4}$ | 3.438254374932465 | 3.443994223237616[b)] |
| $10^{-6}$ | 3.443942835036215[a)] | 3.444002592036073 |
| $10^{-8}$ | 3.444005063329547 | 3.444002592873022 |
| $10^{-10}$ | 3.444082080769128 | 3.444002592873105 |
| $10^{-20}$ | 0.000000000000000 | 3.444002592873111 |

[a)] This is not a mistake. Although the fourth digit does not match the actual error is on the fifth digit

[b)] Same argument as above

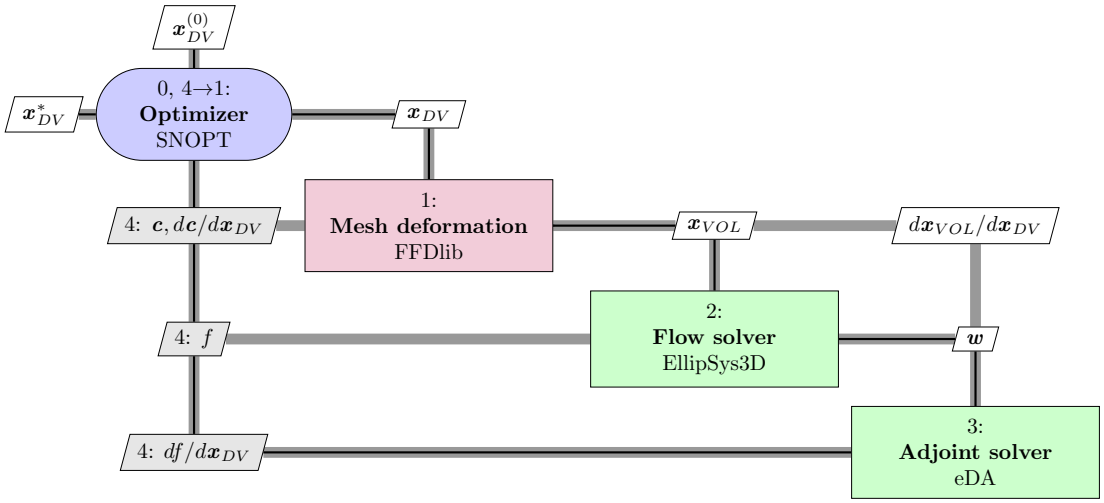



**Figure 7.10:** Figure introduced in Chapter 4 (Fig. 4.1) showing components in the numerical high-fidelity shape optimization framework at DTU Wind Energy. The four components are (0,4) an optimizer, (1) a mesh deformation module, (2) a CFD solver, and (3) a discrete adjoint solver. Gray lines show data flow. Black lines show process flow.

As a natural conclusion to the chapter we test the finalized numerical design optimization framework. Given that Chapter 11 is a presentation of various optimizations using shape design variables we will use this opportunity to showcase some of the other design variables we implemented. Furthermore, we increase the level of difficulty from the optimization in the previous chapter by solving a multipoint (MP) optimization problem and adding a few constraints to check that various parts of the framework function as expected. In the following optimization we will use the eDA*fd* adjoint solver since it provides the lowest gradient precision. Thus, any optimization with the eDA*cs* should be at least equally successful.

We will consider two flow cases in our MP test example where we will reuse the mesh presented above in the adjoint solver development (Fig. 7.4, left). The optimizer is tasked with minimizing the total exerted force, $\mathbf{F} = [F_x, F_y, F_z]^T$, on the airfoil by varying the $u$ and $v$ inflow BC settings for each flow case. Formally, the optimization problem reads:

$$\begin{aligned} \min_{u_{BC}, v_{BC}} \quad & \mathbf{F}(\tilde{\mathbf{w}}, \mathbf{x}), \\ \text{subject to} \quad & \mathbf{R}(\tilde{\mathbf{w}}, \mathbf{x}) = \mathbf{0} && \text{for case 1 and 2,} \\ & {u_{iBC}}^2 + {v_{iBC}}^2 + {w_{iBC}}^2 = 1 && \text{for } i = 1, 2. \end{aligned} \tag{7.17}$$

Notice, that the mesh, $\mathbf{x}$, above naturally is fixed since we use aerodynamic design variables that manipulate the inflow conditions. The starting point for both flow cases can be seen in Fig. 7.11 along with the final optimization result showing the expected $[1, 0, 0]$ BC inflow direction for both cases.

As hinted at earlier in this chapter, we will now operate with two cost functions, $\mathbf{J}_1 = F_x$ and $\mathbf{J}_2 = F_y$, in the adjoint solver to compute the total exerted force, $\mathbf{F} = [F_x, F_y, F_z]^T$, and its derivatives. To be fair we mention that one could indeed avoid the extra adjoint system solve by simply using one scalar objective which would be the force component in the drag direction. This approach should be preferred for industrial scale cases since the solution time of the adjoint system may be several hours. However, the following example is hardly of industrial scale. Given that the setup is an extruded airfoil with the spanwise direction pointing along the $z$-axis, we can focus on the $x$ and $y$ direction. As a measure for the force acting on the airfoil we will for each flow case define the single objective, `unknowns['SingObj'] = (Fx)**2 + (Fy)**2`, meaning that we will need to compute the change in `Fx` and the change in `Fy` with respect to our design variables, $u_{BC}$ and $v_{BC}$. The corresponding lines of Python code are seen in listing 7.1:

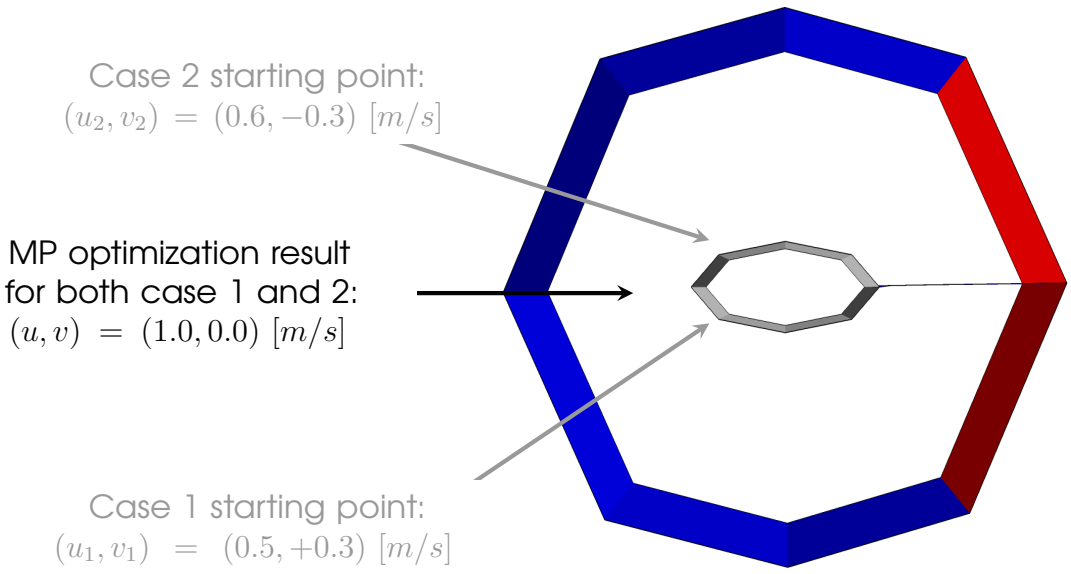



**Figure 7.11:** Visualization of starting point and optimization result of the multipoint (MP) aerodynamic adjoint-based optimization summarized in Fig. 7.12. The mesh (also seen in Fig. 7.4) has an inflow BC zone (blue) and an outflow BC zone (red) to preserve incompressibility. Both cases considered in the MP optimization converge to a horizontal inflow which minimizes the total exerted force, $\mathbf{F} = [F_x, F_y, F_z]^T$ on the airfoil. It should be noted, that both flow cases are constrained to have a total BC inflow length of one, $c_i - 1 = u_{iBC}^2 + v_{iBC}^2 + w_{iBC}^2 - 1 = 0$. As a result, they both start at an infeasible point in the design space.

```
# effects on 'Fx'
self.e3d.adjoint.get_adj_dJdW(3) # 0:fdx, 1:fdy, 2:fdz, 3:Fx, and so on
self.e3d.adjoint.solve_adj_equation()
self.e3d.adjoint.get_adj_dRdX(0) # 0:uinlet
dFx_dU = np.dot(-self.e3d.adjoint.Psi.T,self.e3d.adjoint.dRdX)
self.e3d.adjoint.get_adj_dRdX(1) # 1:vinlet
dFx_dV = np.dot(-self.e3d.adjoint.Psi.T,self.e3d.adjoint.dRdX)

# effects on 'Fy'
self.e3d.adjoint.get_adj_dJdW(4) # 0:fdx, 1:fdy, 2:fdz, 3:Fx, and so on
self.e3d.adjoint.solve_adj_equation()
self.e3d.adjoint.get_adj_dRdX(0) # 0:uinlet
dFy_dU = np.dot(-self.e3d.adjoint.Psi.T,self.e3d.adjoint.dRdX)
self.e3d.adjoint.get_adj_dRdX(1) # 1:vinlet
dFy_dV = np.dot(-self.e3d.adjoint.Psi.T,self.e3d.adjoint.dRdX)
```

**Listing 7.1:** Python code allowing a user to compute the 'fx' and 'fy' derivatives with respect to the $u$-inlet and the $v$-inlet.

As can be seen in the above code snippet, we have an adjoint equation solve[7] both in the `# effects on 'Fx'` section and in the `# effects on 'Fy'` section. Therefore, we should expect to see two PETSc convergences every time the optimizer queries the adjoint solver for the gradient value.

With respect to the constraints we choose to impose a norm of one for the BC inflow vector $c_i - 1 = u_{iBC}^2 + v_{iBC}^2 + w_{iBC}^2 - 1 = 0$ in both cases to avoid the extremely trivial solution, that the optimizer simply chooses to completely turn off the inflow altogether to reduce the exerted force on the airfoil. In other words, both $c_1$ and $c_2$ should be expected to be very close to one at the end of the optimization.

Finally, we note, that while EllipSys certainly can handle moving outflow BC zones, we fix the outflow zones, so that the two single-point cases can share the mesh. Also, while the previous chapter used a RANS fluid model, we will revert to the standard NS equations, since we have not yet shown how to extend the adjoint solver to encompass turbulence.

We are now ready to inspect the result of the MP optimization. To this end, we bring the rather dense plot seen in Fig. 7.12 where we attach the following explanatory comments:

[**Upper graph**:] Optimization history for the boundary condition design variables $(u_{BC}, v_{BC})$ for both flow cases. For case 1 we call the design variables $(u_1, v_1)$ and for case 2 we call the design variables $(u_2, v_2)$. As seen, both $u_{BC}$ design variables converge to 1 whereas $v_{BC}$ converge to 0 for both flow cases just as we would expect.
[**Middle graph**:] Overview of flow solver residual (left, black) and the PETSc adjoint solver residuals (right, green). $\mathbf{R}_u$ is taken as an example. Given that we use the NS equations as our fluid model we could just as well have chosen $\mathbf{R}_v$, $\mathbf{R}_w$, or $\mathbf{R}_p$. As postulated above, we see two PETSc convergences every time the optimizer requests a new gradient from the adjoint solver. To better distinguish the convergence between the two solvers, we have used $10^{-12}$ as threshold in the flow solver and $10^{-6}$ as threshold in the adjoint solver.
[**Lower graph**:] Overview of constraints (left, blue) and cost functions (right, gray). As could be hoped for, the optimization proceeds even though we start at two infeasible points in the design space and at the end of the optimization both constraints are satisfied (i.e. close to one). With respect to the cost functions we note, that SNOPT combines constraint violations and the declared cost function (total exerted force) into a combined merit function. This correlation between constraints and cost functions is evident at the start of the optimization from optimization step 1 to 4. Furthermore, it is evident from the cost function from each flow case (obj1 and obj2) that the MP cost function is a combination of the two.

All considered, the MP optimization progressed as expected. The insertion of the adjoint solver has been completely seamless, and it is now straightforward to develop MP optimization run scripts and add constraints as needed. We have in other words arrived

[7] See the `self.e3d.adjoint.solve_adj_equation()` line

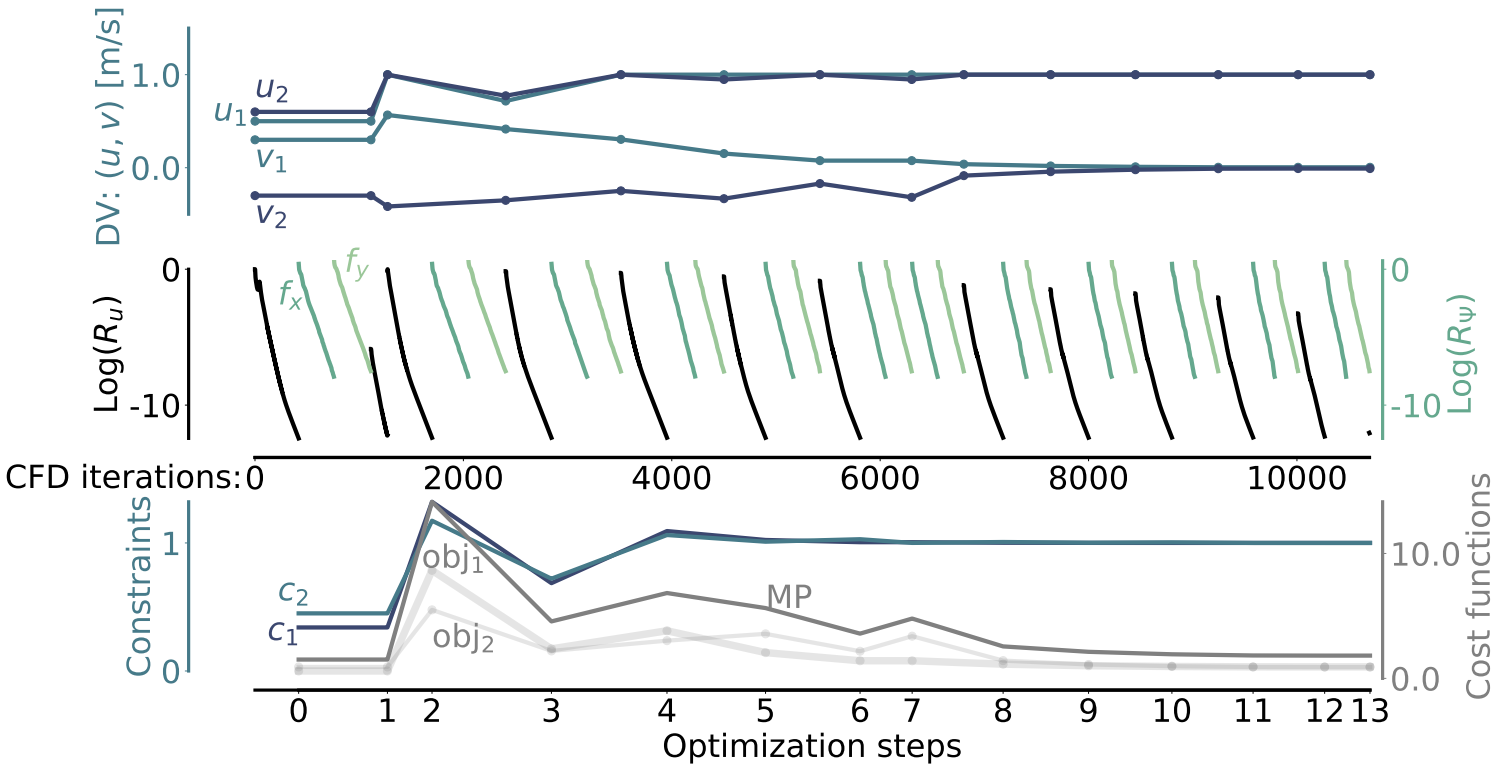



**Figure 7.12:** Summary of the multipoint adjoint-based optimization using inflow design variables $u$ and $v$ to minimize the total force exerted by the fluid on the airfoil. Upper: Design variables for case 1 $(u_1, v_1)$ and case 2 $(u_2, v_2)$. Middle: Convergence history for flow (left, black) and adjoint (right, green) solver. Notice, this is only for case 1. However, a very similar plot could be made for case 2. Lower: Overview of constraints (blue) and objective functions (gray) for both cases ($\text{obj}_1$ and $\text{obj}_2$). The fat gray line is the multipoint objective function that the optimizer constructs from $\text{obj}_1$ and $\text{obj}_2$. The upper, middle, and lower plots are vertically aligned with the $x$-axis in the middle plot. As a result, the optimization steps on the $x$-axis in the lower plot are not equidistantly interspersed. There is in other words not a fixed amount of CFD solver and PETSc solver iterations for each optimization step. Instead, we demand that the flow problem and the adjoint problem are converged to a given tolerance, and we use as many solver iterations as needed at each optimization step.

at a well-functioning foundation for future shape optimization studies. However, before industrially relevant optimizations can be carried out there are a few prominent additions needed for the adjoint solver. These are: i) Algorithmic differentiation, ii) turbulence and transition modelling, and iii) MPI capabilities, and these three topics will be dealt with in the follow three chapters.

# CHAPTER 8
# Algorithmic differentiation

The present chapter describes one way of leveraging algorithmic differentiation (AD) to develop a machine accurate discrete adjoint solver. As we mentioned in Chapter 4 we have planned a rather comprehensive development of the EllipSys discrete adjoint (eDA) solver with four overall architectures (See Tab. 4.3). In Chapter 7 we presented the development of the FD-based eDA*fd* and the CS-based eDA*cs* adjoint solvers. As is evident from the chapter, the former is easiest to implement, but the latter provides machine accurate gradients with up to 16 significant digits (Tab. 7.1), which more than compensates for the extra developmental effort. Before we describe exactly how we chose to use algorithmic differentiation in our adjoint solver we will quickly introduce terms such as operator overloading (OO) and source code transformation (ST) as well as the forward and reverse mode of algorithmic differentiation. The chapter concludes with a gradient verification to test the gradient accuracy.

As mentioned above we have planned a total of four different adjoint solver implementations. The main topic in the present chapter is the eDA*adf* adjoint solver, which is based on forward mode algorithmic differentiation. The fourth and final adjoint solver architecture in Tab. 4.3, eDA*adb*[1], is based on the reverse algorithmic differentiation mode. It is, however, not implemented, and as a result, we cannot yet quantify its performance. For brevity, we therefore leave out its developmental efforts from the present discussion altogether. The performance of eDA*adb* should however mirror that of eDA*cs* and eDA*adf* when it comes to precision, but with respect to memory it is quite another story. The main motivation for the eDA*adb* architecture is namely the ability to compute gradients in a matrix-free fashion without the need to store the state Jacobian matrix. However, as detailed in Chapter 4 it is not a perilless strategy as memory consumption for non-optimized implementations may even increase. Luckily, one can also find mentionings of the opposite in Chapter 4, where in particular the ADflow matrix-free routines seem efficient while still reducing the memory consumption to 67% of the original cost. We refer readers to Chapter 4 for further details on the matrix-free reverse algorithmic differentiation implementations and will in the remainder of the present chapter focus on eDA*adf*, which as mentioned is based on forward algorithmic differentiation. These forward adjoint solvers based on algorithmic differentiation also have advocates in the literature we presented in Chapter 4. Here, Roth and Ulbrich [104] are the most prominent example where an unsteady LES adjoint solver is presented.

Before diving in to the development it should be mentioned that already the FD-based eDA*fd* adjoint solver presents a formidable tool which has been shown to function well in industrial scale cases with up to 10 million cell meshes [38]. Thus, one could certainly

[1] eDA*adb*: EllipSys discrete adjoint algorithmically differentiated backwards

choose to build up a numerical design optimization framework around such an adjoint solver[2]. We have instead chosen to focus on several types of discrete adjoint solvers to highlight their advantages and disadvantages. To this end, we have gone through an extensive developmental phase, but it also allows us to choose between several options when selecting which adjoint solver to use, and in this comparison the FD-based is much less precise than the competition as we will see in the present chapter. Another reasonable choice would be to build up a numerical design optimization framework around the eDA*cs* which does offer machine accuracy once implementation is complete. This would save a considerable amount of development. However, the complexified residual subroutines are typically about twice as slow as the real-valued residual subroutines since we must compute two numbers for every single number computed with the real-valued subroutines. Algorithmically differentiated subroutines can, however, become very efficient in that all variables are not necessarily computed. Depending on the AD-tool it may be possible to only compute active variables which are used downstream. Therefore, algorithmic differentiation offer potentially faster residual derivative computations. We therefore favor the adjoint solvers based on algorithmic differentiation. Besides, although we have paid a high developmental cost in essentially developing four adjoint solvers (where 3 have been completed by now) we would like to point out, that the various adjoint solvers can be used as a powerful tool to debug one another.

We start the discussion on our implementation based on algorithmic differentiation by bringing an overview in Fig. 8.1. Most notably, we have a new makefile, namely `eDA_tapenade.mkf`. This file controls the overall process flow of the differentiation including all instructions to Tapenade [36], which is the differentiation tool we will use in the present work. Keep in mind that we end up with 111 differentiated files (not to mention all the handwritten wrappers) so it is very useful to have the differentiation procedure fully automated. Makefiles are one way of doing this. Also seen in Fig. 8.1 are four new shell scripts (`TPN_prep.sh`, `TPN_post.sh`, `TPN_prep_metric.sh`, and `TPN_post_metric.sh`) which can be compared to previously presented correction scripts (`c_prep.sh` in Fig. 6.3 and `eDAcs_prep.sh` in Fig. 7.3). Where the latter two ensured the complexified code could be compiled the four new scripts ensure that Tapenade sees code that is easily differentiated (`prep.sh` scripts) and that code produced by Tapenade fits in to the overall flow solver (`post.sh` scripts). The folder, `eDAadf_build`, is where all differentiated code is kept. Finally, we have the `tpn_dummy_`-files and `tpn_wrapper_`-files, where only a single file of each type is shown in Fig. 8.1. We have a total of 10 and 13 of these files. In short, these are handwritten routines, which either pertain to MPI or to routines, which are very specific to EllipSys, and therefore better suited for manual differentiation. We will give one example of such a case later on. We now proceed to an introduction of algorithmic differentiation where we cover necessary basic principles to understand our implementation using algorithmic differentiation. For a further details we refer to [34].

[2]However, as the tables in the present chapter will show, one will most likely need to conduct thorough step size studies for each individual partial matrix as well as implement scaling procedures as presented by [38]

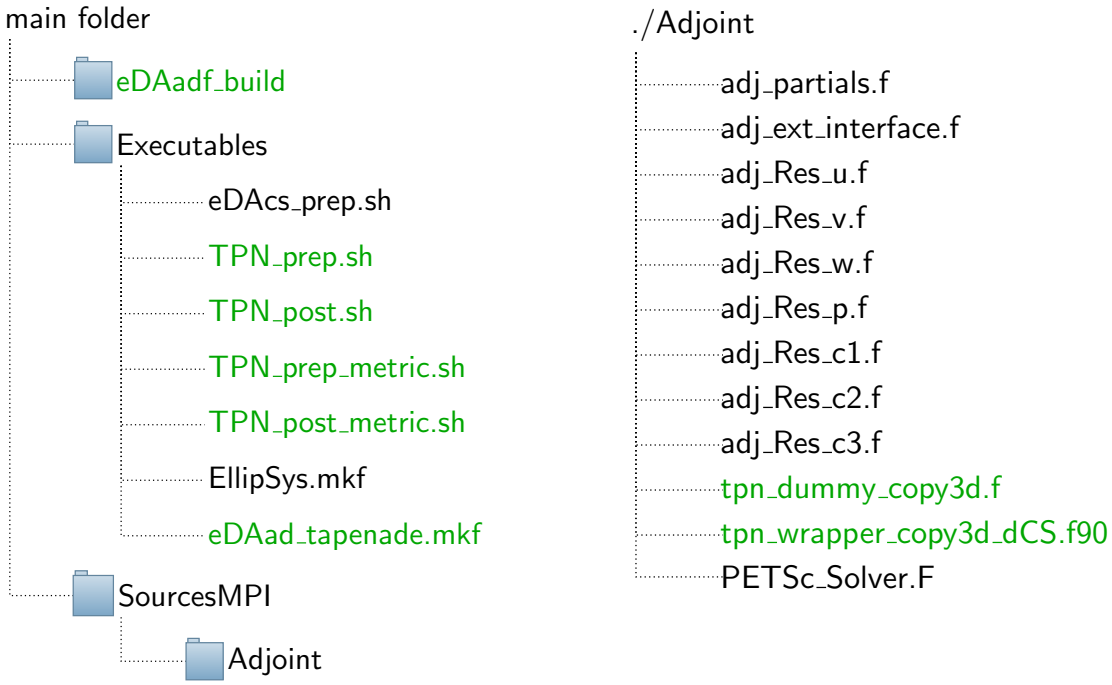



**Figure 8.1:** Visualization of (some of) the code folder structure after implementing an adjoint solver using the forward algorithmic differentiation mode where necessary additions are shown in green. The left hand side is an extension of Fig. 7.3 showing the overall EllipSys code structure. The right hand side shows changes made to the `./Adjoint` folder. We have shown one example of a `tpn_dummy_`-file and a `tpn_wrapper_`-file, but there are 10 and 13 of these handwritten routines, respectively.

## 8.1 Algorithmic differentiation through Tapenade

Algorithmic differentiation is a software-assisted way of obtaining derivatives. One can either compute tangent derivatives by way of the forward mode, or adjoint derivatives by way of the reverse mode. However, before getting to an illustrative example to clarify these terms it is important to introduce two fundamentally different ways to incorporate algorithmic differentiation in the computer code.

### 8.1.1 Operator overloading and source code transformation

The first way is to overload the operators in the code by introducing new data types, hence the name 'operator overloading'. Here, the new data type would carry both the

variable value and its derivative. Overloaded operators would then compute the function value as well as its derivative. One of the benefits of OO is that the overall code structure remains the same. As a result, it is considerably easier to implement than ST. However, as reported elsewhere [55, p. 16] it will also be two to four times slower as well as incur at least twice the memory cost. In this regard, it can be compared to complexified CFD solvers that store two numbers for each variable. Examples of related efforts using OO are easily given since all works in Tab. 4.2 except our own implementation make use of this method. Two of the works do so in C++ using reverse mode [23, 38] whereas [104] use the forward mode in Fortran.

The alternative to OO is source code transformation where new code calculating the derivatives is generated based on the original code. As we will see, the implementation is now more laborious since e.g. the number of arguments for each subroutine may change. One can apply source code transformation algorithmic differentiation in a brute force manner where all code is differentiated in a black-box manner, but this will result in terrible performance. As mentioned (Sec. 3.1.1), ST can produce competitive codes to e.g. handwritten routines with careful implementation [80].

> "AD can then be used to propagate development of the primal through to the tangent and adjoint codes at almost no additional expense."
>
> Müller and Cusdin, [80, p. 945]

However, it does require a careful implementation.

Indeed, algorithmic differentiation – also known as automatic differentiation – is not as automatic as the name would suggest in this case. A statement that is easily confirmed by contemplating the quote from a very recent work:

> "Although source code transformation is a type of automatic differentiation, the ultimate process itself is not entirely automatic. Generally speaking, a considerable amount of work is required before and after differentiation [...] to ensure that no bugs are introduced in the process."
>
> Kaminsky and Ekici, [52, p. 2]

### 8.1.2 Illustrative example on the forward and the reverse mode

Formally, we consider some code that implements a function, $F : X \in \mathcal{R}^m \rightarrow Y \in \mathcal{R}^n$, which takes an input, $X$, and returns an output, $Y$. Here, we have used the notation most custom to the algorithmic differentiation community. We now want to extend the code so that also derivatives of $F$ can be computed. Here, it is quite common that one does not need the entire Jacobian, $F'(X)$, of $F$, but can make do with a projection of it. This is an advantage because $F'(X)$ is a large $n \times m$ matrix whereas the projections are much smaller. The first way to project a Jacobian is the *directional derivative*,

$$\underset{[n\times 1]}{\dot{Y}} = \underset{[n\times m]}{F'(X)} \ \underset{[m\times 1]}{\dot{X}} \ , \tag{8.1}$$

where $\dot{X}$ is the direction we are interested in and $\dot{Y}$ is the amount with which $F$ changes in said direction. We added sizes below Eq. (8.1) to stress that the two dotted variables are vectors. The second projection option is the gradient of $F$,

$$\underset{[1\times m]}{\bar{X}^T} = \underset{[1\times n]}{\bar{Y}^T} \ \underset{[n\times m]}{F'(X)}, \tag{8.2}$$

where sizes again have been inserted to note that the gradient, $\bar{X}^T$, is a vector. More precisely, it is a row in the full Jacobian $F'(X)$, which is seen in the illustration in Fig. 8.2 where the difference between the two methods is visualized. The right hand side of Eq. (8.1) is seen to the left in Fig. 8.2 and the right hand side of Eq. (8.2) is seen to the right in Fig. 8.2.

$$\begin{bmatrix} f'_{1,1} & f'_{1,2} & f'_{1,3} & f'_{1,4} \\ f'_{2,1} & f'_{2,2} & f'_{2,3} & f'_{2,4} \\ f'_{3,1} & f'_{3,2} & f'_{3,3} & f'_{3,4} \end{bmatrix} \begin{bmatrix} \dot{X}_{1,1} \\ \dot{X}_{2,1} \\ \dot{X}_{3,1} \\ \dot{X}_{4,1} \end{bmatrix} \qquad \begin{bmatrix} \bar{Y}_{1,1} & \bar{Y}_{1,2} & \bar{Y}_{1,3} \end{bmatrix} \begin{bmatrix} f'_{1,1} & f'_{1,2} & f'_{1,3} & f'_{1,4} \\ f'_{2,1} & f'_{2,2} & f'_{2,3} & f'_{2,4} \\ f'_{3,1} & f'_{3,2} & f'_{3,3} & f'_{3,4} \end{bmatrix}$$

**Figure 8.2:** Illustrations of the right hand side in Eq. (8.1) (left) and right hand side in Eq. (8.2) (right) for $\dot{X} = [0, 1, 0, 0]^T$ and $\bar{Y}^T = [0, 1, 0]$, respectively.

As mentioned above, we are considering a piece of code that implements $F(X)$, and the algorithmic differentiation tool is applied to said code. To better relate the algorithmic differentiation operations described in Eq. (8.1) and Eq. (8.2) we consider a simple example: Suppose that the entire procedure to calculate the pressure correction, $p^c$, could be encapsulated by the following iterative procedure,

$$p^c = 1.2 \cdot p^c + (u + v)u, \tag{8.3}$$

where $p^c$ is the pressure correction and $u$ and $v$ are the two components in the flow field. Eq. (8.3) has of course nothing to do with the actual (immensely complicated) procedures inside a flow solver, but the overall iterative structure in Eq. (8.3) is very common in CFD solvers, which is why we picked the expression above[3]. Now, to obtain the gradient of $p^c$ we can of course derive the expression for this simple example to get,

[3]What is important in these iterative procedures is, that a variable (in this case $p^c$) is found by repeatedly overwriting the same variable with an ever-improving estimate. At convergence $(p^c)^n \approx (p^c)^{n-1}$

$$\nabla p^c = [1.2, 2u + v, u], \tag{8.4}$$

but assuming Eq. (8.3) was some complicated expression and we did not bother deriving it by hand, we could also implement Eq. (8.3) as a subroutine and leverage a suitable algorithmic differentiation tool. To this end, we write the file, `TPN_pc.f`, containing the subroutine `Calc_pc(u,v,pc)`.

```
      subroutine Calc_pc(u,v,pc)
c-------------------------------
c     [in]
      real*8,intent(in)::u,v
c     [inout]
      real*8,intent(inout)::pc
c-------------------------------
      pc = 1.2d0*pc + (u+v)*u
c-------------------------------
      end subroutine Calc_pc
```

**Figure 8.3:** Eq. (8.3) as a Fortran subroutine.

We then acquire Tapenade[4] and apply the forward mode algorithmic differentiation on `TPN_pc.f` by invoking Tapenade from the terminal as seen below:

```
$TAPENADE -head 'Calc_pc(u,v,pc)>(pc)' -tangent TPN_pc.f
```

**Listing 8.1:** Tapenade command to solicit the forward mode on the `Calc_pc` subroutine from a file called, `TPN_pc.f` (see Fig. 8.3). The command results in the subroutine `CALC_PC_D` which can be found in the file `TPN_pc_d.f` (visualized in Fig. 8.4).

Above, `-head` means the subroutine name, and when we pass it to Tapenade we tell Tapenade that we want the derivative of `>(pc)` with respect to `(u,v,pc)`. The command will generate a new file, `TPN_pc_d.f`, containing the forwardly differentiated routine. To apply the reverse algorithmic differentiation mode instead, we only need to change `-tangent` to `-reverse` as seen below:

```
$TAPENADE -head 'Calc_pc(u,v,pc)>(pc)' -reverse TPN_pc.f
```

**Listing 8.2:** Tapenade command to solicit the reverse mode on the `Calc_pc` subroutine from a file called, `TPN_pc.f` (see Fig. 8.3). The command results in the subroutine `CALC_PC_B` which can be found in the file `TPN_pc_b.f` (visualized in Fig. 8.4).

Again, the command will generate a new file, this time it is called `TPN_pc_b.f` and it will contain the differentiated routine generated with the reverse mode. The two differentiated routines are seen in Fig. 8.4.

[4]Go to `https://www-sop.inria.fr/tropics/tapenade/downloading.html` to obtain a license and follow the instructions

```
1  SUBROUTINE CALC_PC_D(u, ud,&
2 &     v, vd, pc, pcd)
3 C-------------------------------
4  IMPLICIT NONE
5 C-------------------------------
6 C     [in]
7  REAL*8, INTENT(IN) :: u, v
8  REAL*8, INTENT(IN) :: ud, vd
9 C     [inout]
10 REAL*8, INTENT(INOUT) :: pc
11 REAL*8, INTENT(INOUT) :: pcd
12C-------------------------------
13 pcd = 1.2d0*pcd + (ud+vd)*u +&
14&      (u+v)*ud
15 pc = 1.2d0*pc + (u+v)*u
16 END
```

```
1  SUBROUTINE CALC_PC_B(u, ub, v,&
2 &      vb, pc, pcb)
3 C-------------------------------
4  IMPLICIT NONE
5 C-------------------------------
6 C     [in]
7  REAL*8, INTENT(IN) :: u, v
8  REAL*8 ub, vb
9 C     [inout]
10 REAL*8, INTENT(INOUT) :: pc
11 REAL*8, INTENT(INOUT) :: pcb
12C-------------------------------
13 ub = (2*u+v)*pcb
14 vb = u*pcb
15 pcb = 1.2d0*pcb
16 END
```

**Figure 8.4:** Routines generated with the algorithmic differentiation tool, Tapenade. The forward mode, `TPN_pc_d.f` (left), uses `d` for 'dot' as a suffix, whereas the reverse mode, `TPN_pc_b.f` (right) uses `b` for 'bar'. The 'dot' and 'bar' refer to the usage in Eq. 8.1-8.2.

To analyze the output from Tapenade in Fig. 8.4 we start with the forward mode, `CALC_PC_D`, which is easily understood. Our goal was to obtain the gradient seen in Eq. (8.4) where $\nabla p^c =$ `pcd` in the Fortran code in Fig. 8.4. To determine all 3 elements in the gradient in Eq. (8.4) we would have to pass a total of three *seed* vectors, $\dot{X} =$`[pd,ud,wd]`, through `CALC_PC_D`, namely

$\dot{X} = [1,0,0]^T$ to obtain the first element, `1.2d0`,

$\dot{X} = [0,1,0]^T$ to obtain the second element, `2u+v`, and finally

$\dot{X} = [0,0,1]^T$ to obtain the third element, `u`.

Turning now to the reverse mode subroutine, `CALC_PC_B`, it is slightly less intuitive at a first glance. However, from the introduction to Eq. (8.2) we know that we must now set the $\bar{Y}^T$ vector to extract a projection of the Jacobian. In the reverse mode subroutine, `CALC_PC_B`, we have $\bar{Y}^T =$`[pb,ub,wb]` so we simply make one call to `CALC_PC_B` with the seed vector, $\bar{Y}^T = [1,0,0]$, which returns $[1.2d0, 2*u+v, u]$, which is exactly what we wanted. Notice, that both `CALC_PC_D` and `CALC_PC_B` correctly produced the gradient from Eq. (8.2) but we had to call the former three times whereas only one call to the reverse mode subroutine was needed. To reason is obvious when we write out the full Jacobian as seen in Fig 8.5 where the gradient has been highlighted in red.
Comparing Fig. 8.5 to Fig. 8.2 it is obvious that we should choose the reverse mode since we were interested in a row in the Jacobian.

Seen in the light of the simple example above, it is hard to imagine why we did not opt to simply view our flow solver as one (very) large subroutine and simply apply the

$$F'(X) = \begin{bmatrix} \boxed{1.2 & 2u+v & u} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

**Figure 8.5:** The full Jacobian, $F'(X)$, for Eq. (8.3).

reverse mode algorithmic differentiation to the entire flow solver. Alas, there is a cost for the adjoint approach which (using algorithmic differentiation terminology) relates to the reversal of data-flow and control-flow. We saw this in Chapter 2 for a transient heat equation, but it can actually already be inferred from Eq. (8.2). A flow solver should be viewed as a composite function[5] which using the notation typical for Tapenade ([36, Sec. 2]) can be expressed as,

$$F(X) := f_n \circ f_{n-1} \circ ... \circ f_1(X), \tag{8.5}$$

and as a result also the Jacobian would be a sequence of operations,

$$F'(X) := f'_n \circ f'_{n-1} \circ ... \circ f'_1(X). \tag{8.6}$$

The data-flow reversal we referred to above is most easily seen in Eq. (8.2) by taking the transpose of Eq. (8.2) and inserting the now nested transposed Jacobian, $F'^T(X)$,

$$\bar{X} = F'^T(X)\bar{Y} = f'^T_1 \circ ... \circ f'^T_{n-1} \circ f'^T_n(X)\bar{Y}, \tag{8.7}$$

which immediately shows the problem, i.e. we would have to call the $n^{\text{th}}$ reverse mode differentiated subroutine before calling the $(n\text{-}1)^{\text{th}}$ subroutine and so on. It is now clear, that the reverse mode is slightly more involved than forward mode, since an initial *forward sweep* is necessary to make sure all data is available. To handle this conundrum of data-flow reversal algorithmic differentiation tools employ sophisticated checkpointing schemes to find the right balance between a 'store-all' approach and a 'recompute-all' approach. Luckily, our implementation uses the forward mode, so further discussion on these topics are out the scope of the present work.

### 8.1.3 The eDA*adf* implementation

Having already introduced the changes to the code (Fig. 8.1) and presented examples of Tapenade commands (Listing 8.1 and 8.2) as well as the resulting algorithmic differentiation code (Fig. 8.4) it should now be straight forward to grasp our implementation as it is the exact same underlying principle.

[5] Already inserting the iterative $p^c$-example above in a loop would result in a composite function

To give an example of exactly how the algorithmic differentiation procedure from the simple example above translates to an actual implementation on a CFD solver, we return to our own implementation and the `eDAad_tapenade.mkf` from Fig. 8.1 that we mentioned previously. The first task that `eDAad_tapenade.mkf` handles is the preprocessing (cpp) of all source code since Tapenade[6] cannot handle cpp directives[7]. The main reason we use cpp is to only write one set of adjoint source code files which can then accommodate all four adjoint solver architectures we planned to implement.

Once all files are preprocessed it is time to use Tapenade. There are 111 forward mode Tapenade commands in `eDAad_tapenade.mkf` resulting in the 111 differentiated files we mentioned previously. An example is seen below in listing 8.3.

```
1 RhieChow[M](den,u,v,w,p,awm,aem,asm,anm,abm,atm,apu,apv,apw,su,sv,sw,
    app,aw,as,ab,fw,fs,fb,flowksi,floweta,flowzet,p,bccix,bccjx,bcckx,
    bcciy,bccjy,bccky,bcciz,bccjz,bcckz,volcc,bij,bik,bjk,bji,bki,bkj)>
     \
2  (flowksi,floweta,flowzet,p,bccix,bccjx,bcckx,bcciy,bccjy,bccky,bcciz,
     bccjz,bcckz,volcc,bij,bik,bjk,bji,bki,bkj) \
```

**Listing 8.3:** Tapenade command to solicit the forward mode on the Rhie/Chow subroutine from the EllipSys3D flow solver.

As is hardly necessary stating, you do not want to differentiate this subroutine by hand – and certainly not 111 subroutines albeit some less intricate. As seen in listing 8.3 there are 44 independent variables for this particular subroutine and we are interested in the change in 20 dependent variables with respect to these 44 independent variables. We note, that while the initial implementation indeed is extremely laborious it is thankfully very easy to maintain. This experience is as seen shared by leading experts in the field:

> "A major advantage of AD-produced adjoints is the ease of maintenance, as builds can be made fully automatic, albeit for complex codes often requiring a substantial initial code modification."
>
> Müller, Mykhaskiv, and Hückelheim, [81, p. 1]

Another important thing to notice is the `[m]` in listing 8.3. This is a means of invoking the specialization[8] feature in Tapenade. To understand this usage we remind the reader that metric dependencies are irrelevant to the adjoint equation (see Eq. (7.14) and Eq. (7.16)). Here, it is only the dependency of the states, $\tilde{\mathbf{w}}$, that we are interested in. We therefore produce two sets of residual files: A state set where we only differentiate with respect to flow variables, and a metric set which also differentiates the residual routines with respect to metrical terms. The source code differentiated with respect to the metric terms are slower than the source code differentiated with respect to the states so it is important to have two versions of the residual files to not perform redundant

[6] Version 3.14 is used

[7] `https://www-sop.inria.fr/tropics/tapenade/faq.html#SE1`, (last access: 27 June 2019)

[8] `https://www-sop.inria.fr/tropics/tapenade/faq.html#multitop`, (last access: 27 June 2019)

computations. Returning to the specialization feature in Tapenade it is exactly such a specific usage it is dedicated for. The `[m]` results in a new suffix so that we can distinguish the various differentiated routines. Readers are referred to [44] for further information on the specialization feature.

In principle, this is all the information needed to enable an adjoint solver based on forward algorithmic differentiation if a researcher already has followed the road map from the previous chapter where the subroutine files are developed and verified. Instead of the simple $[3 \times 3]$ Jacobian seen in Fig. 8.5 we could write a nested loop to compute the full NS state Jacobian, $\partial \mathbf{R}/\partial \tilde{\mathbf{w}}$, by seeding the extended state vector, $\tilde{\mathbf{w}}$, in one element at a time, allowing us to compute a column for every seed vector.

While the above statement is true, we feel obliged to mention yet a final developmental concern before proceeding to the gradient verification of our adjoint solver using algorithmic differentiation. It has to do with routines which are very specific to the CFD solver in question. Most often, these routines are related to MPI capabilities. We explain how to extend the adjoint solver in this regard in Chapter 10 but still, one will already at the present stage have to handle the way Tapenade interacts with the MPI-related parts of the flow solver code. The way we solved this was to hide MPI related routines specific to EllipSys. Instead we exposed a simple `_dummy`-file to Tapenade which ensured that the correct calls to the problematic routines would still be made in the surrounding code. During our `TPN_post`-scripts we then overwrite the dummy files Tapenade produces with handwritten `_wrapper`-routines. An example of a `_dummy`- and `_wrapper` file set is seen in Fig. 8.6.

## 8.2 Gradient verification of an adjoint solver based on algorithmic differentiation

The above described procedures allow for a computation of all partial derivatives using forward mode algorithmic differentiation. After a thorough debugging phase, we are thus finally ready to assess the accuracy of the new adjoint solver based on algorithmic differentiation. Where we developed the two first adjoint solvers on a simple $8^3 = 512$ cell mesh with regular metric quantities (Fig. 7.4) we now revert back to the slightly larger $4 \cdot 8^3 = 2048$ cell mesh (Fig. 6.5) from Chapter 6. This means that the highest cell volume ratio[9] rises from $\approx 7.5$ to $10^8$, but also the skewness of each cell is much more pronounced. We do this to test the three adjoint solvers and to find out which implementations, that are more susceptible to a lower precision under these more realistic conditions.

Aligning with the approach in Chapter 6 and 7 we choose the force exerted by the fluid in the $x$-direction to be our cost function, $\mathbf{J}$, whereas we use two design variables: A purely aerodynamic design variable, $\mathbf{x}_{\mathrm{BC}}$, where $u$ of the inlet boundary is perturbed, or a shape design variable, $\mathbf{x}_{\mathrm{shape}}$, where a subset of points[10] and perturb them in the $x$-direction.

[9] i.e. $max(vol_{\mathrm{cell}})/min(vol_{\mathrm{cell}})$

[10] We choose all points lying midway in the spanwise direction

```
1  subroutine Copy3D(n,mds,a)
2  !------------------------------
3  use params, only: BPP
4  !------------------------------
5  implicit none
6  !      [in]
7  integer,intent(in)::n,mds
8  !      [inout]
9  real*8,dimension(n,n,n,BPP)&
10 &    ,intent(inout)::a
11 !      [temp]
12 real*8:: dummy
13 !------------------------------
14 dummy = 1.d0
15 a = a * dummy ! nothing happens
16 !------------------------------
17 end
18
19 !------------------------------
20 !The above dummy file is expos-
21 !ed to TPN to ensure correct
22 !calls are made from surround-
23 !ing code. Before compilation,
24 !we then overwrite the file
25 !produced by TPN with the file
26 !to the right, copy3d_d.
27 !   This subroutine correctly
28 !wraps a and ad in one complex
29 !variable, aCS, which in turn
30 !can be copied around using
31 !low-level EllipSys routines.
32 !This is the *only* way to use
33 !these functions. The only
34 !other option is to re-write
35 !new low-level subroutines to
36 !carry out the copying.
37 !------------------------------
```

```
1  subroutine copy3d_d(n,mds,a,ad)
2  !------------------------------
3    use params, only : bpp
4  !------------------------------
5    implicit none
6  ! [in]
7  integer, intent(in) :: n, mds
8  ! [inout]
9  real*8,dimension(n,n,n,bpp)&
10 & ,intent(inout)::a
11 real*8,dimension(n,n,n,bpp)&
12 & ,intent(inout)::ad
13 ! [temp]
14 complex*16,dimension(:,:,:,:)&
15 & ,allocatable::aCS
16 integer::i1,i2,i3,i4
17 !------------------------------
18 ! allocate temporary array
19 allocate(aCS(n,n,n,BPP))
20 ! fill ad and a into the imagi-
21 ! nary and real part of aCS
22 do i4=1,BPP;do i3=1,n;
23  do i2=1,n;do i1=1,n;
24  aCS(i1,i2,i3,i4)=&
25 & DCMPLX(a(i1,i2,i3,i4),&
26 &               ad(i1,i2,i3,i4))
27  enddo;enddo;
28 enddo;enddo;
29 ! make complex copy3d()
30 call c_copy3d(n,mds,aCS)
31 ! split aCS result to a and ad
32 do i4=1,BPP;do i3=1,n;
33    do i2=1,n;do i1=1,n;
34 !------------------------------
35       ad(i1,i2,i3,i4) =&
36 &       AIMAG(aCS(i1,i2,i3,i4))
37       a(i1,i2,i3,i4) =&
38 &        REAL(aCS(i1,i2,i3,i4))
39    enddo;enddo;
40 enddo;enddo;
41 ! deallocation
42 deallocate(aCS)
43 end subroutine copy3d_d
```

**Figure 8.6:** Examples of `_dummy`- and `_wrapper` files in eDA are seen to the left and right, respectively. The purpose of these files are also stated in l. 20-43 in the `_dummy` files so no further elaboration is needed.

