NATIONAL TECHNICAL UNIVERSITY OF ATHENS
School of Mechanical Engineering
Fluids Section
Laboratory of Thermal Turbomachines
Parallel CFD & Optimization Unit

Unsteady Discrete Adjoint Method Formulated in the Time-Domain for Shape Optimization in Turbomachinery

Ph.D. Thesis

Georgios D. Ntanakas

Academic Advisor: Kyriakos C. Giannakoglou, Professor NTUA

Industrial Advisor: Marcus Meyer, Ph.D., CFD Methods,
Rolls-Royce Deutschland

Athens, 2018

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
School of Mechanical Engineering
Fluids Section
Laboratory of Thermal Turbomachines
Parallel CFD & Optimization Unit

Unsteady Discrete Adjoint Method Formulated in the Time-Domain for Shape Optimization in Turbomachinery

Ph.D. Thesis

Georgios Ntanakas

Examination Committee:

1. Kyriakos Giannakoglou (Academic Supervisor)*
   Professor, NTUA, School of Mechanical Engineering
2. Nikolaos Aretakis*
   Associate Professor, NTUA, School of Mechanical Engineering
3. Spyridon Voutsinas*
   Professor, NTUA, School of Mechanical Engineering
4. Konstantinos Mathioudakis
   Professor, NTUA, School of Mechanical Engineering
5. Ioannis Nikolos
   Professor, Technical University of Crete, School of Production Engineering & Management
6. Lambros Kaiktsis
   Professor, NTUA, School of Naval Architecture and Marine Engineering
7. Ioannis Roumeliotis
   Assistant Professor, Hellenic Naval Academy
   Lecturer, Cranfield University, School of Aerospace, Transport and Manufacturing

* Member of the Advisory Committee.

Athens, 2018

This work has been conducted within the AboutFlow ITN on “Adjoint-based optimization of industrial and unsteady flows”
http://aboutflow.sems.qmul.ac.uk

AboutFlow has received funding from the European Union’s Seventh Framework Programme for research, technological development and demonstration under Grant Agreement No. 317006.

Abstract

This PhD thesis deals with the mathematical formulation, solution, programming and validation of the unsteady discrete adjoint method, formulated in the time-domain, for the computation of first-order sensitivity derivatives for objective functions related to the aerodynamics of turbomachinery and their utilization in optimization algorithms. The cases that are tackled involve the constrained optimization of industrial, 3D, multi-row, turbomachinery configurations with transient and periodic flows.

The unsteady adjoint equations are formulated for an objective function in the form of a time-integral over a selected time-interval. The dual time-stepping technique is used to solve the unsteady adjoint equations along with an iterative scheme, which is the adjoint to the 5-stage Runge-Kutta scheme used for the flow equations and which is derived "by-hand". The scheme is formulated so as to ensure same convergence rate as the Unsteady Reynolds-Averaged Navier-Stokes (URANS) solver. Algorithmic Differentiation (AD) is employed in the adjoint solver for the computation of selected differential terms. Its usage is restricted to low level operations and combined with hand-differentiation to ensure efficiency.

To enable communication between adjacent row-domains in the adjoint solver, the adjoint sliding interface is developed to replace the mixing interface technique used in steady state solvers. Its baseline is the sliding interface of the flow solver where grids of adjacent rows are generated so that there is a one-cell overlap. AD along with hand programming ensure that the implementation is consistent with the reverse flow of information in the adjoint solver.

The solver utilizes the SSD disk space instead of RAM to store and read-in, in a parallel manner, the per-time-step flow fields during the adjoint execution. Thus, RAM bottlenecks are avoided while run time is not significantly increased. The temporal coarsening technique is employed in the adjoint solver to decrease the run time and the required storage space when this exceeds the available storage capacity.

Adjoint-based derivatives are computed and used within the optimization workflow. If equality constraints are considered, the component of the objective function's gradient which is normal to the constraints' gradients is used along with the projected gradient descent method to update the design variables and, thus, the geometry. In unconstrained optimization problems, steepest descent is used.