Finally, we are ready to close the chapter by analyzing the measured gradient precision of $d\mathbf{J}/d\mathbf{x}_{BC}$ and $d\mathbf{J}/d\mathbf{x}_{shape}$ presented in Tab. 8.1 and Tab. 8.2, respectively. To calibrate expectations, we first bring a few results from related literature, so the reader more or less knows what to expect. We *only* mention efforts using a machine accurate reference such as a complexified solver or a twin adjoint. We find the use of a FD reference to be close to useless as the error cannot be ascribed to the adjoint solver with full certainty. We do not address the level of difficulty in the various test cases. Both the size of the mesh but certainly also the irregularity/skewness of the mesh should be taken into account when estimating whether or not result A is more impressive than result B. The below list merely serves to give the reader a ballpark estimate of what to expect.

•[**Aero DV**:] Marta et al. [72] report of 7-9 significant digits [72, Tab. 1] for an adjoint based on algorithmic differentiation using a CS reference. The design variable is the free-stream Mach number
•[**Aero DV**:] Mader et al. [68] report of 12-14 significant digits [68, Tab. 2] for an adjoint solver based on algorithmic differentiation. Also using a CS reference and the free-stream Mach number as the design variable
•[**Shape DV**:] Gao, Wu, and Xia [28] report of 12 significant digits [28, Tab. 1] for an adjoint solver based on algorithmic differentiation. Again, a CS-reference (and a tangent-linear solver) is used as reference but in this work a shape design variable is used via the FFD method
•[**Shape DV**:] We have already commented on the extremely thorough effort by Kenway et al. [55]. Several cases are run here. The compressible ADflow solver yields 12 significant digits [55, Tab. 3] and the incompressible DAFoam shows 10 significant digits [55, Tab. 4]. Both results are obtained with algorithmic differentiation

Of the listed works above only the DAFoam solver is based on an incompressible solver. DAFoam is furthermore one of the four works (Tab. 4.2) we know of, favoring our approach of implementing a discrete adjoint solver on a SIMPLE algorithm.

Turning to our own results, we first inspect the performance of our three adjoint solvers for aerodynamic design variables reported in Tab. 8.1. As seen, the FD-based eDA*fd* achieve up to 5 significant digits whereas the CS-based eDA*cs* presents up to 15 significant digits on this mesh. The newly implemented eDA*adf* based on forward algorithmic differentiation shows 14 significant digits. As reported in Chapter 7, we have observed that eDA*cs* can oscillate between 14-16 significant digits depending on where PETSc exits in the GMRES cycle. This phenomenon would also be expected for eDA*adf*. We have yet to fine tune this part.

As a general comment we would like to stress that our 2k cell mesh is an absolute minimum bound for mesh size for the works we listed above. For the eDA*adf* adjoint solver we do present a sensitivity test later on (Tab. 11.2) for a 83 thousand cell mesh which is more easily compared to, e.g., the 103 thousand cell mesh used in [55], but in general we should still carry out more extensive sensitivity tests on large meshes.

Therefore, once Jacobian coloring[11] is fully enabled we should revisit the accuracy tests. However, for these future tests to work out it is a prerequisite to perform well on the present meshes.

**Table 8.1:** Gradient verification of the algorithmically differentiated adjoint solver. We updated all eDA*fd* and eDA*cs* results since we reverted back to using the slightly larger 2k cell mesh from Fig. 6.5 instead of the mesh from Chapter 7. Like in all other gradient verification tables, we have computed the reference gradient with the complexified EllipSys flow solver. The reference gradient was found to be: $\mathrm{d}F_x/\mathrm{d}\mathbf{x}_{\mathrm{u\_BC}} = 2.211972304093335 \cdot 10^{-1}$.

| **Step size**, $h$ | **Significant digits**: | | |
|---|---|---|---|
| | eDA*fd* $(\cdot 10^{-1})$ | eDA*cs* $(\cdot 10^{-1})$ | eDA*adf* $(\cdot 10^{-1})$ |
| $10^{-2}$ | 2.311902740980003 | 2.240870514270167 | – |
| $10^{-4}$ | 2.216219454164728 | 2.211131903527565 | – |
| $10^{-6}$ | 2.212025081848921 | 2.211972213123595 | – |
| $10^{-8}$ | 2.211966529319900 | 2.211972304084249 | – |
| $10^{-10}$ | 2.211572155195292 | 2.211972304093368 | – |
| $10^{-20}$ | 0.000000000000000 | 2.211972304093339 | – |
| seed=`1.d0` | – | – | 2.211972304093325 |

Proceeding to the shape design variables we find an overall reduction in accuracy of 2-4 significant digits as seen in Tab. 8.2. In particular, we observe that the FD-based adjoint solver suffers from the transition to a highly irregular mesh. Where eDA*fd* for the regular mesh (Fig. 7.4) achieved up to 6 significant digits (Tab. 7.1) for a shape variable it now struggles to find even the right sign. Indeed, we only achieve the sign by testing the $10^{-11}$ step since we could see from the convergence rate, that we were close. This could be predicted using the definition of the convergence rate, $p$, from Tab. 5.1 where one can compute eDA*fd*'s $p$ for Tab. 8.2 to be $[-1.07, 0.68, 1.00, 1.00, 0.53]$ for step sizes $h = [10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}]$. Thus, we new that the cancellation error was about to set in and dominate the truncation error. Again, one should find optimal step sizes for each of the four partials and not use one step size for all to further improve precision The reader should notice that the mesh dependencies make it much more difficult to find a correct step size. In Tab. 8.1, the right step size for eDA*fd* clearly is around $10^{-7}$. However, in Tab. 8.2 it seems to be somewhere around $10^{-11}$. The eDA*fd* adjoint solver is of course not broken. Indeed, we know of no other work showing 6 significant digits as in Tab. 7.1 for a FD-based adjoint solver using shape design variables, but it is clear, that more work should be done to fine tune our FD-based adjoint. Again, we stress that the mesh in Chapter 6 also is exceedingly easy, which could explain why other researchers with tougher meshes achieve less significant digits. Still, eDA*fd* is certainly not broken. In the present case, one should conduct step size studies where an optimal step size for each of the four partial matrices from the total derivative equation is determined. Do not confuse this, with the mentionings in Chapter 4 of an optional scaling for the FD-based

[11] Coloring is a way to speed up Jacobian computation (in Sec. 4.1.2)

adjoint solvers. This scaling was clearly related to the convergence properties of the adjoint equation. As also stated in the chapter, we have not tested this, since we have not had the need yet. We believe this could be due to the curvilinear coordinates of the EllipSys flow solver as explained in Chapter 6.

Turning to the eDA*cs* performance in Tab. 8.2 it achieves up to 13 significant digits which is certainly much better, albeit one cannot shy away from the fact, that the overall accuracy seemingly is a bit better for the aerodynamic design variables (Tab. 8.1). This is, however, a well known tendency and can be observed elsewhere [38, Tab. 4-5]. Overall, the eDA*cs* performance is promising, and we are very keen on testing it on larger meshes.

The final adjoint solver in Tab. 8.2 is the eDA*adf* implementation based on forward algorithmic differentiation which achieves 11 significant digits which is three digits less than its 14 significant digits in aerodynamic design variable accuracy. This drop off is perhaps one digit larger than what we would expect but still more than plenty to enable successful high-fidelity shape optimization.

This discussion of the results concludes the chapter on adjoint solver implementations based on algorithmic differentiation. We now proceed to extensions of turbulence and transition modeling.

**Table 8.2:** Gradient verification of the algorithmically differentiated adjoint solver. We updated all eDA*fd* and eDA*cs* results since we reverted back to using the slightly larger 2k cell mesh from Fig. 6.5 instead of the mesh from Chapter 7. Like in all other gradient verification tables, we have computed the reference gradient with the complexified EllipSys flow solver. The reference gradient was found to be: $\mathrm{d}F_x/\mathrm{d}\mathbf{x}_{\text{shape}} = 3.899443324156955 \cdot 10^{-4}$.

| **Step size**, $h$ | **Significant digits**: | | |
|---|---|---|---|
| | eDA*fd* | eDA*cs* ($\cdot 10^{-4}$) | eDA*adf* ($\cdot 10^{-4}$) |
| $10^{-2}$ | $1.331219647360067 \cdot 10^{-0}$ | 3.965850873551496 | – |
| $10^{-4}$ | $-1.828888071703040 \cdot 10^{+2}$ | −1.829342868946546 | – |
| $10^{-6}$ | $-7.811472781076892 \cdot 10^{-0}$ | 3.898618974311258 | – |
| $10^{-8}$ | $-7.777435071017156 \cdot 10^{-2}$ | 3.899443241723080 | – |
| $10^{-10}$ | $-4.023802806026069 \cdot 10^{-4}$ | 3.899443324202739 | – |
| $10^{-11}$ | $+1.557214151703351 \cdot 10^{-4}$ | | |
| $10^{-20}$ | 0.000000000000000 | 3.899443324156330 | – |
| seed=`1.d0` | – | – | 3.899443324185411 |

# CHAPTER 9
# Turbulence and transition modeling

This chapter describes how to incorporate auxiliary models into the adjoint solver developed in Chapters 7 and 8. In Chapters 3 and 4 we learned that it has long been known that the frozen turbulence assumption was not a viable long term option. As a result, it is quite common to find adjoint solvers which also take the turbulence model into account in their adjoint formulation. However, the inclusion of transition modeling is somewhat more of a novelty. The perhaps most relevant work to mention in this regard is the effort by Khayatzadeh and Nadarajah [57] given that it was only the second effort on high-fidelity adjoint method applications within wind energy, which is also pointed out in our literature review [71]. They minimize the drag in 2-D on airfoils by extending the natural laminar region. Just as we will, they used the $k-\omega$ SST turbulence model combined with a $\gamma - \tilde{Re}_{\theta t}$ transition model by Langtry and Menter [75, 63], which is a correlation based transition model.

Another very popular transition model is the $e^n$ method. Rashad and Zingg [96] present a way to augment and adjoint solver to include the new equations. This work is a 2-D multipoint study in natural-laminar-flow airfoils. We will follow this approach closely. More recently, efforts in 3-D have also emerged. As an example we mention, that the ADflow solver has been extended with an $e^n$ formulation in the adjoint solver. The underlying school of thought behind our approach is seen in Fig. 9.1.

$$\begin{bmatrix} \frac{\partial \mathbf{R}}{\partial \tilde{\mathbf{w}}} & \frac{\partial \mathbf{R}}{\partial \tilde{\mathbf{w}}_m} \\ \frac{\partial \mathbf{R}_m}{\partial \tilde{\mathbf{w}}} & \frac{\partial \mathbf{R}_m}{\partial \tilde{\mathbf{w}}_m} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{\Psi} \\ \boldsymbol{\Psi}_m \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathbf{J}}{\partial \tilde{\mathbf{w}}} \\ \frac{\partial \mathbf{J}}{\partial \tilde{\mathbf{w}}_m} \end{bmatrix}^T$$

**Figure 9.1:** Visualization of the underlying principle in extending an adjoint solver. Subscript $m$ refers to the model we are adding to the adjoint solver.

As seen in Fig. 9.1 , we augment the state vector with new variables pertaining to the model, $m$, we want to implement, $[\tilde{\mathbf{w}}, \tilde{\mathbf{w}}_m]^T$. Similarly, the adjoint variable vector is

augmented with new costates, $\mathbf{\Psi}_m$, and the residual vector, $\mathbf{R}$, is extended to include residuals for the new model as well, $[\mathbf{R}, \mathbf{R}_m]^T$.

## 9.1 Residual subroutines for transport equations

The models we deal with in this chapter are all scalar transport equations. To incorporate these in EllipSys, a given transport equation is transformed from Cartesian to curvilinear coordinates similarly to the procedure we described for the NS equations in Chapter 6. Then follows a discretization which results in equations of the form,

$$A_P\phi_P + \sum_{nb} A_{nb}\phi_{nb} = S_F + S_C + S_P \quad , \text{ for } nb = [W, E, S, N, B, T], \tag{9.1}$$

for the scalar variable, $\phi$. Just as in Chapter 6, we use upper case letters, $W, E, S, N, B, T$, to stress that these are neighboring cell centers and not faces. We will not explain the transformation and discretization further but instead refer to [114, Appendix A] where it is carefully treated.

Akin to the residual subroutines for $u$, $v$, and $w$ (Eq. 7.1-7.3) we must produce two handwritten subroutines for both turbulence and transition modeling. Using Eq. 9.1 these subroutines can be written as,

$$\mathbf{R}_\phi = A_P\phi_P + \sum_{nb} A_{nb}\phi_{nb} - S_F + S_C + S_P \quad , \text{ for } nb = [W, E, S, N, B, T]. \tag{9.2}$$

A final point we make with respect to transport equations is, that a formulation based on these equations make use of *local* flow quantities, whereas the $e^n$ formulation (which we have not yet included in the adjoint solver) uses non-local properties in the boundary layer to determine its transition onset functions. The locality of the transport equation formulation is an immense advantage when it comes to the parallelization and differentiation of the code.

## 9.2 Turbulence

The first model we implement is the $k-\omega$ SST turbulence model where the new variables we add to the extended state vector are $k$ for turbulent kinetic energy and $\omega$ for its specific dissipation rate. Thus, our extended state vector is now, $\tilde{\mathbf{w}} = [\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{p}, \mathbf{c1}, \mathbf{c2}, \mathbf{c3}, \boldsymbol{k}, \boldsymbol{\omega}]^T$.

The transport equation for $\boldsymbol{k}$ is,

$$\rho\frac{\partial k}{\partial t} + \rho u_j\frac{\partial k}{\partial x_j} = \tau_{ij}\frac{\partial u_i}{\partial x_j} - \rho\beta^* k\omega + \frac{\partial}{\partial x_j}\left[\left(\mu + \frac{\mu_t}{\sigma_k}\right)\frac{\partial k}{\partial x_j}\right], \tag{9.3}$$

whereas the equation for $\boldsymbol{\omega}$ is,

$$\rho\frac{\partial\omega}{\partial t}+\rho u_j\frac{\partial\omega}{\partial x_j}=\frac{\gamma}{\nu_t}\tau_{ij}\frac{\partial u_i}{\partial x_j}-\beta\rho\omega^2+2\rho\left(1-F_1\right)\frac{1}{\sigma_{\omega 2}\omega}\frac{\partial k}{\partial x_j}\frac{\partial\omega}{\partial x_j}+\frac{\partial}{\partial x_j}\left[\left(\mu+\frac{\mu_t}{\sigma_\omega}\right)\frac{\partial\omega}{\partial x_j}\right]. \tag{9.4}$$

We should mention, that the eddy viscosity can be determined using the relation $\mu_t = (\rho a_1 k)/MAX(a_1\omega; F_2\Omega)$, and refer to [76] for further details on the blending functions, $F_1$ and $F_2$, as well as all constants. Readers digging in to said blending functions will learn that $MAX()$ and $MIN()$ functions pervade the expressions. Whether opting for a CS-based adjoint or adjoint based on algorithmic differentiation it is truly a leisure not to have to derive these expression by hand.

To write the two needed residual files we identify the routines in EllipSys where Eq. (9.3) and Eq. (9.4) are discretized and create subroutines that can compute a residual value as seen in Eq. (9.2) based solely on the converged extended state vector. For this to work out, we had to take the routines computing the production term from the $k$ equation and insert these calls at the very top of the $\omega$ residual file as well, since both subroutines should be fully independent.

As seen in Fig. 9.2, two completely new files are required, namely `adj_Res_tke.f` and `adj_Res_O.f` to implement the turbulence model. We also made changes in the subroutines calculating the partial derivatives. We now proceed with an inspection of the augmented state Jacobian before concluding with a gradient verification.

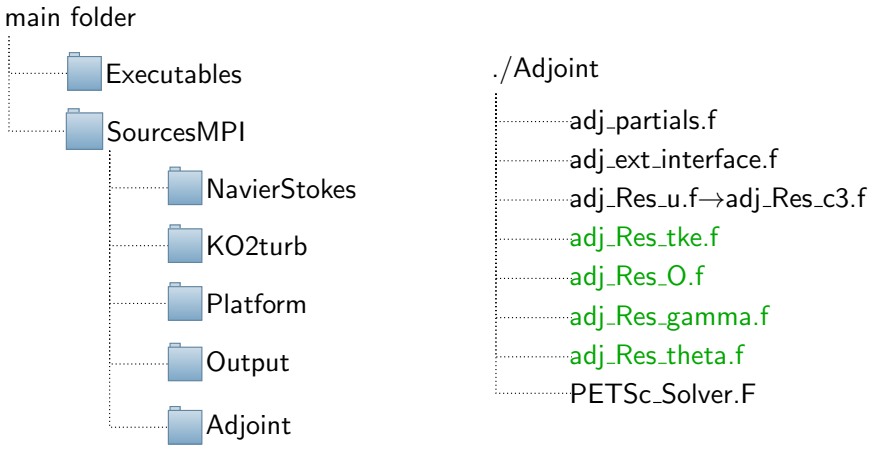



**Figure 9.2:** Visualization of the code folder structure after extending the adjoint solver to encompass models for turbulence and transition. The left hand side is an extension of Fig. 7.3 showing the overall EllipSys code structure. To the right the contents of the `./Adjoint` folder can be seen where `adj_Res_tke.f` and `adj_Res_O.f` are turbulence residual files whereas `adj_Res_gamma.f` and `adj_Res_theta.f` are transition residual files.

Once the implementation of the augmented system from Fig. 9.1 is complete it is useful to inspect the state Jacobian. To this end, we bring Fig. 9.3 to show the original NS state Jacobian (left) and the slightly larger state Jacobian (right) with the added turbulence model. Comparing the state Jacobian from Fig. 9.1 with Fig. 9.3 (right) one can easily recognize the overall pattern with the three added partial derivative sub-matrices, $\partial \mathbf{R}_m / \partial \tilde{\mathbf{w}}$, $\partial \mathbf{R} / \partial \tilde{\mathbf{w}}_m$, and $\partial \mathbf{R}_m / \partial \tilde{\mathbf{w}}_m$, which hints at a correct implementation.

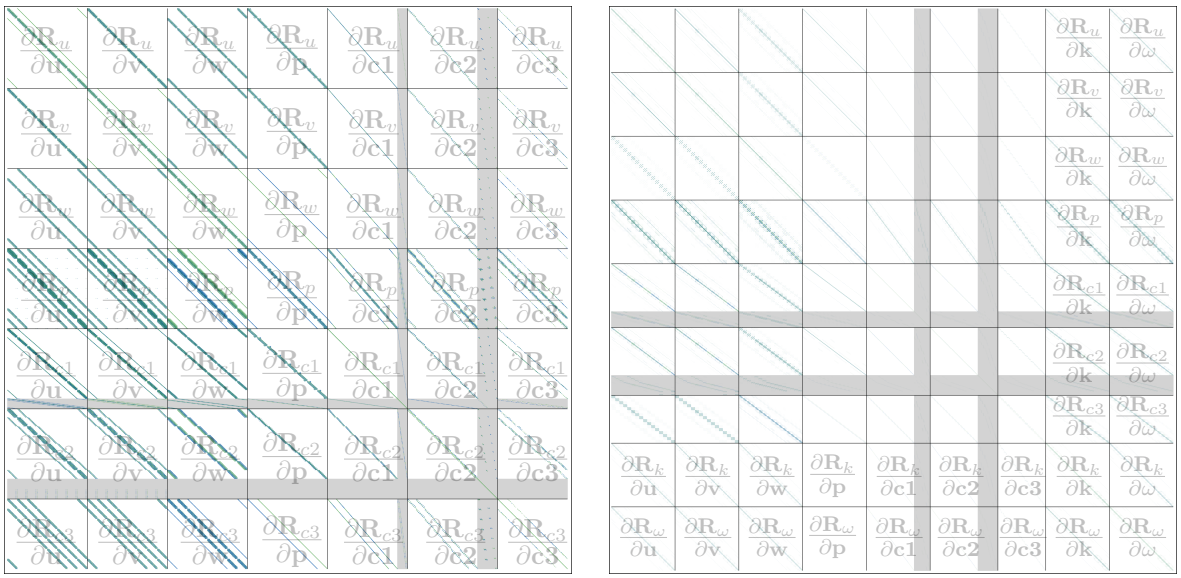


**Figure 9.3:** The left hand side shows the state Jacobian for the core NS adjoint solver without turbulence or transition. Some fluxes ($c1$ and $c2$) are for the mesh seen in Fig. 6.5 classified with inner and outer faces (outer faces are marked with a gray area). See Sec. 7.2.3 for further details. To the right a state Jacobian on the exact same mesh after extending the adjoint solver with the SST turbulence model.

## 9.2.1 Gradient verification after turbulence extension

Finally, we are ready to assess the gradient precision to see whether the extension has been successful. We will again use the cost function, $\mathbf{J} = f_x$, from Chapters 6-8 and also use both a purely aerodynamic design variable, $\mathbf{x}_{\text{BC}}$, where $u$ of the inlet boundary is perturbed, as well as a shape design variable, $\mathbf{x}_{\text{shape}}$, as described in Chapter 8. The choice of cost function means that the added turbulent variables do not directly enter into our cost function. Still, the $\partial \mathbf{R}_m / \partial \tilde{\mathbf{w}}$, $\partial \mathbf{R}_m / \partial \tilde{\mathbf{w}}_m$ and the $\partial \mathbf{R} / \partial \tilde{\mathbf{w}}_m$ matrices from Fig. 9.1 have non-zero entries which should prevent a trivial solution.

As a side note, we mention that Khayatzadeh and Nadarajah [57] propose to use the decrease in turbulent kinetic energy as a possible alternative cost function. We find this

attractive as an extra test since the turbulence variables would directly enter into the expression for the cost function gradient. Will consider this in future work.

Aligning with the procedure in Chapter 8 we start by assessing the aerodynamic gradient precision of $\mathrm{d}\mathbf{J}/\mathrm{d}\mathbf{x}_{\mathrm{BC}}$ in Tab. 9.1. We start with the eDA*fd* adjoint solver since it clearly is the least successful implementation. If we compare with the previous aerodynamic design variable accuracy from Tab. 8.1 the performance is identical for the three largest step sizes. The only difference is that the actual value of the gradient has changed since we now use a RANS model complemented with the SST turbulence model, instead of just using a NS fluid model. However, all step sizes from $10^{-8}$ and below cannot produce a non-zero gradient value. This should of course be possible. At least, we would expect that the effect of the cancellation error would set in gradually, as was the case in Tab. 8.1. Thus, we ascribe this deterioration to bugs in the eDA*fd* code.

Turning to the eDA*cs* performance it is as seen, flawless. The impeccable precision is the more interesting since the actual implementation compared to that of the eDA*fd* is very alike, i.e. all loops in the partial derivative assembly routines have the very same structure. In our implementation, it is mainly the adjoint solver based on algorithmic differentiation which separates itself from the pack with respect to implementation. Due to the similar implementation styles between eDA*fd* and eDA*cs* we are actually surprised that they are so disparate in implementation success. We can only stress that we by far favor the CS-based adjoint solver over the FD-based counterpart despite the fact that the initial developmental effort is substantially higher.

Finally, we note that the adjoint solver, eDA*adf*, using algorithmic differentiation in Tab. 9.1 also exhibits a very successful implementation which is only one significant digit shy of the result from before adding the turbulence model.

**Table 9.1:** Verification of a gradient from an aerodynamic BC design variable with the turbulence extension added to the adjoint solver. We use the 2k cell mesh from Fig. 6.5 instead of the mesh from Chapter 7 we originally developed eDA*fd* and eDA*cs* on. Like in all other gradient verification tables, we have computed the reference gradient with the complexified EllipSys flow solver. The reference gradient was found to be: $\mathrm{d}F_x/\mathrm{d}\mathbf{x}_{\mathrm{u\_BC}} = 2.211972610843547 \cdot 10^{-1}$.

| **Step size**, $h$ | **Significant digits**: | | |
|---|---|---|---|
| | eDA*fd* $(\cdot 10^{-1})$ | eDA*cs* $(\cdot 10^{-1})$ | eDA*adf* $(\cdot 10^{-1})$ |
| $10^{-2}$ | 2.310862692333680 | 2.241381626152641 | – |
| $10^{-4}$ | 2.216265363411347 | 2.211132324144602 | – |
| $10^{-6}$ | 2.212024562919752 | 2.211972519877158 | – |
| $10^{-8}$ | 0.000000000000000 | 2.211972610834458 | – |
| $10^{-10}$ | 0.000000000000000 | 2.211972610843598 | – |
| $10^{-20}$ | 0.000000000000000 | 2.211972610843544 | – |
| seed=`1.d0` | – | – | 2.211972610843425 |

Turning to the accuracy of the gradients based on shape design variables, $\mathrm{d}\mathbf{J}/\mathrm{d}\mathbf{x}_{\mathrm{shape}}$, we inspect Tab. 9.2. Clearly, a similar pattern to Chapter 8 can be observed, where

the inclusion of metric terms is followed by a 2-4 drop in significant digits. Still, the 11 and 10 significant digits for the eDA*cs* and the eDA*adf* adjoint solvers are extremely accurate shape design variable gradients. As a final comment, we mention that to fine tune the step sizes for the eDA*fd* adjoint solver (as we described in Chapter 8) one will now definitely have to fix the bug for step sizes less than $10^{-8}$ since the results from Tab. 8.2 suggested that at least some of these step sizes would be around $10^{-11}$.

**Table 9.2:** Verification of shape gradients the turbulence extension in the adjoint solver. We use a 2k cell mesh from Fig. 6.5 instead of the mesh from Chapter 7 we originally developed eDA*fd* and eDA*cs* on. Like in all other gradient verification tables, we have computed the reference gradient with the complexified EllipSys flow solver. The reference gradient was found to be: $\mathrm{d}F_x/\mathrm{d}\mathbf{x}_{\text{shape}} = 3.89943766325412 \cdot 10^{-4}$.

| **Step size**, $h$ | **Significant digits**: | | |
|---|---|---|---|
| | eDA*fd* | eDA*cs* | eDA*adf* $(\cdot 10^{-4})$ |
| $10^{-2}$ | $1.331145707778760 \cdot 10^{-0}$ | $3.961612021294029 \cdot 10^{-1}$ | – |
| $10^{-4}$ | $-1.828886465094587 \cdot 10^{+2}$ | $-1.827910081228651 \cdot 10^{-4}$ | – |
| $10^{-6}$ | $-7.811499897322844 \cdot 10^{-0}$ | $3.898613342539516 \cdot 10^{-4}$ | – |
| $10^{-8}$ | 0.000000000000000 | $3.899437580888657 \cdot 10^{-4}$ | – |
| $10^{-10}$ | 0.000000000000000 | $3.899437663210748 \cdot 10^{-4}$ | – |
| $10^{-20}$ | 0.000000000000000 | $3.899437663223937 \cdot 10^{-4}$ | – |
| seed=`1.d0` | – | – | 3.899437663182723 |

## 9.3 Transition

As an introductory motivation to this final extension we point out that when optimizing the laminar region to reduce drag it is an absolute must to have a detailed control (i.e. many FFD design variables). Thus, an adjoint solver and transition models are in a sense a perfect match since the adjoint approach with careful implementation is independent of the number of design variables.

The two transitional variables we add are, $\gamma$, for the turbulence intermittency, and $\tilde{Re}_{\theta t}$ for the transport of the momentum thickness Reynolds number. As a result, the extended state vector, $\tilde{\mathbf{w}} = [\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{p}, \mathbf{c1}, \mathbf{c2}, \mathbf{c3}, \boldsymbol{k}, \boldsymbol{\omega}, \boldsymbol{\gamma}, \tilde{\mathbf{Re}}_{\theta t}]^T$, now counts 11 different types of variables.

The transport equation for $\gamma$ is [63, Eq. 3],

$$\frac{\partial(\rho\gamma)}{\partial t} + \frac{(\rho u_i \gamma)}{\partial x_i} = P_\gamma - E_\gamma + \frac{\partial}{\partial x_i}\left[\left(\mu + \frac{\mu_t}{\sigma_f}\right)\frac{\partial \gamma}{\partial x_i}\right], \tag{9.5}$$

where $P_\gamma$ is a transition source term and $E_\gamma$ is a relaminarization term. Similarly, we have a transport equation for $\tilde{Re}_{\theta t}$ that reads [63, Eq. 21],

$$\frac{\partial \rho \tilde{Re}_{\theta t}}{\partial t} + \frac{\partial (\rho u_i \tilde{Re}_{\theta t})}{\partial x_i} = P_{\theta t} + \frac{\partial}{\partial x_j}\left[\sigma_{\theta t}(\mu + \mu_t)\frac{\partial \tilde{Re}_{\theta t}}{\partial x_i}\right], \tag{9.6}$$

which as seen also has a source term, $P_{\theta t}$. The current implementation in EllipSys can handle natural transition, by-pass transition, and separation induced transition. We refer to [63] for details on the model.

Again, two residual subroutines are hand written and verified to reproduce the residuals of the EllipSys flow solver. Similar to the turbulence extension we add two completely new files to the code base, namely `adj_Res_gamma.f` and `adj_Res_theta.f`, as seen in Fig. 9.2. We will omit the inspection of the state Jacobian for now as we also use the full RANS model with models added for both turbulence and transition in the next chapter where we implement MPI. Thus, a state Jacobian where a transition model has been included can be found in Fig. 10.2.

### 9.3.1 Gradient verification after transition extension

The results from testing the aerodynamic gradient, $\mathrm{d}F_x/\mathrm{d}\mathbf{x}_{\mathrm{u\_BC}}$, can be found in Tab. 9.3. In short, the results are almost identical to Tab. 9.1 except for the obvious fact that the reference gradient value changes due to a change in the underlying fluid model. Only the eDA*adf* performance differ where an extra significant digit is now achieved. Thus, eDA*fd*, eDA*cs*, and eDA*adf* obtains up to 5, 15, and 14 significant digits, respectively. Again, the FD-based adjoint solver exhibits a behavior that points to a bug for step sizes smaller than $10^{-8}$.

**Table 9.3:** Verification of a gradient from an aerodynamic BC design variable with the transition extension added to the adjoint solver. We use the 2k cell mesh from Fig. 6.5 instead of the mesh from Chapter 7 we originally developed eDA*fd* and eDA*cs* on. Like in all other gradient verification tables, we have computed the reference gradient with the complexified EllipSys flow solver. The reference gradient was found to be: $\mathrm{d}F_x/\mathrm{d}\mathbf{x}_{\mathrm{u\_BC}} = 2.21197257217237 \cdot 10^{-1}$.

| **Step size**, $h$ | **Significant digits**: | | |
|---|---|---|---|
| | eDA*fd* $(\cdot 10^{-1})$ | eDA*cs* $(\cdot 10^{-1})$ | eDA*adf* $(\cdot 10^{-1})$ |
| $10^{-2}$ | 2.310744901735302 | 2.241075254429230 | – |
| $10^{-4}$ | 2.216262511809254 | 2.211132150859631 | – |
| $10^{-6}$ | 2.211947854314820 | 2.211972481088151 | – |
| $10^{-8}$ | 0.000000000000000 | 2.211972572163201 | – |
| $10^{-10}$ | 0.000000000000000 | 2.211972572172239 | – |
| $10^{-20}$ | 0.000000000000000 | 2.211972572172372 | – |
| seed=`1.d0` | – | – | 2.211972572172337 |

Finally, we arrive at the gradient verification in Tab. 9.4 for shape design variables, $\mathrm{d}F_x/\mathrm{d}\mathbf{x}_{\mathrm{shape}}$, where both turbulence and transition models are included in the adjoint

solver. Comparing to Tab. 9.2 we realize that the performance is exactly the same, i.e. 0, 11, and 10 significant digits are achieved by eDA*fd*, eDA*cs*, and eDA*adf*, respectively.

To sum up, we have demonstrated a feasible way to augment the adjoint solver with models for turbulence and transition. This came at the cost of losing the eDA*fd* as a viable option for high-fidelity shape optimization since a bug is assumed to linger. At the very least, we would have to carry out refined step size studies. However, these considerations are of mere academic interest given that two superior adjoint solvers, eDA*cs* and eDA*adf*, have been successfully developed as well. We therefore proceed using only eDA*cs* and eDA*adf* to the final developmental Chapter 10 where we enable MPI capabilities.

**Table 9.4:** Verification of shape gradients the transition extension in the adjoint solver. We use a 2k cell mesh from Fig. 6.5 instead of the mesh from Chapter 7 we originally developed eDA*fd* and eDA*cs* on. Like in all other gradient verification tables, we have computed the reference gradient with the complexified EllipSys flow solver. The reference gradient was found to be: $\mathrm{d}F_x/\mathrm{d}\mathbf{x}_{\mathrm{shape}} = 3.899438778928179 \cdot 10^{-4}$.

| **Step size**, $h$ | **Significant digits**: | | |
|---|---|---|---|
| | eDA*fd* | eDA*cs* | eDA*adf* ($\cdot 10^{-4}$) |
| $10^{-2}$ | $1.331213989945838 \cdot 10^{-0}$ | $3.961160820397340 \cdot 10^{-1}$ | – |
| $10^{-4}$ | $-1.828889622221733 \cdot 10^{+2}$ | $-1.829862435548700 \cdot 10^{-4}$ | – |
| $10^{-6}$ | $-7.811404399715568 \cdot 10^{-0}$ | $3.898614444282147 \cdot 10^{-4}$ | – |
| $10^{-8}$ | 0.000000000000000 | $3.899438696448658 \cdot 10^{-4}$ | – |
| $10^{-10}$ | 0.000000000000000 | $3.899438778914684 \cdot 10^{-4}$ | – |
| $10^{-20}$ | 0.000000000000000 | $3.899438778978503 \cdot 10^{-4}$ | – |
| seed=`1.d0` | – | – | 3.899438778880244 |

# CHAPTER 10
# MPI implementation

> "Parallel implementation is likely to be particularly important for design using the Navier-Stokes equations, for which much finer meshes are needed to assure sufficient accuracy, with a corresponding increase in the computational cost"[47]
>
> Anthony Jameson, professor, Department of Mechanical and Aerospace Engineering, University of Princeton,
> in his seminal '95 paper when discussing the imminent transition from Euler to Navier–Stokes equations

The more than two decades old statement by Jameson above underlines that there long has been an emphasis on MPI capabilities for the adjoint approach in CFD. We will in the present chapter describe one way of extending an adjoint solver to be run in parallel. There are many ways to do so depending on the flow solver at hand, and once a correct MPI implementation is ensured there are endless ways to optimize the setup, which we cannot hope to cover. Thus, we focus on ensuring correctness and mention a few do's and don't.

To identify an ambitious lower bound to aim at when developing this part of the framework we point out, that a typical multipoint setup where just three flow cases are considered could demand $3 \cdot 216 = 648$ CPUs to cooperate across three sub-comms as we showed recently [71]. For this to be possible one should not only be able to assemble and solve the adjoint equation in parallel but also do so simultaneously in subgroups. Arriving at a developmental stage where the above multipoint optimization problem can be run should allow numerous interesting research questions to be answered, which would all contribute to the absolute forefront of high-fidelity aerodynamic shape optimization of wind energy research.

We first consider the necessary changes to the adjoint solver code base in order to enable MPI. This step is very solver dependent and we make heavy use of the low-level MPI subroutines from the EllipSys flow solver to this end. As seen in Fig. 10.1 we only had to add one new file altogether. Besides the added `PETSc_SolverMPI.f` we also restructured all partial derivative subroutines to adhere to a coloring scheme instead of looping over blocks on a given CPU. Thus, as we implement more and more refined coloring schemes to speed up the Jacobian computation, we need not worry about restructuring the partial derivative subroutines further.

There are many things to keep an eye on when assembling the adjoint system in parallel. One example, is that the ordering of the states in the extended state vector, $\tilde{\mathbf{w}} = [\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{p}, \mathbf{c1}, \mathbf{c2}, \mathbf{c3}, \boldsymbol{k}, \boldsymbol{\omega}, \boldsymbol{\gamma}, \tilde{\mathbf{Re}}_{\theta t}]^T$, will change, now that the same mesh


**Figure 10.1:** Visualization of the code folder structure after enabling MPI capabilities in the adjoint solver. The left hand side shows an excerpt of the overall code structure in EllipSys. To the right some of the content of the `./Adjoint` folder can be seen. The only new file we had to add was the `PETSc_SolverMPI.F` file containing all MPI related subroutines necessary to make the adjoint solver function in parallel.

is distributed on several CPUs. However, the faces between blocks should still not be represented in the adjoint system more than once despite the fact that both CPUs will own a related face in the flow solver. We explained this subtlety in Sec. 7.2.3 and thanks to the strict implementation in EllipSys, we need not worry about a degenerate representation since the block-block structure automatically is preserved whether we split an interface across CPUs or keep it on a single CPU. Still, we know for a fact that this is not the case on all CFD solvers and it should therefore be duly mentioned.

An example of a state Jacobian for the mesh used in Chapters 8 and 9 where both turbulence and transition modeling is included is seen in Fig. 10.2. To the left in Fig. 10.2 we have highlighted (red rectangle) the adjoint subsystem for the second CPU out of a total of 4 CPUs used to assemble the adjoint system. The diagonal sub-matrix pertaining to the second CPU is indicated (see also Fig. 7.6, 7.7 and 9.3). Despite the minuscule mesh ($\approx$ 1k cells) it is completely impossible to discern the sparsity structure of the fill in. This is certainly true for the full state Jacobian (left, Fig. 10.2), but also the zoom of solely the diagonal sub-matrix for the second CPU (right, Fig. 10.2) is difficult to inspect. We note, that the full state Jacobian matrix is $[11072 \times 11072]$ for the $\approx$1k cell mesh. As a consequence of extending the uniqueness rule for the faces to interfaces between CPUs, the $[11072 \times 11072]$ size for the state Jacobian is fixed irregardless of calling 1, 2, or e.g. 16 CPUs to compute the adjoint system.

## 10.1 PETSc MPI solver

A crucial point to mention is that PETSc will split the linear system in equal sizes for optimal load balancing. However, the adjoint system in the Fortran layer is split in a blockwise order. The worst possible example is a mesh with three blocks running on two CPUs. Here, processor 1 would own two thirds of the solution vector ($\mathbf{\Psi}$) in the Fortran layer, but in PETSc it would only own half the solution vector since the linear system

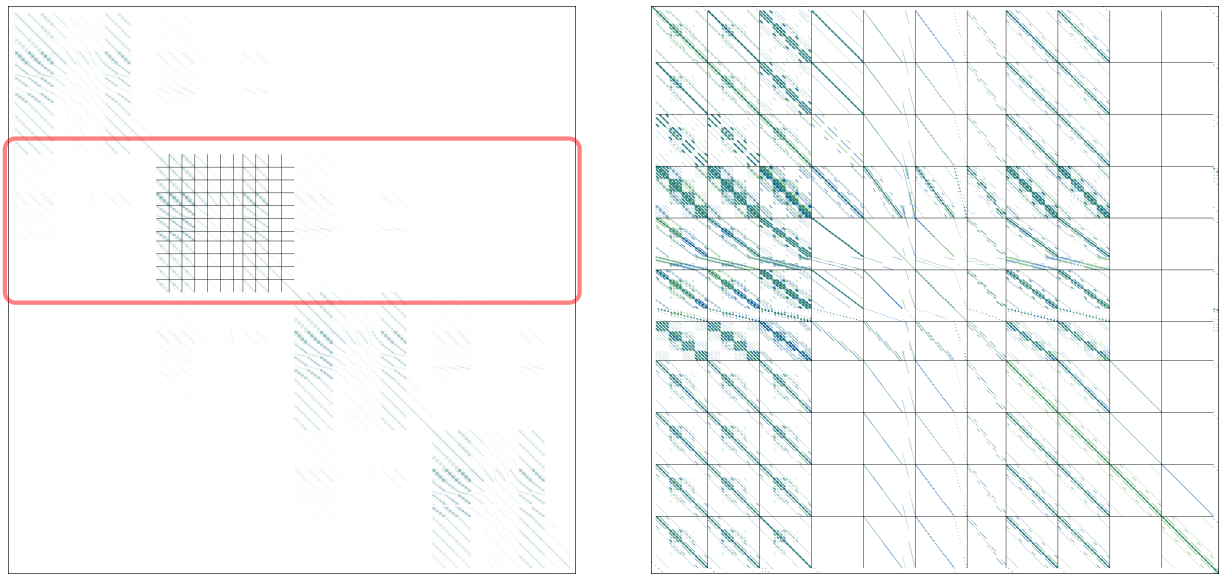


**Figure 10.2:** Visualization of the state Jacobian for the full RANS model complemented with models for turbulence and transition running on four CPUs. To the left, we highlight the sub-matrix for the second CPU with the familiar lines (see also Fig. 7.6, 7.7 and 9.3) separating the various types of variables in the extended state vector, $\tilde{\mathbf{w}} = [\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{p}, \mathbf{c1}, \mathbf{c2}, \mathbf{c3}, \boldsymbol{k}, \boldsymbol{\omega}, \boldsymbol{\gamma}, \tilde{\mathbf{Re}}_{\theta t}]^T$. To the right we show the sub-matrix for the second CPU. Although the mesh is quite small (≈1k cells) it is hard to distinguish the sparsity structure since the resulting Jacobian is a $[11072 \times 11072]$ matrix. Still, a diagonally dominated sparse fill in can be observed in many of the $11 \cdot 11 = 121$ sub-matrices for the second CPU.

would be split evenly. Luckily, writing a few re-distribution MPI subroutines solved the issue for us. One can of course also instruct PETSc to maintain the ordering from the Fortran layer, but we would not recommend it, since it would worsen the load balance.

Once all the technicalities of assembling the adjoint system in parallel have been handled it is time to commit to the final settings for the PETSc solver. We use a setup close to that reported by Kenway et al. [55, p. 21] but without the use of inner/outer Richardson iterations. Thus, we employ a global GMRES solver to the overall adjoint system. The additive Schwartz method (ASM) with one level of overlap is used to split the global system into sub-blocks on each CPU. In Fig. 10.2, the red rectangle would be the second sub-block and PETSc would assign it to the second CPU. After PETSc has split the system into sub-blocks it can now be solved in parallel. For a reasonable convergence rate while still limiting the memory impact we find that a local ILU(1)[1] preconditioning is an efficient compromise.

[1] i.e. the incomplete lower and upper factorization with a level 1 fill in

## 10.2 Gradient verification using MPI

To conclude the final chapter of the second part in the dissertation we present results from testing the gradient precision after enabling the MPI capabilities. The results from testing the aerodynamic gradient, $\mathrm{d}F_x/\mathrm{d}\mathbf{x}_{\mathrm{u\_BC}}$, can be found in Tab. 10.1 whereas the results from testing the gradient using shape design variables, $\mathrm{d}F_x/\mathrm{d}\mathbf{x}_{\mathrm{shape}}$ are seen in Tab. 10.2. Given that both turbulence and transition models are included the tables resembles the results from the previous chapter quite a bit (see Tab. 9.3 and 9.4) with the obvious difference that now we vary the number of CPUs used to compute the gradient instead of inspecting the step size dependency (which is only relevant for eDA*cs*). As already mentioned, we have left out the eDA*fd* results since the inclusion of the turbulence and transition models in the previous chapter suggested that a bug lingered in the code.

As seen in the tables the purely aerodynamic gradients are the most accurate results reaching up to 15 and 13 significant digits for eDA*cs* and eDA*adf*, respectively. This tendency could also be observed in Chapters 7 to 9. The only surprise in Tab. 10.1 related to the bottom two eDA*cs* results. We have no immediate explanation for this 2 digit drop off but thankfully the adjoint solver based on algorithmic differentiation seems very stable.

The shape gradients in Tab. 10.2 also carry over tendencies from previous chapters. The 10-11 significant digits for eDA*cs* are very similar to previously shown performance for running in serial. Turning to the eDA*adf* adjoint solver based on algorithmic differentiation it does seem to oscillate slightly but we ascribe this to the aforementioned lacking fine tuning when exiting the PETSc solver. Still, the 10-12 significant digits are representative of results from previous chapters and should certainly be more than enough to conduct high-fidelity aerodynamic shape optimization.

**Table 10.1:** Table showing gradient precision for an aerodynamic design variable after enabling MPI capabilities. The reference gradient from the complexified EllipSys flow solver was found to be: $\mathrm{d}F_x/\mathrm{d}\mathbf{x}_{\mathrm{u\_BC}} = 2.211972572172374 \cdot 10^{-1}$.

| **CPUs** | **Significant digits**: | |
|---|---|---|
| | eDA*cs* $(\cdot 10^{-1})$ | eDA*adf* $(\cdot 10^{-1})$ |
| 2 | 2.211972572172368 | 2.211972572172907 |
| 4 | 2.211972572172372 | 2.211972572172935 |
| 8 | 2.211972572162977 | 2.211972572172875 |
| 16 | 2.211972572142259 | 2.211972572172897 |

**Table 10.2:** Table showing gradient precision for a shape design variable after enabling MPI capabilities. The reference gradient from the complexified EllipSys flow solver was found to be: $\mathrm{d}F_x/\mathrm{d}\mathbf{x}_{\mathrm{shape}} = 3.899438778928179 \cdot 10^{-4}$.

| **CPUs** | **Significant digits**: | |
|---|---|---|
| | eDA*cs* $(\cdot 10^{-4})$ | eDA*adf* $(\cdot 10^{-4})$ |
| 2 | 3.89943877862031 | 3.89943877846912 |
| 4 | 3.89943877878503 | 3.89943877892463 9 |
| 8 | 3.89943877829690 9 | 3.89943877889565 6 |
| 16 | 3.89943877826309 4 | 3.89943877849396 |

# Part III

# High-fidelity shape optimization of wind turbine blades

# CHAPTER 11

# High-fidelity shape optimization of wind turbine blades

This chapter is the first and only chapter in Part III. Here, we have chosen two applications which we present in two consecutive sections. The first application is a single-point drag minimization of a 3-D wing subject to a lift constraint where we use five twist design variables. This optimization is carried out using our newly developed numerical shape optimization framework which was presented in Part II. In Sec. 11.2 we bring the second application which is a multipoint shape optimization of a full rotor configuration which we carried out using the MACH framework in corporation with the University of Michigan. This application is one of several optimizations one can find in a recent publication [71]. We chose this application since it represents exactly what we hope to carry out in the near future using our own framework, i.e., robust adjoint-based high-fidelity multipoint shape optimizations of full rotor configurations.

## 11.1 3-D wing optimization

In this section, we use a test case to evaluate our numerical design framework described in great detail in Part II. The framework can be seen in Fig. 11.1 (also shown in Chapter 4, Fig. 4.1).

We chose the 3-D wing optimization for the obvious reason that it is one of the few aerodynamic shape optimization problems with an analytical solution: We know, that the lift distribution along the wing should become elliptic for the downwash to be constant across the wing span:

$$L(z) = L_0 \sqrt{\left(1 - \left(\frac{z}{(b/2)}\right)^2\right)}. \tag{11.1}$$

Above, $L_0$ is the spanwise lift at the root of the wing, $b$ is the span, and $z$ is the position along the $z$-axis which is the spanwise direction for the wing. Thus, we can easily quantify how well the optimization problem is solved.

This section will show that the framework certainly is a work in progress. As will be evident in the benchmark results it is at present not feasible to handle meshes much larger than $100 \cdot 10^3$ cells with the adjoint solver since we have yet to finalize our coloring acceleration. However, it will also be evident that the framework already in the present

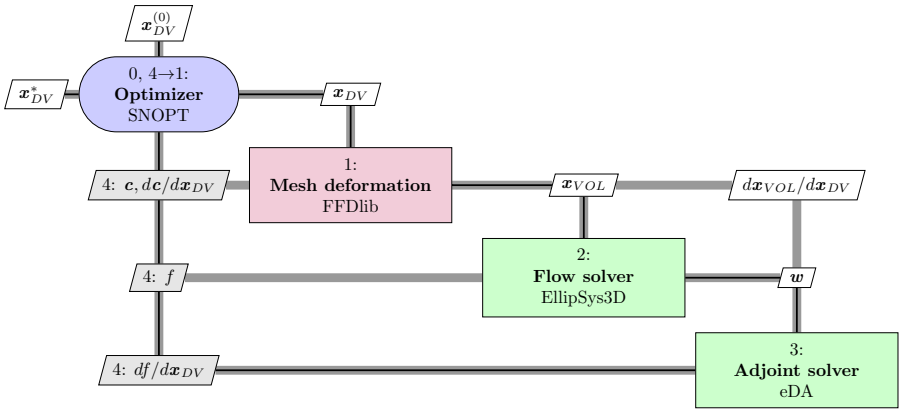



**Figure 11.1:** Components in the numerical high-fidelity shape optimization framework at DTU Wind Energy. The four components are (0,4) an optimizer, (1) a mesh deformation module, (2) a CFD solver, and (3) a discrete adjoint solver. Gray lines show data flow. Black lines show process flow.

state is fully functional to carry out shape optimizations. The largest case handled by the framework in the following is a 664 thousand cell wing mesh using the complex step method to compute gradients, whereas the largest case handled by the adjoint solver is a mesh with 83 thousand cells.

The present section is structured as follows. First, we conduct a mesh convergence study (Sec. 11.1.1) to quantify how well the physics are captured on the mesh levels we are currently able to handle with our framework. Then follows a short description (Sec. 11.1.2) of the design optimization problem. Here, we also present the FFD setup. Finally, the results are presented (Sec. 11.1.3) including a benchmark (Sec. 11.1.3.1) before we conclude by identifying future work for the framework development (Sec. 11.1.4).

## 11.1.1 Mesh convergence study

Fig. 11.2 shows the 3-D wing mesh used in the present section. An overview of the half sphere mesh is given to the left whereas the right hand side shows a close-up of the wing. Also in the right hand side of Fig. 11.2 one can find an inset with a view down the spanwise direction of the wing where a 4 degree angle of attack is visible.

The surface mesh for the wing was generated using the in-house Parametric Geometry Library (PGL). Here, a wing was generated with a chord of 1 m, and a half-span of 8 m using a NACA0015 airfoil. The wing was twisted by a constant four degrees about the $z$-axis. A chordwise resolution of 384 cells is achieved using 8 surface mesh blocks, each with $48 \times 48$ cells. The wing tip is made up of 4 blocks with similar resolution. This results in a surface mesh with 12 blocks and a total of 27648 cells.

The volume mesh is a half sphere generated using the hyperbolic in-house mesh generator HypGrid [115]. Given that we run all wing optimizations at a Reynolds number of $Re = 10 \cdot 10^6$ we set the first cell height to $2 \cdot 10^{-6}$ [m] which ensures a $y^+$ below 1 for

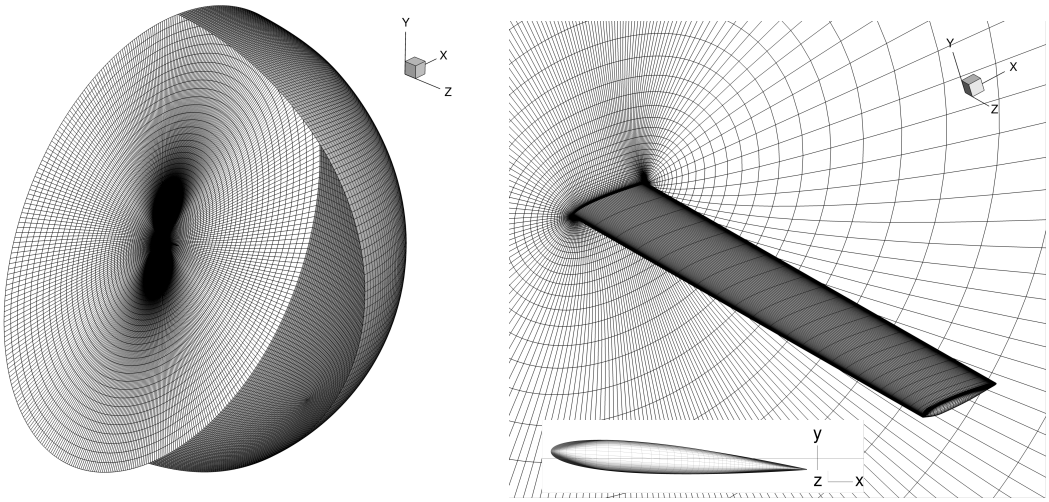



**Figure 11.2:** The left hand side is an overview of the L0 3-D wing mesh. The right hand side shows a close-up of the wing and from the inset one can verify the 4 degree angle of attack. All mesh level resolutions are found in Tab. 11.1.

the L0 mesh level. The entire outer half sphere seen in the left hand side of Fig. 11.2 is an inlet zone (gray) whereas the $xy$-plane intersecting origin (white mesh) is a symmetry plane. An outlet zone is prescribed on the spherical outer boundary of appropriate size downstream of the wing. By growing 192 cell layers from the surface wing mesh seen to the right we arrive at a 48 block volume mesh totaling 5.308416 million cells. The outermost vertices are placed in a radius of 90 [m] from origin.

The above described mesh is the finest level, L0. To obtain the next mesh level in the series, L1, which is coarser than L0, we remove every second cell in all coordinate directions. Correspondingly, one can obtain the L2 mesh by coarsening the L1 mesh and so on.

Given that the mesh has 48 blocks we will use 48 CPUs in the following. All computations were carried out on the Jess[1] computation cluster which has 320 nodes with each 20 CPUs[2] connected with InfiniBand.

Having described the computational mesh we are now ready to carry out our mesh convergence study. The result can be seen in Tab. 11.1.

In Tab. 11.1, we have set the EllipSys residual limit to $10^{-10}$ and listed the computed lift and drag metrics for the wing using both a first-order scheme (UDS) and a third-order scheme (QUICK). In order to quantify the error in percentages we must estimate the continuum mesh value, $f_c$, i.e., the value a mesh with infinitely fine cells would have. To this end we use a Richardson extrapolation where $f_c$ is given by [103, Eq. 3]:

$$f_c \approx f_0 + \frac{f_0 - f_1}{r^2 - 1}. \tag{11.2}$$

[1] `https://docs.hpc.ait.dtu.dk/Jess/`,(last access: 10 Dec 2019)

[2] Xeon E5-268v2 running at 2.8 GHz

**Table 11.1:** Mesh convergence study for the incompressible EllipSys3D flow solver. The Richardson extrapolations from Fig. 11.3 have been used to obtain the error percentages.

| | | UDS[a)] | | | | QUICK[a)] | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Mesh** | **Cells** | **Drag** | **Error** | **Lift** | **Error** | **Drag** | **Error** | **Lift** | **Error** |
| | [million] | $\cdot 10^{-2}$[N] | [%] | [N] | [%] | $\cdot 10^{-2}$[N] | [%] | [N] | [%] |
| L3 | 0.010 | 19.76 | 234.2 | 1.36 | 9.6 | 6.28 | 43.0 | 1.47 | 2.8 |
| L2 | 0.083 | 12.42 | 110.1 | 1.45 | 3.7 | 4.66 | 6.1 | 1.53 | 1.0 |
| L1 | 0.664 | 8.60 | 45.4 | 1.48 | 1.6 | 4.43 | 0.9 | 1.52 | 0.5 |
| L0 | 5.308 | 6.59 | 11.4 | 1.50 | 0.4 | 4.40 | 0.2 | 1.52 | 0.1 |
| Extrapolation | $\infty$ | 5.9 | 0.0 | 1.50 | 0.0 | 4.39 | 0.0 | 1.51 | 0.0 |

[a)] Computational schemes for the EllipSys3D flow solver. UDS is first-order accurate and QUICK is third-order accurate.

Above, $f_c$ is the continuum value and $f_0$ and $f_1$ are values we obtained from meshes L0 and L1, respectively. The $r$ represents the mesh refinement ratio which in our case is two. The finest mesh we use in optimizations is L1. Thus, L0 is only used in the analysis part in the present section to carry out the mesh convergence study.

In order to assess the mesh convergence study and the related Richardson extrapolations from Tab. 11.1 we visualize these results in Fig. 11.3.

There are several important conclusions to draw from Fig. 11.3. First of all one should expect the values obtained with UDS and QUICK stencils to agree at the infinitely fine mesh levels. Certainly, it is acceptable that the third-order QUICK stencil is much better at estimating the continuum value at a given mesh level, but for infinitely fine meshes the values should agree. This is, however, not the case. To improve the mesh convergence study we could introduce finer mesh levels, L-1, L-2, and so on until the Richardson extrapolations were in better agreement. This is, however, outside the scope of the present section. Another point to make is that the lines in Fig. 11.3 are not straight. The higher order terms we neglected in the Eq. 11.2 approximations are in other words not negligible for the chosen mesh levels and therefore we do not see a straight line as we should for the chosen $x$-axis. Examples of more linear convergence rates can be found in [71, Fig. 5] but that mesh convergence study was also conducted with mesh sizes close to 50 million cells. For now, it will suffice that we have quantified how much the various mesh levels are in agreement for each stencil type. Here, we note that the third-order QUICK stencil already at the L2 mesh level have errors less than 10% for both lift and drag.

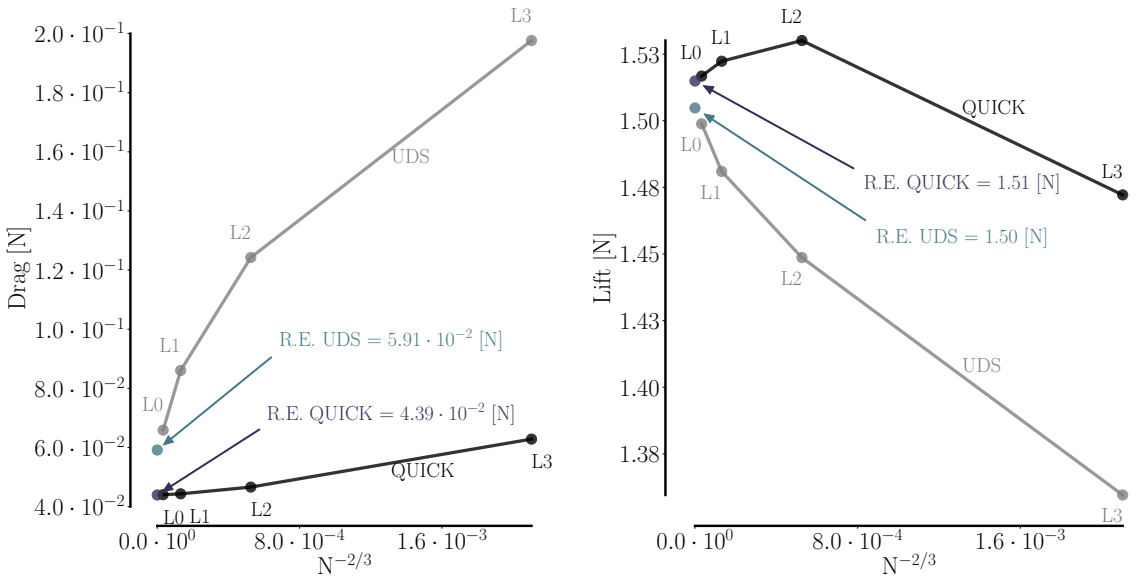



**Figure 11.3:** A mesh convergence study for the first-order stencil, UDS, and the third-order stencil, QUICK, to quantify how well the four mesh levels from Tab. 11.1 describe the flow. N along the $x$-axis is the number of mesh cells. The abbreviation R. E. stands for Richardson extrapolation (see Eq. 11.2). The left hand side compares drag values to the Richardson extrapolation drag values where we obtain L0 error percentages of 0.2% and 11.4% for the QUICK and UDS scheme, respectively. To the right the lift values can be inspected. Here, the lift values for the L0 mesh compared to the Richardson extrapolation values gives error percentages of 0.1% and 0.4% for the QUICK and UDS scheme, respectively.

### 11.1.2 Design optimization problem

We are now ready to present the optimization problem,

$$\begin{aligned} \text{minimize} \quad & \text{drag} \\ \text{with respect to} \quad & \text{twist} \\ \text{subject to} \quad & \text{lift} = L_{\text{init}}, \end{aligned} \tag{11.3}$$

where $L_{\text{init}}$ is the lift obtained on the unperturbed mesh.

We use the free-form deformation parameterization described in Chapter 5 to impose shape changes on the geometry, where we define a box with dimensions $x = 2 \times$ chord, $y = 0.5 \times$ chord, and $z = 10 \times$ chord, thus the box extends 2 m beyond the tip of the wing. In this setup, there are five twist design variables distributed evenly along the FFD box in the spanwise direction. In the two other coordinate directions the FFD box is only resolved with 2 points, since we only seek to deform the blade with a rigid body

twist change. Once the optimizer (see Fig. 11.1) updates any of the design variables FFDlib deforms both the surface and the volume mesh. It also sends its analytical mesh derivatives further down the process stream so that these can be included in the total derivative computation. To see how FFDlib is setup to undertake also the volume deformation and to better inspect where the five twist design variables are positioned we show the FFD setup (blue) for the wing mesh in Fig. 11.4.

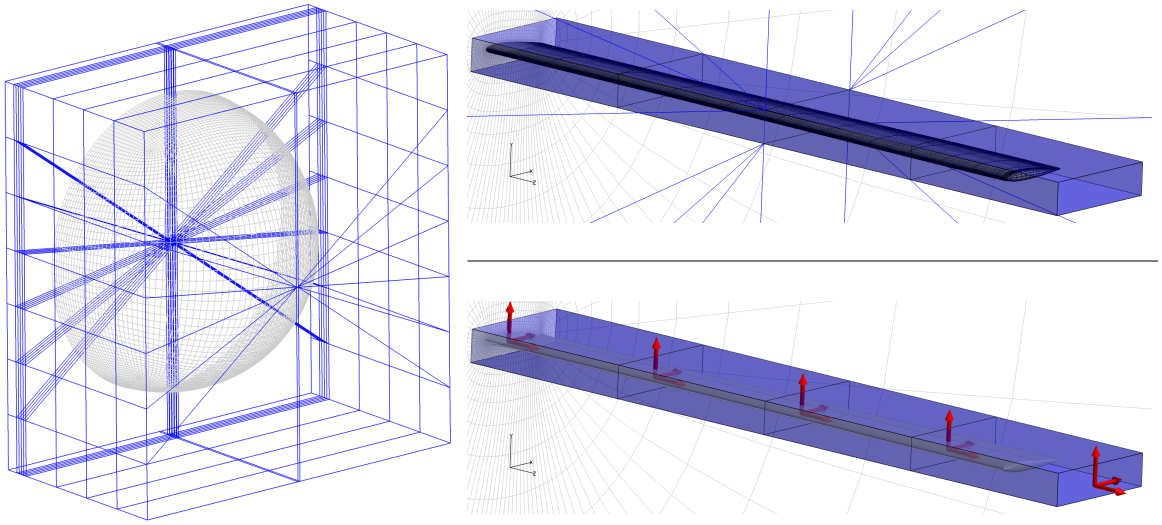


**Figure 11.4:** The left hand side is an overview of the outer FFD box encapsulating the entire volume mesh. The right hand side shows two insets: Upper inset shows how the inner FFD control points form another box (blue) which is closely set around the wing to ensure a high deformation control of the surface mesh. Blue connecting lines are only shown for the third twist section. Lower inset shows the positions of the five twist design variables marked by red unit vectors.

To the left in Fig. 11.4 one can see the outline of the rather large FFD box encapsulating the entire volume mesh. The two insets on the right hand side of the figure shows two close-ups of the wing surface. In the upper inset a smaller closely set box is located around the wing to ensure a high degree of local deformation control. This inner FFD box can be seen to have many blue lines connecting it to the outer box. In the lower inset we have removed all the blue FFD lines to highlight the position of the five twist design variables. These are visualized as five sets of rotation axis with red unit vectors. The unit vector used for twist deformation is the one pointing in the spanwise direction. The other unit vectors are used by FFDlib in case chord or thickness variables are needed.

### 11.1.3 Results

We solve the optimization problem from Eq. 11.3 with our framework using both the CS-method and the adjoint method on mesh levels L3 and L2, using the SNOPT optimizer [32]. Using both the UDS and the QUICK stencils this amounts to eight optimizations (four CS-based and four adjoint-based). Furthermore, we used the CS-method to run some of the finer mesh levels, to investigate whether this could further improve the final shape.

Before presenting the results we first inspect the measured sensitivity reported in Tab. 11.2.

**Table 11.2:** Table showing the eDA*adf* adjoint solver's gradient precision of the drag with respect to twist design variables. The reference gradient was found with the complexified EllipSys flow solver.

| **DV** | **Significant digits on L3**: | | | |
|---|---|---|---|---|
| | **UDS** | | **QUICK** | |
| | Complex-step ($\cdot 10^{-3}$) | eDA*adf* ($\cdot 10^{-3}$) | Complex-step ($\cdot 10^{-3}$) | eDA*adf* ($\cdot 10^{-3}$) |
| 1 | −4.286618086230437 | −4.286618926152518 | −1.408557838341691 | −1.245911104858749 |
| 2 | −7.026560520373469 | −7.026563181416801 | −2.392148621857154 | −2.161250803024587 |
| 3 | −7.117837054371840 | −7.117840175321519 | −2.694458248236351 | −2.552794935225120 |
| 4 | −5.744341121075573 | −5.744341749535324 | −2.547573784819831 | −2.527148716446644 |
| 5 | −1.368841963195332 | −1.368841648291211 | −0.670748350143506 | −0.678506961537469 |
| | **Significant digits on L2**: | | | |
| | Complex-step ($\cdot 10^{-3}$) | eDA*adf* ($\cdot 10^{-3}$) | Complex-step ($\cdot 10^{-3}$) | eDA*adf* ($\cdot 10^{-3}$) |
| 1 | −2.982491725740956 | −2.982493284253812 | −0.959493471302423 | −0.608572548209246 |
| 2 | −5.127000046226491 | −5.127003415526973 | −1.756318454237536 | −1.197168181262014 |
| 3 | −5.234955074364305 | −5.234959360007437 | −2.119915357151773 | −1.675398236377225 |
| 4 | −4.310142223003504 | −4.310143879783881 | −2.152997531833337 | −1.907686718060984 |
| 5 | −1.015239510176375 | −1.015239461874037 | −0.570873787230461 | −0.528092278660713 |

As seen, we have 5-7 significant digits for the UDS scheme whereas we only muster a mere 0-2 digits for the QUICK scheme. Importantly, however, the sign is always right. Evidently, we have an error somewhere in our differentiated QUICK stencil subroutines which accumulates with mesh size. As of now, we can only speculate where the error might be, but a possible explanation is an incorrect handling of internal boundaries in the hand-written residual functions, possibly both in the handling of the face flux and vertex perturbations. While we are able to converge the present optimization case reasonably well with this level of accuracy, it is clearly of very high priority for us to rectify this discrepancy, since other mesh and optimization problem sizes may result in the error being further pronounced than in the current case.

Turning to the eight aforementioned optimizations (four on L3 and four on L2) we first present the four optimizations carried out on L3 in Fig. 11.5.
Fig. 11.5 shows the normalized SNOPT merit functions to the left and the SNOPT optimality convergence to the right. The merit functions are augmented Lagrangian

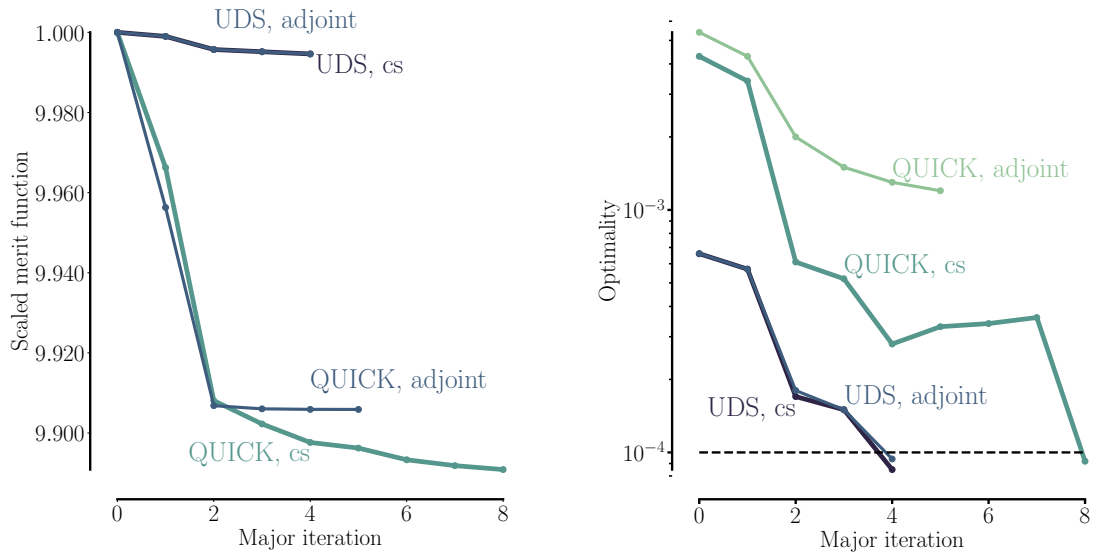



**Figure 11.5:** L3 optimization of the drag minimization problem from Eq. 11.3 using both the CS method and the adjoint method for two different computational stencils (UDS and QUICK). The scaled merit functions from the SNOPT optimizer (left). To the right the SNOPT optimality is seen. The optimality value is a measure for how tightly the optimization problem (Eq. 11.3) is converged. The black dashed line resembles the chosen convergence criteria.

functionals where infeasibility is incorporated into the final score. Whereas the UDS merit function trajectories to the left are almost identical it is noteworthy that the adjoint-based QUICK optimization only achieves 86% of the CS-based drag reduction in merit function. The reason is the impaired gradient precision for the QUICK stencil seen in Tab. 11.2. In the interest of saving space we showed both the UDS and the QUICK merit function results on the same plot although the scaling for the UDS optimizations is less than ideal. However, we will later on bring a comparative plot (Fig. 11.11) of an adjoint-based and a CS-based UDS optimization so that the almost identical behavior can be better inspected.

Turning to the optimality plot on the right hand side of Fig. 11.5 we find exactly what one would expect, i.e., both UDS optimizations as well as the CS-based QUICK optimization are converged below our chosen threshold of $10^{-4}$ (black dashed line). In these three cases less than 10 steps are used by the optimizer. We expect the optimizer is able to converge these three optimization problems very tightly since we know the gradient for these three optimizations is very precise (Tab. 11.2). Correspondingly, we see that the adjoint-based QUICK optimization is not able to obtain below the threshold given that the gradient precision here is somewhat compromised.

To show that the QUICK gradients indeed are useful although they do not allow for a tight convergence of the optimization problem we now inspect the lift distributions in Fig. 11.6.

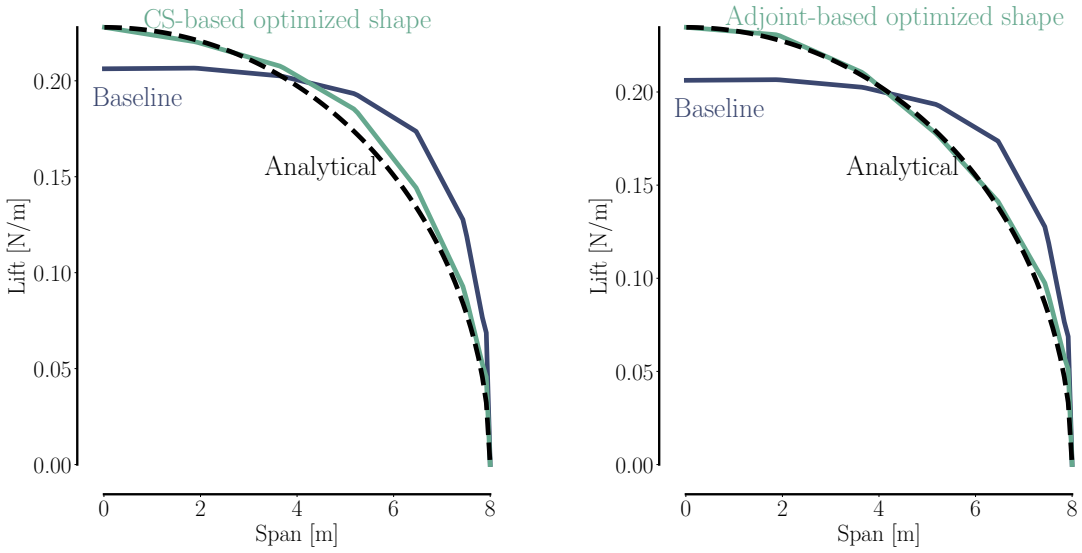



**Figure 11.6:** Resulting lift distributions from drag minimizations (Eq. (11.3)) of a 3-D wing carried out on the L3 mesh using QUICK stencils. The left hand side is a comparison of baseline, CS-based result, and analytical (elliptic Eq. 11.1) lift distribution. The right hand side shows a similar comparison but for the adjoint-based optimization. Evidently, both optimization results are approaching the elliptic solution.

As is hopefully evident from Fig. 11.6 the adjoint-based QUICK result provides a lift distribution that matches the analytical solution surprisingly well considering that we only had 1-2 significant digits. The adjoint-based optimization result only has a small disagreement with the elliptic distribution around 7 [m]. The CS-based optimization result has a clear discrepancy focused at mid span. However, we simply suspect this is owed to the coarseness of the L3 mesh.

Based on inspecting the L3 results in Fig. 11.5-11.6 we can conclude that adjoint-based optimizations are functional but that the final 4-5 significant digits needed for the adjoint QUICK gradient precision to match that for the UDS stencil are simply necessary before we can tightly converge our adjoint-based QUICK optimizations. Another noteworthy finding is, that Fig. 11.5 suggests that the 5-7 significant digits on the UDS stencil seem more than enough in order for us to achieve borderline identical behavior from the optimization regardless of gradient estimation method. This may, however, be problem dependent. This will as mentioned be easier to inspect and confirm in Fig. 11.11.

Turning to the L2 results we find both the merit functions as well as the optimality visualizations in Fig. 11.7 to highly resemble the L3 results from Fig. 11.5. The main difference is perhaps that the curves in general are more smooth. The one exception is the optimality curve in Fig. 11.7 for the adjoint-based QUICK optimization, which seems to exhibit more kinks than the L3 counterpart. Assessing the sensitivity in Tab. 11.2, this corresponds to a slight reduction in the adjoint-based QUICK gradient precision.

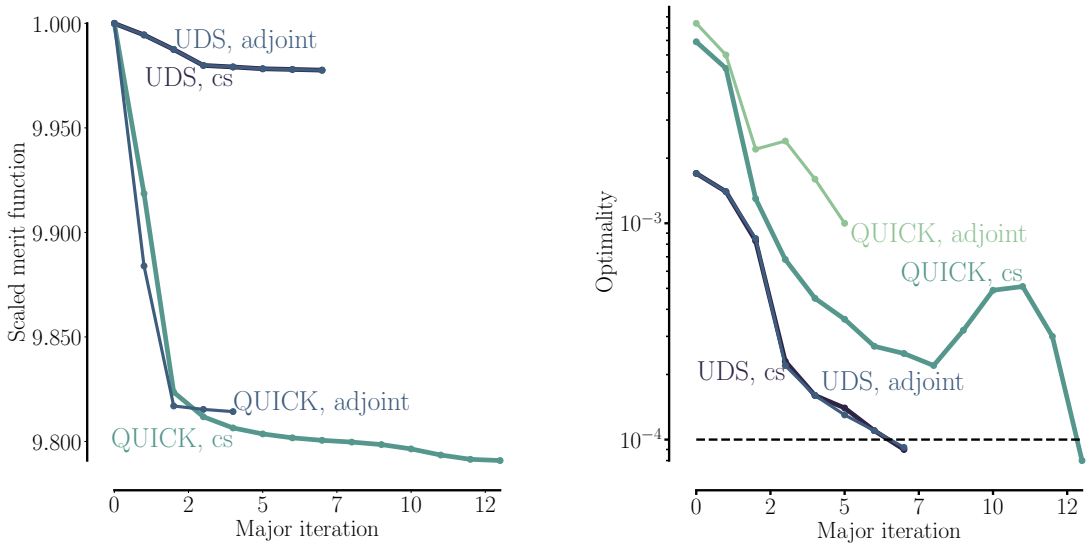



**Figure 11.7:** L2 optimization of the drag minimization problem from Eq. 11.3 using both the CS method and the adjoint method for two different computational stencils (UDS and QUICK). The left hand side shows the scaled merit functions from the SNOPT optimizer. The right hand side shows the SNOPT optimality which is a measure for how tightly the optimization problem is converged. The black dashed line resembles the chosen convergence criteria. A similar plot of the L3 results can be found in Fig. 11.5.

Again, we find the QUICK adjoint-based optimization to stagnate, this time it achieves 89% of the CS-based optimization result. In this context, it should be mentioned that the chosen optimality threshold ($10^{-4}$) is a rather tall order. We used the same threshold in our recent shape optimization study [71] and also here we experienced difficulty converging below said threshold with the MACH framework [71, Fig. 14a and 17a] – albeit these optimization where much more complex.

Next, we inspect the baseline and optimized pressure fields in Fig. 11.8 for L2 optimization results. Here, it is apparent that we are looking at rather small shape changes given that it is hard to distinguish one plot from the other when comparing the baseline (upper) and the optimized (lower) pressure fields. However, we can visually confirm that the tip of the blade is unloaded, with lower pressure peaks, while the inner part of the wing sees an increased loading.

Turning to the lift distributions (Fig. 11.9) for the L2 mesh we see that the tables now have turned. Now, it clearly is the CS-based optimization result that best resembles the elliptic lift distribution. However, the adjoint-based optimization is not far off.

Starting from the right in Fig. 11.9 there certainly is room for improvement for the adjoint-based result although it does approximately achieve an elliptical lift distribution. Clearly, the precision mustered by the adjoint solver for the L2 mesh level is at the absolute minimum for a successful shape optimization. This comes as no surprise given

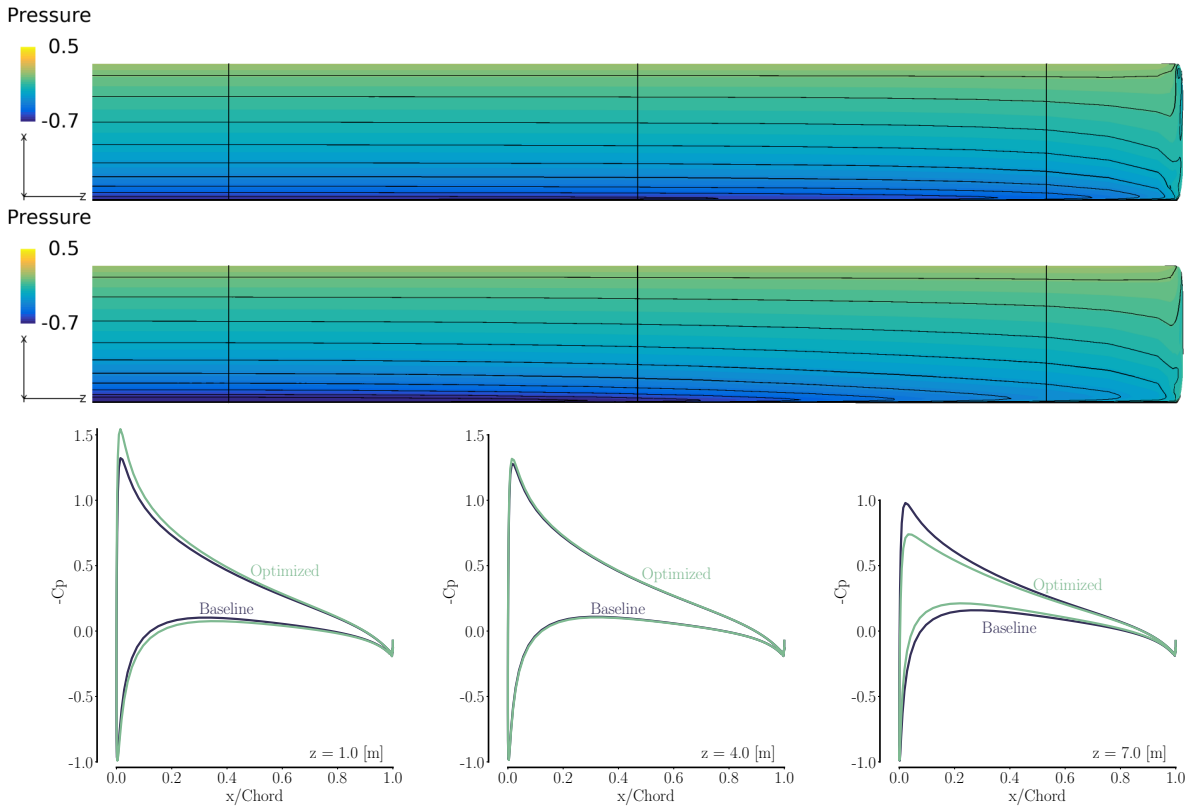



**Figure 11.8:** L2 optimization results. Upper: Baseline suction side of wing. Middle: Optimized suction side of wing. Leading edges are closest to the bottom of the page. Lower plot shows $C_p$-curves at $z = 1$ [m], $z = 4$ [m], and $z = 7$ [m]. These slices are visualized as black lines in the upper and middle plot.

that Tab. 11.2 showed 0-1 significant digits, where we only achieved the correct sign right in the worst cases.

Turning to the CS-based results we find a very close approximation of the analytical lift distribution. There still seems to be a very small mismatch right at the center of the wing for the CS-based optimization although it should provide machine accurate gradients. However, the deviation from the analytical distribution is much less pronounced than for the L3 results from Fig. 11.6. To ensure we approach the analytical lift distribution for finer meshes we compare the results from CS-based optimizations for mesh levels, L3, L2, and L1, in Fig. 11.10.

The comparison across grid levels in Fig. 11.10 for CS-based optimizations clearly shows that the analytical distribution is captured rather well already at the L2 mesh level. The L1 lift distribution does follow the analytical distribution somewhat better, but it hardly seems worth the extra effort in computation time.

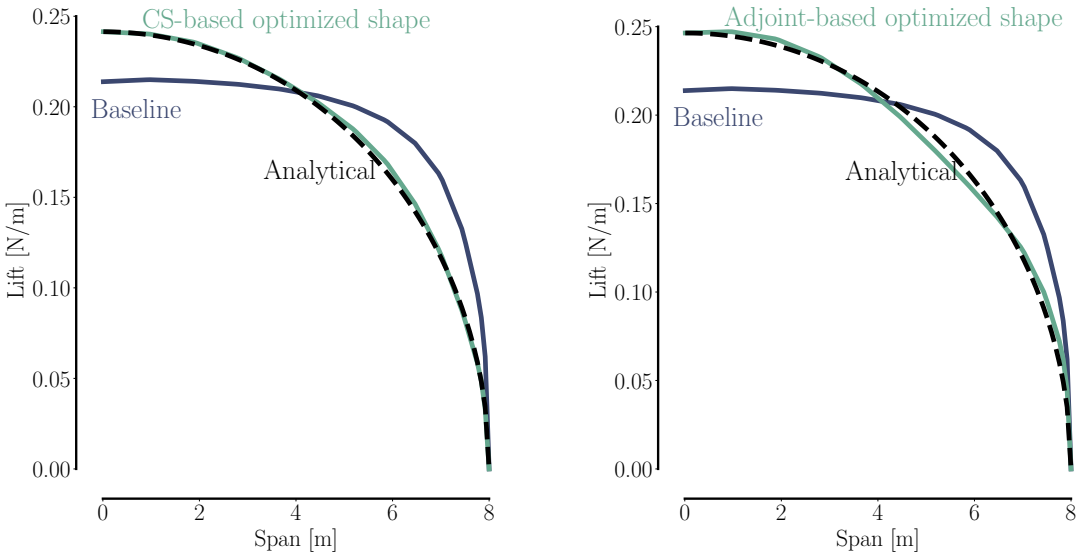



**Figure 11.9:** Lift distributions from the L2 mesh using QUICK stencils. To the left the baseline, CS-based result, and elliptic (Eq. 11.1) lift distribution can be seen. The right hand side shows a similar comparison but for the adjoint-based optimization result.

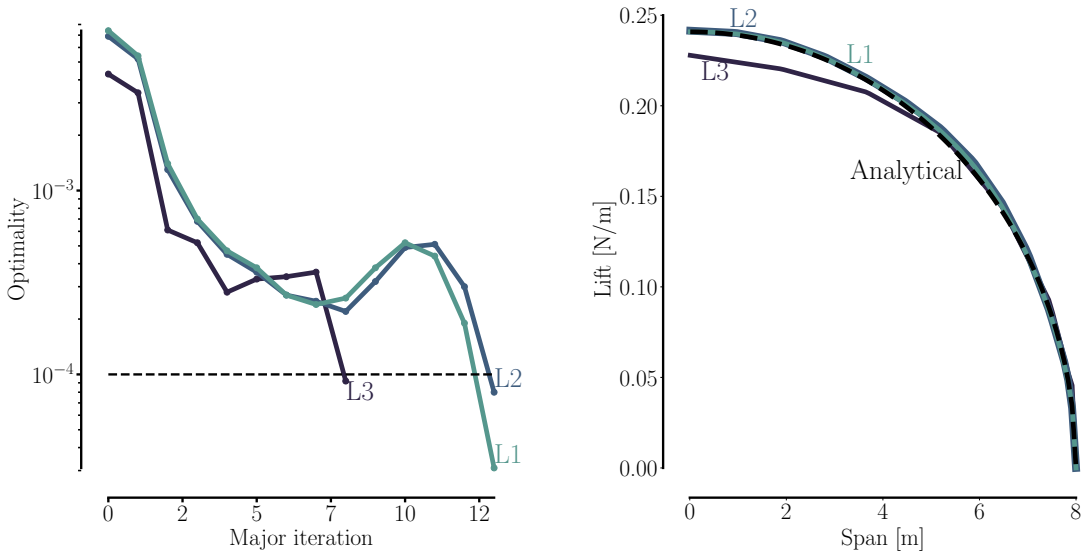



**Figure 11.10:** Optimality (left) and lift distributions (right) from CS-based optimizations on L3, L2, and L1. Mesh sizes can be found in Tab. 11.1 The lift distributions show that already the L2 mesh approximates the analytical distribution very well.

We have now presented the L3 and L2 CS-based and adjoint-based optimization results in great detail where we have focused on the QUICK optimizations to assess the situation with the worst possible gradient precision. Furthermore, we used CS-based results on finer mesh levels to check that the optimization result approaches the analytical lift distribution as the mesh is refined. Here, it was evident that the optimization results approached the analytical distribution as the mesh resolution increased. Thus, any deviation from the analytical distributions is likely due to the lacking gradient precision.

As promised, we now turn to focus on the UDS results where precision was much higher to show that vastly different gradient estimation methods (CS and adjoint) will lead to practically identical optimization trajectories taken by the optimizer due to their high gradient precision. The comparison of the L2 CS-based and adjoint-based optimizations using the UDS stencil is seen in Fig. 11.11. There is a significant amount of information to digest in Fig. 11.11, which shows complex flow solver iterations for the CS-based optimization (upper), optimality of both CS-based and adjoint-based optimization (middle) and iterations for both CFD solver and adjoint solver (lower).

Some of the things to keep in mind when contemplating Fig. 11.11 are the following:

- The three plots (upper, middle, and lower) are aligned with the middle plot's $x$-axis which shows steps taken by the optimizer.
- As a result the upper/lower $x$-axis have been scaled accordingly: Here one can consider the amount of flow solver iterations (black, left for both upper/lower plot) where the CS-based optimization reaches close to 200 thousand iterations and the adjoint-based optimization incurs only 12909 iterations. The flow solver is tasked by the optimizer to compute more or less the exact same flow problems but in the CS-based case the iteration counter also counts the gradient computations.
- The very first flow solver convergence (black, upper/lower) ends in a small extra spike in the upper/lower plots. This is the SNOPT optimizer's gradient check.
- Considering the middle plot it is stunning how closely the optimization trajectories mirror one another before meeting the threshold requirement (black dashed line).
- The gradient computations in the CS-based optimization always have 1-2 computations with a very smooth line before the final 3-4 computations become very noisy. This is a deficiency in our current complexification causing a slowdown. This is further explained in the next section.
- The adjoint solver convergence in the lower plot seems jumpy both for the drag (green, $f_x$) and for the lift (blue, $f_y$). This is the GMRES restart setting in PETSc, i.e., it is the subspace size which determines the number of directions used to orthogonalize against. Once PETSc restarts one will see a slight jump in the residuals before convergence continues.
- Just below the lower plot we list a three-layered iteration axis showing CFD iterations (black) as well as PETSc iterations for drag (green) and PETSc iterations for lift (blue).

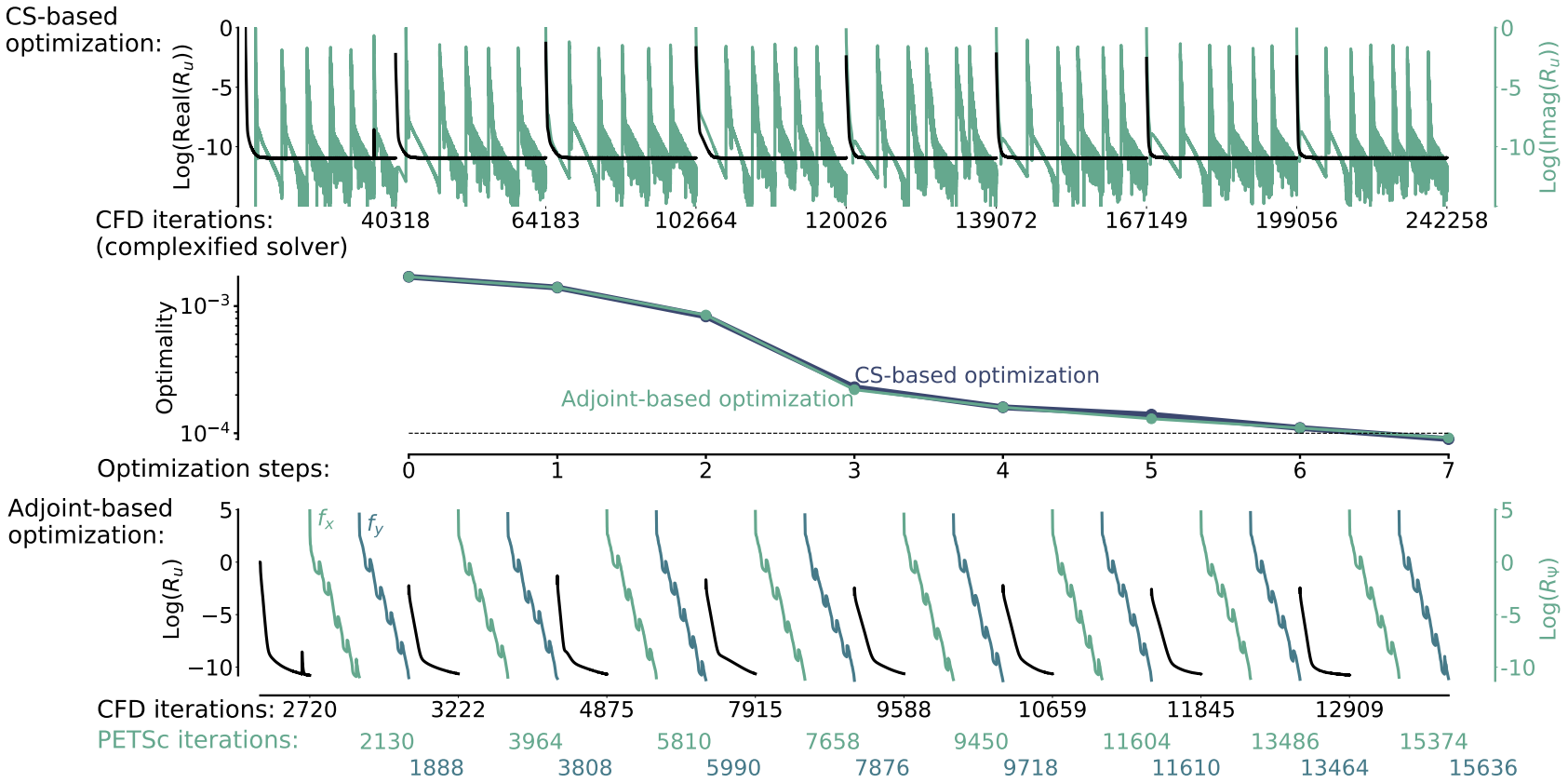



**Figure 11.11:** Overview of L2 drag minimizations of a 3-D wing using the UDS stencil. There are five complex steps for the CS-based optimization for each optimization step since the CS method scales with the number of design variables. Correspondingly, we only see two PETSc convergences for the adjoint-based optimization since this method scales with the number of functions of interest, of which we have two (lift and drag). Even in this page size plot it is hard to discern the two optimization optimality trajectories in the middle graph. This was achieved with the 5-7 significant digits gradient precision evident from Tab. 11.2. In the lower plot $f_x$ corresponds to drag and $f_y$ corresponds to lift.

To conclude the visual assessment in Fig. 11.11 of the UDS optimizations we end with a final note on the fundamental difference in the CS-based and the adjoint-based approach by reiterating that the complex-step method scales with the number of design variables (five) whereas the adjoint methods scales with the number of functions of interest (two: lift and drag). As a result one will in Fig. 11.11 always find five green complex-step convergences and two green/blue adjoint convergences in the upper/lower plot for each optimizer step in the middle plot. Once you think about it, it is indeed quite stunning that the adjoint method allows for a gradient computation with a computation cost which is independent of the number of design variables. This is possible, since the adjoint variables make up the *sensitivity* field of a given flow field. A comparison of a flow field (upper) and a sensitivity field (lower) computed during one of the L2 adjoint-based optimizations is seen in Fig. 11.12.

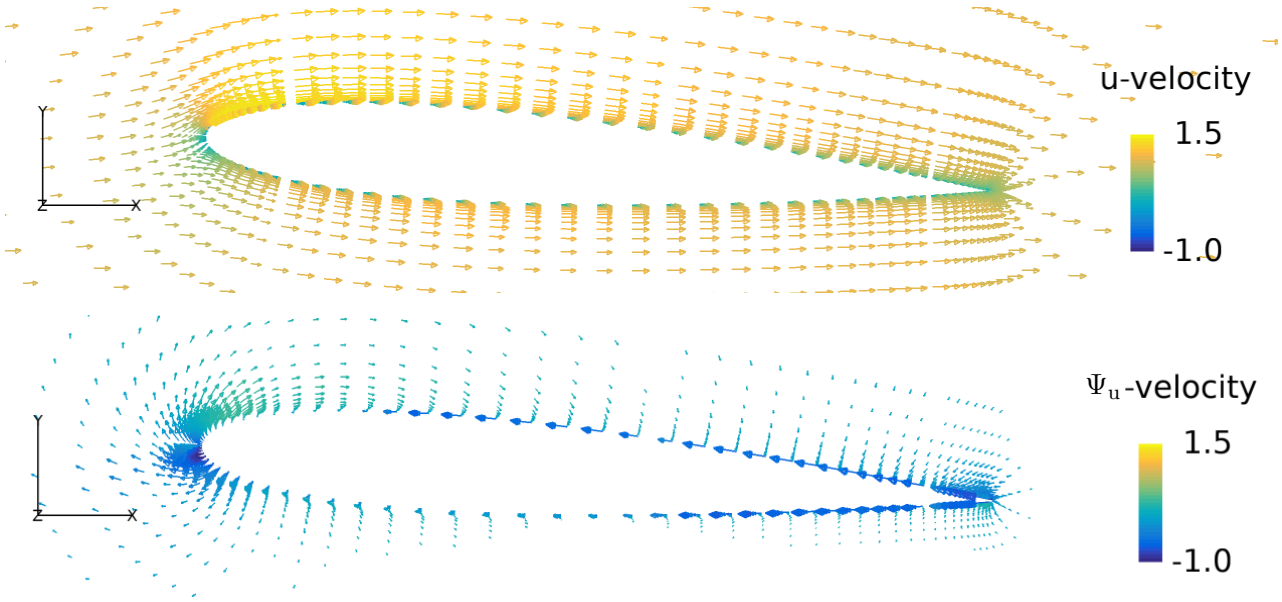



**Figure 11.12:** Comparison of the flow field (upper) and the adjoint field (lower). The adjoint variables make up a type flow field but of completely different orientation and magnitude. It is the *sensitivity* field. Not surprisingly, it has its largest magnitudes at the surface. The wing slice is from $z = 4$[m].

There are many noteworthy observations to be made in Fig. 11.12. Clearly, the intuition about the sensitivity field is not a given. We see completely different magnitudes in the sensitivity field when comparing to the flow field just above it. Indeed, the largest magnitudes in the sensitivity field are at the surface. This is very different from a typical flow field where we have no-slip boundaries and always expect to see higher velocities far removed from surface boundaries.

Also the direction of the two fields are disparate. For the sensitivity field close to the surface the flow direction even seems to be reversed. Here, we note that the sensitivities

on the suction side (upper surface) are stronger than the sensitivities on the pressure side (lower surface) since we have a 4 degree angle of attack.

Although sensitivity analysis, i.e., analysis related to the sensitivity field, is underprioritized in many aerodynamic shape optimization efforts it is crucial to arrive at a successful optimization. As an example hereof we mention that the magnitudes in the sensitivity field should be used when setting up the FFD control points. This would ensure that areas with a high sensitivity (e.g., wind turbine blade tips) would be parameterized to a very fine degree whereas other areas with only minor influence (e.g., the wind turbine hub) would be parameterized using fewer FFD control points.

With time, one can indeed nurture the intuition of theses sensitivity fields As is hopefully evident from the above discussion this will inevitably lead to better suited parameterizations and in turn more useful final results.

#### 11.1.3.1 Benchmarking

In this section we try to quantify the performance of the framework. To this end, we will use the UDS optimizations since the optimizer here takes the exact same steps, which allows for the most fair one-to-one comparison. An overview of timing and memory consumption for the UDS drag minimization runs can be seen in Tab. 11.3.

The complexified flow solver seems remarkably slower than the standard flow solver. The speed difference is most pronounced in the L2 results where the EllipSys timing in Tab. 11.3 is much worse for the CS optimization ($[hh : mm : ss] = [02 : 16 : 30]$) than for the adjoint optimization where EllipSys runs in the normal configuration ($[00 : 27 : 23]$). We remind the reader, that for these simulations the optimizer takes essentially the exact same steps. Thus, the flow solver is tasked with converging more or less the exact same problems – albeit in one case it must compute complex values. A reasonable result would be a complexified flow solver which is twice as slow, but the above given timings suggest that the complexified flow solver is about five times slower. This can be further improved. Indeed, the complexified flow solver is usually a factor 2 slower than the normal flow solver, but when the real-valued residuals reach machine precision in EllipSys the maximum number of allowed pressure iterations for each segregated step is reached. This is exactly what happens when EllipSys is tasked with computing gradients for more than one design variable in a row using the CS method. In fact, one can observe this phenomenon in the upper graph in Fig. 11.11: After each step the optimizer takes we first observe a bump in the real-valued residuals. This is the optimizer prompting the flow solver to compute new flow state values. Then comes a bump in the imaginary part of the residuals. One bump for each design variable. Noticeably, the first imaginary residual convergence is just as smooth and thin in line thickness as the real-valued residual convergence but the convergence behavior for the remaining four design variables are much more noisy. For these iterations the performance of EllipSys is reduced due to the mentioned limit on the maximum amount of pressure iterations which explains the slowdown for the complexified solver. Improving our flow solver complexification to circumvent this issue is a high priority.

The runtime ratio is hard to deduce from Tab. 11.3. Indeed, both the flow solver

**Table 11.3:** Table showing timing and memory consumption for the four UDS drag minimizations of the 3-D wing.

| **Computation:** | **Wall clock:** [hhh:mm:ss] | **Percentage:** % |
|---|---|---|
| **CS-based optimization on L3** (7.945 GB[c]): | | |
| Flow solve | [ :44:48] | 15.8 |
| Gradient | [ 3:51:32] | 81.6 |
| Etc. | [ :07:29] | 2.6 |
| **Total:** | [ 4:43:50] | 100 |
| **Adjoint-based optimization on L3** (9.094 GB[c]): | | |
| Flow solve | [ :17:17] | 5.6 |
| $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$ | [ 1:51:13] | 36.2 |
| $\partial\mathbf{J}/\partial\tilde{\mathbf{w}}$ | [ :13:27] | 4.4 |
| $\partial\mathbf{R}/\partial\mathbf{x}$ | [ 2:25:21] | 47.3 |
| $\partial\mathbf{J}/\partial\mathbf{x}$ | [ :13:43] | 4.5 |
| Gradient[a] | [ :03:26] | 1.1 |
| Etc. | [ :03:13] | 1.0 |
| **Total:** | [ 5:07:30] | 100 |

| **Computation:** | **Wall clock:** [hhh:mm:ss] | **Percentage:** % |
|---|---|---|
| **CS-based optimization on L2** (8.148 GB[c]): | | |
| Flow solve | [ 2:16:30] | 20.5 |
| Gradient | [ 8:32:16] | 77.1 |
| Etc.[b] | [ :16:03] | 2.4 |
| **Total:** | [ 11:04:49] | 100 |
| **Adjoint-based optimization on L2** (59.450 GB[c]): | | |
| Flow solve | [ :27:23] | 0.3 |
| $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$ | [ 68:57:00] | 46.7 |
| $\partial\mathbf{J}/\partial\tilde{\mathbf{w}}$ | [ 3:24:12] | 2.3 |
| $\partial\mathbf{R}/\partial\mathbf{x}$ | [ 67:39:13] | 45.8 |
| $\partial\mathbf{J}/\partial\mathbf{x}$ | [ 4:48:53] | 3.3 |
| Gradient[a] | [ 1:50:16] | 1.2 |
| Etc.[b] | [ :28:57] | 0.3 |
| **Total:** | [147:37:43] | 100 |

[a] PETSc solution procedure to obtain the adjoint variables. The given time is the total time for both drag- and lift gradients.

[b] The Etc. row are very minor computational efforts such as FFD-related procedures, optimization algorithm decisions, and so on.

[c] We have yet to carry out extensive memory tests and memory improvements for the framework given that i) it has not been an issue yet, and ii) the coloring acceleration is by far the bigger need for us to carry out shape optimizations for meshes on a few hundred thousand cells. However, we did measure the peak memory consumption for the four optimizations in the present table. We do not put much stock in the L3 memory results since these meshes are simply too small for any useful comparison. However, the L2 memory results are more relevant. Here, the memory usage ratio between the optimizations ($\approx 6.5$) is rather reasonable. To obtain an estimate of our adjoint-flow solver memory ratio one could half the CS-based memory consumption which gives an estimate of 13 for the ratio between the adjoint solver and the flow solver. This estimate compares favorably to the lowest ratio reported for DAFoam which is 23.7 [55, Tab. 2] albeit obtained on a slightly bigger mesh ($83 \cdot 10^3$ cells vs. $103 \cdot 10^3$ cells).

and the adjoint solver use previous solutions as starting guesses for ensuing solution procedures, and it may be difficult to determine which solver benefits most from such an advantage. What one should do to properly determine the adjoint-flow ratio, i.e., runtime ratio between the adjoint solver and the flow solver, is to time them against each other starting from a zero flow- and adjoint field. These timings are given in Tab. 11.4.

| | EllipSys3D [hh:mm:ss] | eDA*adf* [hh:mm:ss] | Runtime ratio (Adjoint-flow ratio) |
|---|---|---|---|
| $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$ | - | [ 9:01:54] | - |
| Solution | [ :06:12] | [ :11:11] | 1.8 |
| Total | [ :06:12] | [ 9:13:05] | 89.2 |

**Table 11.4:** Runtime ratio between adjoint solver and flow solver calculated on the L2 mesh (see Tab. 11.1). For these timings we have not used any acceleration in the Jacobian computation. Thus, every single state in $\tilde{\mathbf{w}}$ is perturbed one at a time. This naive representation results in 750528 colors and a runtime of 89.2. This number can be reduced to below 1000 once our coloring scheme is fully implemented [55]. Using 1000 colors would result in a runtime of 1.9.

One could argue that the $\partial\mathbf{R}/\partial\mathbf{x}$ partial derivative matrix should be taken into account as well when computing runtime ratios. One complicating factor here is, that the size of $\partial\mathbf{R}/\partial\mathbf{x}$ drastically depends on whether we use shape design variables or aerodynamic BC design variables. However, as seen in Tab. 11.3 the computation time for $\partial\mathbf{R}/\partial\mathbf{x}$ and $\partial\mathbf{R}/\partial\tilde{\mathbf{w}}$ are somewhat similar, so one can obtain such a runtime ratio using a factor of two.

A runtime ratio around one is one of the hallmarks of mature adjoint solver implementations. As an example we mention ADflow's runtime ratio of 0.8 [55, Tab. 2], also achieved on a 3-D wing. We mention this example since the mesh sizes are comparable: Their wing mesh has 102912 cells whereas our L2 wing mesh has 82944 cells. Instead of focusing on the ADflow numbers it is perhaps more relevant to mention DAFoam's runtime ratios ranging from 4.7 to 6.2 on the same mesh [55, Tab. 2] given that the DAFoam adjoint solver is one of the few adjoint solvers for SIMPLE algorithms mentioned in Tab. 4.2 which highly resembles the implementation strategy we chose. Thus, a runtime ratio of about 5 should certainly be realistic for our implementation once we have finalized our coloring implementation. Currently, we have only implemented an inner-block analytical coloring which only reduces the number of colors with a factor of about 2 (no timings presented in the present work). Thus, we will have to complement this analytical coloring scheme with a coloring scheme that handles the block-block interfaces. A quick back-of-the-envelope calculation[3] using Tab. 11.4 shows that we for

[3]Aiming at a runtime ratio of 5 gives us: $5 \cdot 372$ [sec] $= 1860$ [sec] for the entire adjoint solver ($[mm:ss] = [06:12] = 372$ [sec], see Tab. 11.4). Given that all 750528 colors took 32514 seconds we estimate one color to take $\approx 0.043$ seconds. Deducting the adjoint solution time we find the maximum number of colors we can handle for a runtime of 5 to be: $(1860\text{ [sec]} - 671\text{ [sec]})/(0.043\text{ [sec]}) \approx 27436$ colors

the L2 mesh would have to reduce the naive number of colors (750528) to around 27436 to obtain a runtime ratio of 5. This is certainly possible. Indeed, we aim at a coloring scheme which results in a fixed number of colors around 500-600. We believe this to be realistic given that ADflow can muster 162 colors on the aforementioned mesh whereas DAFoam require 954 colors. We cannot hope to compete with the ADflow coloring scheme since it is a compressible flow solver where there is no need to introduce the three fluxes as independent variables. However, we do on the other hand expect to compare favorably to DAFoam's coloring scheme since DAFoam uses a heuristic coloring solver to handle unstructured meshes [38]. Finally, it is realistic to aim at a fixed number regardless of mesh size given that He et al. shows that even their heuristic coloring solver is almost independent of mesh size [38, Fig. 7]. Even in the worst case scenario where we mirror the coloring performance of the unstructured DAFoam and obtain a coloring number around 1000 this would still result in a favorable runtime ratio comparison when comparing eDA*adf* and DAFoam. The resulting runtime ratio when using 1000 colors in eDA*adf* would be 1.9 whereas the runtime ratios for DAFoam as mentioned range from 4.7 to 6.2.

In the same vein, we can use the timings back in Tab. 11.3 to estimate when the two methods (CS and adjoint) would be equally effective, i.e., we can estimate how many design variables the complex-step method would be able to handle without incurring a computation time that exceeds that of the adjoint-based method. The answer is a mere 86 design variables which is quite a stunning result. This means, that even though we have not used any coloring acceleration we should still have to prefer the adjoint method on L2 for optimizations with more than 86 design variables. Remembering the overview Tab. 3.1 from Chapter 3 we note that the number of design variables for rotor studies easily reach 100 design variables. Then again, we cannot hope to carry out a rotor study for a mesh less than 200 thousand cells (L2 in [71] had 221 thousand cells) and even such a small mesh would be intractable without coloring acceleration.

A final thing to note when discussing these benchmarks is, that we have chosen very unfavorable convergence limits for the adjoint solver to obtain the worst possible outcome. We set the relative residual tolerance to $10^{-16}$ for our adjoint solver which can actually be observed in Fig. 11.11. This means it is tasked with reducing the relative residual 16 orders of magnitude. It is not straight forward to relate this setting to the absolute residual threshold of $10^{-10}$ for the summed residuals we use in EllipSys. We are therefore in the process of unifying the residual formats in the two solvers. It is, however, straightforward to relate the $10^{-16}$ residual setting in our adjoint solver to the settings for ADflow and DAFoam we cited above. Here, the settings are "$10^{-10}$ and $10^{-8}$ for the flow and adjoint solutions, respectively" [55, p. 29]. That is; the adjoint solver is in their setup only tasked with reducing the residuals 8 orders of magnitude. Thus, the runtimes reported by Kenway et al. [55] would most likely worsen a bit if the residual settings were more strict.

### 11.1.4 Future work

This section sums up the first test case which we presented in Sec. 11.1. The test was aimed at validating our developed framework, and the presentation included a mesh convergence study (Sec. 11.1.1), a description of the design optimization problem (Sec. 11.1.2), and a results section (Sec. 11.1.3) including a benchmark (Sec. 11.1.3.1) to quantify the performance of our adjoint solver.

Overall, we have showed that the framework is functional and that the adjoint can be leveraged to carry out shape optimization studies. However, we have also identified numerous areas of future work. The most important are listed below:

**Coloring:** It became apparent in the benchmark results (Tab. 11.3) that acceleration of our Jacobian computation is perhaps the most dire need for the framework. This work is underway and is our top priority.

**Accuracy**: The sensitivity test (Tab. 11.2) clearly showed that the precision should be further improved. Particularly the third-order stencil (QUICK) must be improved. Once the coloring is completed, we will be able to run debug tests much faster thus reducing debugging time.

**Solver**: Here we aim at the way we solve the adjoint system, i.e., the solution procedure, which currently is handled in PETSc. While we have not presented much data to quantify these issues we find that the current MPI implementation leveraging PETSc is less than ideal. On one hand we have proof that these kind of setups can be used to carry out shape optimizations on meshes larger than 10 million cells [38]. On the other hand we find that it can be a tedious process tuning the settings[4] to make sure PETSc exhibits a robust convergence. We admit, that this could be due to lack in experience. Likewise, we admit that we have yet to encounter an adjoint problem which we could not solve with our PETSc implementation. However, we have not tested meshes above 100 thousand cells. All told, this is a topic we focus on more than we expected. Of the options we consider we mention: Aligning with the solution procedure used by Dilgen et al. [23] (still PETSc), aligning with the solution procedure used by Roth and Ulbrich [104], and testing fixed-point methods such as that by Müller, Mykhaskiv, and Hückelheim [81].

**Memory**: The memory considerations are linked to the above topic. Most likely, we will pursue the absolute limit of what our current solver architecture can offer, which should be successful shape optimizations using well above 10 million cell meshes. Again, this has been shown by He et al. [38] to be feasible and with such a setup one could easily contribute to the absolute forefront of rotor studies. However, it is indisputable that the approach used by He et al. [38] which we also implemented is much more memory intensive than, e.g., fixed-point methods. Therefore, thinking long term it might be necessary to have a more memory efficient architecture. The fixed-point approach would certainly be very attractive in this respect, but this

[4]PETSc settings are, e.g., preconditioner type, fill in for the preconditioner, amount of overlap for the additive Schwartz method, Krylov subspace size before restart is triggered in GMRES and so on

would only be attractive for us in the event that we would not run into convergence issues. In that case, we would rather use more memory to obtain a robust solution procedure. Another approach would be to align with ADflow. As shown in [55] they have successfully implemented a reverse algorithmically differentiated adjoint solver using the transpose-matrix-vector matrix-free operations which are much more memory effective. However, for now this topic is not resulting in any implementation changes.

## 11.2 Multipoint shape optimization of a 10 MW offshore wind turbine

To briefly revisit the motivation mentioned in the thesis summary we reiterate that one of the benefits from optimizing with free-form deformation and 3-D RANS models is a concurrent shaping of planform and cross-sectional shape to generate novel rotor shapes. This is exactly what we achieved in this multipoint rotor study and once our own framework is fully matured we aim at carrying out further rotor studies to advance the absolute forefront of wind turbine aerodynamics.

The following is a result of a collaboration with the MDOLab at University of Michigan, where we carried out developmental work on their framework and subsequently used it for design studies. The framework being maintained at the MDOLab is one of the world leaders within the field with a proven published track record. Said framework, called MACH (MDO for aircraft configurations with high-fidelity) [53], provides a basis for aerostructural design optimization and the compressible flow solver, ADflow[5], is now open source which allows for an inspection of its adjoint solver source code.

Our collaboration with the MDOLab have led to several design studies, many of which can be found in a recent publication [71]. These design studies include a rotor planform optimization where the optimization problem was solved both with CFD and BEM models which allowed us to carry out a rare one-to-one comparison across fidelities. We will not present these results for the sake of brevity. We therefore refer readers interested in our other design studies to appendix (Sec. A) for the paper itself.

We will, however, include one of the design studies, where we choose the final multipoint aerodynamic shape optimization of a full rotor configuration. Our work related to this study included:

- Conducting a comparative analysis between ADflow (tailored for aerospace research) and EllipSys3D, which is the flow solver at DTU Wind Energy, to establish how well the physics were captured when applying ADflow to a wind turbine rotor.
- To (re)establish forward algorithmically differentiated gradients for rotating problems.
- To implement reverse algorithmically differentiated gradients in order to use the transpose-matrix-vector matrix-free operations that due to the memory reduc-

[5] `https://github.com/mdolab/adflow`,(last access: 11 July 2019)

tion allow for very large CFD meshes ($\mathcal{O}(10^7)$ cells) to be used in CFD-based optimizations.

The rotor design study is aligned with the IEA Wind Task 37 rotor aerodynamic design case study as explained in [71]. The objective in the design optimization problem is to maximize the annual energy production (AEP) while constraining thrust and bending moment to a maximum increase of 14% and 11%, respectively. We had to modify the design optimization problem slightly since the original IEA design study was prepared for BEM-based codes. As a result, the design problem:

$$\begin{aligned} \text{maximize} \quad & \text{AEP} \\ \text{with respect to} \quad & \text{twist} \\ & \text{chord} \\ & \text{shape} \\ \text{subject to} \quad & T \leq 1.14 \cdot T_{\text{init}} \\ & M_{\text{bend}} \leq 1.11 \cdot M_{\text{bend}_{\text{init}}} \end{aligned} \tag{11.4}$$

includes the full shape design variables instead of the relative thickness used in BEM. In Eq. (11.4) AEP is the annual energy production, and $T_{\text{init}}$ and $M_{\text{bend}_{\text{init}}}$ are initial values for thrust and flapwise bending moment, respectively.

Another change compared to the original design problem is, that we evaluate the rotor performance at three wind speeds, 5, 8, and 11 m/s, instead of generating a full power curve. The multipoint objective function is then computed in a weighted fashion using a Weibull distribution to take the wind speed distribution into account.

To inspect the baseline rotor design and the FFD setup including the geometrical constraints we had to impose to avoid negative cell volumes we reproduce Figure 6 from the paper, see Figure 11.13:

There are three FFD boxes in the left hand side of Fig. 11.13: one box for each blade. The three boxes are linked to ensure that the three blades undergo the exact same deformation. Each FFD box has a total of $10 \times 2 \times 9$ FFD control points distributed over nine spanwise sections where each section has ten control points from leading edge to trailing edge. The final factor of two can be explained by the fact that each section has an upper and a lower row of control points A spanwise section can be seen in the lower right hand side of Fig. 11.13. An obvious difference compared to the FFD setup we presented for the 3-D wing in Fig. 11.4 is that we now only deform surface mesh points with FFD whereas we deformed both surface and volume mesh points in the 3-D wing setup.

Returning to Fig. 11.13 we note that the two FFD box sections closest to the root are closely set and kept fixed throughout the optimization. This ensures a $C^1$-continuous transition from the deforming surface mesh at the inner part of the blade to the non-deforming surface mesh at the root of the blade. This leaves seven of the nine FFD box sections free to move. We therefore have seven twist design variables. Each twist design variable can twist one of the FFD box sections about a point located at 35% from the

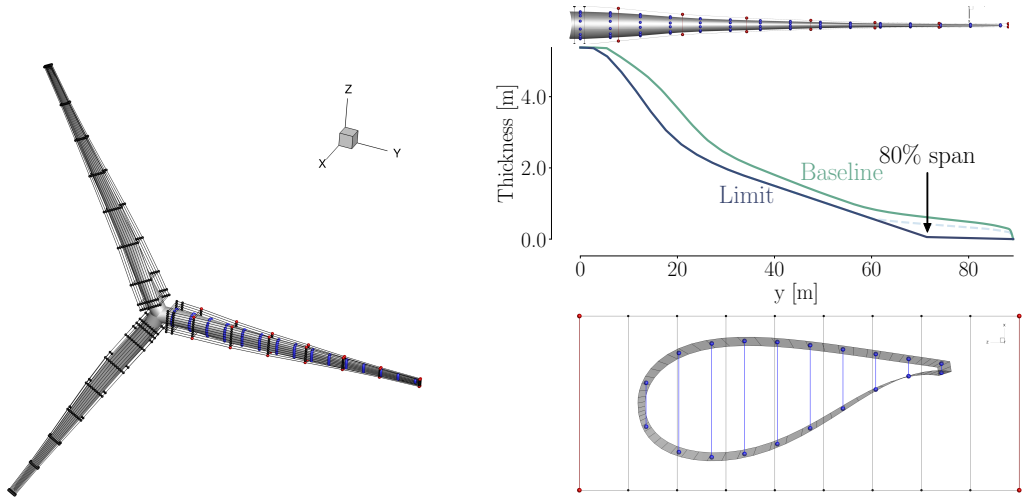



**Figure 11.13:** This figure has been published elsewhere [71, Fig. 6]. Overview of baseline geometry and FFD boxes (left). Each FFD box has nine spanwise sections. Each blade (upper right) has 15 thickness constraints (blue) and seven LE/TE constraints (red). Thickness distributions (mid-right) are for the baseline thickness (green) and minimum allowed thickness (purple). Profile section (bottom right) at 36 m span shows the shape control points (20), the thickness constraints (ten blue segments) and LE/TE constraints (two red segments) .

leading edge. Likewise we have seven chord design variables. Each chord design variables scales one of the FFD box sections both in the chord and thickness direction to maintain the relative thickness of the airfoil. The only way to change the relative thickness for the optimizer is therefore to make use of the shape design variables of which there are twenty per spanwise section. Each shape design variable moves one FFD control point in the direction perpendicular to the chord. This amounts to a total of 154 design variables as seen in Tab. 11.5.

**Multipoint rotor study**:

| design variable | chord | twist | shape | total |
|---|---|---|---|---|
| Amount | 7 | 7 | 140 | 154 |

**Table 11.5:** Overview of design variables in the multipoint rotor study.

We now turn to the geometrical constraints. There are 15 thickness constraint sections shown in blue in the upper right hand side of Fig. 11.13. Just below we compare the imposed thickness constraint limit (blue) to the thickness of the initial baseline rotor geometry (green). The original design problem only had a thickness constraint on the inner most 80% of the span but we had to impose the thickness constraint also on the final 20% of the blade to avoid negative cell volumes. This added thickness constraint

is marked with a dashed blue line. Finally, we mention the leading edge and trailing edge constraints which are shown in red in the lower right hand side of Fig. 11.13. These constraints ensure that the upper and a lower FFD control points move in opposite directions. This is done in order to prevent the optimizer from introducing a skewing twist deformation through the shape design variables. Thus, a twisting of the blade should only be introduced through our twist variables.

Before proceeding to the design studies we did carry out an extensive comparison between the compressible solver, ADflow, used in the MACH framework and the incompressible solver, EllipSys3D, we use in our design framework. This includes a mesh convergence study with meshes up to 48 M cells. We omit these sections for the sake of brevity and refer readers to [71, Sec. 4] and [71, Appendix A] for these details. However, we do bring a summary in Tab. 11.6. In short, the mesh convergence study showed that the L0 mesh with more than 14 million cells was a reasonable compromise between speed and accuracy (less than 10% error compared to continuum values). Therefore, the L0 is the finest mesh we use in optimization. For the multipoint rotor study we have used three consecutive mesh levels where the solution from a coarser mesh is used as an initial guess on a finer mesh. This reduces the amount of optimization steps we have to take on the fine mesh. Readers consulting Tab. 3.1 where we gave an overview of all high-fidelity adjoint-based studies found within wind energy will notice, that such a high mesh resolution certainly pertains to the upper echelons of the group. In the same chapter we also brought a table focusing solely on the full rotor studies (Tab. 3.2). We refer readers to said table to better compare the various studies.

**Meshes used in the multipoint rotor study**:

| Mesh | Cells (million) | ADflow error (%) | EllipSys error (%) |
|---|---|---|---|
| L2 | 0.221 | 134.5 | 3.2 |
| L1 | 1.769 | 38.3 | 1.7 |
| L0 | 14.155 | 9.6 | 0.5 |

**Table 11.6:** A summary of the mesh convergence study [71, Sec. 4] preceding the multipoint rotor study. The errors given in the present table are highest occurring errors in [71, Tab. 4]. Note, that ADflow used a second-order stencil whereas EllipSys used a third-order stencil which can explain some of the difference in accuracy.

Turning to the results, we start by analyzing the output from the SNOPT optimizer shown in Fig. 11.14.

There is a some familiarity with the shown optimality and merit functions compared to the results from the 3-D wing problem – albeit we now work with a maximization of AEP resulting in the merit functions to trend upwards. As is evident from the optimality plot the complexity of the design optimization problem makes it very hard to converge the problem tightly and meet the requested threshold (black, dashed line). More simple problems from the paper [71, Fig. 8 and 11] presented no difficulty to converge below the

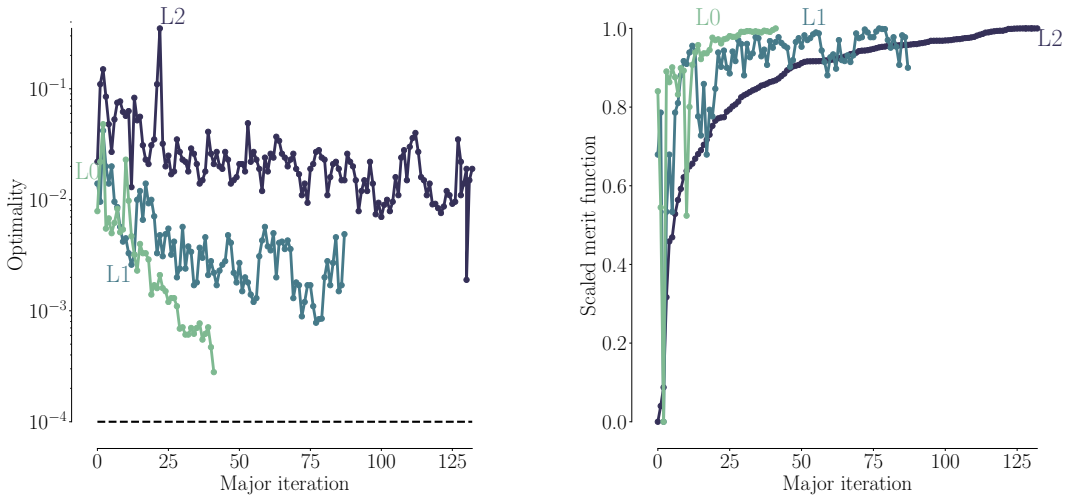



**Figure 11.14:** This figure has been published elsewhere [71, Fig. 17]. History of convergence (left) and scaled merit functions (right) for the multipoint shape optimizations. Mesh sizes for L0, L1, and L2 can be found in Tab. 11.6.

requested threshold but for the multipoint study we only obtain close to the threshold on the finest mesh level. However, looking at the merit functions in Fig. 11.14 we see that they all tend to plateau at the end.

After having quantified how well the optimization problem has been solved, we turn to the resulting design seen in Fig. 11.15. This figure actually shows the single-point shape optimization results obtained at 8 [m/s]. We start by addressing these single-point results to assess the benefits of shifting to multipoint optimization which allegedly should produce more robust results. It is relevant to try to quantify the benefits from transitioning to multipoint optimization since most available rotor studies in the literature are single-point studies. Indeed, only one other work in Tab. 3.1 uses multipoint optimizations.

The most noticeable change in shape in the upper part of Fig. 11.15 is the increased chord towards the root of the blade. This result agrees with the planform optimization [71, Fig. 10] from the paper. Furthermore, it is an expected design trend since the initial rotor geometry intentionally was made worse to allow the optimizer room for improvements. Indeed, one of the changes made to the initial rotor design was a decrease in chord [71, Fig. 2] near the root of the blade. The lower part of Fig. 11.15 is a $C_p$ comparison. Here, the suction peak clearly is reduced along the blade. The airfoil comparison just below clearly shows that the change is owed to a thickness reduction and a slight increase in camber. Of all changes, the thickness reduction is most easy to predict. Given, that we carry out purely aerodynamic shape optimizations and use thickness constraints as a surrogate for structural feasibility the optimizer simply chooses the thinnest possible airfoils that satisfy our constraints. Finally, we mention the sharper leading edge shapes. These shapes might work well for this particular wind speed (8 [m/s]) but they would perform poorly for other wind speeds.

As is evident from the above, we certainly have to make amends for the pointy leading

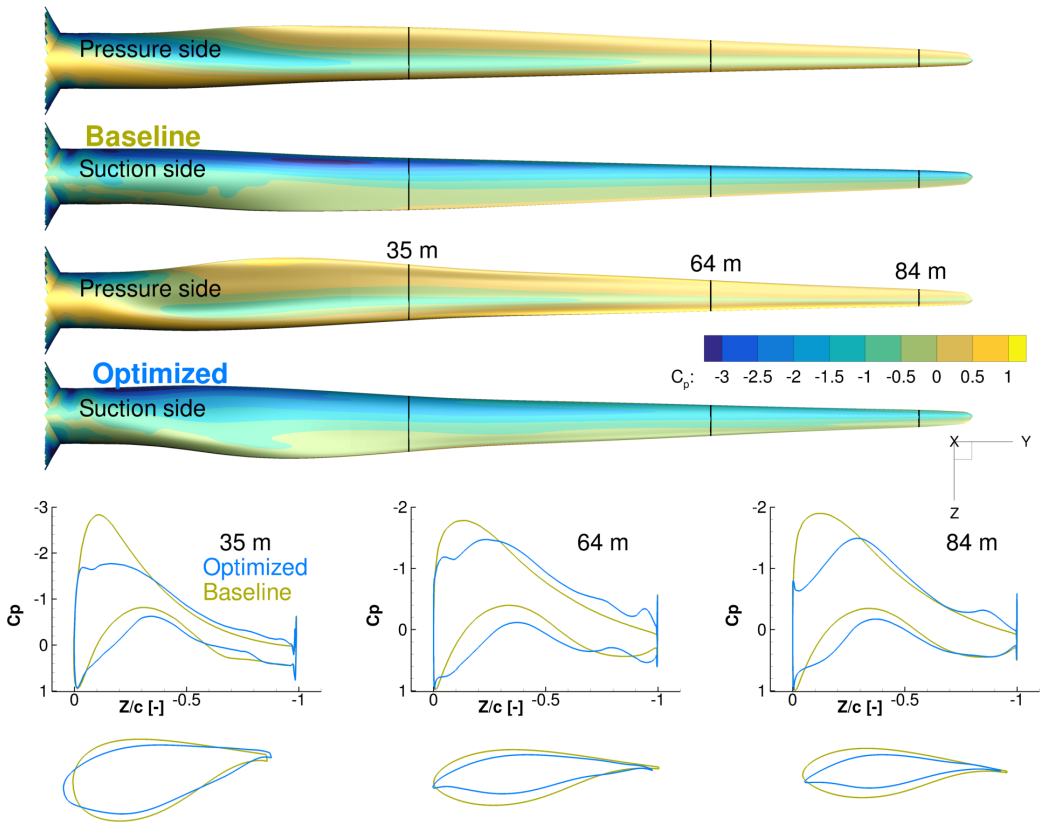



**Figure 11.15:** This figure has been published elsewhere [71, Fig. 15]. Comparison of $C_p$ distributions for the baseline and optimized result from the single-point shape optimization. There is an increase in trailing edge camber, especially at the root, as well as a less pronounced suction peak.

edge shape of the freely formed airfoils. To this end we explored two avenues: First, we transition to a multipoint optimization where a range of wind speeds (5, 8, and 11 [m/s]) as mentioned is taken into account. The comparison between single-point and multipoint results can be seen in Fig. 11.16.


**Figure 11.16:** This figure has been published elsewhere [71, Fig. 18] Comparison of airfoil profiles obtained from single-point and multipoint optimizations. The profiles are taken from 35, 64, and 84 m spanwise positions .

While it certainly did help to take more wind speeds into account it is evident that we still have work to do in order to arrive at an industrially relevant design. We therefore

chose to implement additional geometrical leading edge thickness constraints. This finally resolved the issues of obtaining too pointy leading edge shapes as seen in Fig. 11.17.


**Figure 11.17:** This figure has been published elsewhere [71, Fig. 19] Comparison of airfoil profiles obtained from multipoint optimizations with and without leading edge geometric constraint .

The final multipoint result including the extra geometric leading edge constraints improve the wind turbine performance with an astonishing 23.76% which is of course *not* realistic. As mentioned above, the IEA Task 37 case had a starting point which was intentionally made worse to make room for improvement in the ensuing optimization which explains the large improvement.

A final performance comparison at 8 [m/s] of i) baseline, ii) single-point, and iii) multipoint is given in Fig. 11.18.

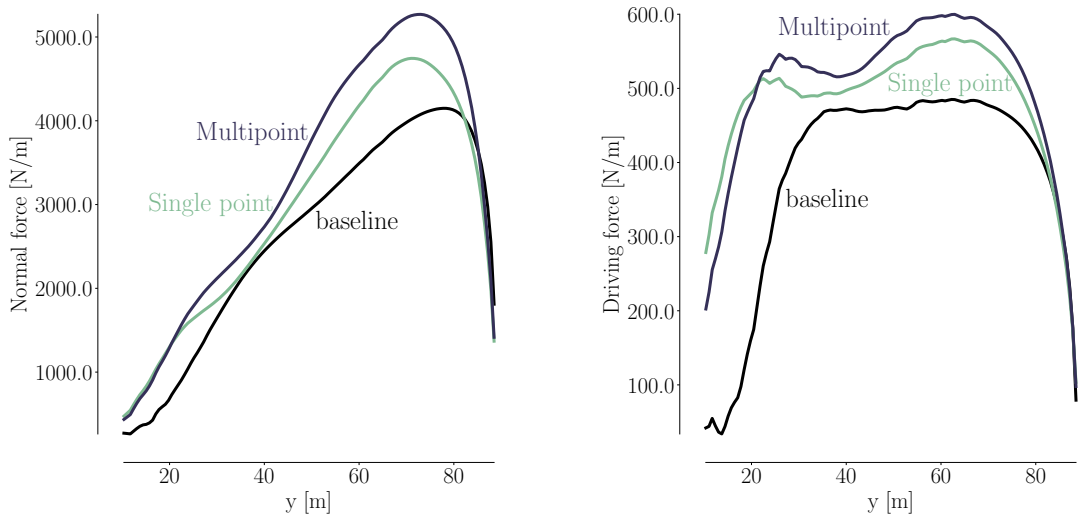



**Figure 11.18:** This figure has been published elsewhere [71, Fig. 20] Comparison of normal (left) and driving (right) forces for baseline and optimized designs at 8 [m/s]. The shape optimization increases the normal force, and the peak has also moved further inboard. The driving force is increased considerably both at the root and close to the tip region.

The comparison of the spanwise forces in Fig. 11.18 shows the 14% and 11% allowed increase in thrust and bending moment has resulted in a general increase in the spanwise distributions. The thrust constraint is instrumental in lowering the overall thrust which in turn ensures a structural feasible design. The bending moment constraint moves the

normal force peak farther inboard to limit the high loads at the tip region. Here, it should be noted that the comparison favors the multipoint result in a biased manner since the constraints in the single-point optimization is determined based on the 8 [m/s] case whereas the multipoint optimization uses the 12 [m/s] which is where the thrust peak occurs [71, Fig. A1]. Given that the thrust is the normal force integrated along all three blades it is to be expected that a relaxed constraint on thrust results in changes in the normal force for the optimized shape. This explains why the more robust multipoint design can outperform the single-point design at 8 [m/s]. Turning to the driving force we see a general increase in tangential loading where especially the root region seems improved. This is in part due to the mentioned increase in chord in this region but also that ability to freely form the airfoils to take account of the complex flow field in this region factors in.

# CHAPTER 12

# Conclusion

This thesis presents a methodology to enable gradient-based high-fidelity shape optimization using CFD. For the long term, the present study aims at developing a framework which is relevant both in the wind energy research community and in the industry. The thesis revolves around the in-house EllipSys3D flow solver which is used by the industry for analysis of wind turbine airfoil sections and full rotors. However, to expand the area of application the design and optimization capability is desired. In this process new numerical tools were developed and existing tools were further enhanced as presented throughout the three thesis parts.

Breaking new ground is a tiring effort. Two components, i.e., a deformation module and a discrete adjoint solver, have been developed from scratch. Then a numerical optimization framework was assembled and validated on a 3-D wing test case. Now that the underlying framework with all its components is established it should be more manageable to advance the framework further.

Below, results from the three parts of the thesis are summarized to show that the six goals set at the very beginning (Summary) have all been met.

Part I starts with an overall introduction to the adjoint approach (Chapter 2) followed by a detailed literature study (Chapter 3) where focus was laid on high-fidelity adjoint-based efforts within wind energy research (Sec. 3.1.2.2).

The adjoint method can at first come across as counterintuitive which may limit the amount of practitioners. This is a pity. In part because of its elegance and underlying mathematical ingenuity, but most of all because it is an extremely powerful way to carry out gradient-based optimization. Therefore, the intuitive adjoint introduction in Chapter 2 should certainly be seen as a contribution to further expand the high-fidelity shape optimization field within wind energy.

The second chapter in Part I (i.e., Chapter 3) contains a comprehensive literature study on the adjoint method. The first declared goal at the onset of this project was to carry out such a study with focus on the wind energy research community. The literature study includes a historical review (Sec. 3.1.1) which helped identify developmental trends throughout the years. Not surprisingly, these trends make up most of the topics we seek to address in future research (Sec. 12.1). The review focused specifically on wind energy research (Sec. 3.1.2) and documents that it is a young but promising research field. Here, six rotor studies were found (Tab. 3.1) where only one study was transient. Studying the six efforts, it is evident that much still has to be done before even the steady-state rotor studies become trivial tasks.

Part II of the thesis opens with a presentation in Chapter 4 of the numerical framework developed during the course of the project. It is in this part, that the development of all components in the framework are presented before Part III presents two test case applications. Part II is by far the largest part in the thesis and comprises Chapters 4 to 10.

After the initial framework presentation (Chapter 4) a literature review (Sec. 4.1) on discrete adjoint solvers can be found. In this part of the review the focus is on a particular approach to develop discrete adjoint solvers called the Krylov approach. Only three other works (Tab. 4.2) of this specific discrete adjoint solver type were found to be applied on CFD solvers using the SIMPLE algorithm. This final part of the literature review helped place the developed discrete adjoint solver from the present work in a broader context. This thesis actually documents the development of three versions of the discrete adjoint solver and they are all presented (Tab. 4.3) for the first time at the very end of Chapter 4. The literature review showed that steady-state cases up to 10 million cell meshes [38] could be achieved with a mature framework for the chosen adjoint solver type. Furthermore, one [104] of the efforts presented a methodology with transient capabilities using Large Eddy Simulations. The chosen strategy should therefore be promising long-term.

The second chapter (5) in Part II presents the development of the free-form deformation (FFD) module, called FFDlib, used to deform meshes and provide analytical mesh gradients to the surrounding framework. This was the second declared goal for the project. After the initial presentation, FFDlib is used in test cases in Chapter 6 (Fig. 6.7) and Chapter 7 (Fig. 7.12) before finally being used in Part III for the 3-D wing validation test case (Sec. 11.1). FFDlib has already at this early stage of the framework development proven to be a vital component and useful for carrying out optimizations using either adjoint solvers or the complex-step method.

Chapter 6 presents aspects of the EllipSys3D flow solver which were deemed relevant for the ensuing adjoint solver development in the final 4 chapters of Part II. Another goal for the project was to enhance the EllipSys3D flow solver with the ability to compute complex-valued flow fields. This process, known as complexification, is carried out in Chapter 6. The inclusion of the EllipSys3D flow solver into the numerical optimization framework is demonstrated on a test case (Fig. 6.7) using the complex-step method for gradient estimation. The presented ability to conduct shape optimization using the complex-step method is later on demonstrated for the 3-D wing validation test case (Sec. 11.1).

A focal point throughout the project was to develop the EllipSys discrete adjoint solver, eDA. However, this thesis presents three versions of eDA. The versions using finite-differences (eDA*fd*) or the complex-step method (eDA*cs*) are developed in Chapter 7. Sensitivity results on small meshes for the two versions align with the predicted performance (Fig. 7.9) where only the eDA*cs* gradient approaches machine accuracy. Another important contribution found in Chapter 7 is the road map guide (Sec. 7.2) to a systematic development procedure. Akin to the pedagogical adjoint introduction in Chapter 2 the road map aim is to simplify the development of a discrete adjoint as

much as possible. Certainly, something that is not easily done for such a daunting task. It should be emphasized that the detailed derivation of the pressure and flux residual subroutines (Eq. 7.6, and 7.7) is a pivotal point in the adjoint solver development. Yet, the related literature is rather scarce on this as regards SIMPLE algorithms which is why that part of the documentation certainly also is a contribution. Chapter 7 concludes with a multipoint adjoint-based optimization (Fig. 7.12) on a test case to demonstrate a functioning implementation.

Chapter 8 focuses on algorithmic differentiation and describes the development of the third eDA version, eDA*adf*, which is based on forward algorithmic differentiation. A gradient precision comparable to eDA*cs* is demonstrated for small meshes at the end of the chapter (Tab. 8.1 and 8.2). In Chapter 11 the eDA*adf* adjoint solver is furthermore used on the 3-D wing validation test case (Sec. 11.1).

Chapter 9 presents a methodology (Fig. 9.1) to include auxiliary models for turbulence and transition into an existing adjoint implementation. One can find several gradient verifications for small meshes in the chapter. These tests are reasonable for eDA*cs* and eDA*adf* whereas the adjoint solver based on finite differences, eDA*fd*, shows a reduced performance compared to previous chapters. These tests should be improved in the future by increasing mesh size and by incorporating the turbulence and transition variables into the cost function. However, later on there is a sensitivity test (Tab. 11.2) of the eDA*adf* adjoint solver on a 83 thousand cell mesh where the turbulence model is included. These results are discussed below.

Chapter 10 concludes Part II by describing one way of running in parallel in an MPI environment. This description should be seen as the absolute minimum. The reason is, that it is currently an active area of development in the eDA adjoint solver. In particular, there is a present focus on improving the parallel solution procedure as well as on accelerating the Jacobian computation. However, the chapter does conclude with sensitivity tests carried out in parallel to demonstrate a functional implementation. This is followed up by a sensitivity test in Chapter 11 computed in parallel using 48 CPUs. Also the 3-D wing validation test case (Sec. 11.1) is computed in parallel with the same number of CPUs.

Part III shifts the focus from development to application. The first application (Sec. 11.1) is a test case of a 3-D wing to validate the developed framework whereas the second application (Sec. 11.2) is a rotor design study which was carried out in collaboration with experts from the aerospace community.

The first test case (Sec. 11.1) is a single-point drag minimization of a 3-D wing subject to a lift constraint. The case is designed as a validation case for the overall framework and in particular for the developed discrete adjoint solver. The presented results are obtained using the eDA*adf* adjoint solver version. Also the ability to carry out optimizations based on the complex-step method using a complexified flow solver is demonstrated in this section. Preceding the drag minimization is a mesh convergence study (Sec. 11.1.1) followed by a sensitivity test (Tab. 11.2). Both first-order (UDS) and third-order (QUICK) stencils are used in the sensitivity test. The largest mesh used for

the adjoint solver is 83 thousand cells (L2 in Tab. 11.1). The eDA adjoint solver obtains 5 to 7 significant digits using UDS and 0 to 2 significant digits using QUICK (sign always correct). The difference for the two stencils in gradient precision in the adjoint solver affects the optimizations. The adjoint-based optimizations with the UDS stencil resulted in a tight convergence of the optimization problem (Fig. 11.5 and 11.5). This was not the case for the adjoint-based optimizations using the QUICK stencil which stopped approximately one order of magnitude short of the desired threshold.

To inspect the final wing design a comparison (Fig. 11.6 and 11.9) between adjoint-based optimizations using QUICK stencils were then made to the optimizations based on complex-step gradients. The results show that both the adjoint-based optimizations and the optimizations based on the complex-step method obtain the analytical elliptic lift distributions. However, the results from the complex-step optimization match the analytical distribution more closely due to the reduced gradient precision for the adjoint solver when using the QUICK stencil. For the optimizations based on the complex-step method it was furthermore shown (Fig. 11.10) that lift distributions from optimizations approach the analytical lift distribution as the mesh is refined. Here, meshes up to 664 thousand cells where used (L1 in Tab. 11.1). It was then investigated how closely the adjoint-based optimizations using UDS would follow optimizations based on the complex-step method. The results (Fig. 11.11) showed that hardly any difference could be discerned thanks to the adjoint solver's high gradient precision (i.e., 5-7 digits) for the UDS stencil. This ability to carry out identical gradient-based optimizations while interchanging the complex-step and the adjoint method has already proven extremely valuable.

The final part of the results from the 3-D wing case is a benchmark (Tab. 11.3) between the complexified EllipSys3D solver and the eDA adjoint solver. This benchmark clearly shows that an acceleration of the Jacobian computation in the adjoint solver is necessary. The computed runtime ratio between adjoint solver and flow solver was 89.2 on the L2 mesh. These timings were obtained without any coloring acceleration[1]. However, calculations in the discussion also showed, that once the coloring acceleration is implemented a runtime ratio of 1.9 should be within reach. These eDA estimates compare favorably to runtime ratios from the related literature for similar adjoint solvers which are 4.7 to 6.2 [38]. The benchmark furthermore entails overall memory consumption measurements. This allowed for an adjoint-flow memory ratio estimate of 13 (Tab 11.3) which also compare favorably to the lowest ratio, 23.7, we found reported from a similar adjoint solver [55, Tab. 2].

The presentation of the first application ends with identified areas of future work (Sec. 11.1.4). These improvements are necessary to carry out 3-D high-fidelity shape optimization of full rotor configurations. Among the identified areas of future work is an acceleration of the Jacobian as mentioned above. Another identified area is the reduced precision for the adjoint solver which must be improved. Also the solution procedure to the adjoint equation and the overall memory usage are important areas. These two topics are less dire than the precision and the acceleration. However, for industrial scale

[1] Coloring is a way to speed up Jacobian computation (in Sec. 4.1.2)

applications with meshes above 10 million cells one would most likely have to improve these areas.

The second case presented in Part III is a multipoint rotor design study (Sec. 11.2). We believe it to be the most comprehensive shape optimization study for a full wind turbine rotor. A comparison of the six known rotor studies is found in Tab. 3.1 and 3.2. The rotor study is carried out with the MACH framework developed at the MDOLab at the University of Michigan. The study is part of a series of optimization problems presented in a recent paper [71] which can be found in the appendix (Sec. A).

In short, it shows that one can optimize a modern 10 MW offshore wind turbine rotor using high-fidelity. Importantly, this allows for a simultaneous shaping of blade planform and cross-sectional shape. By taking the complex 3-D rotor flow at blade root and tip into account a novel free-form blade shape is obtained.

The excerpt presented in the present thesis (Sec. 11.2) starts by outlining the work carried out on the MACH framework in order to conduct the study. This is followed by a presentation of the optimization problem (Eq. 11.4) which is to optimize AEP using 154 design variables (Tab. 11.5) which are distributed out along the blade to manipulate twist, chord and shape (Fig. 11.13). There are constraints on both thrust an bending moment which proved vital to ensure a relevant final design. The multipoint study takes three wind speeds (5, 8, and 11 m/s) into account. A 14 million cell mesh is chosen based on a preceding mesh convergence study (Tab. 11.6). The optimization problem is shown to be deeply converged (Fig. 11.14) and optimizations are stopped close to the desired threshold ($10^{-4}$). By first inspecting single-point (8 m/s) results (11.15) it is seen that especially the leading edge shape is too pointy. A subsequent comparison to multipoint results (Fig. 11.16) reveals that the resulting multipoint design does have rounder leading edge shapes. However, it is also found that more should be done to ensure a robust design. To this end additional leading edge constraints are implemented leading to a comparison (Fig. 11.17) where the leading edge shape is much improved. By comparing spanwise forces (Fig. 11.18) between initial geometry and the final multipoint result with added geometrical constraints a clear improvement can be observed. Furthermore, the constraints on thrust and bending moment are observed to effectively reduce the overall loading and move the normal force peak farther inboard to ensure a relevant final design. The reported final AEP improvement of 24% does *not* suggest one can expect such large increases in performance relative to current state-of-the-art wind turbine designs. Indeed, the baseline for the multipoint case study is made much worse intentionally ([71, Fig. 2]) in order to make room for improvement in following optimization.

Although it is a comprehensive study, there are several areas where one could further improve the multipoint rotor study. One option could be to include the laminar to turbulent boundary layer transition. It would also be relevant to include turbulent inflow and to transition from a steady-state RANS formulation to an unsteady RANS formulation akin to the work by [85].

## 12.1 Outlook

Four of the most enticing avenues to pursue are discussed in the present section.

**Robust convergence**: The adjoint method hinges on a deeply converged flow problem. This is one of the reasons that enhanced convergence is an extremely active area of research at present day. A robust convergence should be seen as the gateway to the three other topics listed below. Therefore, an enhanced solution procedure is currently being developed for the EllipSys3D flow solver at DTU Wind Energy.

**Aerostructural optimization**: Extending the framework to other disciplines is very attractive. Here, specifically the inclusion of structural models is a high priority. As pointed out in the introduction (Chapter 1) the main motivation for carrying out high-fidelity shape optimizations is a concurrent shaping of the blade planform and cross-sectional shape. In a purely aerodynamic design it is custom to account for structures using constraints. However, geometric constraints are a poor surrogate for structural design feasibility. In particular, it is the combination of aerodynamic and the structural design of a rotor that may lead to large improvements. Here, the optimizer can to an even greater extent tailor the aerodynamic and structural responses to lower loads and increase power production.

**Transient formulation**: An extensive literature survey (Tab. 3.1) was carried out in Chapter 3. Here, only one [86] out of the six known rotor studies was found to be transient. Still, more transient studies should emerge as the steady-state rotor study case is mastered. As stated by Nielsen and Diskin in [86] one must be able to evaluate the linearized code at each physical time step to arrive at an adjoint formulation for unsteady flows. Nielsen and Diskin point to an efficient infrastructure to store and access the needed solutions as the predominant challenge to successfully transition from steady-state to unsteady adjoint capabilities. The type of discrete adjoint solver developed in this work is modelled after the work by Roth and Ulbrich [104] who already demonstrated that one can indeed arrive at a functioning unsteady adjoint formulation with the adjoint architecture described in the present work. It is therefore reasonable to believe a transition to an unsteady adjoint formulation to be feasible for the developed framework. This transition from steady-state to unsteady formulation will be pursued in future work.

**Multifidelity optimization**: An enhanced framework with i) improved convergence, ii) multiple disciplines, and iii) capabilities to carry out transient optimizations will inevitably need to reduce computation time. One very popular way to do so is to span multiple fidelities. As stated in the motivation (Sec. 1.1) the high-fidelity approach incurs a prohibitive computational cost to completely replace lower-fidelity methods in the wind energy industry. Instead, a complementary use of low- and high-fidelity should be very beneficial. This would bring the overall long-term goal of the developed framework to fruition, i.e., to provide a high-fidelity shape optimization framework which is relevant both in the wind energy research community and in the industry.

# APPENDIX A

# Multipoint high-fidelity CFD-based aerodynamic shape optimization of a 10 MW wind turbine

This research has already been published. The paper can be found on the next page.

Wind Energ. Sci., 4, 163–192, 2019
https://doi.org/10.5194/wes-4-163-2019


eawe european academy of wind energy
Open Access
WIND ENERGY SCIENCE

# Multipoint high-fidelity CFD-based aerodynamic shape optimization of a 10 MW wind turbine

**Mads H. Aa. Madsen[1], Frederik Zahle[1], Niels N. Sørensen[1], and Joaquim R. R. A. Martins[2]**

[1]Aerodynamic Design Section, DTU Wind Energy, Risø Campus, Frederiksborgvej 399, 4000 Roskilde, Denmark
[2]Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI 48109, USA

**Correspondence:** Mads H. Aa. Madsen (mham@dtu.dk)

Received: 3 October 2018 – Discussion started: 26 October 2018
Revised: 29 January 2019 – Accepted: 22 February 2019 – Published: 3 April 2019

**Abstract.** The wind energy industry relies heavily on computational fluid dynamics (CFD) to analyze new turbine designs. To utilize CFD earlier in the design process, where lower-fidelity methods such as blade element momentum (BEM) are more common, requires the development of new tools. Tools that utilize numerical optimization are particularly valuable because they reduce the reliance on design by trial and error. We present the first comprehensive 3-D CFD adjoint-based shape optimization of a modern 10 MW offshore wind turbine. The optimization problem is aligned with a case study from International Energy Agency (IEA) Wind Task 37, making it possible to compare our findings with the BEM results from this case study and therefore allowing us to determine the value of design optimization based on high-fidelity models. The comparison shows that the overall design trends suggested by the two models do agree, and that it is particularly valuable to consult the high-fidelity model in areas such as root and tip where BEM is inaccurate. In addition, we compare two different CFD solvers to quantify the effect of modeling compressibility and to estimate the accuracy of the chosen grid resolution and order of convergence of the solver. Meshes up to $14 \times 10^6$ cells are used in the optimization whereby flow details are resolved. The present work shows that it is now possible to successfully optimize modern wind turbines aerodynamically under normal operating conditions using Reynolds-averaged Navier–Stokes (RANS) models. The key benefit of a 3-D RANS approach is that it is possible to optimize the blade planform and cross-sectional shape simultaneously, thus tailoring the shape to the actual 3-D flow over the rotor. This work does not address evaluation of extreme loads used for structural sizing, where BEM-based methods have proven very accurate, and therefore will likely remain the method of choice.

## 1 Introduction

Wind turbine rotor optimization aims to maximize wind energy extraction and has been an important area of research for decades. A common metric is to minimize the levelized cost of energy (LCoE) (Ning et al., 2014), which can be decreased by lowering installation costs and operating expenses or by increasing the annual energy production (AEP). Simply upscaling the turbine leads to an increase in swept area, which in turn extracts more energy. However, a naive upscaling does not capture the complexity of the problem (Ashuri, 2012).

A major drawback of naive upscaling is that mass increases with the cube of the rotor radius. The industry avoids the prohibitive mass increase by improving the blade design, which has resulted in blades that are more slender for a given power rating, where the increase in loads (and therefore mass) can be kept low. This further results in blades with increased capacity factors.

Traditionally, the blade design optimization process has been sequential, where the optimization of airfoils and planform are performed in two distinct steps. In the present work, we optimize the airfoils and the planform concurrently using 3-D computational fluid dynamics (CFD). This

concurrent design optimization process is vital for the industry because, as previously shown, concurrent design processes result in a larger gain compared to sequential counterparts (Barrett and Ning, 2018), which is the main principle in multidisciplinary design optimization (MDO) (Martins and Lambe, 2013).

The use of 3-D CFD is particularly valuable near the turbine blade root and tip, since the blade element momentum (BEM) method uses empirical models to capture 3-D effects for these regions. The increase in fidelity also allows us to explore out-of-plane features such as blade pre-bend and winglets, which is outside the scope of traditional BEM approaches.

Industry still relies heavily on BEM, given that the 3-D CFD shape design of rotors poses several challenges. One of these challenges is modeling all the load cases that drive the design during an optimization. Much work has been done in steady-state computations with steady uniform inflow, but to truly generate realistic loads, one should transition to turbulent inflow and accurately resolve the time domain. This poses an immense challenge in terms of memory and computation time and is an active area of research.

In this paper, we present results from a high-fidelity aerodynamic shape optimization of a 10 MW offshore wind turbine rotor. By "high-fidelity", we mean a detailed modeling of the rotor in 3-D and the use of Reynolds-averaged Navier–Stokes (RANS) equations to model the aerodynamics throughout the optimization. The optimization is based on the case study from the International Energy Agency (IEA) Wind Task 37[1], which allows for a comparison with the low-fidelity BEM results from this case study. Low-fidelity tools offer a fast and reliable modeling approach. However, BEM does not capture the physics as completely as high-fidelity CFD-based tools that solve the RANS equations. In the present work, we aim to quantify the pros and cons of each approach.

Ideally, one would include all the relevant disciplines in such an optimization. This has been addressed in previous work using BEM-based aeroelastic tools combined with various cross-sectional analytical or finite-element-based structural tools.

Zahle et al. (2016) showed that simultaneous design of the aerodynamic shape and structural layout of a blade leads to passive load alleviation. This was achieved through bend–twist coupling, which increased the AEP without increasing loads and blade mass. The LCoE has been minimized by other researchers while taking aerodynamics, structures, and controls into account, thereby truly treating it as an MDO problem both for 5 MW turbines (Ashuri et al., 2014) and for 20 MW turbines (Ashuri et al., 2016). While we could tackle high-fidelity aerostructural optimization using tools that have already been demonstrated in aircraft wing design (Kenway and Martins, 2014; Brooks et al., 2018; Kenway et al., 2014), we focus solely on aerodynamic shape optimization in the present work.

We start the remainder of this paper with a literature review on wind turbine optimization. We then explain the methodology (Sect. 3), followed by a comparison between the compressible flow solver and an incompressible flow solver (Sect. 4). The design optimization problem is presented in Sect. 5, followed by the optimization results in Sect. 6. We end with our conclusions in Sect. 7.

## 2 Literature review

This literature review on wind turbine optimization is divided into three overall approaches: those that use low-fidelity and multi-fidelity models (Sect. 2.1), approaches that use CFD models without adjoint sensitivities (Sect. 2.2), and approaches that use CFD models with adjoint solvers (Sect. 2.3).

### 2.1 Low-fidelity and multi-fidelity shape optimization

CFD-based aerodynamic shape optimization is still rarely used in wind energy research, but both the aerospace and the automotive communities have been using it increasingly often (Chen et al., 2016; He et al., 2018). However, when it comes to low-fidelity shape optimization, the wind energy community has a large body of work.

BEM codes have been used extensively throughout the wind energy community for aerodynamic optimization. These codes are easy to implement and incur low computational cost. Robustness has been an issue in BEM codes, as they do not always converge (Maniaci, 2011). Robustness is critical, especially when the analysis is part of an optimization cycle. A lack of robustness will slow down the convergence in the best case, and interrupt the optimization altogether in the worst case. To address this issue, Ning (2014) re-parameterized the BEM equations using a single local inflow angle, resulting in guaranteed convergence.

It has long been known that the design of wind turbines is inherently a multidisciplinary endeavor. There have been more than two decades of research where BEM has been coupled with elastic beam models to account for structural deflections and material failure (Fuglsang and Madsen, 1999), including work in wind turbine optimization considering site-specific winds (Fuglsang and Thomsen, 1998, 2001; Fuglsang et al., 2002; Kenway and Martins, 2008).

BEM has also been coupled to structural models with different levels of fidelity. This allowed Bottasso et al. (2013) to study possible configurations to achieve bend–twist coupling resulting in load alleviation. They found that the highest load reduction is obtained by combining (passive) bend–twist coupling and (active) individual pitch control instead of using only a single approach. Another example where BEM is part of a larger multidisciplinary toolkit applied to the

[1] https://community.ieawind.org/tasks/taskdirectory (last access: 18 March 2019)

study of load alleviation is that of Zahle et al. (2016), who maximized AEP without exceeding the original overall loads of a 10 MW reference wind turbine (RWT). They achieved a 8.7 % AEP increase through passive load alleviation without an increase in the blade mass and only minor increases in the loads, despite blades that were 9 % longer. The parameterization was comprised of 60 design variables and just as in the work of Bottasso et al. (2013), they computed the gradients with finite differences. After an initial step size study, they ran a reduced set of design load cases to obtain the final turbine design, which was then evaluated on the full design load basis. Their work is a demonstration of the power of integrating design approaches.

As we will detail later, gradient-based optimization algorithms, combined with an adjoint method for computing the gradients, provide a powerful approach to address large-scale problems. For multidisciplinary systems, it is necessary to compute coupled derivatives, which presents additional challenges (Martins et al., 2005; Hwang and Martins, 2018). Ning and Petch (2016) introduced the application of the coupled adjoint method to the MDO of wind turbines.

One obstacle in using BEM codes is that the lift and drag data must be at hand. Typically, one uses data from wind tunnel experiments or low-fidelity numerical models, such as a panel code (Kenway and Martins, 2008). Another option is to use high-fidelity methods such as RANS CFD to generate the lift and drag coefficients for the BEM code (Barrett and Ning, 2016, 2018). Barrett and Ning (2018) combine BEM with both panel and 2-D RANS CFD in a comparison between two integrated blade design approaches ("precomputational" and "free-form") and a sequential approach. They used a panel code iteratively to converge the BEM residual and then either a panel code or CFD to generate the final lift and drag coefficients. Like Zahle et al. (2016), they argued for the integrated design approach, but they found that the precomputational approach achieved most of the benefits yielded by the free-form approach. This is impressive, since the precomputational approach took marginally more computation time than the sequential approach.

Gradient-based, gradient-free, and hybrid approaches have all been used to optimize airfoils using panel codes. An example of a gradient-based optimization approach is the Risø-B1 airfoil family, which currently is in commercial use by several manufacturers. Fuglsang et al. (2004) described the design and experimental verification process, where they used an in-house MDO tool. They carried out the numerical design studies using XFOIL (Drela, 1989) and used the VELUX wind tunnel for 2-D experimental verification. Due to concerns with XFOIL's accuracy in predicting separation, they opted to verify the optimization results using the CFD code EllipSys2D, thus combining fidelities in an attempt to balance speed and accuracy.

Grasso et al. optimized airfoils dedicated to both the blade tip (Grasso, 2011) and the blade root (Grasso, 2012), using gradient-based and hybrid approaches, respectively, both based on the panel code RFOIL, which is based on XFOIL. More recent airfoil studies have turned to large, offshore, pitch-controlled wind turbines, including tests with vortex generators that resulted in the development of a new airfoil family (Grasso, 2016).

Medium-fidelity vortex methods are popular aerodynamic models in wind turbine applications. Vortex theory is based on potential flow, which does not model the viscous effects modeled in RANS CFD. However, it does provide a more realistic solution than BEM codes while still keeping the computational cost low compared to CFD. Well-established vortex codes in the wind energy community include the GENeral Unsteady Vortex Particle (GENUVP) code (Voutsinas, 2006), the Aerodynamic Wind Turbine Simulation Module (AWSM) (Garrel, 2003), and the method for interactive rotor aeroelastic simulations (MIRAS) (Ramos García et al., 2016).

These vortex codes have been widely used in analysis, but applications to design optimization have been less frequent. Early optimization studies were performed by Zhiquan et al. (1992), Chattot (2003), and Badreddinne et al. (2005). More recent efforts based on vortex codes include those of Sessarego et al. (2016), who report on a surrogate-based optimization methodology, and of Lawton and Crawford (2014), who use the complex-step method to carry out the gradient-based optimization of a winglet. Researchers have also developed analytic gradient computation for vortex methods by reformulating the vortex dynamics using the finite element method (FEM) (McWilliam, 2015). However, BEM is still well entrenched and is currently the default choice for optimization.

### 2.2 High-fidelity CFD-based shape optimization without adjoint gradients

Barrett and Ning (2016) compared two numerical models of different fidelities (a panel code and RANS CFD) to wind tunnel data. They found that the choice of aerodynamic model had a large impact on the optimal design, which thereby stresses the need for high-fidelity models such as RANS. This agrees with Lyu et al. (2014), who report serious issues with Euler-based aircraft wing design due to missing viscous effects (compared to RANS-based design). They found that while Euler-based design yields some insights, the RANS-based optimization is needed to achieve a realistic design. Therefore, we limit the discussion in the present section to RANS CFD optimization.

Kwon et al. (2012) used 2-D RANS with a transition model to carry out gradient-based optimization using finite differences with nine design variables and achieved an 11 % increase in torque. Similarly, Ribeiro et al. (2012) used nine design variables and a gradient-free method (a genetic algorithm) to perform both multi-objective and single-objective optimizations. By training a surrogate model, they sped up the optimization process by almost 50 % while

achieving similar results. Liang and Li (2018) used two design variables (thickness and camber) to carry out 2-D shape optimization with a gradient-free method of airfoils (NACA0015) for vertical axis wind turbines (VAWTs). A subsequent 3-D modeling and CFD evaluation of the VAWT using the optimized airfoils achieved power coefficient increases up to 7 %. Finally, Zahle et al. (2014) carried out an airfoil optimization and wind tunnel validation. The developed optimization framework, based on the open-source framework OpenMDAO (Gray et al., 2019), included a combination of panel (XFOIL) and CFD (EllipSys2D) codes for the analysis, where the turbulence is described using the $k-\omega$ shear-stress transport (SST) turbulence model (Menter, 1993) and two transition models: $\gamma-\widetilde{Re_\theta}$ (Menter et al., 2004; Langtry et al., 2004; Sørensen, 2009) and the $e^N$ Drela–Giles transition model (Drela and Giles, 1986) described by Madsen (2002, chap. 6). They used a total of 21 design variables and computed the gradients using finite differences. They ran 20 optimizations under various conditions, and since each optimization involved 2640 CFD simulations, they split the procedure into two steps of increasing fidelity to save time. First, they optimized using XFOIL, and then, they used this intermediate result as a starting point for a subsequent CFD-based optimization. Such "warm starts" are now common practice, and we also use them in the present work. Using this framework, Zahle et al. (2014) completed the optimization of a 30 % and a 36 % airfoil called LRP2[2]-30[3] and LRP2-36, respectively. Finally, through experimental results from the Stuttgart Laminar Wind Tunnel for both LRP2-30 and LRP2-36, as well as the FFA-W3 counterparts (FFA-W3-301 and FFA-W3-360), they demonstrated that the new airfoils exhibit a superior performance compared to the FFA-W3 airfoils.

Shape optimization has also been used to optimize turbine blades using 3-D CFD in conjunction with gradient-free and gradient-based methods. Vucina et al. (2016) used 3-D RANS and a genetic algorithm to optimize the shape of wind turbine blades with up to 25 design variables. They concluded that their gradient-free framework was functional and robust, but also that many CFD evaluations were needed for the optimizer to converge due to the high number of variables. As a final example of the use of gradient-free methods with 3-D CFD models, Elfarra et al. (2014) optimized a winglet, also using a genetic algorithm. They used two design variables (cant and twist angle) to optimize the torque, resulting in a 9 % increase in power production. The results were obtained by training a surrogate model (an artificial neural network) using 24 CFD samples to reduce computational time.

There has been an increasing interest in blade extensions and winglets for wind turbines, since they can offer a cost-effective alternative to a complete blade redesign for site-specific performance enhancements. Zahle et al. (2018) explore such a design problem. They used 12 design variables to maximize the energy production while satisfying certain load constraints from the original blade design. Like Elfarra et al. (2014), they also used a surrogate model that they trained using a random sampling strategy. Here, they seek a more balanced design by using multiple wind speeds throughout the sampling. Using gradient-based optimization on the resulting surrogate model, they obtain a power increase of 2.6 % by adding a winglet, while not increasing the flapwise bending moment at 90 % radius.

To optimize with respect to large numbers of variables, gradient-based algorithms are the only hope if one wishes to achieve convergence to an optimum in a reasonable amount of time (Yu et al., 2018). The efficiency of gradient-based optimization is dependent in large part on the cost and accuracy of computing the gradients. Finite differences provide a way to compute gradients that is easy to implement, but they are subject to numerical errors, and they scale poorly with the number of design variables (Martins et al., 2003).

The complex-step derivative approximation method is an alternative to finite differences that is much more accurate but still scales linearly with the number of variables (Martins et al., 2003). This method has been widely used, including in some wind energy applications (Barrett and Ning, 2018; Kenway and Martins, 2008). Some efforts tried to reduce the computational cost by using semi-empirical gradients (Fuglsang and Madsen, 1999), surrogate models (Ribeiro et al., 2012; Elfarra et al., 2014; Zahle et al., 2018), and mixed-fidelity models (Barrett and Ning, 2018; Zahle et al., 2014).

For large numbers of variables, the adjoint method provides an efficient way to compute the required gradients (Peter and Dwight, 2010; Martins and Hwang, 2013), a fact that has also been verified in the wind energy community (Vorspel et al., 2017). The adjoint method is the subject of the next section.

### 2.3 High-fidelity CFD-based optimization using the adjoint method

We now detail previous efforts on RANS CFD-based shape optimization using the adjoint method, which we also use in the present work. These efforts are listed in Table 1.

Ritlop and Nadarajah (2009) were the first to use a high-fidelity shape optimization method with an adjoint solver for wind turbine profiles. They optimize the lift-to-drag ratio starting from the S809 airfoil using a compressible solver, a low-Mach preconditioner (both for flow and adjoint solver), and the Spalart–Allmaras (SA) turbulence model and find a tendency to increase camber to gain more lift. Finally, they point to the $k-\omega$ SST turbulence model and a transition model as needed improvements. Khayatzadeh and Nadarajah (2011) optimized the same airfoil using an enhanced frame-

[2]LRP stands for Light Rotor Project.

[3]https://energiteknologi.dk/node/1197 (last access: 18 March 2019)

**Table 1.** Overview of related work using the adjoint method.

| Reference | Turbine[g] | Adjoint | Dim. | $Re$[f] | Mesh size[a] | Design variables | Iterations[b] |
|---|---|---|---|---|---|---|---|
| Ritlop and Nadarajah (2009) | – | Discrete | 2-D | $2.0 \times 10^6$ | $3.3 \times 10^4$ | 385 | 100–200 |
| Khayatzadeh and Nadarajah (2011) | – | Discrete | 2-D | $2.1 \times 10^6$ | $1.3 \times 10^5$ | 385 | – |
| Schramm et al. (2014) | – | Continuous | 2-D | $3.0 \times 10^6$ | $5.5 \times 10^4$ | 720 | – |
| Schramm et al. (2016) | – | Continuous | 2-D | $7.9 \times 10^6$ | – | 480 | – |
| Barrett and Ning (2016) | – | Continuous[f] | 2-D | $1.0 \times 10^6$ | $1.4 \times 10^4$ | 10–22 (Table 2) | – |
| Vorspel et al. (2017) | – | Continuous | 2-D | $5.0 \times 10^4$ | $5.0 \times 10^4$ | 2–364 (Table 1) | – |
| Schramm et al. (2018) | – | Continuous | 2-D | $2.0 \times 10^6$ | $2.1 \times 10^5$ | 20–50 | 0–30 (Figs. 5, 7) |
| Barrett and Ning (2018) | – | Continuous[f] | 2-D | $1.0 \times 10^6$ | $1.4 \times 10^4$ | 10–68 (Table 1) | – |
| Economon et al. (2013) | – | Continuous | 2-D | $1.0 \times 10^3$ | $3.2 \times 10^4$ | 50 | 10 |
| | NREL Phase VI | | 3-D | $1.0 \times 10^6$ | $7.9 \times 10^6$ | 84 | 3 |
| Vorspel et al. (2016) | – | Continuous | 2-D | $5.0 \times 10^4$ | – | 2 | 30 (Fig. 3) |
| | – | | 3-D | $1.2 \times 10^6$ | $2.4 \times 10^6$ | – | < 8 (Fig. 6) |
| Dhert et al. (2017) | NREL Phase VI | Discrete | 3-D | $1.0 \times 10^6$ | $2.6 \times 10^6$[d] | 1–252 | 9–23 |
| Vorspel et al. (2018) | NREL Phase VI | Continuous | 3-D | $1.0 \times 10^6$ | $5.2 \times 10^6$[e] | 5–9 | < 8 (Fig. 5) |
| Tsiakas et al. (2018) | MEXICO | Continuous | 3-D | $1.0 \times 10^6$ | $2.5 \times 10^6$[c] | 135 | 10 (Fig. 4) |
| This study | IEA | Discrete | 3-D | $1.0 \times 10^7$ | $1.4 \times 10^7$ | 1–154 | 100–200 |

[a] Number of cells in largest mesh used for optimization. [b] Not all papers state the number of optimization iterations explicitly. In some cases, we report the number of iterations estimated from the cited figures. As mentioned in Vorspel et al. (2017), this number depends on the optimization problem and optimizer settings, meaning that cross-setup comparison is difficult. [c] Tsiakas et al. (2018) only gives the number of mesh nodes. [d] Reduced geometry where the root section was removed. [e] Applied symmetric boundary conditions (BCs) double the mesh size compared to others. [f] In cases where a range of Reynolds numbers were used, we report the maximum values. [g] We only found high-fidelity shape optimization for three turbine configurations in the literature: two smaller turbines – NREL Phase VI and MEXICO (Schepers et al., 2012) – and the large, commercial-scale IEA 10 MW wind turbine. We find it reasonable to assume that the simulations for NREL Phase VI and MEXICO have a Reynolds number on the order of $Re = 10^6$ (Sørensen et al., 2002, p. 152; Schepers et al., 2012, p. 10), while we estimate the Reynolds number for the IEA turbine to be on the order of $Re = 10^7$ (Bak et al., 2013, p. 15–16).

work that included the cited improvements, where they attempted to postpone the onset of transition. They concluded that both the capability and accuracy of the discrete adjoint optimization framework improved by including the new adjoint variables for the transition model.

There have been several contributions to 2-D RANS shape optimization that use the continuous adjoint approach (Schramm et al., 2014, 2015, 2016, 2018). In these efforts, the continuous adjoint implemented for ducted flows in the flow solver OpenFOAM (Weller et al., 1998) was extended to handle external aerodynamics. First, Schramm et al. (2014) optimized the lift-to-drag ratio of the DU 91-W2-250 profile using 720 design variables while constraining cross-sectional area. They use the "frozen turbulence" assumption, which means that no adjoint equation is used for the turbulence model. Since each surface point in this work is a design variable, they smooth the gradient for stability. The result is a 5.7 % to 59 % increase in lift-to-drag ratio for angles of attack ranging from 6.15 to 9.66°.

In a later work, Schramm et al. (2015, 2016) presented a finite-difference verification of the adjoint gradients. The same group performed the shape optimization of an upstream leading edge (LE) slat for the DU 91-W2-250 airfoil and a validation of the framework using wind tunnel data, showing good agreement below stall (Schramm et al., 2016). The optimization, which used 480 design variables, resulted in a 2 % decrease in drag. As mentioned previously, there have been other efforts in turbine blade design using 2-D RANS CFD with the adjoint method (Barrett and Ning, 2016, 2018) that used the open-source compressible solver SU2 (Palacios et al., 2013). These works couple the 2-D RANS and adjoint model to BEM, panel, and beam element analysis codes to arrive at a 3-D multi-fidelity and multidisciplinary design framework.

Vorspel et al. (2017) present a benchmark of different optimization algorithms (Nelder–Mead, steepest descent, and quasi-Newton) for unconstrained shape optimization in 2-D, where the continuous adjoint solver within OpenFOAM is used. The benchmark optimization problem is to find a target lift coefficient from any baseline shape. However, they both consider computation time and ease of use to grade the algorithms. As already mentioned, they point to the use of the adjoint method to compute the gradient for a large number of design variables. They recommend that further analysis be done within constrained optimization and within multidisciplinary optimization.

In a more recent work within unconstrained optimization, Schramm et al. (2018) investigated the effect of the "frozen turbulence" assumption in 2-D. They carried out their investigations on the NACA 0012 and DU 93-W-210 airfoils. In this single-point study, they concluded that the implementation of adjoint turbulence models results in better gradients than those obtained through the frozen turbulence assump-

tion. Finally, they specifically mention thickness constraints as a future work topic.

OpenFOAM with a continuous adjoint solver has also been used in 3-D. This was done by Vorspel et al. (2016), who first performed a 2-D test case with two design variables. The 3-D test case consisted of an extruded airfoil with a spanwise length of five chords and a mesh of $2.4 \times 10^6$ cells with an $y^+$ of 2.5. They investigated both a twist and a bend–twist coupling case but found that the bending had no discernible effect. This is something they expect to change for future rotating blades applications.

The above work does not model the rotation, which is important to get the correct local angle of attack along the blade and thus accurately compute the forces acting on the blade. Several 3-D adjoint-based optimization efforts model rotation effects, three of which studied the NREL Phase VI rotor (Economon et al., 2013; Dhert et al., 2017; Vorspel et al., 2018), and another which studied the MEXICO rotor (Tsiakas et al., 2018). Economon et al. (2013) used a continuous adjoint formulation to perform single-point aerodynamic shape optimization using a compressible RANS model. In 2-D, they reduced drag starting from a NACA 4412 profile baseline by 4.86 % under imposed thickness constraints. They used a total of 50 design variables and completed 10 design iterations. In 3-D, they improved the torque coefficient on a mesh with 7.9 million cells by 4 % using 84 shape variables with no constraints imposed on geometry or loads. The free-form deformation (FFD) box covered part of the blade such that both the trailing edge and the innermost part of the blade could not deform.

The optimization was not fully converged, as only three design iterations were performed. One drawback in this early work is the use of the frozen turbulence assumption, which they also identified as an area of future work.

Dhert et al. (2017) used a discrete adjoint solver to carry out a multipoint optimization of a two-bladed rotor using a 2.6 million cell mesh, where they maximized the torque coefficient using up to 252 design variables. They used pitch, twist, and local shape design variables while constraining the thickness between 15 % and 50 % of the local blade chord to ensure adequate space for a structural box. The final multipoint optimization resulted in a 22.1 % increase in torque coefficient but also increased the thrust by 69 %. The original design was meant to be a three-bladed rotor, which explains the low thrust coefficients in the reported results (Dhert et al., 2017, Table 1). They found the optimized shapes for both single and multipoint optimization exhibited highly cambered trailing edges at the root region where the wind speed is reduced. While this does agree with what has been reported in 2-D cases (Ritlop and Nadarajah, 2009), it is also exactly what one would expect when chord is not included as a design variable.

The present work builds on Dhert et al. (2017), who used the same CFD solver, ADflow. Our improvements are summarized in Table 2. One major improvement has to do with the adjoint implementation that was used. As we will explain in more detail later (in Sect. 3.2.2), our adjoint implementation uses the automatic differentiation (AD) technique to compute certain derivatives (Mader et al., 2008). One major improvement is that we implemented the more memory-efficient reverse automatic differentiation. Dhert et al. (2017) was forced to use the less memory-efficient forward automatic differentiation because the reverse option did not include the rotating terms required to model wind turbine rotors. We also added constraints on maximum thrust and flapwise bending moment to align with the IEA case study and enlarged the design space to include chord design variables. Furthermore, Dhert et al. (2017) carried out their studies on the turbine blade excluding the root because of flow solution convergence issues, whereas we include the root. This was made possible by the robustness of the new approximate Newton–Krylov (ANK) solver in ADflow, which also increases its speed (Yildirim et al., 2018). Finally, we achieve an optimality convergence tolerance that is up to 5 orders of magnitude lower.

**Table 2.** Overview of differences between the work by Dhert et al. (2017) and the present work.

| | Dhert et al. (2017) | Present work |
|---|---|---|
| Geometry | Reduced geometry (no root) | Entire geometry included |
| Adjoint derivatives | Forward AD | Reverse AD |
| Convergence of optimization | $[10^{-1}, 10^{-2}]$ | $[10^{-2}, 10^{-7}]$ |
| Optimization iterations | $\mathcal{O}(10^1)$ | $\mathcal{O}(10^2)$ |
| Maximum mesh size (million) | 2.60 | 14.16 |

Another recent effort is that of Vorspel et al. (2018), who performed unconstrained optimization of the NREL Phase VI rotor where they minimized the thrust by varying up to nine twist design variables using a steepest descent optimization algorithm. Not surprisingly, they mention convergence issues, in part due to the turbine being stall regulated and exhibiting separated flow at some inflow speeds. Vortices at tip and root further impaired the convergence, which in turn resulted in poor gradient quality. They addressed this issue by limiting the deformable area to only the outer 50 % of the blade length, which limited the shape design optimization. Like Economon et al. (2013), they used the frozen turbulence assumption. However, they differed in choice of turbulence model: Vorspel et al. (2018) used the $k-\omega$ SST model, while Economon et al. (2013) used the SA turbulence model. For future work, they suggested the use of more efficient optimization algorithms, and mentioned the inclusion of the adjoint turbulence equations and the study of turbines that are not stall regulated.

**Table 3.** Overview of aerodynamic optimization works of wind turbine rotors using the adjoint method.

| Reference | Multi | Turbulence | Deformation (✓ = full blade) | Geometry (✓ = full blade) | Load constraints | | Geometric constraints | Design variables | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | Thrust | Moment | | Twist | Chord | Shape |
| Economon et al. (2013) | | | | ✓ | | | ✓ | | | ✓ |
| Dhert et al. (2017) | ✓ | ✓ | | | | | ✓ | ✓ | | ✓ |
| Vorspel et al. (2018) | | | | ✓ | | | | ✓ | | |
| Tsiakas et al. (2018) | | ✓ | ✓ | ✓ | | | | | | ✓ |
| This study | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Multi: multipoint optimization; turbulence: whether the turbulence model is included in the adjoint solver; deformation: whether the entire blade was allowed to deform; geometry: whether the entire blade was modeled; geometric constraints: whether any geometric constraints were imposed.

Finally, Tsiakas et al. (2018) used a continuous adjoint approach that included the SA turbulence equations to optimize the MEXICO RWT. The flow was modeled by the incompressible RANS equations and solved in a co-moving frame of reference. They maximized the power for a single wind speed of $10\,\mathrm{m\,s^{-1}}$. Compared to the present work, they used a different parameterization technique based on volumetric non-uniform rational B splines (NURBS), which confine the blade in a small volume. NURBS are used both for the deformation of the surface and the volume meshes, and the outermost NURBS control points are fixed to keep the outer volume mesh fixed. This only ensures $C^0$ continuity. They use 385 NURBS, resulting in 135 design variables, which are only allowed to move in the direction perpendicular to the rotor plane. This choice of parameterization limits the design space; for example, no chord increase can be obtained without simultaneously changing profile shape. The flow and adjoint solvers take advantage of graphics processing unit (GPU) hardware, resulting in fast solutions. Indeed, they state that the overall optimization process can run up to 50 times faster on GPUs than on CPUs. They obtained a 3 % increase in the objective function and attribute this modest improvement to the limited freedom in the parameterization.

In spite of the contributions cited above, many improvements are needed before we achieve the ultimate goal of providing a "push-button solution" for wind turbine manufacturers. This paper contributes with some of these improvements by including all of the following features in a comprehensive high-fidelity 3-D RANS-based shape optimization framework:

1. enforcement of geometric constraints to ensure structural feasibility,
2. normal operation rotor load constraints limiting thrust and flapwise bending moment,
3. more precision and stability in the convergence of flow and adjoint solvers,
4. inclusion of a turbulence model in the adjoint solver,
5. a comprehensive set of design variables, and
6. modeling and deformation of the entire blade shape.

In Table 3, the present work is compared to the above-cited 3-D shape optimization efforts on wind turbine rotors.

As previously mentioned, structural considerations are crucial in wind turbine design. Anderson et al. (2018) partially addressed this issue by coupling the NSU3D RANS solver with the AStrO structural finite element solver through a fluid–structure interface to converge on realistic, steady-state loads on the SWiFT RWT. They used Abaqus to make a finite element model with shell elements. They performed a purely structural optimization of the composite blade, with the loads computed by the CFD. The optimization's objective was to, using gradient-based optimization, minimize the off-axis stress with respect to 16 310 ply orientation variables. They completed 10 optimization iterations considering five different load cases and achieved a reduction in the maximum fatigue stress between 40 % and 60 %. They did so without adding any constraints, but they did assume the material to be a single-ply, unidirectional fiber composite for each blade section. The logical next step would be to perform the simultaneous optimization of the structural sizing and aerodynamic shape optimization, as is already done in aircraft wing design (Kenway and Martins, 2014).

## 3 Methodology

We now briefly describe all components of the optimization framework. The overall workflow is shown in Fig. 1 using an extended design structure matrix (XDSM) diagram (Lambe and Martins, 2012). An initial set of design variable values, $\boldsymbol{x}^{(0)}$, is given to the optimizer. The optimizer passes the current design variables to the surface deformation module, prompting it to update the surface mesh (except for the very first iteration). The surface deformation module also provides analytic derivatives of the surface mesh with respect to the design variables, $\mathrm{d}\boldsymbol{x}_\mathrm{s}/\mathrm{d}\boldsymbol{x}$. After the surface mesh has been updated, it is passed to the volume deformation module, which updates the volume mesh and computes its analytic derivatives with respect to the surface mesh, $\mathrm{d}\boldsymbol{x}_v/\mathrm{d}\boldsymbol{x}_\mathrm{s}$. Then, the flow solver computes the flow states, $\boldsymbol{w}$. These states are passed to the adjoint solver, which computes the total derivative. Finally, the objective function, $f$ (e.g., torque), as well as its derivatives, $\mathrm{d}f/\mathrm{d}\boldsymbol{x}$, are provided to the opti-

mizer, which computes a new step for another optimization iteration. Both the surface and volume deformation steps are fast explicit operations. On the other hand, the flow and adjoint solvers are costly iterative operations that take up the vast majority of the computation time. The optimization process involves $\mathcal{O}(10^2)$ major iterations, which is an absolute minimum bound on the number of CFD solutions and mesh updates; there are additional CFD solutions within each major iteration.

### 3.1 Geometry and mesh deformation

To deform the surface geometry and mesh, we use the Python module pyGeo developed by Kenway et al. (2010), which implements the FFD (Sederberg and Parry, 1986) technique. Some of the key features of FFD are analytic derivatives and a machine-precision representation of the baseline geometry.

The volume deformation tool is called IDWarp and is based on the inverse distance weighting function (Edward et al., 2012). IDWarp is a fast and unstructured deformation algorithm that has been demonstrated in aerodynamic (Yu et al., 2018; He et al., 2018) and aerostructural applications (Brooks et al., 2018).

### 3.2 Flow and adjoint solvers

#### 3.2.1 EllipSys3D

EllipSys3D is an in-house, structured, multi-block, finite volume method (FVM) flow solver developed at DTU Wind Energy by Michelsen (1992, 1994) and Sørensen (1995), and we use it in the present work to perform the comparison between CFD solvers. It discretizes the incompressible RANS equations using general curvilinear coordinates and couples velocity and pressure through the SIMPLE algorithm.

In this study, we run EllipSys3D using the third-order quadratic upwind interpolation for convection kinematics (QUICK) scheme and the $k-\omega$ SST (Menter, 1993) model to calculate the turbulent eddy viscosity, which compares favorably to other turbulence models for wind turbine applications (Reggio et al., 2011).

EllipSys3D has been validated against experimental data for the MEXICO RWT (Bechmann et al., 2011) and the NREL Phase VI RWT (Sørensen et al., 2002; Sørensen and Schreck, 2014), and also in a blind comparison (Simms et al., 2001). In addition, the unsteady interaction between tower and blade has been simulated for the NREL Phase VI RWT with EllipSys3D using overset grid capabilities, and an overall good agreement was found with experimental data (Zahle et al., 2009). EllipSys3D has been used in various rotor applications to perform computations, such as aerodynamic power (Johansen et al., 2009) and fluid–structure interaction (Heinz et al., 2016). The latter work also encompasses a comparison across fidelities between the CFD-based tool, HAWC2CFD, and the BEM-based HAWC2 solvers where a good agreement was found. EllipSys3D has also been compared with OpenFOAM for a case with atmospheric flow over complex terrain. EllipSys3D was found to be 2–6 times faster while producing almost identical numerical results (Cavar et al., 2016). More recent sources also show that these two solvers yield comparable results (Sørensen et al., 2016; Yilmaz et al., 2017; Boorsma et al., 2018).

#### 3.2.2 ADflow

ADflow is a compressible RANS solver based on SUmb (Weide et al., 2006), a structured FVM CFD solver written in Fortran 90 that uses cell-centered variables on a multi-block grid. Unlike EllipSys3D, ADflow uses the Spalart–Allmaras (SA) turbulence model (Spalart and Allmaras, 1994) and works with state variables computed using the Jameson–Schmidt–Turkel (JST) scheme. More recently, Kenway et al. (2017) implemented overset mesh capability. ADflow is wrapped with Python to provide a more convenient user interface and to facilitate integration with optimization algorithms and other components of an MDO framework.

ADflow has been coupled to a structural finite element solver in the MACH (MDO for aircraft configurations with high fidelity) framework (Kenway et al., 2014), which has been used to perform not only aerostructural optimization of aircraft configurations (Kenway and Martins, 2014; Brooks et al., 2018; Burdette and Martins, 2018, 2019) but also hydrostructural optimization of hydrofoils (Garg et al., 2019, 2017).

As previously mentioned, we use an ANK solver, which is implemented in ADflow to provide robustness. The ANK implementation is important since it is crucial to properly converge the flow field in order to obtain accurate gradients. Newton–Krylov (NK) methods are not robust because they might not converge if the starting point is outside the basin of attraction. ANK addresses this convergence issue using a globalization method called pseudo-transient continuation, which starts with the stable but inefficient backward Euler method with a small time step, and then increases the time step to approach the higher convergence rate of the NK solver. The ANK method involves the solution of large linear systems using preconditioners. These systems are solved in a matrix-free fashion with the generalized minimal residual (GMRES) algorithm (Saad and Schultz, 1986) using the Portable, Extensible Toolkit for Scientific Computation (PETSc) library (Balay et al., 2018a, b, 1997). The adjoint solver linear systems are solved using the same algorithm. ADflow is considered converged when the ratio of the L2 norm of the residuals at iteration $n$ and the same norm of the free stream residual is below a given tolerance, i.e., when

$$\eta \leq \frac{\|\mathcal{R}^n\|_2}{\left\|\mathcal{R}^{\mathrm{fs}}\right\|_2}. \tag{1}$$

For the optimizations presented below, we typically set $\eta = 10^{-9}$, whereas the L2 convergence for the adjoint equation

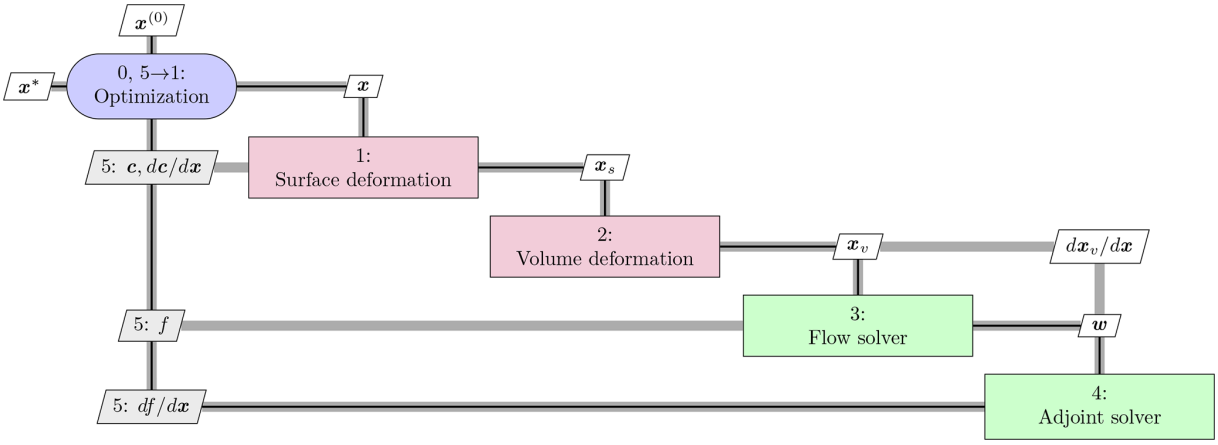



**Figure 1.** Extended design structure matrix (XDSM) showing the optimization framework. Green blocks are iterative solvers, whereas red boxes represent explicit functions. Black lines represent the process flow in the order of the numbers; gray lines represent data dependencies.

is set to $10^{-7}$. These convergence thresholds are not to be confused with the optimality tolerance, which we set to $10^{-4}$.

One crucial capability in ADflow is the efficient computation of gradients through its adjoint solver. Together with the geometry and mesh deformation tools mentioned above, and the optimization software mentioned in the next section, this enables aerodynamic shape optimization with respect to hundreds of design variables (Lyu et al., 2014; Chen et al., 2016; Dhert et al., 2017). All the optimization results in Sect. 6 are obtained with the ADflow framework.

We now derive the adjoint equations and briefly explain how they are assembled and solved. A detailed description of the implementation is provided in previous work (Mader et al., 2008; Mader and Martins, 2011; Lyu et al., 2013). The CFD solver computes the flow field, $\boldsymbol{w}$, for a given set of design variables, $\boldsymbol{x}$, by converging the residuals $\mathbf{R}(\boldsymbol{x}, \boldsymbol{w})$ of the governing equations to zero. Then, any function of interest, $f(\boldsymbol{x}, \boldsymbol{w})$, can be computed. Gradient-based optimizers require the gradient of the objective and constraint functions with respect to the design variables. To compute this gradient, we use the equation for the total derivative:

$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \frac{\partial f}{\partial \boldsymbol{x}} + \frac{\partial f}{\partial \boldsymbol{w}} \frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}\boldsymbol{x}}. \tag{2}$$

Here, the partial derivatives correspond to derivatives of explicit functions, while the total derivative involves the iterative solution of the governing equations. Thus, the partial derivatives can be found analytically at a low computational cost, but the direct computation of the total derivative $\mathrm{d}\boldsymbol{w}/\mathrm{d}\boldsymbol{x}$ should be avoided. A similar total derivative equation can be written for the residuals, which must remain zero for the CFD solution to hold, and thus

$$\frac{\mathrm{d}\mathbf{R}}{\mathrm{d}\boldsymbol{x}} = \frac{\partial \mathbf{R}}{\partial \boldsymbol{x}} + \frac{\partial \mathbf{R}}{\partial \boldsymbol{w}} \frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}\boldsymbol{x}} = 0. \tag{3}$$

We can now substitute the solution of the Jacobian given by the above equation into the total derivative equation (Eq. 2) to obtain

$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \frac{\partial f}{\partial \boldsymbol{x}} - \underbrace{\frac{\partial f}{\partial \boldsymbol{w}} \left[\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}\right]^{-1}}_{\boldsymbol{\Psi}^T} \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}, \tag{4}$$

where we have only partial derivative terms that can be found analytically at a low computational cost. The linear system in this equation can either be solved by computing the solution Jacobian, $\mathrm{d}\boldsymbol{w}/\mathrm{d}\boldsymbol{x}$, from the linear system from Eq. (3) or by solving the adjoint system:

$$\left[\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}\right]^T \boldsymbol{\Psi} = \left[\frac{\partial f}{\partial \boldsymbol{w}}\right]^T, \tag{5}$$

where $\boldsymbol{\Psi}$ is the adjoint vector, which can be substituted into the total derivative equation (Eq. 2), i.e.,

$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \frac{\partial f}{\partial \boldsymbol{x}} - \boldsymbol{\Psi}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}. \tag{6}$$

The cost of the adjoint method is independent of the number of design variables because the adjoint equation (Eq. 5) does not contain $\boldsymbol{x}$. However, if there are multiple functions of interest $f$, we need to solve Eq. (5) for each $f$ with a different right-hand side. Given that our problem has $\mathcal{O}(10^2)$ design variables and only a few functions of interest, the adjoint method is particularly advantageous.

In the adjoint equation (Eq. 5) and total derivative equation (Eq. 6), we need to provide two matrices and two vectors of partial derivatives. As mentioned above, these derivatives involve only explicit operations and are in principle cheap to compute. However, they still require the differentiation of parts of a complex CFD code, and a good implementation is essential to preserve the accuracy and efficiency of the adjoint approach. Traditionally, adjoint method developers have derived these partial derivatives by differentiating the equations or code manually and programming new functions that

compute those derivatives. This process is labor intensive and prone to programming errors. To address this drawback, Mader et al. (2008) pioneered the use of automatic differentiation to compute the partial derivatives. Automatic differentiation is a technique that takes a given code and produces new code that computes the derivatives of the outputs with respect to the inputs (Griewank, 2000). Using a pure automatic differentiation approach to compute our derivatives of interest, $\mathrm{d}f/\mathrm{d}\boldsymbol{x}$, would mean applying the automatic differentiation tool to the whole CFD code, including the iterative solver. While this produces accurate derivatives, it is not an efficient approach. By selectively using automatic differentiation to produce code that computes only the partial derivatives, which do not involve the iterative solver, we lower the adjoint implementation effort while keeping the efficiency of the traditional adjoint implementation approach. There are still many details involved in making our adjoint implementation approach efficient; these details have been presented in previous work (Mader et al., 2008; Lyu et al., 2013).

As briefly mentioned in the introduction, there are two modes for automatic differentiation: the forward mode and the reverse mode. Dhert et al. (2017) had used automatic differentiation in forward mode to compute and store the flow Jacobian, $\partial\mathbf{R}/\partial\boldsymbol{w}$, as well as the other partial derivatives. Then, these stored matrices are used by the adjoint solver to compute transpose-matrix-vector products to converge the adjoint solution, $\boldsymbol{\Psi}$. Using the reverse mode, no storage of the Jacobian is needed. Instead, a matrix-free approach is used, where the transpose-matrix-vector products required to converge the adjoint solution are computed directly through the reverse mode derivative routines. While the reverse mode is more efficient in terms of memory usage, the reverse mode implementation was missing the rotation terms required for wind turbine modeling. We have fixed this for the implementation in the present work and use the reverse mode instead. The implemented reverse AD routines may also lead to speed up depending on the number of Krylov iterations needed to converge the adjoint system.

### 3.3 Optimizer

We use the Sparse Nonlinear OPTimizer (SNOPT) (Gill et al., 2002) for all optimizations herein. SNOPT implements a sequential quadratic programming (SQP) algorithm. We use it through the open-source Python wrapper pyOptSparse[4], which provides a common interface to this and other optimization software. The convergence in SNOPT is set through the "major optimality tolerance" setting (Gill et al., 2007). We aim at converging all optimization problems to $10^{-4}$.

[4]https://github.com/mdolab/pyoptsparse (last access: 18 March 2019)

## 4 Flow solver comparison

In this work, we use ADflow as the CFD solver in the design optimization due to its adjoint gradient computation and integration with geometry parameterization, mesh deformation, and optimization tools. However, EllipSys3D has been more thoroughly validated for wind turbine rotor flows, so in this section, we verify ADflow against EllipSys3D for a three-bladed pitch-regulated rotor geometry. In this section, we only include a mesh convergence study for one operational condition. A more detailed flow comparison is included in Appendix A.

### 4.1 Fluid model and computational mesh

All simulations are steady-state 3-D RANS of the rotor only, where effects from both tower and nacelle have been neglected. Since we study a rigid upwind turbine, neglecting tower and nacelle should have a limited effect. We also note that we compute the flow field using a co-rotating, non-inertial reference frame that is attached to the rotor. Therefore, the RANS equations have additional terms to account for Coriolis and centripetal forces. Just as for the IEA Wind Task 37 case, the three-bladed pitch-regulated rotor geometry in the analysis is a design based on the DTU 10 MW RWT (Bak et al., 2013), where both chord and twist distributions have been altered to allow for more room for improvement using design optimization. We compare the twist and thickness distributions for the DTU 10 MW RWT and the IEA Wind Task 37 baseline in Fig. 2

The surface mesh consists of three blades, each with 36 blocks. For each blade, there are 256 cells in the chordwise direction and 128 in the spanwise direction (tip excluded). The surface mesh is generated using the in-house Parametric Geometry Library (PGL). The tip was constructed using four blocks of $32 \times 32$ cells each, resulting in a total surface mesh with 110 592 mesh cells.

The spherical volume mesh has an O–O topology generated with the hyperbolic in-house mesh generator HypGrid (Sørensen, 1998). Setting the first boundary layer cell height to $10^{-6}$ m yields a $y^+$ of around 1 for the given operational conditions, and a total of 128 cell layers are grown from the surface mesh where the farthest vertices reach a distance of 1740 m. This results in a total of 432 blocks, each with $32 \times 32 \times 32$ cells, which is equivalent to 14.155776 million cells. Given a span of $R = 89.166$ m, the surrounding spherical mesh expands to about 20 times the blade span.

The mesh we just described above is the finest mesh we use, which we call the L0 mesh. A coarser (L1) mesh is obtained by coarsening L0 once, i.e., by removing every second cell in all three directions. Similarly, the L2 mesh is obtained by coarsening L1. Unless otherwise stated, we use these three meshes in all the work herein. The turbine geometry and the surrounding spherical mesh are shown in Fig. 3, and a more detailed view of the rotor is shown in Fig. 4.

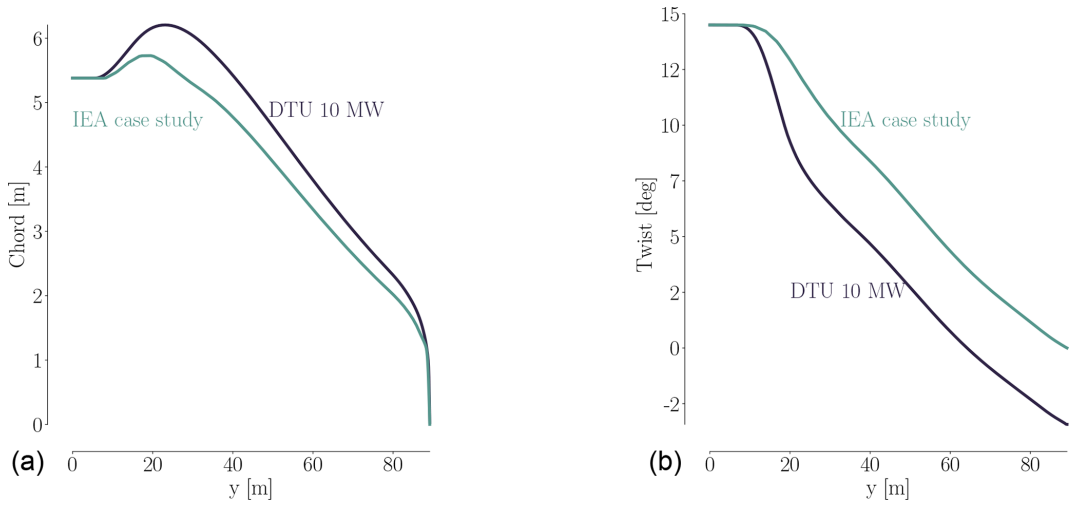



**Figure 2.** Comparison of chord **(a)** and twist **(b)** for the DTU 10 MW RWT and the perturbed design used as the starting point for the IEA optimization case study. Both the chord and twist are reduced. The baseline blade design is based on the FFA-W3 airfoil family with relative thicknesses in the range of [24 %, 36 %].

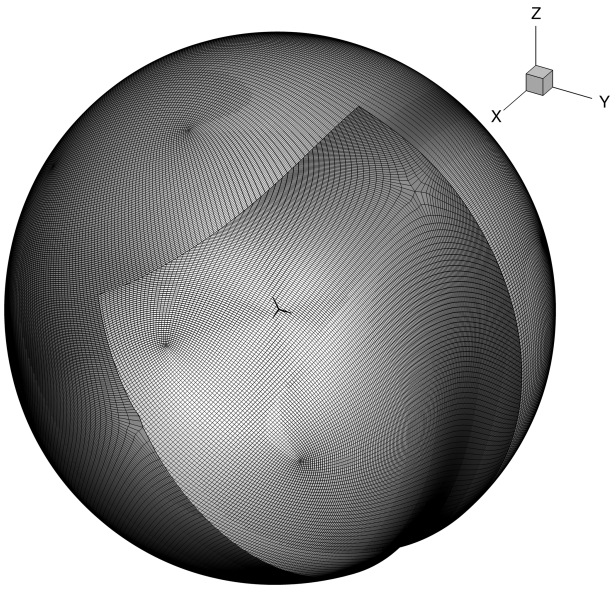



**Figure 3.** The baseline wind turbine design with the spherical L0 mesh around it. The blade span is 89.166 m, and the spherical mesh stretches to 20 times the blade span.

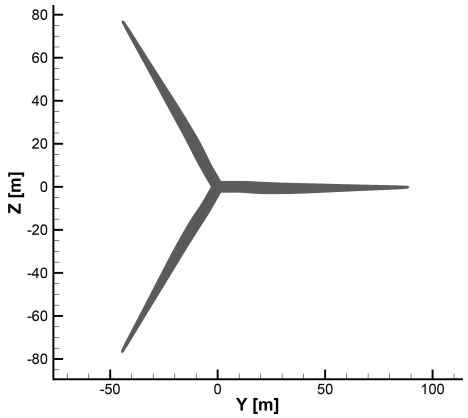



**Figure 4.** Baseline geometry used in the flow solver comparison and as starting point for the optimization. Each blade has a surface mesh with 36 square blocks. Each block has $32 \times 32$ cells, resulting in 110 592 surface mesh cells.

## 4.2 Mesh convergence study

To quantify the mesh dependence for each solver, we compute the integrated metrics – torque and thrust – for the three mesh levels (L0, L1, and L2) and list them in Table 4. The operational condition corresponds to a wind speed of 8 m s$^{-1}$ and rotor speed of 6.69 rpm at zero blade pitch, which is one of the conditions listed in Table A1 in Appendix A. As is evident from the results for meshes L2, L1, and L0 in Table 4, ADflow does not produce a sufficiently mesh-independent solution on mesh L0. This agrees with an earlier mesh convergence study (Dhert et al., 2017, Table 1), where up to 22 million cells were used without reaching convergence. Therefore, we generated a finer mesh with more than 47 million cells called L-1. The L-1 mesh is made exclusively for the present grid convergence study and will not be used in the ensuing optimizations.

Table 4 shows that error reduction from L0 to L-1 for ADflow is much lower (with reductions of about 4 % in thrust and 7 % in torque) than the error reduction from L2 to L1 (15 % and 21 %) or from L1 to L0 (22 % and 41 %). The errors are computed using the Richardson extrapolation values from Fig. 5, which are based on an estimate of the continuum value (in the limit of an infinitely fine mesh), given by

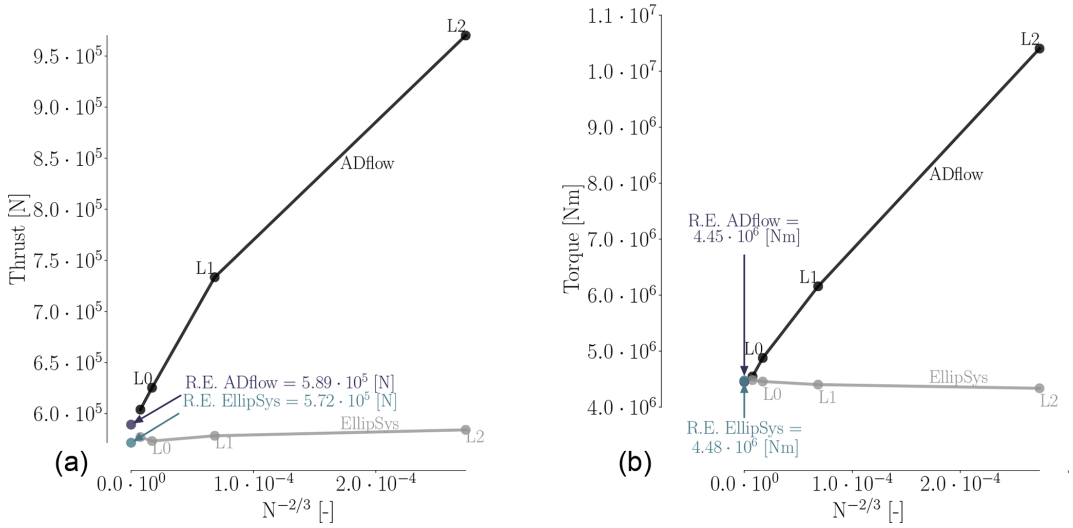



**Figure 5.** Richardson extrapolation (Eq. 7) for the grid convergence study for thrust **(a)** and torque **(b)**. Between the two solvers, the extrapolated continuum values for thrust differ by 3 %, whereas the errors for the torque values vary by less than 0.7 %.

**Table 4.** Mesh convergence study for the compressible solver ADflow and the incompressible solver EllipSys3D. The operational conditions for the convergence study correspond to the $8\,\mathrm{m\,s^{-1}}$ case listed in Table A1. The error percentages are estimated using the Richardson extrapolations from Fig. 5.

| | | ADflow | | | | EllipSys3D | | | |
|---|---|---|---|---|---|---|---|---|---|
| Mesh | Cells (million) | Thrust (kN) | Error (%) | Torque (kNm) | Error (%) | Thrust (kN) | Error (%) | Torque (kNm) | Error (%) |
| L2 | 0.221 | 934 | 58.6 | 10 403 | 134.5 | 584 | 2.1 | 4336 | 3.2 |
| L1 | 1.769 | 733 | 24.4 | 6156 | 38.3 | 578 | 1.0 | 4402 | 1.7 |
| L0 | 14.155 | 625 | 6.1 | 4877 | 9.6 | 573 | 0.2 | 4457 | 0.5 |
| L-1 | 47.776 | 603 | 2.4 | 4547 | 2.2 | 577 | 0.9 | 4471 | 0.2 |
| Extrapolation | $\infty$ | 589 | 0.0 | 4451 | 0.0 | 572 | 0.0 | 4475 | 0.0 |

Roache (1994):

$$f_c \approx f_1 + \frac{f_1 - f_2}{r^2 - 1}, \tag{7}$$

where $f_c$ is the continuum value, $f_1$ and $f_2$ are the values obtained using the L0 and L1 meshes, respectively, and $r$ is the grid refinement ratio.

In Table 4, we can also see that the two solvers tend to converge towards the same thrust and torque continuum values – 0.3 % difference for thrust and 0.7 % difference for torque. Based on the results in this table, we determine that the L0 mesh represents a reasonable compromise between accuracy (less than 10 % error) and speed.

It is clear from Fig. 5 that mesh level L2 is very coarse and yields very different results. As we will demonstrate later, the suggested design trends from such a coarse mesh can sometimes lead to savings in computation time and, other times, lead to completely wrong design trends. Thus, one should use such coarse meshes with care. We report the results obtained with L2 throughout the presented work to substantiate this claim.

There is a slight increase in error for EllipSys3D in the thrust value on the finest mesh level, which is unexpected. It is also surprising that the compressible solver seems to benefit so drastically from an increase in cell count. Recent studies have suggested this can be the case for some compressible solvers (Sørensen et al., 2016). From the expressions for the Prandtl–Glauert compressibility corrections (Glauert, 1928), one would expect that compressible effects could be at play, which agrees with our results. Compressibility effects in wind turbine applications have become increasingly significant as turbine rotor sizes have increased. One of the conclusions from the AVATAR project was that compressibility effects play a role in large wind turbines (Sørensen et al., 2017, p. 9). In the AVATAR project, results from EllipSys3D were compared to results from a compressible CFD code. Here, they studied a case with an inflow speed of $14\,\mathrm{m\,s^{-1}}$ and a Mach number of 0.2457 (Sørensen et al., 2017, Fig. 8), where the obtained $C_p$ curves differed in particular on the

suction side close to the trailing edge (TE). The resulting sectional forces on the blade differed up to 12.9 % (Sørensen et al., 2017, Table 3). The cited Mach number of 0.2457 is within the Mach number range of the present work, where we have Mach numbers close to 0.3 at the tip depending on the inflow speed. The effects of compressibility near the tip region have recently been studied by Sørensen et al. (2018). This work also includes results obtained with EllipSys3D. They find that classical compressibility corrections to incompressible results can be applied in a post-processing step in order to reduce the lift and drag error to within 2.5 % for Mach numbers up to 0.3. The cases studied by Sørensen et al. (2018) include Mach numbers ranging from 0 to 0.5.

This suggests that we could hope to further align the results between ADflow and EllipSys3D in future work by using classical compressibility corrections. Based on the grid convergence study above and in Appendix A, where we provide more details on the flow phenomena and solver performance, we conclude that while there are discrepancies due to different turbulence models, compressibility effects, and numerical scheme order, the trends for the two solvers largely agree.

## 5 Implementation

In this section, we first introduce the design optimization problems for all the CFD and BEM cases we solve. We then explain the FFD parameterization, geometric constraints, and rotor load constraints.

### 5.1 Design optimization problem

We adapt and extend the design optimization problem from the IEA Wind Task 37 case study, which is to maximize the AEP for a range of wind speeds by varying chord and twist, while constraining the increase in thrust and bending moment to be no more than 14 % and 11 %, respectively. Thickness constraints are enforced over the blade to ensure structural integrity. Mathematically, the IEA Wind Task 37 design optimization problem can be expressed as follows:

$$\begin{aligned} &\text{maximize} && \text{AEP} \\ &\text{with respect to} && \text{twist} \\ &&& \text{chord} \\ &\text{subject to} && T \leq 1.14 \cdot T_{\text{init}} \\ &&& M_{\text{bend}} \leq 1.11 \cdot M_{\text{bend}_{\text{init}}}. \end{aligned} \tag{8}$$

The AEP is computed using a specified Weibull distribution (with scale and shape parameters $A = 8$ and $k = 2$, respectively) and the power produced for each wind speed, which is computed from the torque, $Q$, produced by the turbine ($P = \omega \cdot Q$).

We solve four different CFD-based optimizations derived from the problem above:

**Single-point pitch optimization:** This is used to maximize torque on the turbine with respect to blade pitch for a single wind speed (which in this case is equivalent to maximizing AEP). The purpose of this case is to validate the newly implemented rotational terms in the adjoint solver.

**Single-point planform optimization:** This is the same as the IEA Wind Task 37 problem (Eq. 8), except with the objective of maximizing torque for a single wind speed. We solve this problem because it is well suited for comparison with BEM.

**Single-point full shape optimization:** This is the same as the single-point planform optimization but with the addition of blade shape variables. This problem takes advantage of the additional design freedom that is not available for BEM-based models.

**Multipoint full shape optimization:** This is the same as the IEA Wind Task 37 problem (Eq. 8) but with the addition of blade shape variables.

The single-point optimizations are all performed for a wind speed of 8 m s$^{-1}$ and rotational rate of 6.69 rpm at zero blade pitch, which is one of the conditions listed in Table A1 in Appendix A. For the multipoint optimizations, we use the wind speeds 5, 8, and 11 m s$^{-1}$, and the relevant operational conditions can again be found in Table A1 in Appendix A. Furthermore, we use the initial values at 12 m s$^{-1}$ in the thrust and flapwise bending moment constraint for the multipoint optimizations because we know from the solver comparison (Appendix A, Fig. A1) that the maximum thrust occurs at that speed.

In addition to the CFD-based optimizations, we solve two BEM-based optimization problems for comparison with the CFD-based planform optimization:

**BEM1:** identical to the single-point planform optimization.

**BEM2:** identical to the multipoint full shape optimization, except the shape variables are replaced by spanwise thickness distribution variables.

The thickness is handled by interpolating between the predefined airfoil data. While both BEM1 and BEM2 use specified airfoil polar data, BEM2 can change the relative thickness of the airfoils. The airfoils vary from 72 % to 24 % in relative thickness.

### 5.2 Parameterization

The baseline design is shown in Fig. 6a, along with the three FFD boxes used to parameterize the geometry. The FFD boxes have $10 \times 2 \times 9$ control points (shown in black), where 10 is the number of control points from LE to TE, 9 is the number of spanwise sections, and 2 corresponds to the top

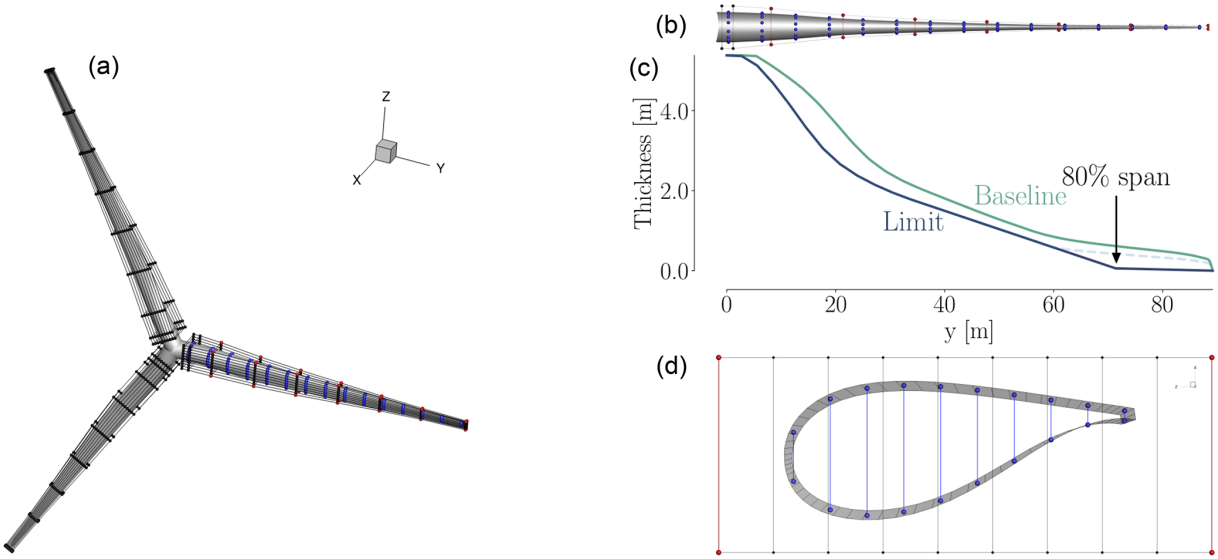



**Figure 6.** Overview of baseline geometry and FFD boxes **(a)**. Each FFD box has nine spanwise sections. Each blade **(b)** has 15 thickness constraints (blue) and seven LE/TE constraints (red). Thickness distributions **(c)** are for the baseline thickness (green) and minimum allowed thickness (purple). Profile section **(d)** at 36 m span shows the shape control points (20), the thickness constraints (10 blue segments) and LE/TE constraints (two red segments). The LE/TE constraints are only relevant for the full shape optimizations.

and bottom of the FFD box. Our approach to deciding on the number of control points is to use the largest number possible to provide maximum freedom in the optimization. However, as the density of control points approaches that of the CFD mesh, numerical issues occur because the physical model no longer resolves the effect of the geometry change. We have found that, as a rule of thumb, we should have no more than one control point for every four CFD mesh points.

The FFD boxes are used to apply the pitch, twist, chord, and shape variables to each blade. Since we want all three blades to have the same pitch and shape, the variables are forced to be the same. Furthermore, the FFD boxes have two fixed sections close to each other at the root to ensure $C^1$ continuity there, while the seven outer sections are free to move and deform the blades. Pitch, $\boldsymbol{x}_{\text{pitch}}$, is achieved by rotating all free FFD sections by the same amount along the reference axis, which is at 35 % of the chord from the LE. Twist, $\boldsymbol{x}_{\text{twist}}$, is achieved by rotating each spanwise section of FFD control points independently. The chord variables, $\boldsymbol{x}_{\text{chord}}$, are achieved by scaling each spanwise section in the chord and thickness direction. Thus, the relative thickness at each section is preserved during the CFD planform optimization. Only for the full shape optimizations, where the shape variables are added, can the relative thickness change. The shape variables, $\boldsymbol{x}_{\text{shape}}$, move each control point independently in the direction perpendicular to the chord to control the airfoil shape.

In Fig. 6, the thickness constraints are highlighted in blue. The thickness constraints in the BEM comparison are only enforced on the inner 80 % of the blade, as detailed in the definition of the IEA case study. This is also visualized in Fig. 6c.

There were a few necessary changes we made to the IEA case study, but only for the full shape optimizations. One such deviation is the dashed segment connected to the thickness limit curve in Fig. 6, which prevents negative cell volumes. Furthermore, there are constraints applied to the LE and TE of the FFD box. The LE/TE constraints (shown in red in Fig. 6) are only implemented for the single-point and multipoint full shape optimizations. These constraints force each pair of points to move exactly the same amount in opposite directions, so that the midpoint in the segment remains stationary. This ensures that the individual FFD control points do not apply skewing twist, since they are meant to control only airfoil profiles. Finally, we mention that the thickness limit is fully imposed only for the fourth thickness constraint (counting from the LE), while the remaining nine constraints in a section are relaxed to not unnecessarily restrict the possible design space.

## 6 Results

The results are split into the four main problems listed in Table 5. First, we perform a single design variable optimization where pitch is varied to maximize the torque (Sect. 6.1). This simple optimization is included to validate the adjoint formulation for rotating frame of reference flows. Second, we perform a planform optimization where chord and twist are varied (Sect. 6.2). This optimization is well suited for comparison with BEM results because the airfoil shapes do not change. The two final optimizations are full shape optimiza-

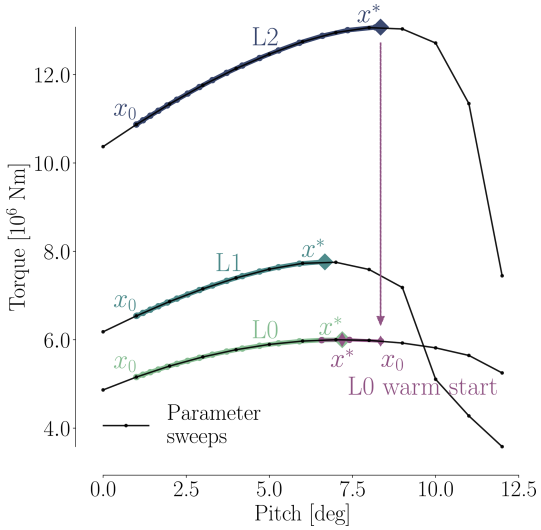



**Figure 7.** Variation of torque with the pitch design variable.

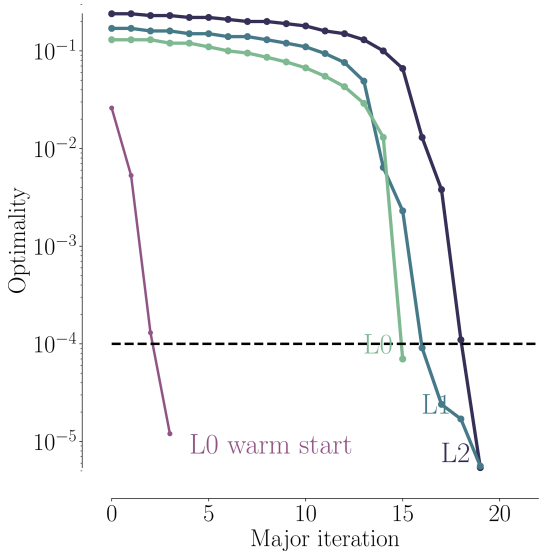



**Figure 8.** Convergence history for the pitch optimizations.

tion problems where all variables, including airfoil shape variables, are allowed to change. First, we solve the problem as a single-point optimization (Sect. 6.3). Then, we solve it as a multipoint optimization (Sect. 6.4).

### 6.1 Pitch optimization

In the pitch optimization, the pitch angle for the seven outer FFD sections on each blade is controlled by a single design variable. The optimization result is an increase in torque of 25.7 %, 26.1 %, and 23.0 % for mesh levels L2, L1, and L0, respectively. Figures 7–9 summarize the optimization history for the three mesh levels.

Figure 7 shows the torque as a function of pitch. Before optimizing, we performed a sweep of CFD evaluations of the torque for the whole range of pitch values, for all three mesh levels. These are represented by black dots in Fig. 7. The thin black lines are linearly interpolated from these points. Although the torque value varies between mesh levels, the trends are consistent, and the maximum torque is achieved around 7° of pitch. The optimization histories for each mesh are shown in color; they start from the initial pitch, $\boldsymbol{x}_0$, and end at the optimal one, $\boldsymbol{x}^*$. The purple line shows the optimization history on the finest (L0) mesh that was obtained by using the result from a coarser mesh (L2) as a starting point. We use this "warm start" technique since coarser meshes are much faster to converge. This technique leads to a reduction of computation time since fewer steps are taken by the optimizer on the finest mesh level. This is seen in Table 6, where only four steps were needed instead of the 16 steps taken in the original optimization. In this case, it reduced the computation time at approximately 50 %. As expected, the result of this warm start optimization is identical to the result of optimizing solely on the finest mesh level. Now that we have introduced (and visualized in Fig. 7) the use of warm starts, we will start using them regularly. This means that an L1 optimization from now on uses the result of an L2 optimization, and an L0 optimization uses the result of an L1 optimization.

As shown in Fig. 8, all optimizations converged to an optimality of at least $10^{-4}$ (black dashed line). Figure 9 shows the merit function, which combines the scaled objective function value and constraint feasibility. The merit function value is equivalent to the scaled objective function value when all constraints are satisfied towards the end of the optimization process. As we can see in Fig. 9, the curves flatten towards the end, and further iterations are not worthwhile because the optimizer reaches the limit of what it can achieve with the provided precision of the function evaluations. The pitch optimizations are summarized in Table 6.

### 6.2 Planform optimization

For the planform optimization, described in Sect. 5.1, both twist and chord are controlled at the seven outer FFD sections along the blade, which results in 14 design variables. The high-fidelity planform optimization results are visualized in Figs. 10–12, which show the final chord and twist distributions as well as the history of the convergence and merit functions.

As we can see in Fig. 10, the optimized shape for the finest mesh level has a large increase in chord towards the root and a decrease in chord towards the tip, just as we would expect for an aerodynamically optimized blade. The optimized chord distribution is reminiscent of the DTU 10 MW turbine's chord distribution from Fig. 2, which was also designed for maximum power. However, the DTU 10 MW root chord is not as high due to a constraint on maximum chord of 6.2 m. Turning to the optimized twist (green curve) in the lower plot in Fig. 10, we see it exhibits a large variation to-

**Table 5.** Overview of optimization problems.

| | | Design variables | | | | | |
|---|---|---|---|---|---|---|---|
| Optimization problem | Objective | Pitch | Twist | Chord | Shape | Total | BEM comparison |
| Single-point pitch | Torque | 1 | | | | 1 | |
| Single-point planform | Torque | | 7 | 7 | | 14 | ✓ |
| Single-point full shape | Torque | | 7 | 7 | 140 | 154 | |
| Multipoint full shape | AEP | | 7 | 7 | 140 | 154 | |

**Table 6.** Pitch design variable optimization. All runs used 216 processors. This means that, for example, the L2 optimization had an actual wall-clock time just under 30 min.

| Mesh | Cells (million) | Twist (°) | Torque ($\boldsymbol{x}^0$) (kNm) | Torque ($\boldsymbol{x}^*$) (kNm) | CPU time (h) | Iterations | Improvement |
|---|---|---|---|---|---|---|---|
| L2 | 0.221 | 8.35 | 10 403 | 13074 | 106.9 | 20 | 25.7 % |
| L1 | 1.769 | 6.67 | 6156 | 7763 | 1004.1 | 19 | 26.1 % |
| L0* | 14.155 | 7.19 | 4877 | 6001 | 6436.3 | 4 | 23.0 % |
| L0 | 14.155 | 7.19 | 4877 | 6001 | 12 734.7 | 16 | 23.0 % |

* Warm start with the L2 optimum, resulting in a total CPU time of 106.9 h + 6436.3 h = 6543.2 h.

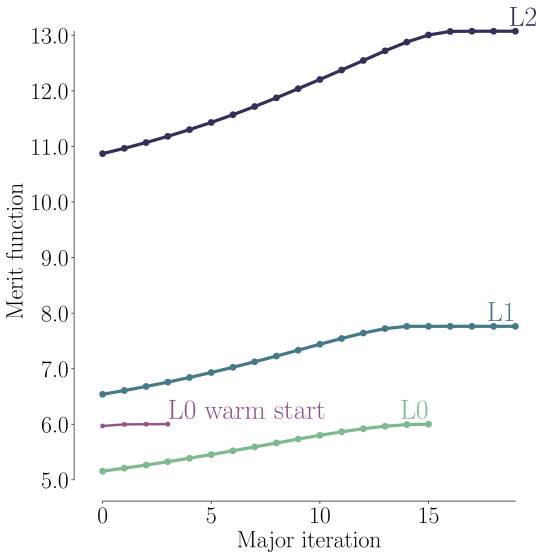



**Figure 9.** Merit function history as a function of steps taken by the optimizer.

wards the tip compared to its baseline. The result is a more aggressive twist distribution.

Comparing the results across mesh levels, there is a much larger spread than for the pitch optimization. The result using the coarsest (L2) mesh is significantly different from the ones obtained with the finer meshes (L1 and L0); therefore, the L2 mesh is too coarse to obtain physically representative results, which is consistent with the mesh convergence study (Table 4). We cannot rule out that, in some cases, the L2 result can be useful to perform a warm start sequence, as shown for the pitch optimization (Fig. 7 and Table 6). However, the planform results certainly show that one should use the L2 mesh with care and not for final results.

Figure 11 shows the convergence history for the three mesh levels. Again, all optimizations were converged to at least $10^{-4}$. In Fig. 12, we see a similar trend to that of the pitch optimization (Fig. 9), where much of the improvement is gained in the first half of the optimization. Thus, an easy way to speed up the design process would be to take an intermediate design. However, one should make sure to check the constraint feasibility, since SQP methods often explore infeasible regions before fully converging. The sharp initial decrease for L1 is due to the (infeasible) warm start from L2. Note that the function is scaled differently for each mesh level to accommodate all the results in one figure.

We now compare our L0 result from the planform optimization to our results from the BEM1 and BEM2 optimization problems. We obtain the BEM results by running HAWTOpt2 (Zahle et al., 2016), which uses HAWCStab2 (Hansen, 2004) as the underlying analysis code. Since this is a comparison between results obtained with completely different models, we do not expect an exact match, but we expect similar trends. As previously mentioned, the CFD planform optimization problem and the BEM1 optimization are completely identical in problem definition, and the relative thickness is fixed in both optimizations. For the BEM2 optimization, the main difference is that it is solved as a multipoint optimization and that the relative thicknesses can be changed through interpolation. We refer to Sect. 5 for further information.

The BEM optimizations are performed with SNOPT. The baseline and optimized chord and twist distributions are

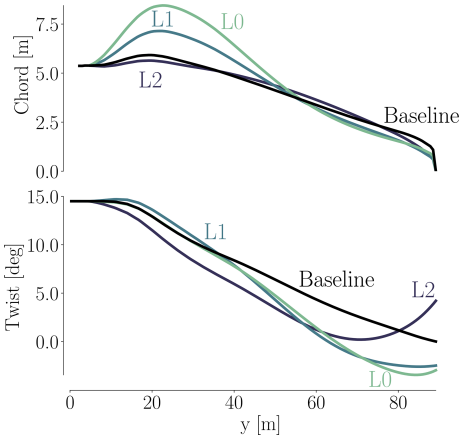



**Figure 10.** Final chord and twist distributions for the CFD-based planform optimizations.

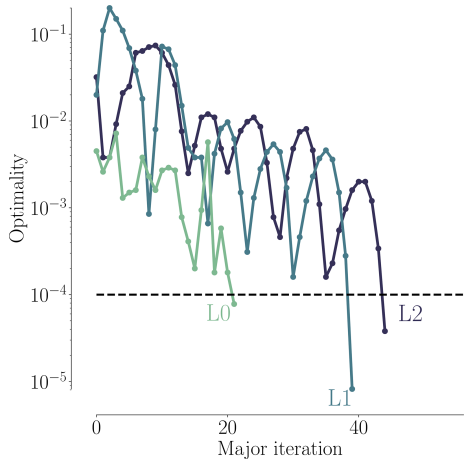



**Figure 11.** Convergence history for all three mesh levels.

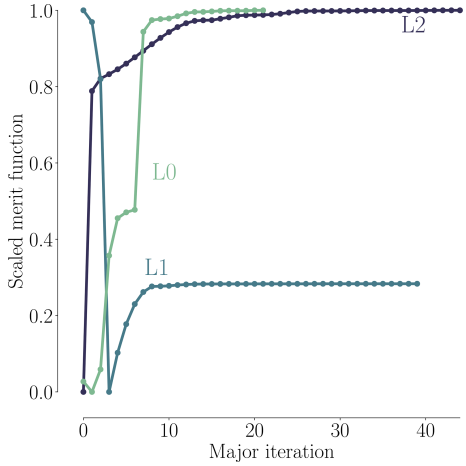



**Figure 12.** Scaled merit function history for the CFD-based planform optimizations.

shown in Fig. 13. Although both chord and twist distributions show clear discrepancies for the final designs, there are several similar traits. When it comes to chord, there is a large difference in maximum chord. BEM1 converges to a 26 % increase, BEM2 converges to a 74 % increase, and the CFD optimization converges to somewhere between these two (43 %). BEM1 is the surprising result of the three, because it seems that the relation between power and thrust is so poor that it makes little sense to increase the chord at the root. This is owed to the fact that BEM1 has fixed relative thickness for all sections. It makes sense that BEM2 can increase the chord further since it can change the relative thickness. Given that our CFD-based planform optimization also has fixed relative thickness, it also makes sense that the BEM2 chord values are larger than those from the CFD-based planform optimization.

Both the BEM1 and the BEM2 results have a steeper, more pronounced increase in chord values, which we suspect our CFD framework could not reproduce due to difference in the parameterization. The two innermost fixed FFD sections ensuring the $C^1$ mesh continuity make such a steep increase in chord impossible so close to the root. As a final comment on the discrepancies at the root, we suspect that BEM profile data for such thick airfoils are far from precise. Besides, the empirical 3-D correction used on said 2-D profile data is also likely to be imprecise. Needless to say, the combination of the two could yield shaky results. To make matters worse, we know from the comparative analysis (Fig. A3) that separation reaches up to about 37 m span, which further complicates the situation. A more uniform picture is seen for the tip region where the chord distributions have converged to a reduced chord, where only minor differences can be seen. In conclusion, the overall trends in optimal chord distribution are mirrored across the BEM and CFD models, and the discrepancies are less pronounced towards the tip.

As for the twist comparison (Fig. 13b), both CFD and BEM results exhibit the overall trend of decreasing the twist relative to the baseline, but the BEM twist is consistently 1–2° lower than the CFD result. This difference is likely due to the different modeling. The CFD parameterization is limited near the root due to the two fixed sections that enforce $C^1$ continuity, so it cannot match the more abrupt change in twist for the BEM result in that region. The BEM2 result exhibits an increase in twist near the root, which is very different from the BEM1 trend. This is because BEM2 is free to control the chord while lowering the relative thickness. Thus, BEM2 uses a large chord increase near the root to optimize the loading, instead of using twist. The planform optimization and BEM comparison are summarized in Table 7.

Using values for torque from Table 7, we can obtain the power coefficient, $C_P$, defined as

$$C_P = \frac{P}{(1/2)\rho V^3 A}, \tag{9}$$

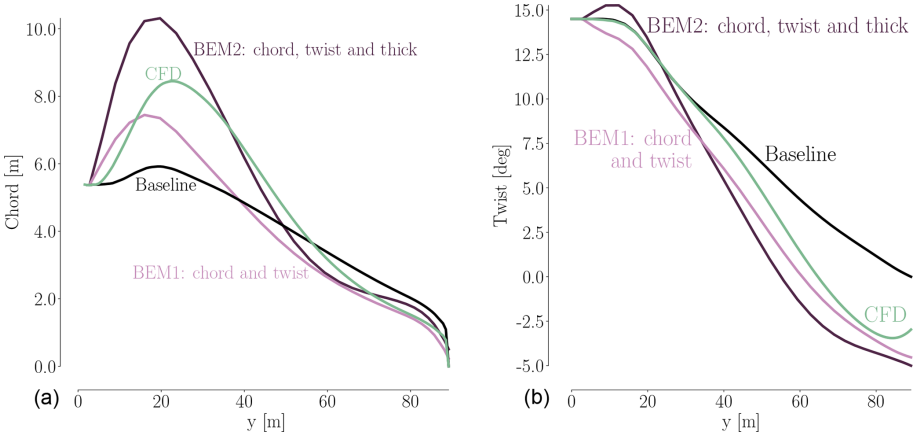



**Figure 13.** Comparison between optimal chord **(a)** and twist **(b)** distributions for the IEA Wind Task 37 case study. The three design optimization problems – (i) single-point planform optimization, (ii) BEM1, and (iii) BEM2 – are further described in Sect. 5.1.

**Table 7.** Planform optimization comparison between CFD and BEM results.

| | | CFD[1] | | | | |
|---|---|---|---|---|---|---|
| Mesh | Cells (million) | Torque ($\boldsymbol{x}^0$) (kNm) | Torque ($\boldsymbol{x}^*$) (kNm) | Improvement | BEM1[1] Improvement | BEM2[2] Improvement |
| L2 | 0.221 | 10 403 | 11 573 | 11.25 % | | |
| L1 | 1.769 | 6156 | 6928 | 12.54% | 8.06 % | 22.46 % |
| L0 | 14.155 | 4877 | 5417 | 11.07 % | | |

[1] Relative improvement in torque. [2] Relative improvement in AEP.

where $P$ is power, $\rho$ is the air density, $V$ is wind speed, and $A$ is the area swept by the rotor. The resulting coefficients are $C_P = [1.04, 0.62, 0.48]$ for mesh levels L2, L1, and L0, respectively. Clearly, the coarser the mesh, the more unphysical the coefficient. The Betz limit for power coefficients ($C_{P_{Betz}} = 0.59$) is violated for L2 and L1, which draws the results from coarse mesh levels into doubt. Judging from the huge spread in these coefficients, it is not surprising that the optimized designs differ greatly across mesh levels.

### 6.3 Single-point shape optimization

We now solve the full shape optimization problem as a single-point optimization. As stated in Sect. 5, this problem is equivalent to optimizing for torque, when only a single wind speed is used. Figure 14 shows convergence (left) and scaled merit function (right) histories for the free-form shape optimizations. Since we typically request an optimization convergence tolerance that is smaller than what is possible for the level of the CFD solver convergence, the optimizer stops before the optimization convergence tolerance is met. Comparing the convergence history to similar plots for the pitch and planform optimizations (Figs. 8 and 11), we see that as the mesh is refined, the optimization is better converged, and the finest mesh level almost meets the requested tolerance (black dashed line). However, the scaled merit function plots (Fig. 14b) do seem flat for L2 and L1 (albeit the latter curve is less smooth), hinting that the merit function could have plateaued.

Table 8 shows the improvement achieved by the optimization. The achieved improvement on the finest mesh (15.89 %) is higher than that of the planform optimization (11.07 %, Table 7), which is expected because this case includes all the planform design optimization variables plus the additional freedom to optimize the airfoil shapes. One should not compare these results to the pitch optimization results since they do not include any thrust constraint. A comparison to the BEM code results is given farther down in Table 9 once the multipoint optimization results have been presented.

We now turn to the shape and pressure ($C_p$) distributions for the baseline and optimized geometries in Fig. 15. The optimized blade increases the chord near the root. This design trend agrees with the planform optimization result.

Comparing the airfoil shapes and corresponding $C_p$ distributions at the bottom of Fig. 15, we can see that the optimization reduced the thickness and slightly increased the camber. The thickness reduction is expected when considering only the aerodynamics with no structural strength constraints. Since we use thickness constraints as a surrogate for structural feasibility, the optimizer exploits this by produc-

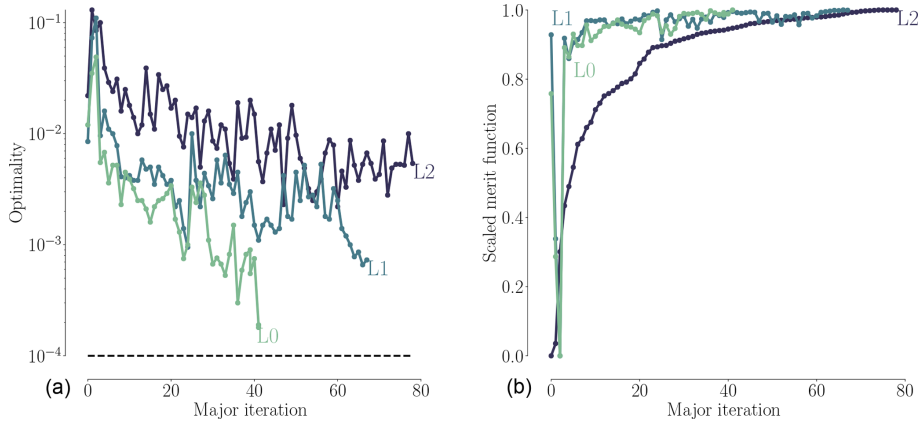



**Figure 14.** Convergence history **(a)** and scaled merit function history **(b)** for the single-point shape optimizations.

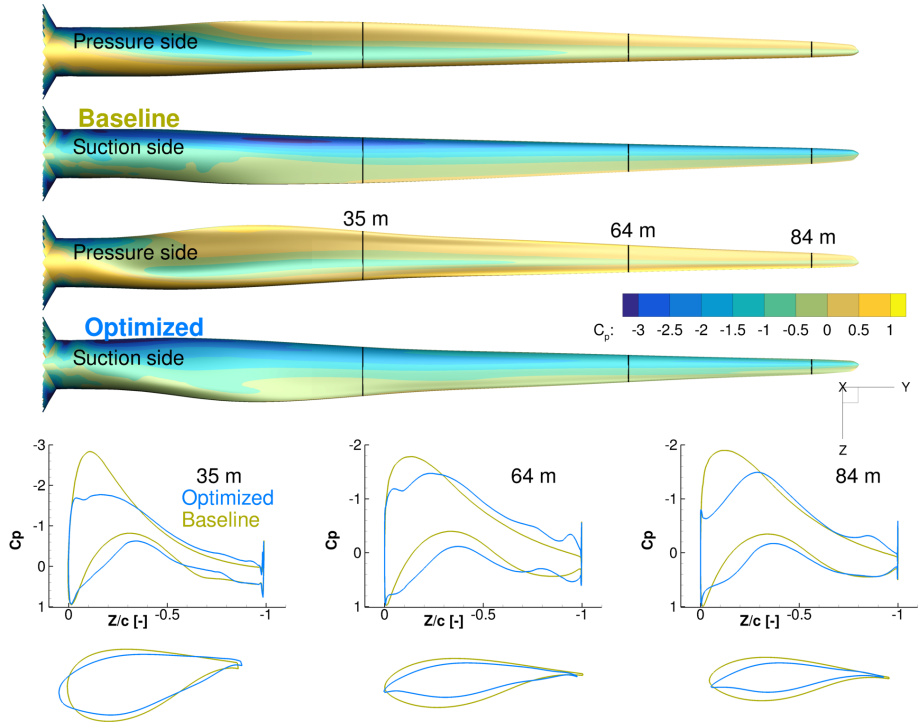



**Figure 15.** Comparison of $C_p$ distributions for the baseline and optimized result from the single-point shape optimization. There is an increase in TE camber, especially at the root, as well as a less pronounced suction peak.

**Table 8.** Overview of single-point optimization results for the operational conditions of the 8 m s$^{-1}$ case listed in Table A1.

| Mesh | Cells (million) | Torque ($\boldsymbol{x}^0$) (kNm) | Torque ($\boldsymbol{x}^*$) (kNm) | Improvement |
|---|---|---|---|---|
| L2 | 0.221 | 10 403 | 12705 | 22.13 % |
| L1 | 1.769 | 6156 | 7373 | 19.77 % |
| L0 | 14.155 | 4877 | 5652 | 15.89 % |

ing the thinnest airfoils that satisfy these constraints. The increased camber, owed to the physical incentive to generate more lift, is consistent with the results of Dhert et al. (2017), but the increase in camber here is more modest because the optimizer can increase the torque by tailoring camber, chord, and twist instead of just camber. The incentive to operate at high lift coefficient is due to the fact that high $C_l/C_d$ is most easily achieved by operating at high $C_l$, especially for airfoils designed assuming a fully turbulent boundary layer.

Another feature of the optimized airfoil shapes is the sharper LE. This is expected due to the fact that we are max-

imizing the performance at a single wind speed. This shape is not robust to changes in wind speed and would perform poorly at other wind speeds. This issue can be addressed by enforcing the LE radius constraints or by considering the performance for multiple wind speeds in the objective function, as we will see in the next section.

### 6.4 Multipoint shape optimization

The motivation for this multipoint optimization is to take a whole range of wind speeds into consideration to achieve a more robust design. We consider both cases for normal power production and also cases leading to peak loading conditions. The design optimization problem and model are the same as those for the single-point optimizations (detailed in Sect. 5.1), except for the objective function. The objective function here is the AEP estimate, which we describe in Sect. 5.1.

When it comes to selecting the wind speeds in a multipoint optimization, it is important to consider speeds that lie outside the ideal operational range. Typically, the rotational rate of the wind turbine rotor is controlled to match a target tip speed ratio, which is the ratio of the tip speed and the wind speed, given by $\lambda = \omega R/V$. As long as the tip speed ratio is the same, the blade angle of attack is the same, and a given design has similar aerodynamic performance. However, for low wind speeds, the rotational speed has a lower bound to avoid tower excitation, and at higher wind speeds, the rotational speed is kept constant, and the turbine starts regulating pitch to maintain rated mechanical power. In our case, the target tip speed ratio is $\lambda = 7.8$, and the rotor speeds corresponding to the minimum and maximum limits are 6.0 and 9.6 rpm, respectively. The variation of rotor speed with wind speed is shown in Fig. 16. There are two reasons we consider wind speeds outside the constant tip speed ratio range. First, the angles of attack are different at these operating points, which should lead to a more balanced design. Additionally, we need to consider the load constraints defined in the optimization case study. For this reason, we choose $5\,\mathrm{m\,s^{-1}}$ as the lower wind speed, and $11\,\mathrm{m\,s^{-1}}$, because this is just below the wind speed at which the rotor reaches rated rotational speed and rated power and thus peak thrust and flapwise moment.

Research has shown that, in reality, the angle of attack varies significantly (more than 4°) over just one rotor revolution (Madsen et al., 2014, Fig. 5). The explanation for this can be found in the complex operating conditions for turbines containing, for example, turbulent inflow and inflow wind shear. To simulate these effects, it would be ideal to add turbulent inflow and transition from steady-state RANS to unsteady RANS. A cheaper way could be a multipoint optimization with a fixed rpm for all turbines operating at slightly different wind speeds. We leave this for future work.

The history of convergence and merit functions are shown in Fig. 17. Just as for the single-point optimization, the selected threshold is not quite met. However, as before, the scaled merit function flattens enough that we determined that the design is close enough to the optimum.

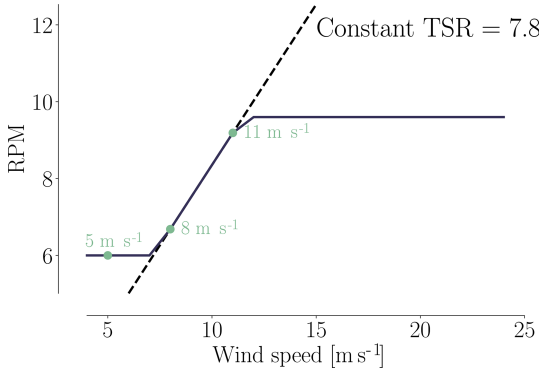



**Figure 16.** Rotational rate schedule with wind speed, showing the ideal constant tip speed rate. The green dots are the wind speeds used in the multipoint optimizations.

We first turn to the airfoil shape to assess the effect of adding geometrical constraints while taking multiple angles of attack into consideration. The airfoil shapes for the multipoint optimizations are compared to the single-point ones in Fig. 18. As we can see, the LE shapes are somewhat improved but still unrealistically sharp. This points towards the necessity of including off-design operational cases resulting in wider ranges of angles of attack, where such a sharp LE would result in deterioration in performance.

To obtain more realistic LE shapes, we added an LE thickness constraint to the optimization problem. The geometric constraint was enforced as a thickness constraint close to the LE. The resulting shapes are shown in Fig. 19, where we compare them to the shape obtained by the multipoint optimization without the LE constraints. While we choose to focus solely on the 2-D profile improvement from single-point to multipoint optimizations, the optimizations are indeed all 3-D rotor optimizations. As we can see, enforcing the geometric constraint results in a more round LE shape that is much more similar to previously published wind turbine airfoil shapes.

Having verified that the resulting shapes for the multipoint full shape optimizations are much improved, we now compare the multipoint optimization results to other optimization results in Table 9. Whereas the single-point BEM1 result (8.06 %) is close to the single-point planform optimization result (11.07 %), the multipoint BEM2 result (22.46 %) is comparable to the multipoint full shape optimizations result (23.76 %) since relative thicknesses can change in both cases. The multipoint result (23.76 %) is somewhat higher than the single-point full shape optimization result (15.89 %), which can be explained by the relaxed thrust constraint for multipoint optimizations. Here, we use the thrust from the $12\,\mathrm{m\,s^{-1}}$ case instead of the $8\,\mathrm{m\,s^{-1}}$ case to define the initial constraint values for thrust and bending moment. Indeed, the

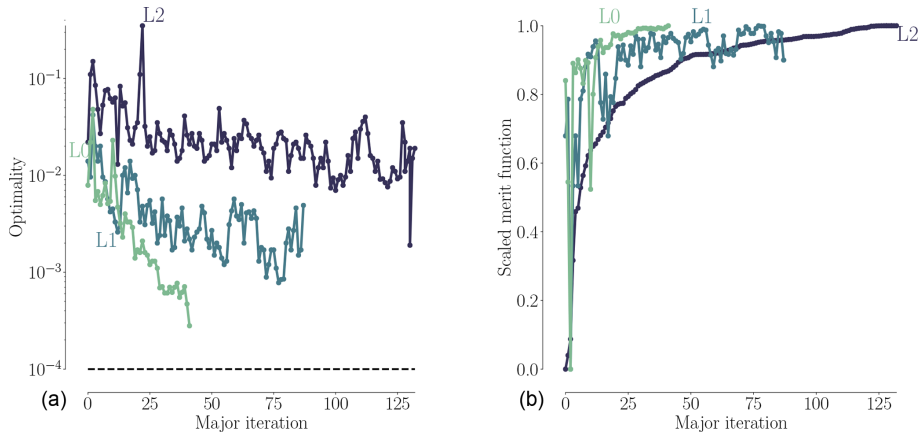



**Figure 17.** History of convergence **(a)** and scaled merit functions **(b)** for the multipoint shape optimizations.


**Figure 18.** Comparison of airfoil profiles obtained from single-point and multipoint optimizations. The profiles are taken from 35, 64, and 84 m spanwise positions.


**Figure 19.** Comparison of airfoil profiles obtained from multipoint optimizations with and without LE geometric constraint.

thrust constraint relaxation results in the constraint not being active at convergence for the CFD-based multipoint full shape optimization, as seen in Table 9.

These results do not show that the industry can necessarily gain a 20 % increase simply by using high-fidelity optimization. Indeed, the amount of improvement depends on the performance of the baseline turbine. Since we study an intentionally poor baseline design, we therefore get a large improvement.

To analyze the optimized designs from single-point and multipoint shape optimizations in more detail, we plot the spanwise forces for both optimized and baseline designs in Fig. 20.

The normal force acts normal to the rotor plane and, integrated over all three blades, yields the rotor thrust. Likewise, the torque can be derived from the driving force by integrating its first moment along all three blades.

For the single-point shape optimization results, we see, as expected, an overall large increase in tangential loading across the blade, and we observe that a high loading is achieved in the root region of the blade as well. This is partially due to the chord increase but also due to the fact that the blade is optimized based on modeling that accounts for the complex three-dimensional flow field, which is particularly dominant in the root region. The thrust constraint and moment constraint were both essential for the design to be industrially relevant for the single-point result: the thrust constraint helped lower the overall thrust values to maintain structural feasibility. The bending moment constraint resulted in a change in the normal force distribution, where the peak moved farther inboard to reduce high loads close to the tip region, as one would expect. Based on the optimization output, we can verify that both constraints are active for the single-point optimization, meaning that thrust and moment have reached the upper limits of 14 % and 11 % increase in thrust and moment, respectively. In the multipoint full shape optimization, the moment constraint is again active at an 11 % increase in bending moment. However, the thrust constraint is only at 11 % and is, as mentioned, not active at convergence due to the relaxed constraint. With these constraints, we could add span as a design variable in future work.

We find the same overall trends for the multipoint results as we did for the single-point optimization. The relaxed thrust constraint for the multipoint optimization results in a rotor with slightly higher loads, which explains why the more robust design from the multipoint optimization outperforms the single-point result.

The multipoint optimization problem presented in this section is functional but should be further improved in the future

**Table 9.** Overview of optimization results. As further detailed in Sect. 5.1, the single-point and multipoint optimizations use the operational conditions for the 8 m s$^{-1}$ case and 5, 8, and 11 m s$^{-1}$ cases, respectively. Operational conditions are listed in Table A1.

| | | | Cells | Constraints (✓ = active) | | |
|---|---|---|---|---|---|---|
| | | Mesh | (million) | Thrust | Bending moment | Improvement |
| Single-point results | BEM1 | – | – | ✓ | ✓ | 8.06 % |
| | Planform | L0 | 14.155 | ✓ | ✓ | 11.07 % |
| | Full shape | L0 | 14.155 | ✓ | ✓ | 15.89 % |
| Multipoint results | BEM2 | – | – | ✓ | ✓ | 22.46 % |
| | Full shape | L0 | 14.155 | – | ✓ | 23.76 % |

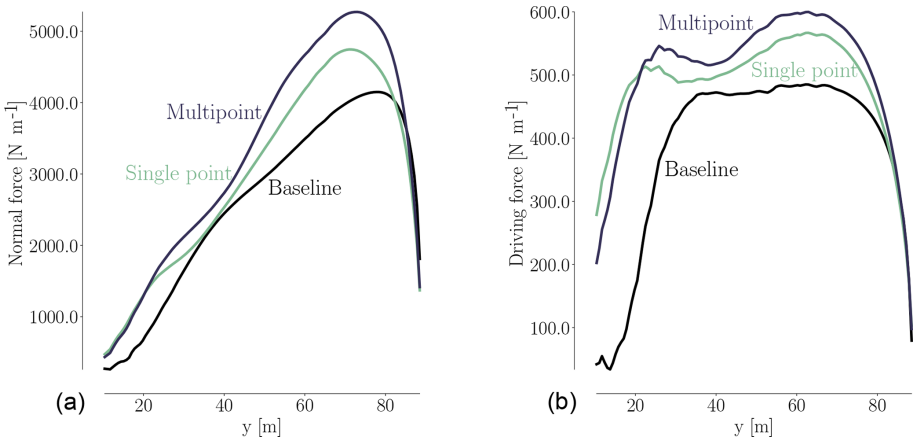



**Figure 20.** Comparison of normal **(a)** and driving **(b)** forces for baseline and optimized designs. The shape optimization increases the normal force, and the peak has also moved further inboard. The driving force is increased considerably both at the root and close to the tip region.

to obtain truly practical wind turbines. First, the laminar to turbulent boundary layer transition should be modeled, since this affects the optimal airfoil shapes. In this work, we just assumed the boundary layer to be turbulent throughout. Second, a wider range of operating points should be considered by, for example, varying the rotation rate or pitch setting for a given wind speed.

## 7 Conclusions

In this work, we presented results from the high-fidelity RANS-based shape optimization of a 10 MW RWT. Based on our literature review of the high-fidelity shape optimization efforts in wind turbine design, we determined that this was a promising area of research.

We compared two state-of-the-art compressible and incompressible CFD solvers to quantify the mesh dependence and discrepancies across different RANS models applied on the same rotor. The results were compatible, and future work involving classical compressibility corrections was identified.

We investigated the advantage of using higher-fidelity models by comparing our optimization results to low-fidelity BEM results from the same case study. We did this through a planform optimization with chord and twist variables, where shape changes were restricted to keep the design case comparable with the BEM-based optimization. The overall design trends were the same across fidelities, with differences due the parameterizations and models. The same overall amount of improvement was observed.

Finally, full shape optimization was performed with respect to twist, chord, and airfoil shape design variables, which raised the number of design variables from 14 to 154. Here, the planform results were further improved with a factor of 1.44. The improvement was enabled by a decrease in relative thickness as well as the novel airfoil shapes.

While further developments are required to obtain truly practical wind turbine blade shapes, this work shows that with the right tools, we can model the entire geometry, including the root, and optimize modern wind turbine rotors at the cost of $\mathcal{O}(10^2)$ CFD evaluations.

**Data availability.** Data are available upon request to the corresponding author.

**Table A1.** Operational conditions for the simulations in the analysis. For the compressible solver (ADflow), we use velocity, density, and temperature as input parameters. ADflow then computes the complete thermodynamic conditions. The density is set to $1.225\,\mathrm{kg\,m^{-3}}$, temperature to 15 °C, and dynamic viscosity to $1.784 \times 10^{-5}\,\mathrm{kg\,m^{-1}\,s^{-1}}$.

| Wind speed ($\mathrm{m\,s^{-1}}$) | rpm (–) | Rotational rate *, $\omega$ ($\mathrm{rad\,s^{-1}}$) | Pitch (°) |
|---|---|---|---|
| 4 | 6.00 | 0.63 | 0 |
| 6 | 6.00 | 0.63 | 0 |
| 8 | 6.69 | 0.70 | 0 |
| 10 | 8.36 | 0.88 | 0 |
| 11 | 9.20 | 0.96 | 0 |
| 12 | 9.60 | 1.01 | 0 |
| 15 | 9.60 | 1.01 | 6.74 |
| 25 | 9.60 | 1.01 | 19.00 |

* Based on a target tip speed ratio of $\gamma = 7.8$, where $6.0 \leq \mathrm{rpm} \leq 9.6$.

## Appendix A: Extended flow solver comparison

The following is a continuation of Sect. 4 to extend the comparison between the flow solvers: EllipSys3D and ADflow.

### A1 Operational conditions

The case study is defined with a cut-in speed of $4\,\mathrm{m\,s^{-1}}$ and a cut-out speed of $25\,\mathrm{m\,s^{-1}}$. Within this range, we use the eight operational conditions defined in Table A1 to compare the solvers.

### A2 Integrated loads

Integrated loads, in the form of thrust and torque, have been computed for each simulation in Table A1 and are visualized in Fig. A1. As seen, the ADflow results are consistently higher than the EllipSys3D results. This trend could partially be accounted for by applying the mentioned Prandtl–Glauert correction to the incompressible computations but is also a result of ADflow results on mesh L0 not being fully mesh independent, as shown in Table 4. As a low-fidelity reference, we have added the integrated loads (in gray) from steady-state BEM results using HAWCStab2. A general agreement between the CFD results and the HAWCStab2 results can be seen, save for the torque value at $25\,\mathrm{m\,s^{-1}}$, which could be corrected with a slight change in pitch setting given in Table A1. Agreement is expected between EllipSys3D and BEM since the airfoil data used in BEM are computed using EllipSys2D.

### A3 Spanwise forces, pressure distribution and flow visualization

Figure A2 shows the spanwise forces and shows that the difference between solvers is more or less spread out over the entire span. Not surprisingly, the ADflow values are consistently higher. We will revisit the distribution of spanwise forces after the optimization to inspect where performance increase occurs on the blade.

Turning to the surface-restricted streamlines in Fig. A3, we first note the rather large amount of separation. Even the pressure side shows a distinct area of separation from 19 to 41 m span. Comparing said area with the pressure side separation for the unperturbed DTU 10 MW rotor in Fig. A4, where only a small separation area at the root is seen, it is clear that the perturbed design we use as a starting point for the optimization seen in Fig. A3 suffers a more poor aerodynamic design owed to the reduced chord distribution and increase in relative thickness. The suction side in Fig. A3 looks more like one would expect, save for the expanded separation area reaching just above 37 m in the spanwise direction. Here, the DTU 10 MW only has separation below the 32 m span, as seen in Fig. A4.

In Fig. A5, we compare the obtained $C_\mathrm{p}$ curves at three spanwise positions: 35, 64, and 84 m (positions marked in red in Fig. A3), where the $C_\mathrm{p}$ distribution is found using the dynamic pressure, and the far-field pressure, $p_\infty$:

$$C_\mathrm{p} = \frac{p - p_\infty}{(1/2)\rho(V_\infty^2 + (r\omega)^2)}. \tag{A1}$$

The slice at 35 m shows the least consistent comparison, which we suspect is due to the large amount of separation present both at suction and pressure side. Given that the solvers use different turbulence models, it would be surprising to find a perfect match at this position. We also note that the pressure side separation results in a $C_\mathrm{p}$ curve with a typical flat, squeezed shape in the 30 % closest to the trailing edge (TE). The $C_\mathrm{p}$ curves for the sections at 64 m span and 84 m span show, in general, a better likeness to one another. Early investigations showed that the chordwise distribution of cells has a distinct impact on the solvers' ability to capture the stagnation point and suction peak. Therefore, we chose a distribution that seemed to have enough cells close to the stagnation point while still having an adequate amount of cells to resolve the TE area. In general, the ADflow suction peaks seem to be more pronounced than those from EllipSys3D. The same can be said for the blunt TE, where the ADflow $C_\mathrm{p}$ curve again has a more pronounced spike.

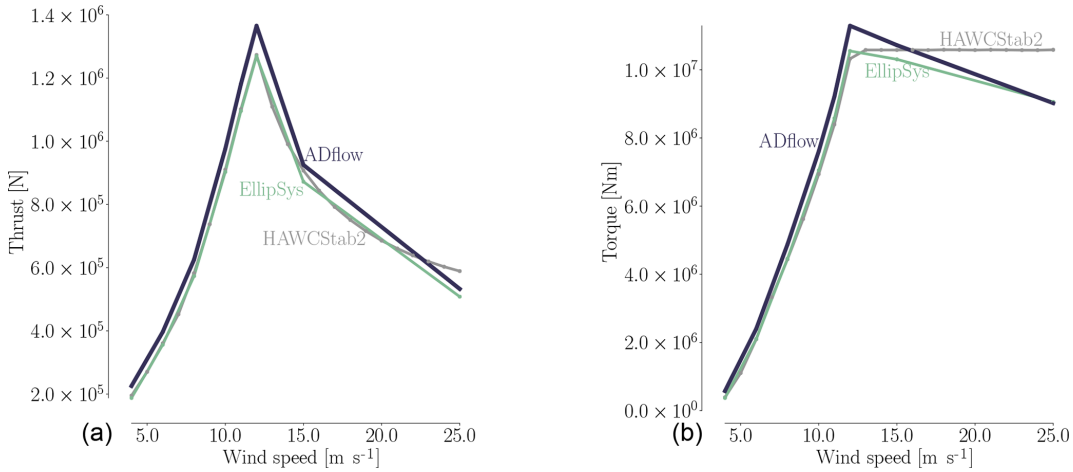



**Figure A1.** Total thrust **(a)** and torque **(b)** as a function of wind speed for the rotor geometry used as the starting point for the optimization computed using mesh L0. As expected, the torque increases rapidly from cut-in speed to the rated speed at 12 m s$^{-1}$, which is also where the thrust peak occurs. From rated to cut out, the torque curve flattens. Here, the pitch setting found with steady-state BEM results using HAWCStab2 (seen in gray) clearly does not result in the CFD solvers tracking rated power accurately due to the model changes. ADflow consistently overshoots the EllipSys results, which is consistent with the trend seen in Table 4. Operational conditions for the eight simulations are given in Table A1.

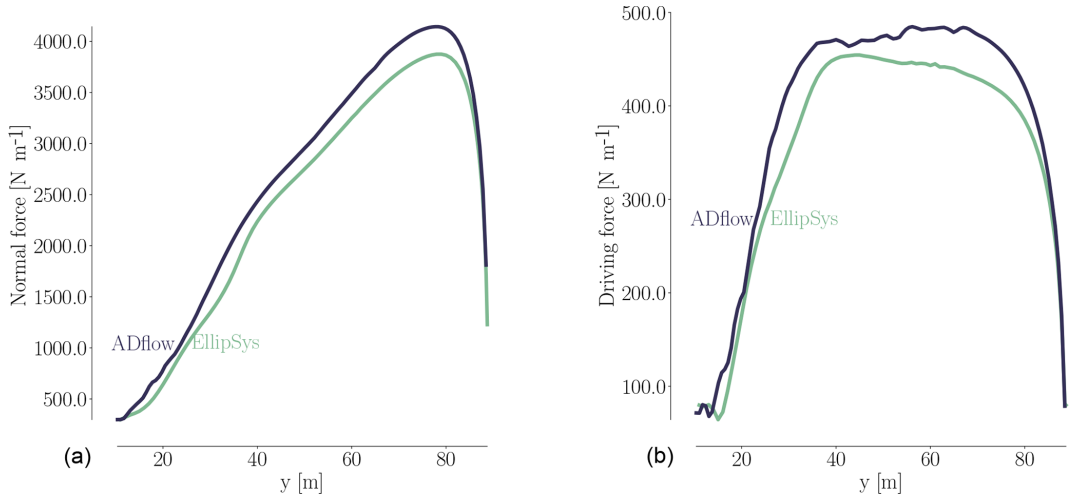



**Figure A2.** Spanwise distribution of the normal force **(a)** and driving force **(b)** for the 8 m s$^{-1}$ case listed in Table A1.

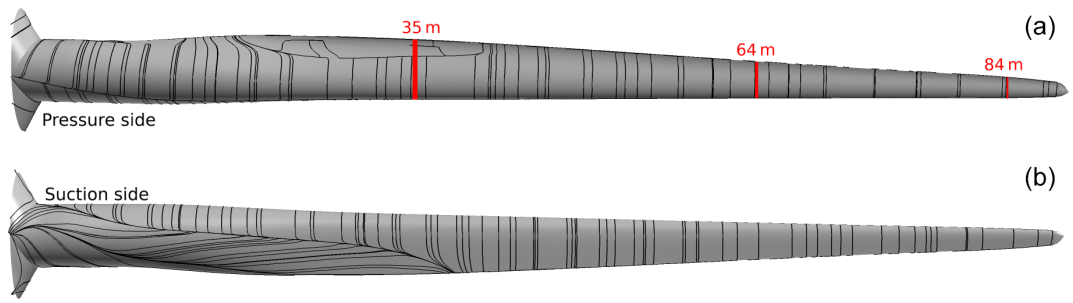



**Figure A3.** Surface-restricted streamlines from the EllipSys solution for a wind speed of 8 m s$^{-1}$, both for the pressure side **(a)** and the suction side **(b)** for the perturbed design we use as a starting point for the optimization. The operational conditions are listed in Table A1.

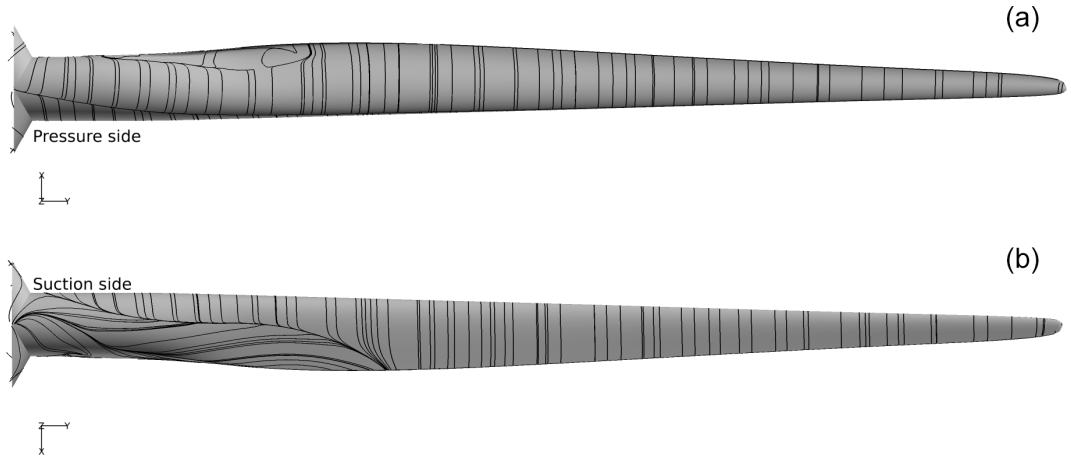



**Figure A4.** Surface-restricted streamlines from the EllipSys solution obtained using the original DTU 10 MW wind turbine geometry for the 8 m s$^{-1}$ case in Table A1 both for the pressure side **(a)** and the suction side **(b)**.

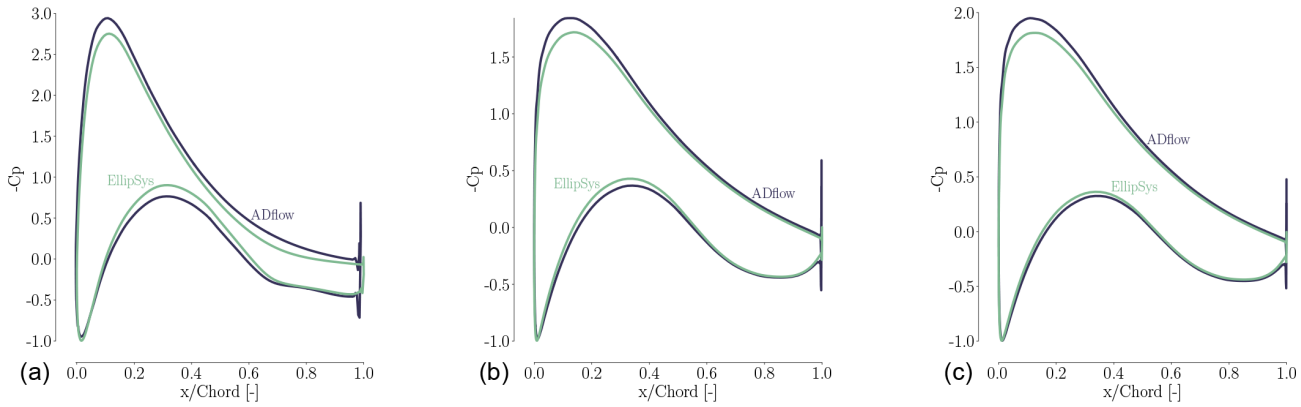



**Figure A5.** $C_p$ curves for 35 m **(a)**, 64 m **(b)**, and 84 m **(c)**.

**Author contributions.** MHAM compiled the literature review, implemented the geometry constraint and the algorithmic differentiation contributions, created the FFD boxes and CFD meshes for the test cases, ran the analysis and optimizations, postprocessed the cases, and wrote the bulk of the paper. FZ assisted in work related to CFD meshes, analysis and optimization, and carried out all BEM optimizations. NNS assisted with CFD expertise within mesh generation, CFD solver settings, and issues related to compressibility effects. JRRAM guided the project, secured access to clusters, and gave technical advice on the entire tool chain at the University of Michigan. All authors took part in writing and editing the paper.

**Competing interests.** The authors declare that they have no conflict of interest.

**Acknowledgements.** We would like to thank the members of the MDO Lab for their support. We thank, in particular, Eirikur Jonsson for assistance with programs in the MACH framework, Nicholas Bons for helping with pyGeo, and Anil Yildirim for his assistance with the ANK solver. We also thank Charles A. Mader for consulting on the AD improvements implemented in the adjoint solver. Finally, researcher Michael McWilliam at DTU kindly assisted with IEA material.

**Review statement.** This paper was edited by Alessandro Bianchini and reviewed by Joseph Saverin and one anonymous referee.

## References

Anderson, E., Bhuiyan, F., Mavriplis, D., and Fertig, R.: Adjoint-Based High-Fidelity Aeroelastic Optimization of Wind Turbine Blade for Load Stress Minimization, 2018 Wind Energy Symposium, AIAA SciTech Forum (AIAA 2018-1241), https://doi.org/10.2514/6.2018-1241, 2018.

Ashuri, T.: Beyond Classical Upscaling: Integrated Aeroservoelastic Design and Optimization of Large Offshore Wind Turbines, PhD thesis, Delft University of Technology, Delft, the Netherlands, 2012.

Ashuri, T., Zaaijer, M. B., Martins, J. R. R. A., van Bussel, G. J. W., and van Kuik, G. A. M.: Multidisciplinary Design Optimization of Offshore Wind Turbines for Minimum Levelized Cost of Energy, Renew. Energ., 68, 893–905, https://doi.org/10.1016/j.renene.2014.02.045, 2014.

Ashuri, T., Martins, J. R. R. A., Zaaijer, M. B., van Kuik, G. A., and van Bussel, G. J.: Aeroservoelastic Design Definition of a 20 MW Common Research Wind Turbine Model, Wind Energy, 19, 2071—2087, https://doi.org/10.1002/we.1970, 2016.

Badreddinne, K., Ali, H., and David, A.: Optimum project for horizontal axis wind turbines "OPHWT", Renew. Energ., 30, 2019–2043, https://doi.org/10.1016/j.renene.2004.12.004, 2005.

Bak, C., Zahle, F., Bitsche, R., Kim, T., Yde, A., Henriksen, L., Natarajan, A., and Hansen, M.: Description of the DTU 10 MW Reference Wind Turbine, Tech. rep., Risø National Laboratory, 2013.

Balay, S., Gropp, W., McInnes, L., and Smith, B.: Efficient management of parallelism in object oriented numerical software libraries, in: Modern Software Tools in Scientific Computing, edited by: Arge, E., Bruaset, A. M., and Langtangen, H. P., Birkhäuser Press, 163–202, 1997.

Balay, S., Abhyankar, S., Adams, M., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Eijkhout, V., Gropp, W., Kaushik, D., Knepley, M., May, D., McInnes, L., Mills, R., Munson, T., Rupp, K., Sanan, P., Smith, B., Zampini, S., Zhang, H., and Zhang, H.: PETSc Web page, available at: http://www.mcs.anl.gov/petsc (last access: 1 March 2019), 2018a.

Balay, S., Abhyankar, S., Adams, M., JBrown, Brune, P., Buschelman, K., Dalcin, L., Eijkhout, V., Gropp, W., Kaushik, D., Knepley, M., May, D., McInnes, L., Mills, R., Munson, T., Rupp, K., PSanan, Smith, B., Zampini, S., Zhang, H., and Zhang, H.: PETSc Users Manual, Tech. Rep. ANL-95/11 - Revision 3.9, Argonne National Laboratory, available at: https://www.mcs.anl.gov/petsc/documentation/index.html (last access: 1 March 2019), 2018b.

Barrett, R. and Ning, A.: Comparison of airfoil precomputational analysis methods for optimization of wind turbine blades, IEEE T. Sustain. Energ., 7, 1081–1088, https://doi.org/10.1109/TSTE.2016.2522381, 2016.

Barrett, R. and Ning, A.: Integrated free-form method for aerostructural optimization of wind turbine blades, Wind Energy, 21, 663–675, https://doi.org/10.1002/we.2186, 2018.

Bechmann, A., Sørensen, N., and Zahle, F.: CFD simulations of the MEXICO rotor, Wind Energy, 14, 677–689, https://doi.org/10.1002/we.450, 2011.

Boorsma, K., Schepers, J., Gomez-Iradi, S., Herraez, I., Lutz, T., Weihing, P., Oggiano, L., Pirrung, G., Madsen, H., Shen, W., Rahimi, H., and Schaffarczyk, P.: Final Report of IEA Wind Task 29 Mexnext (Phase 3), Tech. Rep. ECN-E–18-003, https://www.ecn.nl/publications/ECN-E--18-003, 2018.

Bottasso, C., Campagnolo, F., Croce, A., and Tibaldi, C.: Optimization-based study of bend-twist coupled rotor blades for passive and integrated passive/active load alleviation, Wind Energy, 16, 1149–1166, https://doi.org/10.1002/we.1543, 2013.

Brooks, T. R., Kenway, G. K. W., and Martins, J. R. R. A.: Benchmark Aerostructural Models for the Study of Transonic Aircraft Wings, AIAA J., 56, 2840–2855, https://doi.org/10.2514/1.J056603, 2018.

Burdette, D. and Martins, J. R. R. A.: Design of a Transonic Wing with an Adaptive Morphing Trailing Edge via Aerostructural Optimization, Aerosp. Sci. Technol., 81, 192–203, https://doi.org/10.1016/j.ast.2018.08.004, 2018.

Burdette, D. A. and Martins, J. R. R. A.: Impact of Morphing Trailing Edge on Mission Performance for the Common Research Model, J. Aircraft, 56, 369–384, https://doi.org/10.2514/1.C034967, 2019.

Cavar, D., Réthoré, P.-E., Bechmann, A., Sørensen, N. N., Martinez, B., Zahle, F., Berg, J., and Kelly, M. C.: Comparison of OpenFOAM and EllipSys3D for neutral atmospheric flow over complex terrain, Wind Energ. Sci., 1, 55-70, https://doi.org/10.5194/wes-1-55-2016, 2016.

Chattot, J.: Optimization of wind turbines using helicoidal vortex model, J. Sol. Energ.-T. ASME, 125, 418–424, https://doi.org/10.1115/1.1621675, 2003.

Chen, S., Lyu, Z., Kenway, G. K. W., and Martins, J. R. R. A.: Aerodynamic Shape Optimization of the Common Research Model Wing-Body-Tail Configuration, J. Aircraft, 53, 276–293, https://doi.org/10.2514/1.C033328, 2016.

Dhert, T., Ashuri, T., and Martins, J. R. R. A.: Aerodynamic Shape Optimization of Wind Turbine Blades Using a Reynolds-Averaged Navier–Stokes Model and an Adjoint Method, Wind Energy, 20, 909–926, https://doi.org/10.1002/we.2070, 2017.

Drela, M.: XFOIL: An Analysis and Design System for Low Reynolds Number Airfoils, in: Low Reynolds Number Aerodynamics, edited by: Mueller, T. J., Springer Berlin Heidelberg, Berlin, Heidelberg, 1–12, 1989.

Drela, M. and Giles, M.: Viscous-inviscid analysis of transonic and low Reynolds number airfoils, AIAA Paper, 131–140, 1986.

Economon, T., Palacios, F., and Alonso, J.: A viscous continuous adjoint approach for the design of rotating engineering applications, 21st AIAA Computational Fluid Dynamics Conference, San Diego, CA, 24–27 June 2013.

Edward, L., Eric, C., and Eric, B.: A fast mesh deformation method using explicit interpolation, J. Comput. Phys., 231, 586–601, https://doi.org/10.1016/j.jcp.2011.09.021, 2012.

Elfarra, M., Sezer-Uzol, N., and Akmandor, I.: NREL VI rotor blade: numerical investigation and winglet design and optimization using CFD, Wind Energy, 17, 605–626, https://doi.org/10.1002/we.1593, 2014.

Fuglsang, P. and Madsen, H.: Optimization method for wind turbine rotors, J. Wind Eng. Ind. Aerod., 80, 191–206, https://doi.org/10.1016/S0167-6105(98)00191-3, 1999.

Fuglsang, P. and Thomsen, K.: Site specific design optimization of wind turbines, A Collection of the 1998 Asme Wind Energy Symposium Technical Papers, 294–303, 1998.

Fuglsang, P. and Thomsen, K.: Site-specific design optimization of 1.5-2.0 MW wind turbines, J. Sol. Energ.-T ASME, 123, 296–303, https://doi.org/10.1115/1.1404433, 2001.

Fuglsang, P., Bak, C., Schepers, J., Bulder, B., Cockerill, T., Claiden, P., Olesen, A., and Rossen, R.: Site-specific design optimization of wind turbines, Wind Energy, 5, 261–279, https://doi.org/10.1002/we.61, 2002.

Fuglsang, P., Bak, C., Gaunaa, M., and Antoniou, I.: Design and verification of the Risø-B1 airfoil family for wind turbines, J. Sol. Energ.-T ASME, 126, 1002–1010, https://doi.org/10.1115/1.1766024, 2004.

Garg, N., Kenway, G. K. W., Martins, J. R. R. A., and Young, Y. L.: High-fidelity Multipoint Hydrostructural Optimization of a 3-D Hydrofoil, J. Fluid Struct., 71, 15–39, https://doi.org/10.1016/j.jfluidstructs.2017.02.001, 2017.

Garg, N., Pearce, B. W., Brandner, P. A., Phillips, A. W., Martins, J. R. R. A., and Young, Y. L.: Experimental Investigation of a Hydrofoil Designed via Hydrostructural Optimization, J. Fluid Struct., 84, 243–262, https://doi.org/10.1016/j.jfluidstructs.2018.10.010, 2019.

Garrel, A.: Development of a wind turbine aerodynamics simulation module, Tech. Rep. ECN-C–03-079, available at: https://www.ecn.nl/publications/E/2003/ECN-C--03-079 (last access: 18 March 2019), 2003.

Gill, P. E., Murray, W., and Saunders, M. A.: SNOPT: An SQP algorithm for large-scale constrained optimization, SIAM Journal of Optimization, 12, 979–1006, https://doi.org/10.1137/S1052623499350013, 2002.

Gill, P. E., Murray, W., and Saunders, M. A.: User's Guide for SNOPT Version 7: Software for Large-Scale Nonlinear Programming, Systems Optimization Laboratory, Stanford University, California, 94305-4023, Technical Report, 2007.

Glauert, H.: The Effect of Compressibility on the Lift of an Aerofoil, P. Roy. Soc. A-Math. Phy., 118, 113–119, https://doi.org/10.1098/rspa.1928.0039, 1928.

Grasso, F.: Usage of numerical optimization in wind turbine airfoil design, J. Aircraft, 48, 248–255, https://doi.org/10.2514/1.C031089, 2011.

Grasso, F.: Hybrid optimization for wind turbine thick airfoils, 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, AIAA 2012-1354, https://doi.org/10.2514/6.2012-1354, 2012.

Grasso, F.: ECN-G1-21 Airfoil: Design and wind-tunnel testing, J. Aircraft, 53, 1478–1848, https://doi.org/10.2514/1.C033089, 2016.

Gray, J. S., Hwang, J. T., Martins, J. R. R. A., Moore, K. T., and Naylor, B. A.: OpenMDAO: An open-source framework for multidisciplinary design, analysis, and optimization, Struct. Multidiscip. O., 1–30 https://doi.org/10.1007/s00158-019-02211-z, in press, 2019.

Griewank, A.: Evaluating Derivatives: Principles and techniques of algorithmic differentiation, SIAM, Philadelphia, ISBN 0-89871-451-6, 2000.

Hansen, M. H.: Aeroelastic stability analysis of wind turbines using an eigenvalue approach, Wind Energy, 7, 133–143, https://doi.org/10.1002/we.116, 2004.

He, P., Mader, C. A., Martins, J. R. R. A., and Maki, K. J.: An Aerodynamic Design Optimization Framework Using a Discrete Adjoint Approach with OpenFOAM, Comput. Fluids, 168, 285–303, https://doi.org/10.1016/j.compfluid.2018.04.012, 2018.

Heinz, J., Sørensen, N., and Zahle, F.: Fluid-structure interaction computations for geometrically resolved rotor simulations using CFD, Wind Energy, 19, 2205–2221, https://doi.org/10.1002/we.1976, 2016.

Hwang, J. T. and Martins, J. R. R. A.: A computational architecture for coupling heterogeneous numerical models and computing coupled derivatives, ACM T. Math. Software, 44, 37, https://doi.org/10.1145/3182393, 2018.

Johansen, J., Madsen, H., Gaunaa, M., Bak, C., and Sørensen, N.: Design of a wind turbine rotor for maximum aerodynamic efficiency, Wind Energy, 12, 261–273, https://doi.org/10.1002/we.292, 2009.

Kenway, G. K., Kennedy, G. J., and Martins, J. R. R. A.: A CAD-Free Approach to High-Fidelity Aerostructural Optimization, in: Proceedings of the 13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference, AIAA 2010-9231, Fort Worth, TX, https://doi.org/10.2514/6.2010-9231, 2010.

Kenway, G. K. W. and Martins, J. R. R. A.: Aerostructural Shape Optimization of Wind Turbine Blades Considering Site-Specific Winds, in: Proceedings of the 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Victoria, BC, 10–12 September 2008, AIAA 2008-6025, https://doi.org/10.2514/6.2008-6025, 2008.

Kenway, G. K. W. and Martins, J. R. R. A.: Multipoint High-Fidelity Aerostructural Optimization of a Transport Aircraft Configuration, J. Aircraft, 51, 144–160, https://doi.org/10.2514/1.C032150, 2014.

Kenway, G. K. W., Kennedy, G. J., and Martins, J. R. R. A.: Scalable Parallel Approach for High-Fidelity Steady-State Aeroelastic Analysis and Derivative Computations, AIAA Journal, 52, 935–951, https://doi.org/10.2514/1.J052255, 2014.

Kenway, G. K. W., Secco, N., Martins, J. R. R. A., Mishra, A., and Duraisamy, K.: An Efficient Parallel Overset Method for Aerodynamic Shape Optimization, in: Proceedings of the 58th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, AIAA SciTech Forum, Grapevine, TX, 9–13 January 2017, https://doi.org/10.2514/6.2017-0357, 2017.

Khayatzadeh, P. and Nadarajah, S.: Aerodynamic Shape Optimization via Discrete Viscous Adjoint Equations for the $k-\omega$ SST Turbulence and $\gamma-\widetilde{Re_\theta}$ Transition Models, 49th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, Orlando, Florida, 4–7 January 2011, https://doi.org/10.2514/6.2011-1247, 2011.

Kwon, H., You, J., and Kwon, O.: Enhancement of wind turbine aerodynamic performance by a numerical optimization technique, J. Mech. Sci. Technol., 26, 455–462, https://doi.org/10.1007/s12206-011-1035-2, 2012.

Lambe, A. B. and Martins, J. R. R. A.: Extensions to the Design Structure Matrix for the Description of Multidisciplinary Design, Analysis, and Optimization Processes, Struct. Multidiscip. O., 46, 273–284, https://doi.org/10.1007/s00158-012-0763-y, 2012.

Langtry, R., Menter, F., Likki, S., Suzen, Y., Huang, P., and Vølker, S.: A correlation-based transition model using local variables part II – Test cases and industrial applications, Proceedings of the Asme Turbo Expo 2004, 4, 69–79, 2004.

Lawton, S. and Crawford, C.: Investigation and Optimization of Blade Tip Winglets Using an Implicit Free Wake Vortex Method, J. Phys. Conf. Ser., 524, 012033, https://doi.org/10.1088/1742-6596/524/1/012033, 2014.

Liang, C. and Li, H.: Effects of optimized airfoil on vertical axis wind turbine aerodynamic performance, J. Braz. Soc. Mech. Sci., 40, 88, https://doi.org/10.1007/s40430-017-0926-2, 2018.

Lyu, Z., Kenway, G. K., Paige, C., and Martins, J. R. R. A.: Automatic Differentiation Adjoint of the Reynolds-Averaged Navier–Stokes Equations with a Turbulence Model, in: 21st AIAA Computational Fluid Dynamics Conference, San Diego, CA, 24–27 June 2013, https://doi.org/10.2514/6.2013-2581, 2013.

Lyu, Z., Xu, Z., and Martins, J. R. R. A.: Benchmarking Optimization Algorithms for Wing Aerodynamic Design Optimization, in: Proceedings of the 8th International Conference on Computational Fluid Dynamics, Chengdu, Sichuan, China, iCCFD8-2014-0203, 2014.

Mader, C. A. and Martins, J. R. R. A.: Computation of Aircraft Stability Derivatives Using an Automatic Differentiation Adjoint Approach, AIAA J., 49, 2737–2750, https://doi.org/10.2514/1.J051147, 2011.

Mader, C. A., Martins, J. R. R. A., Alonso, J. J., and van der Weide, E.: ADjoint: An Approach for the Rapid Development of Discrete Adjoint Solvers, AIAA J., 46, 863–873, https://doi.org/10.2514/1.29123, 2008.

Madsen, H., Bertagnolio, F., Fischer, A., and Bak, C.: Correlation of amplitude modulation to inflow characteristics, Proceedings of 43rd International Congress on Noise Control Engineering, 2014.

Madsen, H. A.: Forskning i aeroelasticitet EFP-2001, 2002.

Maniaci, D.: An investigation of WT_Perf convergence issues, 49th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, https://doi.org/10.2514/6.2011-150, 2011.

Martins, J. R. R. A. and Hwang, J. T.: Review and Unification of Methods for Computing Derivatives of Multidisciplinary Computational Models, AIAA J., 51, 2582–2599, https://doi.org/10.2514/1.J052184, 2013.

Martins, J. R. R. A. and Lambe, A. B.: Multidisciplinary Design Optimization: A Survey of Architectures, AIAA J., 51, 2049–2075, https://doi.org/10.2514/1.J051895, 2013.

Martins, J. R. R. A., Sturdza, P., and Alonso, J. J.: The Complex-Step Derivative Approximation, ACM T. Math. Software, 29, 245–262, https://doi.org/10.1145/838250.838251, 2003.

Martins, J. R. R. A., Alonso, J. J., and Reuther, J. J.: A Coupled-Adjoint Sensitivity Analysis Method for High-Fidelity Aero-Structural Design, Optim. Eng., 6, 33–62, https://doi.org/10.1023/B:OPTE.0000048536.47956.62, 2005.

McWilliam, M.: Towards Multidisciplinary Design Optimization Capability of Horizontal Axis Wind Turbines, PhD, University of Victoria, Department of Mechanical Engineering, available at: https://dspace.library.uvic.ca//handle/1828/6441 (last access: 18 March 2019), 2015.

Menter, F.: Zonal Two Equation Kappa-Omega Turbulence Models for Aerodynamic Flows, 24th Fluid dynamics conference, Orlando, Florida, 6–9 July 1993, AIAA93-2906, 1993.

Menter, F., Langtry, R., Likki, S., Suzen, Y., Huang, P., and Völker, S.: A correlation-based transition model using local variables part I - Model formulation, Proceedings of the Asme Turbo Expo 2004, 4, 57–67, 2004.

Michelsen, J.: Basis3D – a Platform for Development of Multiblock PDE Solvers, Tech. Rep. AFM 92-05, Department of Fluid Mechanics, Technical University of Denmark, 1992.

Michelsen, J.: Block structured multigrid solution of 2D and 3D elliptic PDE's, Tech. Rep. AFM 94-06, Department of Fluid Mechanics, Technical University of Denmark, 1994.

Ning, A.: A simple solution method for the blade element momentum equations with guaranteed convergence, Wind Energy, 17, 1327–1345, https://doi.org/10.1002/we.1636, 2014.

Ning, A. and Petch, D.: Integrated design of downwind land-based wind turbines using analytic gradients, Wind Energy, 19, 2137–2152, https://doi.org/10.1002/we.1972, 2016.

Ning, A., Damiani, R., and Moriarty, P.: Objectives and constraints for wind turbine optimization, J. Sol. Energ.-T. ASME, 136, 041010, https://doi.org/10.1115/1.4027693, 2014.

Palacios, F., Colonno, M., Aranake, A., Campos, A., Copeland, S., Economon, T., Lonkar, A., Lukaczyk, T., Taylor, T., and Alonso, J.: Stanford University Unstructured (SU2): An open-source integrated computational environment for multi-physics simulation and design, 51st AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition 2013, Grapevine, Texas, 7–10 January 2013.

Peter, J. E. V. and Dwight, R. P.: Numerical Sensitivity Analysis for Aerodynamic Optimization: A Survey of Approaches, Comput. Fluids, 39, 373–391, https://doi.org/10.1016/j.compfluid.2009.09.013, 2010.

Ramos García, N., Sørensen, J., and Shen, W.: Three-dimensional viscous-inviscid coupling method for wind turbine computations, Wind Energy, 19, 67–93, https://doi.org/10.1002/we.1821, 2016.

Reggio, M., Villalpando, F., and Ilinca, A.: Assessment of turbulence models for flow simulation around a wind turbine airfoil, Model. Simul. Eng., 2011, 714146, https://doi.org/10.1155/2011/714146, 2011.

Ribeiro, A., Awruch, A., and Gomes, H.: An airfoil optimization technique for wind turbines, Appl. Math. Model., 36, 4898–4907, https://doi.org/10.1016/j.apm.2011.12.026, 2012.

Ritlop, R. and Nadarajah, S.: Design of wind turbine profiles via a preconditioned adjoint-based aerodynamic shape optimization, 47th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, Orlando, Florida, 5–8 January 2009, aIAA 2009-1547, 2009.

Roache, P.: Perspective – A method for uniform reporting of grid refinement studies, J. Fluid Eng.-T ASME, 116, 405–413, https://doi.org/10.1115/1.2910291, 1994.

Saad, Y. and Schultz, M.: GMRES – A generalized minimal residual algorithm for solving nonsymmetric linear-systems, SIAM J. Sci. Stat. Comp., 7, 856–869, https://doi.org/10.1137/0907058, 1986.

Schepers, J., Boorsma, K., Cho, T., Gomez-Iradi, S., Schaffarczyk, P., Jeromin, A., Shen, W., Lutz, T., Meister, K., Stoevesandt, B., Schreck, S., Micallef, D., Pereira, R., Sant, T., Aagaard Madsen, H., and Sørensen, N.: Analysis of Mexico wind tunnel measurements: Final report of IEA Task 29, Mexnext (Phase 1), Energy Research Centre of the Netherlands (ECN), 2012.

Schramm, M., Stoevesandt, B., and Peinke, J.: Adjoint optimization of 2D-airfoils in incompressible flows, 11th World Congress on Computational Mechanics, Wccm 2014, 5th European Conference on Computational Mechanics, Eccm 2014 and 6th European Conference on Computational Fluid Dynamics, ECFD 2014, 20–25 July 2014, Barcelona, Spain, 6200–6211, 2014.

Schramm, M., Stoevesandt, B., and Peinke, J.: Lift optimization of airfoils using the adjoint approach, European Wind Energy Association Annual Conference and Exhibition 2015, EWEA 2015 – Scientific Proceedings, Paris, France, 17–20 November 2015.

Schramm, M., Stoevesandt, B., and Peinke, J.: Simulation and Optimization of an Airfoil with Leading Edge Slat, J Phys. Conf. Ser., 753, 022052, https://doi.org/10.1088/1742-6596/753/2/022052, 2016.

Schramm, M., Stoevesandt, B., and Peinke, J.: Optimization of airfoils using the adjoint approach and the influence of adjoint turbulent viscosity, Computation, 6, 5, https://doi.org/10.3390/computation6010005, 2018.

Sederberg, T. and Parry, S.: Free-form deformation of solid geometric models, Comp. Graph., 20, 151–160, https://doi.org/10.1145/15886.15903, 1986.

Sessarego, M., Shen, W., Sørensen, J., and Ramos García, N.: Design of Large Wind Turbines using Fluid-Structure Coupling Technique, DTU Wind Energy, Denmark, 2016.

Simms, D., Schreck, S., Hand, M., and Fingersh, L.: NREL Unsteady Aerodynamics Experiment in the NASA-Ames Wind Tunnel: A Comparison of Predictions to Measurements, Technical Report, NREL/TP-500-29494, https://doi.org/10.2172/783409, 2001.

Sørensen, N.: General purpose flow solver applied to flow over hills, PhD thesis, Risø National Laboratory, Roskilde, Denmark, 1995.

Sørensen, N.: HypGrid2D. A 2-d mesh generator, Tech. Rep. RisøR-1035(EN), Risø National Laboratory, 1998.

Sørensen, N.: CFD modelling of laminar-turbulent transition for airfoils and rotors using the gamma-(Re)over-tilde (theta) model, Wind Energy, 12, 715–733, https://doi.org/10.1002/we.325, 2009.

Sørensen, N. and Schreck, S.: Transitional DDES computations of the NREL Phase-VI rotor in axial flow conditions, J. Phys. Conf. Ser., 555, 012096, https://doi.org/10.1088/1742-6596/555/1/012096, 2014.

Sørensen, N., Michelsen, J., and Schreck, S.: Navier-Stokes predictions of the NREL phase VI rotor in the NASA Ames 80-by-120 wind tunnel, 2002 Asme Wind Energy Symposium, 40. AIAA Aerospace Sciences Meeting and Exhibit, 94–105, 2002.

Sørensen, N., Méndez, B., Muñoz, A., Sieros, G., Jost, E., Lutz, T., Papadakis, G., Voutsinas, S., Barakos, G., Colonia, S., Baldacchino, D., Baptista, C., and Ferreira, C.: CFD code comparison for 2D airfoil flows, J. Phys. Conf. Ser., 753, 082019, https://doi.org/10.1088/1742-6596/753/8/082019, 2016.

Sørensen, N., Gonzalez-Salcedo, A., Martin, R., Jost, E., Pirrung, G., Rahimi, H., Schepers, G., Sieros, G., Madsen, H., Boorsma, K., Garcia, N., Voutsinas, S., and Lutz, T.: Aerodynamics of large rotors WP 2, Deliverable 2.8; Engineering models for complex inflow situations, Tech. rep., ECN, Faruenhofer IWES, CENER, NTUA, DTU, US, UO, 2017.

Sørensen, N., Bertagnolio, F., Jost, E., and Lutz, T.: Aerodynamic effects of compressibility for wind turbines at high tip speeds, J. Phys. Conf. Ser., 1037, 022003, https://doi.org/10.1088/1742-6596/1037/2/022003, 2018.

Spalart, P. and Allmaras, S.: A one-equation turbulence model for aerodynamic flows, Recherche Aerospatiale, 5–21, 1994.

Tsiakas, K., Trompoukis, X., Asouti, V., and Giannakoglou, K.: Shape Optimization of Wind Turbine Blades Using the Continuous Adjoint Method and Volumetric NURBS on a GPU Cluster, in: Advances in Evolutionary and Deterministic Methods for Design, Optimization and Control in Engineering and Sciences, Springer, Cham, 48, 131–144, https://doi.org/10.1007/978-3-319-89988-6_8, 2018.

Vorspel, L., Herráez, I., Peinke, J., and Stoevesandt, B.: Towards the optimization of wind turbine rotor blades by means of computational fluid dynamics and the adjoint approach, 34th AIAA Applied Aerodynamics Conference, AIAA AVIATION Forum, Washington, DC, 13–17 June 2016, AIAA 2016-3728, https://doi.org/10.2514/6.2016-3728, 2016.

Vorspel, L., Schramm, M., Stoevesandt, B., Brunold, L., and Bünner, M.: A benchmark study on the efficiency of unconstrained optimization algorithms in 2D-aerodynamic shape design, Cogent Engineering, 4, 1354509, https://doi.org/10.1080/23311916.2017.1354509, 2017.

Vorspel, L., Stoevesandt, B., and Peinke, J.: Optimize rotating wind energy rotor blades using the adjoint approach, Appl. Sci., 8, 1112, https://doi.org/10.3390/app8071112, 2018.

Voutsinas, S. G.: Vortex methods in aeronautics: how to make things work, Int. J. Comput. Fluid D., 20, 3–18, https://doi.org/10.1080/10618560600566059, 2006.

Vucina, D., Marinic-Kragic, I., and Milas, Z.: Numerical models for robust shape optimization of wind turbine blades, Renew. Energ., 87, 849–862, https://doi.org/10.1016/j.renene.2015.10.040, 2016.

Weide, E., Kalitzin, G., Schluter, J., and Alonso, J.: Unsteady turbomachinery computations using massively parallel platforms, 44th AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada, 9–12 January 2006, https://doi.org/10.2514/6.2006-421, 2006.

Weller, H., Tabor, G., Jasak, H., and Fureby, C.: A tensorial approach to computational continuum mechanics using object-oriented techniques, Comput. Phys., 12, 620–631, https://doi.org/10.1063/1.168744, 1998.

Yildirim, A., Kenway, G. K. W., Mader, C. A., and Martins, J. R. R. A.: A Jacobian-free approximate Newton–Krylov startup strategy for RANS simulations, J. Comput. Phys., submitted, 2018.

Yilmaz, Ö., Pires, O., Xabier, M., Sørensen, N., Reichstein, T., Schaffarczyk, A., Diakakis, K., Papadakis, G., Daniele, E., Schwarz, M., Lutz, T., and Prieto, R.: Summary of the blind test campaign to predict the high Reynolds number performance of DU00-W-210 airfoil, 35th Wind Energy Symposium, AIAA SciTech Forum, Grapevine, Texas, 9–13 January 2017, AIAA 2017-0915, https://doi.org/10.2514/6.2017-0915, 2017.

Yu, Y., Lyu, Z., Xu, Z., and Martins, J. R. R. A.: On the influence of optimization algorithm and starting design on wing aerodynamic shape optimization, Aerosp. Sci. Technol., 75, 183–199, https://doi.org/10.1016/j.ast.2018.01.016, 2018.

Zahle, F., Sørensen, N., and Johansen, J.: Wind turbine rotor-tower interaction using an incompressible overset grid method, Wind Energy, 12, 594–619, https://doi.org/10.1002/we.327, 2009.

Zahle, F., Bak, C., Sørensen, N., Vronsky, T., and Gaudern, N.: Design of the LRP airfoil series using 2D CFD, J. Phys. Conf. Ser., 524, 012020, https://doi.org/10.1088/1742-6596/524/1/012020, 2014.

Zahle, F., Tibaldi, C., Pavese, C., McWilliam, M., Blasques, J., and Hansen, M.: Design of an aeroelastically tailored 10 mw wind turbine rotor, J. Phys. Conf. Ser., 753, 062008, https://doi.org/10.1088/1742-6596/753/6/062008, 2016.

Zahle, F., Sørensen, N., McWilliam, M., and Barlas, A.: Computational fluid dynamics-based surrogate optimization of a wind turbine blade tip extension for maximising energy production, J. Phys. Conf. Ser., 1037, 042013, https://doi.org/10.1088/1742-6596/1037/4/042013, 2018.

Zhiquan, Y., Zhaoxue, C., Jingyi, C., and Shibao, B.: Aerodynamic optimum design procedure and program for the rotor of a horizontal-axis wind turbine, J. Wind Eng. Ind. Aerod., 39, 179–186, https://doi.org/10.1016/0167-6105(92)90544-K, 1992.

# A.1 Nomenclature

$$\mathbf{x}_{\mathrm{DV}} : \text{design variables,} \tag{A.1}$$
$$\mathbf{w} : \text{state solution,} \tag{A.2}$$
$$\mathbf{F}(\mathbf{w}, \mathbf{x}_{\mathrm{DV}}) : \text{PDE relating design variables and states,} \tag{A.3}$$
$$\mathrm{J}(\mathbf{w}, \mathbf{x}_{\mathrm{DV}}) : \text{functional of interest, and} \tag{A.4}$$
$$\hat{\mathrm{J}}(\mathbf{x}_{\mathrm{DV}}) : \text{reduced functional.} \tag{A.5}$$

## A.2 Abbreviations

DV: Design variable

LHS: Left hand side of an equation

RHS: Right hand side of an equation

AD: Algorithmic differentiation

IC: Initial conditions

BC: Boundary conditions

CFD: Computational fluid dynamics

LCO: Limit Cycle Oscillations

# Bibliography


[1] Siamak Akbarzadeh, Yang Wang, and Jens-Dominik Müller. *Fixed point discrete adjoint of SIMPLE-like solvers*. eng. In: *22nd AIAA Computational Fluid Dynamics Conference* (2015), 11 pp.

[2] Evan M. Anderson et al. *Adjoint-based high-fidelity aeroelastic optimization of wind turbine blade for load stress minimization*. eng. In: *Wind Energy Symposium, 2018* 210029 (2018), 17 pp. DOI: 10.2514/6.2018-1241.

[3] Evan M. Anderson et al. *Adjoint-based high-fidelity structural optimization of wind-turbine blade for load stress minimization*. eng. In: *AIAA Journal* 57.9 (2019), pages 4057–4070. ISSN: 1533385x, 00011452. DOI: 10.2514/1.j057756.

[4] W. Kyle Anderson et al. *Sensitivity analysis for Navier-Stokes equations on unstructured meshes using complex variables*. eng. In: *AIAA Journal* 39 (2001), pages 56–63. ISSN: 1533385x, 00011452. DOI: 10.2514/3.14697.

[5] Mikko Auvinen. *A Modular Framework for Generation and Maintenance of Adjoint Solvers Assisted by Algorithmic Differentiation - with Applications to an Incompressible Navier-Stokes Solver*. eng. G4 Monografiaväitöskirja. 2014, 183 pp. ISBN: 978-952-60-5883-2 (electronic); 978-952-60-5882-5 (printed). URL: https://aaltodoc.aalto.fi/handle/123456789/14050.

[6] Christian Bak et al. *Description of the DTU 10 MW Reference Wind Turbine*. Technical report. Risø National Laboratory, 2013.

[7] S Balay et al. *Efficient management of parallelism in object oriented numerical software libraries*. In: *Modern Software Tools in Scientific Computing*. Edited by E. Arge, A. M. Bruaset, and H. P. Langtangen. Birkhäuser Press, 1997, pages 163–202.

[8] S Balay et al. *PETSc Users Manual*. Technical report ANL-95/11 - Revision 3.9. https://www.mcs.anl.gov/petsc/documentation/index.html. Argonne National Laboratory, 2018.

[9] S Balay et al. *PETSc Web page*. http://www.mcs.anl.gov/petsc. 2018.

[10] R Barrett and A Ning. *Comparison of airfoil precomputational analysis methods for optimization of wind turbine blades*. eng. In: *IEEE Transactions on Sustainable Energy* 7.3 (2016), pages 1081–1088. ISSN: 19493037, 19493029. DOI: 10.1109/TSTE.2016.2522381.

[11] R Barrett and A Ning. *Integrated free-form method for aerostructural optimization of wind turbine blades*. In: *Wind Energy* 21.8 (2018), pages 663–675. DOI: 10.1002/we.2186. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/we.2186. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/we.2186.

[12] O. Baysal and M. E. Eleshaky. *AERODYNAMIC SENSITIVITY ANALYSIS-METHODS FOR THE COMPRESSIBLE EULER EQUATIONS*. eng. In: *Journal of Fluids Engineering-transactions of the Asme* 113.4 (1991), pages 681–688. ISSN: 1528901x, 00982202. DOI: 10.1115/1.2926534.

[13] Oktay Baysal, Mohamed E. Eleshaky, and Greg W. Burgreen. *Aerodynamic shape optimization using sensitivity analysis on third-order Euler equations*. In: *Journal of Aircraft* 30.6 (1993), pages 953–961. DOI: 10.2514/3.46439. eprint: https://doi.org/10.2514/3.46439. URL: https://doi.org/10.2514/3.46439.

[14] Franck Bertagnolio and Niels N. Sørensen. *Extension of EllipSys to compressible flows - Implementation and verifications*. eng. Technical report. 2018.

[15] Christian Bischof et al. *ADIFOR-Generating Derivative Codes from Fortran Programs*. In: *Sci. Program.* 1.1 (January 1992), pages 11–29. ISSN: 1058-9244. DOI: 10.1155/1992/717832. URL: http://dx.doi.org/10.1155/1992/717832.

[16] Joel Brezillon, Richard P. Dwight, and Markus Widhalm. *Aerodynamic Optimization for Cruise and High-Lift Configurations*. eng. In: *Notes on Numerical Fluid Mechanics and Multidisciplinary Design (nnfm)* 107 (2009), pages 249–262. ISSN: 10600824, 16122909.

[17] M. O. Bristeau et al. *On the numerical solution of nonlinear problems in fluid dynamics by least squares and finite element methods. II. Applications to transonic flow simulations*. eng. In: *Computer Methods in Applied Mechanics and Engineering* 51.1-3 (1985), pages 363–94, 363–394. ISSN: 18792138, 00457825. DOI: 10.1016/0045-7825(85)90039-8.

[18] MS Casale and EL Stanton. *An overview of analytic solid modeling*. eng. In: *IEEE Computer Graphics and Applications* 5.2 (1985), pages 45–56. ISSN: 15581756, 02721716. DOI: 10.1109/MCG.1985.276402.

[19] Bruce Christianson. *Reverse accumulation and attractive fixed points*. eng. In: *Optimization Methods and Software* 3.4 (1994), pages 311–326. ISSN: 10294937, 10556788. DOI: 10.1080/10556789408805572.

[20] Francois Courty et al. *Reverse automatic differentiation for optimum design: From adjoint state assembly to gradient computation*. eng. In: *Optimization Methods and Software* 18.5 (2003), pages 615–627. ISSN: 10294937, 10556788. DOI: 10.1080/10556780310001610501.

[21] Juan Carlos De los Reyes. *Numerical PDE-Constrained Optimization, SpringerBriefs in Optimization*. eng. Springer International Publishing, 2015, 123 pp. ISBN: 9783319133942.

[22] Tristan Dhert, Turaj Ashuri, and Joaquim R. R. A. Martins. *Aerodynamic Shape Optimization of Wind Turbine Blades Using a Reynolds-Averaged Navier–Stokes Model and an Adjoint Method*. In: *Wind Energy* 20.5 (May 2017), pages 909–926. DOI: 10.1002/we.2070.

[23] Cetin B. Dilgen et al. *Topology optimization of turbulent flows*. eng. In: *Computer Methods in Applied Mechanics and Engineering* 331 (2018), pages 363–393. ISSN: 18792138, 00457825. DOI: 10.1016/j.cma.2017.11.029.

[24] Richard P. Dwight and Joël Brezillon. *Efficient and robust algorithms for solution of the adjoint compressible Navier-Stokes equations with applications*. eng. In: *International Journal for Numerical Methods in Fluids* 60.4 (2009), pages 365–389. ISSN: 10970363, 02712091. DOI: 10.1002/fld.1894.

[25] TD Economon, F Palacios, and JJ Alonso. *A viscous continuous adjoint approach for the design of rotating engineering applications*. eng. In: *21st AIAA Computational Fluid Dynamics Conference* (2013).

[26] MA Elfarra, N Sezer-Uzol, and IS Akmandor. *NREL VI rotor blade: numerical investigation and winglet design and optimization using CFD*. eng. In: *Wind Energy* 17.4 (2014), pages 605–626. ISSN: 10991824, 10954244. DOI: 10.1002/we.1593.

[27] Jonathan Elliott and Jaime Peraire. *Practical 3d aerodynamic design and optimization using unstructured meshes*. eng. In: *6th Symposium on Multidisciplinary Analysis and Optimization* (1996), pages 1819–1828.

[28] Yisheng Gao, Yizhao Wu, and Jian Xia. *Automatic differentiation based discrete adjoint method for aerodynamic design optimization on unstructured meshes*. In: *Chinese Journal of Aeronautics* 30.2 (2017), pages 611–627. ISSN: 1000-9361. DOI: https://doi.org/10.1016/j.cja.2017.01.009. URL: http://www.sciencedirect.com/science/article/pii/S1000936117300377.

[29] Ralf Giering and Thomas Kaminski. *Recipes for Adjoint Code Construction*. In: *ACM Trans. Math. Softw.* 24.4 (December 1998), pages 437–474. ISSN: 0098-3500. DOI: 10.1145/293686.293695. URL: http://doi.acm.org.proxy.findit.dtu.dk/10.1145/293686.293695.

[30] M.B. Giles, D. Ghate, and M.C. Duta. *Using automatic differentiation for adjoint CFD code development*. In: *Recent Trends in Aerospace Design and Optimization*. Edited by B. Uthup et al. Tata McGraw-Hill, New Delhi, 2006, pages 426–434.

[31] Michael B. Giles and Niles A. Pierce. *An introduction to the adjoint approach to design*. eng. In: *Flow, Turbulence and Combustion* 65.3-4 (2000), pages 393–415. ISSN: 15731987, 13866184. DOI: 10.1023/A:1011430410075.

[32] Philip E. Gill, Walter Murray, and Michael A. Saunders. *SNOPT: An SQP algorithm for large-scale constrained optimization*. In: *SIAM Journal of Optimization* 12.4 (2002), pages 979–1006. DOI: 10.1137/S1052623499350013.

[33] Justin S. Gray et al. *OpenMDAO: an open-source framework for multidisciplinary design, analysis, and optimization*. eng. In: *Structural and Multidisciplinary Optimization* 59.4 (2019), pages 1075–1104. ISSN: 16151488, 1615147x. DOI: 10.1007/s00158-019-02211-z.

[34] A. Griewank. *Evaluating derivatives :* eng. Volume 19. SIAM, 2000, 369 pp. ISBN: 0898714516.

[35] Max D. Gunzburger. *Perspectives in flow control and optimization*. eng. SIAM, 2003, 261 pp. ISBN: 089871527x.

[36] Laurent Hascoet and Valérie Pascual. *The tapenade automatic differentiation tool: Principles, model, and specification*. eng. In: *Acm Transactions on Mathematical Software* 39.3 (2013), page 20. ISSN: 15577295, 00983500. DOI: 10.1145/2450153.2450158.

[37] Laurent Hascoët, Mariano Vázquez, and Alain Dervieux. *Automatic differentiation for optimum design, applied to sonic boom reduction*. eng. In: *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 2668 (2003), pages 85–94. ISSN: 16113349, 03029743.

[38] Ping He et al. *An aerodynamic design optimization framework using a discrete adjoint approach with OpenFOAM*. eng. In: *Computers and Fluids* 168 (2018), pages 285–303. ISSN: 18790747, 00457930. DOI: 10.1016/j.compfluid.2018.04.012.

[39] Ping He et al. *An Object-oriented Framework for Rapid Discrete Adjoint Development using OpenFOAM*. In: *AIAA Science and Technology Forum (SciTech)*. January 2019.

[40] P Heimbach, C Hill, and R Giering. *Automatic generation of efficient adjoint code for a parallel Navier-Stokes solver*. eng. In: *Lecture Notes in Computer Science* 2330 (2002), pages 1019–1028. ISSN: 16113349, 03029743.

[41] JC Heinz, NN Sørensen, and F Zahle. *Fluid-structure interaction computations for geometrically resolved rotor simulations using CFD*. eng. In: *Wind Energy* 19.12 (2016), pages 2205–2221. ISSN: 10991824, 10954244. DOI: 10.1002/we.1976.

[42] Raymond M. Hicks, Earll M. Murman, and Garret N. Vanderplaats. *An assessment of airfoil design by numerical optimization*. eng. In: *NASA, report TM X-3092* (1974).

[43] Sergio González Horcas et al. *Suppresing Vortex Induced Vibrations of Wind Turbine blades with flaps*. English. In: *Proceedings of the 7th European Conference on Computational Fluid Dynamics (ECFD 7)*. European Community on Computational Methods in Applied Sciences, 2018.

[44] Jan Christian Hückelheim, Laurent Hascoët, and Jens Dominik Müller. *Algorithmic differentiation of code with multiple context-specific activities*. eng. In: *Acm Transactions on Intelligent Systems and Technology* 43.4 (2016), page 35. ISSN: 21576912, 21576904, 00983500, 15577295. DOI: 10.1145/3015464.

[45] III James Newman et al. *Aerodynamic shape sensitivity analysis and design optimization of complex configurations using unstructured grids*. In: *15th Applied Aerodynamics Conference* (1997). DOI: 10.2514/6.1997-2275. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.1997-2275. URL: https://arc.aiaa.org/doi/abs/10.2514/6.1997-2275.

[46] Antony Jameson. *Aerodynamic design via control theory*. In: *Journal of Scientific Computing* 3.3 (September 1988), pages 233–260. ISSN: 1573-7691. DOI: 10.1007/BF01061285. URL: https://doi.org/10.1007/BF01061285.

[47] Antony Jameson. *Optimum aerodynamic design using CFD and control theory*. In: *AIAA-95-1729-CP* (1995).

[48] Antony Jameson, Luigi Martinelli, and Niles A. Pierce. *Optimum aerodynamic design using the Navier–Stokes equations*. eng. In: *Theoretical and Computational Fluid Dynamics* 10.1-4 (1998), pages 213–237. ISSN: 14322250, 09354964. DOI: 10.1007/s001620050060.

[49] Dominic Jones, Jens-Dominik Mueller, and Sami Bayyuk. *CFD Development with Automatic Differentiation*. In: *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. DOI: 10.2514/6.2012-573. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2012-573. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2012-573.

[50] Dominic Jones, Jens-Dominik Müller, and Faidon Christakopoulos. *Preparation and assembly of discrete adjoint CFD codes*. eng. In: *Computers and Fluids* 46.1 (2011), pages 282–286. ISSN: 18790747, 00457930. DOI: 10.1016/j.compfluid.2011.01.042.

[51] S. H. Jongsma. *On a Method for Simulation-BasedWind Turbine Blade Design*. PhD thesis. University of Twente, The Netherlands, 2014, 248 pp. ISBN: 978-90-365-3698-1. DOI: 10.3990/1.9789036536981. URL: https://www2.ts.ctw.utwente.nl/rob/TS_publications/PDF/S.H.Jongsma.pdf.

[52] Andrew L. Kaminsky and Kivanc Ekici. *Reduced-order model-based convergence acceleration of reverse mode discrete adjoint solvers*. eng. In: *Aerospace Science and Technology* 93 (2019), page 105334. ISSN: 16263219, 12709638. DOI: 10.1016/j.ast.2019.105334.

[53] Gaetan K. W. Kenway, Graeme J. Kennedy, and Joaquim R. R. A. Martins. *Scalable Parallel Approach for High-Fidelity Steady-State Aeroelastic Analysis and Derivative Computations*. In: *AIAA Journal* 52.5 (May 2014), pages 935–951. DOI: 10.2514/1.J052255.

[54] Gaetan K. W. Kenway and Joaquim R. R. A. Martins. *Multipoint High-Fidelity Aerostructural Optimization of a Transport Aircraft Configuration*. In: *Journal of Aircraft* 51.1 (January 2014), pages 144–160. DOI: 10.2514/1.C032150.

[55] Gaetan K. W. Kenway et al. *Effective Adjoint Approaches for Computational Fluid Dynamics*. In: *Progress in Aerospace Sciences* (2019). DOI: 10.1016/j.paerosci.2019.05.002.

[56] Gaetan K.W. Kenway, Graeme. J. Kennedy, and Joaquim R. R. A. Martins. *A CAD-Free Approach to High-Fidelity Aerostructural Optimization*. In: *Proceedings of the 13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*. AIAA 2010-9231. Fort Worth, TX, September 2010. DOI: 10.2514/6.2010-9231.

[57] P Khayatzadeh and SK Nadarajah. *Aerodynamic Shape Optimization via Discrete Viscous Adjoint Equations for the k-ω SST Turbulence and $\gamma - \widetilde{Re_\theta}$ Transition Models*. eng. In: *49th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition* (2011). DOI: 10.2514/6.2011-1247.

[58] William L. Kleb et al. *Collaborative software development in support of fast adaptive aerospace tools (FAAST)*. eng. In: *16th AIAA Computational Fluid Dynamics Conference* (2003).

[59] D. A. Knoll and David E. Keyes. *Jacobian-free Newton-Krylov methods: A survey of approaches and applications*. eng. In: *Journal of Computational Physics* 193.2 (2004), pages 357–397. ISSN: 10902716, 00219991. DOI: 10.1016/j.jcp.2003.08.010.

[60] HI Kwon, JY You, and OJ Kwon. *Enhancement of wind turbine aerodynamic performance by a numerical optimization technique*. eng. In: *Journal of Mechanical Science and Technology* 26.2 (2012), pages 455–462. ISSN: 19763824, 1738494x. DOI: 10.1007/s12206-011-1035-2.

[61] Paul van der Laan and Niels N. Sørensen. *A 1D version of EllipSys*. eng. Technical report. 2017.

[62] Andrew B. Lambe and Joaquim R. R. A. Martins. *Extensions to the Design Structure Matrix for the Description of Multidisciplinary Design, Analysis, and Optimization Processes*. In: *Structural and Multidisciplinary Optimization* 46 (August 2012), pages 273–284. DOI: 10.1007/s00158-012-0763-y.

[63] Robin B. Langtry and Florian R. Menter. *Correlation-based transition modeling for unstructured parallelized computational fluid dynamics codes*. eng. In: *AIAA Journal* 47.12 (2009), pages 2894–2906. ISSN: 1533385x, 00011452. DOI: 10.2514/1.42362.

[64] C Liang and H Li. *Effects of optimized airfoil on vertical axis wind turbine aerodynamic performance*. eng. In: *Journal of the Brazilian Society of Mechanical Sciences and Engineering* 40.2 (2018), page 88. ISSN: 18063691, 16785878. DOI: 10.1007/s40430-017-0926-2.

[65] Michael J. Lighthill. *A New Method of Two-dimensional Aerodynamic Design*. Technical report. Aeronautical Research Council, 2112, 1945.

[66] J. N. Lyness. *Numerical algorithms based on the theory of complex variable*. eng. In: *Proceedings of the 1967 22nd National Conference* (1967), pages 125–133. DOI: 10.1145/800196.805983.

[67] J. N. Lyness and C. B. Moler. *Numerical Differentiation of Analytic Functions*. eng. In: *Siam Journal on Numerical Analysis* 4.2 (1967), pages 202–210. ISSN: 10957170, 00361429. DOI: 10.1137/0704019.

[68] C. A. Mader et al. *ADjoint: An Approach for the Rapid Development of Discrete Adjoint Solvers*. In: *AIAA Journal* 46.4 (2008), pages 863–873. DOI: 10.2514/1.29123. eprint: https://doi.org/10.2514/1.29123. URL: https://doi.org/10.2514/1.29123.

[69] Charles A. Mader, Joaquim R.R.A. Martins, and Juan J. Alonso. *ADjoint: An approach for rapid development of discrete adjoint solvers*. eng. In: *Collection of Technical Papers - 11th AIAA/issmo Multidisciplinary Analysis and Optimization Conference* 4 (2006), pages 2506–2518.

[70] Mads Holst Aagaard Madsen. *Onset of turbulence*. Master's thesis. Denmark: Niels Bohr Institute, University of Copenhagen, 2016.

[71] Mads Holst Aagaard Madsen et al. *Multipoint high-fidelity CFD-based aerodynamic shape optimization of a 10 MW wind turbine*. In: *Wind Energy Science* 4.2 (2019), pages 163–192. DOI: 10.5194/wes-4-163-2019. URL: https://www.wind-energ-sci.net/4/163/2019/.

[72] A. C. Marta et al. *A methodology for the development of discrete adjoint solvers using automatic differentiation tools*. eng. In: *International Journal of Computational Fluid Dynamics* 21.9-10 (2007), pages 307–327. ISSN: 10290257, 10618562. DOI: 10.1080/10618560701678647. eprint: https://doi.org/10.1080/10618560701678647. URL: https://doi.org/10.1080/10618560701678647.

[73] Joaquim R. R. A. Martins, Peter Sturdza, and Juan J. Alonso. *The Complex-Step Derivative Approximation*. In: *ACM Transactions on Mathematical Software* 29.3 (September 2003), pages 245–262. DOI: 10.1145/838250.838251.

[74] Geoffrey B. Mcfadden. *An artificial viscosity method for the design of supercritical airfoils*. eng. In: *Mathematics and Computing COO-3077-ISS* (1979). URL: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19790019965.pdf.

[75] F. R. Menter, R. Langtry, and S. Voelker. *Transition modelling for general purpose CFD codes*. eng. In: *Flow Turbulence and Combustion* 77.1-4 (2006), pages 277–303. ISSN: 15731987, 13866184. DOI: 10.1007/s10494-006-9047-1.

[76] FR Menter. *Zonal Two Equation Kappa-Omega Turbulence Models for Aerodynamic Flows*. eng. In: *24th Fluid dynamics conference* (1993).

[77] JA Michelsen. *Basis3D - a Platform for Development of Multiblock PDE Solvers*. Technical report AFM 92-05. Department of Fluid Mechanics, Technical University of Denmark, 1992.

[78] JA Michelsen. *Block structured multigrid solution of 2D and 3D elliptic PDE's*. Technical report AFM 94-06. Department of Fluid Mechanics, Technical University of Denmark, 1994.

[79] Bijan Mohammadi. *Optimal shape design, reverse mode of automatic differentiation and turbulence*. eng. In: *35th Aerospace Sciences Meeting and Exhibit* (1997), A-97-0099.

[80] J. D. Müller and P. Cusdin. *On the performance of discrete adjoint CFD codes using automatic differentiation*. eng. In: *International Journal for Numerical Methods in Fluids* 47.8-9 (2005), pages 939–945. ISSN: 10970363, 02712091. DOI: 10.1002/fld.885.

[81] Jens-Dominik Müller, Orest Mykhaskiv, and Jan Christian Hückelheim. *STAMPs: A finite-volume solver framework for adjoint codes derived with source-transformation AD*. eng. In: *2018 Multidisciplinary Analysis and Optimization Conference* (2018), AIAA 2018-2928, AIAA 2018-2928. DOI: 10.2514/6.2018-2928.

[82] S. K. Nadarajah. *The discrete adjoint approach to aerodynamic shape optimization*. PhD thesis. Stanford University, 2003. URL: http://aero-comlab.stanford.edu/Papers/nadarajah.thesis.pdf.

[83] Siva K. Nadarajah and Antony Jameson. *A comparison of the continuous and discrete adjoint approach to automatic aerodynamic optimization*. eng. In: *38th Aerospace Sciences Meeting and Exhibit* (2000).

[84] Eric J. Nielsen and Kyle W. Anderson. *Aerodynamic Design Optimization on Unstructured Meshes Using the Navier-Stokes Equations*. eng. In: *AIAA 98-4809* (1998).

[85] Eric J. Nielsen and Boris Diskin. *Discrete adjoint-based design for unsteady turbulent flows on dynamic overset unstructured grids*. eng. In: *50th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition* (2012), AIAA 2012-0554. DOI: 10.2514/6.2012-554.

[86] Eric J. Nielsen and Boris Diskin. *Discrete adjoint-based design for unsteady turbulent flows on dynamic overset unstructured grids*. eng. In: *AIAA Journal* 51.6 (2013), pages 1355–1373. ISSN: 1533385x, 00011452, 10810102. DOI: 10.2514/1.J051859.

[87] Eric J. Nielsen and William L. Kleb. *Efficient construction of discrete adjoint operators on unstructured grids using complex variables*. eng. In: *AIAA Journal* 44.4 (2006), pages 827–836. ISSN: 1533385x, 00011452. DOI: 10.2514/1.15830.

[88] C. Othmer. *A continuous adjoint formulation for the computation of topological and surface sensitivities of ducted flows*. eng. In: *International Journal for Numerical Methods in Fluids* 58.8 (2008), pages 861–877. ISSN: 10970363, 02712091. DOI: 10.1002/fld.1770.

[89] Carsten Othmer, Eugene De Villiers, and Henry G. Weller. *Implementation of a continuous adjoint for topology optimization of ducted flows*. eng. In: *Collection of Technical Papers - 18th AIAA Computational Fluid Dynamics Conference* 1 (2007), pages 407–413.

[90] Francisco Palacios et al. *Stanford University Unstructured (SU2): An open-source integrated computational environment for multi-physics simulation and design*. eng. In: *51st AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition 2013* (2013).

[91] S.V. Patankar. *Numerical heat transfer and fluid flow*. eng. Hemisphere, 1980, 197 pp. ISBN: 0070487405.

[92] John Pearson. *Fast iterative solvers for PDE-constrained optimization problems*. PhD thesis. Oxford University, UK, 2013.

[93] Ruben E. Perez, Peter W. Jansen, and Joaquim R. R. A. Martins. *pyOpt: A Python-Based Object-Oriented Framework for Nonlinear Constrained Optimization*. In: *Structural and Multidisciplinary Optimization* 45.1 (January 2012), pages 101–118. DOI: 10.1007/s00158-011-0666-3.

[94] Jacques E V Peter and Richard P. Dwight. *Numerical sensitivity analysis for aerodynamic optimization: A survey of approaches*. eng. In: *Computers and Fluids* 39.3 (2010), pages 373–391. ISSN: 18790747, 00457930. DOI: 10.1016/j.compfluid.2009.09.013.

[95] O. Pironneau. *On optimum profiles in Stokes flow*. In: *Journal of Fluid Mechanics* 59.1 (1973), pages 117–128. DOI: 10.1017/S002211207300145X.

[96] Ramy Rashad and David W. Zingg. *Aerodynamic Shape Optimization for Natural Laminar Flow Using a Discrete-Adjoint Approach*. In: *AIAA Journal* 54.11 (2016), pages 3321–3337. DOI: 10.2514/1.J054940. eprint: https://doi.org/10.2514/1.J054940. URL: https://doi.org/10.2514/1.J054940.

[97] Tyrone Rees. *Preconditioning Iterative Methods for PDE Constrained Optimization*. PhD thesis. Oxford University, UK, 2010, 197 pp.

[98] James Reuther and Antony Jameson. *Control theory based airfoil design using the Euler equations*. eng. In: *5th Symposium on Multidisciplinary Analysis and Optimization* (1994). DOI: 10.2514/6.1994-4272. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.1994-4272. URL: https://arc.aiaa.org/doi/abs/10.2514/6.1994-4272.

[99] James Reuther et al. *Aerodynamic Shape Optimization of Complex Aircraft Configurations via an Adjoint Formulation*. eng. In: *34th Aerospace Sciences Meeting and Exhibit* (1996). DOI: 10.2514/6.1996-94. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.1996-94. URL: https://arc.aiaa.org/doi/abs/10.2514/6.1996-94.

[100] CM Rhie and WL Chow. *Numerical study of the turbulent flow past an airfoil with trailing edge separation*. eng. In: *AIAA Journal* 21.11 (1983), pages 1525–1532. ISSN: 1533385x, 00011452.

[101] AFP Ribeiro, AM Awruch, and HM Gomes. *An airfoil optimization technique for wind turbines*. eng. In: *Applied Mathematical Modelling* 36.10 (2012), pages 4898–4907. ISSN: 18728480, 0307904x. DOI: 10.1016/j.apm.2011.12.026.

[102] R Ritlop and SK Nadarajah. *Design of wind turbine profiles via a preconditioned adjoint-based aerodynamic shape optimization*. eng. In: *47th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition* (2009). AIAA 2009-1547.

[103] PJ Roache. *Perspective - A method for uniform reporting of grid refinement studies*. eng. In: *Journal of Fluids Engineering-transactions of the Asme* 116.3 (1994), pages 405–413. ISSN: 17784166, 12900729, 1528901x, 00982202. DOI: 10.1115/1.2910291.

[104] Rolf Roth and Stefan Ulbrich. *A discrete adjoint approach for the optimization of unsteady turbulent flows*. eng. In: *Flow, Turbulence and Combustion* 90.4 (2013), pages 763–783. ISSN: 15731987, 13866184. DOI: 10.1007/s10494-012-9439-3.

[105] Y Saad and MH Schultz. *GMRES - A generalized minimal residual algorithm for solving nonsymmetric linear-systems*. eng. In: *Siam Journal on Scientific and Statistical Computing* 7.3 (1986), pages 856–869. ISSN: 21683417, 01965204. DOI: 10.1137/0907058.

[106] Jamshid A. Samareh. *Status and Future of Geometry Modeling and Grid Generation for Design and Optimization*. In: *Journal of Aircraft* 36.1 (1999), pages 97–104. DOI: 10.2514/2.2417. eprint: https://doi.org/10.2514/2.2417. URL: https://doi.org/10.2514/2.2417.

[107] JG Schepers et al. *Analysis of Mexico wind tunnel measurements: Final report of IEA Task 29, Mexnext (Phase 1)*. English. Energy Research Centre of the Netherlands (ECN), 2012.

[108] M Schramm, B Stoevesandt, and J Peinke. *Adjoint optimization of 2D-airfoils in incompressible flows*. eng. In: *11th World Congress on Computational Mechanics, Wccm 2014, 5th European Conference on Computational Mechanics, Eccm 2014 and 6th European Conference on Computational Fluid Dynamics, ECFD 2014* (2014), pages 6200–6211.

[109] M Schramm, B Stoevesandt, and J Peinke. *Lift optimization of airfoils using the adjoint approach*. eng. In: *European Wind Energy Association Annual Conference and Exhibition 2015, EWEA 2015 - Scientific Proceedings* (2015).

[110] M Schramm, B Stoevesandt, and J Peinke. *Optimization of airfoils using the adjoint approach and the influence of adjoint turbulent viscosity*. eng. In: *Computation* 6.1 (2018). ISSN: 20793197. DOI: 10.3390/computation6010005.

[111] M Schramm, B Stoevesandt, and J Peinke. *Simulation and Optimization of an Airfoil with Leading Edge Slat*. In: *Journal of Physics: Conference Series* 753.2 (2016), page 022052. URL: http://stacks.iop.org/1742-6596/753/i=2/a=022052.

[112] Thomas W. Sederberg and Scott R. Parry. *Free-form deformation of solid geometric models*. eng. In: *ACM Siggraph Computer Graphics* (1986), pages 151–160. ISSN: 15584569, 00978930. DOI: 10.1145/15922.15903.

[113] G. R. Shubin and P. D. Frank. *A comparison of two closely-related approaches to aerodynamic design optimization*. eng. In: *Third International Conference on Inverse Design Concepts and Optimization in Engineering Sciences (ICIDES-III)* (1991).

[114] NN Sørensen. *General purpose flow solver applied to flow over hills*. PhD thesis. Frederiksborgvej 399, 4000 Roskilde: Risø National Laboratory, 1995.

[115] NN Sørensen. *HypGrid2D. A 2-d mesh generator*. eng. Technical report Risø-R-1035(EN). Risø National Laboratory, 1998, 36 pp.

[116] NN Sørensen, JA Michelsen, and S Schreck. *Navier-Stokes predictions of the NREL phase VI rotor in the NASA Ames 80-by-120 wind tunnel*. eng. In: *2002 Asme Wind Energy Symposium; 40. AIAA Aerospace Sciences Meeting and Exhibit* (2002), pages 94–105.

[117] PR Spalart and SR Allmaras. *A one-equation turbulence model for aerodynamic flows*. eng. In: *Recherche Aerospatiale* 1 (1994), pages 5–21. ISSN: 00341223.

[118] Arthur Stück. *Adjoint Navier-Stokes methods for hydrodynamic shape optimisation*. doctoralThesis. Technische Universität Hamburg, 2012. ISBN: 978-3-89220-661-3. DOI: 10.15480/882.1061. URL: http://tubdok.tub.tuhh.de/handle/11420/1063.

[119] M. Towara and U. Naumann. *A Discrete Adjoint Model for OpenFOAM*. In: *Procedia Computer Science* 18 (2013). 2013 International Conference on Computational Science, pages 429–438. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2013.05.206. URL: http://www.sciencedirect.com/science/article/pii/S1877050913003499.

[120] M. Towara and U. Naumann. *SIMPLE adjoint message passing*. eng. In: *Optimization Methods and Software* 33.4-6 (2018), pages 1232–1249. ISSN: 10294937, 10556788. DOI: 10.1080/10556788.2018.1435653.

[121] Fredi Tröltzsch. *Optimal control of partial differential equations : Theory, methods and applications*. eng. American Mathematical Society, 2010, 399 pp. ISBN: 9780821849040.

[122] Konstantinos T. Tsiakas et al. *Shape Optimization of Wind Turbine Blades Using the Continuous Adjoint Method and Volumetric NURBS on a GPU Cluster*. eng. In: *Computational Methods in Applied Sciences* 48 (2019), pages 131–144. ISSN: 18713033. DOI: 10.1007/978-3-319-89988-6_8.

[123] Michael Ulbrich and Stefan Ulbrich. *Automatic Differentiation: A Structure-Exploiting Forward Mode with Almost Optimal Complexity for Kantorovic Trees*. eng. Technical report IAMS1996.1TUM. Technical University of München, 1996. DOI: 10.1.1.29.2886.

[124] Tom Verstraete, Lasse Müller, and Jens Dominik Müller. *Adjoint-based design optimisation of an internal cooling channel U-bend for minimised pressure losses*. eng. In: *International Journal of Turbomachinery, Propulsion and Power* 2.2 (2017), ijtpp2020010. ISSN: 2504186x. DOI: 10.3390/ijtpp2020010.

[125] L Vorspel, B Stoevesandt, and J Peinke. *Optimize rotating wind energy rotor blades using the adjoint approach*. eng. In: *Applied Sciences* (2018). ISSN: 20763417.

[126] L Vorspel et al. *A benchmark study on the efficiency of unconstrained optimization algorithms in 2D-aerodynamic shape design*. eng. In: *Cogent Engineering* 4.1 (2017). Edited by Pandian Vasant, page 1354509. ISSN: 23311916. DOI: 10.1080/23311916.2017.1354509.

[127] L Vorspel et al. *Towards the optimization of wind turbine rotor blades by means of computational fluid dynamics and the adjoint approach*. eng. In: *34th AIAA Applied Aerodynamics Conference, AIAA AVIATION Forum, (AIAA 2016-3728)* (2016). DOI: 10.2514/6.2016-3728.

[128] D Vucina, I Marinic-Kragic, and Z Milas. *Numerical models for robust shape optimization of wind turbine blades*. eng. In: *Renewable Energy* 87 (2016), pages 849–862. ISSN: 18790682, 09601481. DOI: 10.1016/j.renene.2015.10.040.

[129] HG Weller et al. *A tensorial approach to computational continuum mechanics using object-oriented techniques*. eng. In: *Computers in Physics* 12.6 (1998), pages 620–631. ISSN: 15584208, 08941866. DOI: 10.1063/1.168744.

[130] D. C. Wilcox. *Formulation of the k-w Turbulence Model Revisited*. In: *AIAA Journal* 46.11 (2008), pages 2823–2838. DOI: 10.2514/1.36541. eprint: https://doi.org/10.2514/1.36541. URL: https://doi.org/10.2514/1.36541.

[131] Shenren Xu and Sebastian Timme. *Robust and efficient adjoint solver for complex flow conditions*. eng. In: *Computers and Fluids* 148 (2017), pages 26–38. ISSN: 18790747, 00457930. DOI: 10.1016/j.compfluid.2017.02.012.

[132] Shenren Xu et al. *Stabilisation of discrete steady adjoint solvers*. eng. In: *Journal of Computational Physics* 299 (2015), pages 175–195. ISSN: 10902716, 00219991. DOI: 10.1016/j.jcp.2015.06.036.

[133] F Zahle, NN Sørensen, and J Johansen. *Wind turbine rotor-tower interaction using an incompressible overset grid method*. In: *Wind Energy* 12.6 (2009), pages 594–619. DOI: 10.1002/we.327. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/we.327. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/we.327.

[134] Frederik Zahle. *Wind turbine aerodynamics using an incompressible overset grid method*. eng. PhD thesis. Denmark: DTU Wind Energy, 2006, 149 pp.

[135] F Zahle et al. *Computational fluid dynamics-based surrogate optimization of a wind turbine blade tip extension for maximising energy production*. In: *Journal of Physics: Conference Series* 1037.4 (2018), page 042013. URL: http://stacks.iop.org/1742-6596/1037/i=4/a=042013.

[136] F Zahle et al. *Design of the LRP airfoil series using 2D CFD*. In: *Journal of Physics: Conference Series* 524.1 (2014), page 012020. URL: http://stacks.iop.org/1742-6596/524/i=1/a=012020.

**Technical University of Denmark**
Department of Wind Energy
Frederiksborgvej 399
Building 118
4000 Roskilde
Denmark
Telephone  46 77 50 85

info@vindenergi.dtu.dk
www.vindenergi.dtu.dk