The developed software is applied to the shape optimization of 3D, multi-row, turbomachinery cases for the first time in the literature. The application cases include one single row turbine case (transient operation), one stage turbine case (periodic flow study) and one 3-row compressor case (periodic flow study). The computed derivatives are validated against the derivatives computed via finite differences and, then, used in optimization setups with and without equality constraints.

Key words: Computational Fluid Dynamics, Unsteady Discrete Adjoint Method, Sensitivity Derivatives, Shape Optimization, Transient Flow, Multi-row Turbomachinery

Acknowledgments

I would like to thank all those who have contributed to the completion of this thesis. First of all, I would like to thank the academic advisor, Kyriakos C. Giannakoglou, Professor NTUA, for giving me the opportunity to engage in such an interesting project. Our cooperation during my Diploma thesis, inspired me to pursue a Ph.D. and his scientific and personal guidance was essential during the development of this thesis. I would like to thank him also for the diligent proofreading of the published scientific papers and this text.

I would also like to express my gratitude to the industrial advisor, Dr. Marcus Meyer, Rolls-Royce Deutschland, for the supervision of the research work conducted in Rolls-Royce Deutschland. I truly appreciate his daily investment in time and effort to reach our goal. Without his constant support and guidance, starting from my first day in RRD, the completion of this work would be impossible.

I would like to thank Nikolaos Aretakis, Associate Professor NTUA, and Spiridon Voutsinas, Professor NTUA, members of the advisory comittee and also Konstantinos Mathioudakis, Professor NTUA, for their suggestions for improving the content of this thesis and its presentation.

I would like to thank the AboutFlow ITN, funded by Marie Skłodowska-Curie Actions, a Research Fellowship Programme of the European Commission, for funding the first 3 years of this work. Thanks also to all the participants, advisors and research fellows, for the cooperation and exchange of ideas for the duration of the project. The experience of working towards similar directions in such a scientific community will remain unforgettable. Many thanks to Dr. Jens D. Müller, Senior Lecturer QMUL, and Dr. Laurent Hascoët, INRIA, for kindly hosting me during my secondments in QMUL and INRIA.

I would like to thank Rolls-Royce Deutschland for funding the last 2 years of this work and the DSE group for creating a pleasant and supportive working environment.

More specifically, my sincere thanks to Prof. Marius Swoboda, Head of DSE, for being available when needed and for the facilitation of the 2-year extension of my contract with RRD. I am also grateful to David Radford and Paolo Adami for helping me familiarize with HYDRA and our meaningful communication during the development of the software. Thanks also to Ilias Vasilopoulos both for the scientific cooperation and the daily chit-chats.

I would like to thank the members of LTT/NTUA for the useful discussions and e-mails during this work and the friendly working environment during my secondments at NTUA.

Finally, I would like to thank my parents, Δημήτρη and Παναγιώτα, and my sister for their love and support and also my close friends and Lisa.

Contents

List of Figures

List of Tables

Nomenclature

This section lists the most important symbols and abbreviations used within this thesis. All symbols are defined within the thesis upon their first appearance. In case a symbol represents more than a single quantity, this is re-specified whenever its definition is not self-explanatory.

Scalar Quantities

δ_{ij}	Kronecker delta function
γ	Ratio of specific heats
μ	Dynamic viscosity
ν	Molecular kinetic viscosity
τ	Pseudo-time
$\tilde{\nu}$	Spalart-Allmaras turbulence model variable
ϱ	Density
c_p	Specific heat capacity under constant pressure
c_v	Specific heat capacity under constant volume
J	Objective function
j_{st}	Objective function integrand
p	Pressure
Pr	Prandtl number
q_i	Heat flow
R_g	Constant of perfect gas
T	Temperature
t	Real time
V	Volume

Vectors & Matrices

$\boldsymbol{\psi}$	Adjoint variables' vector
$\boldsymbol{v}$	Absolute velocity vector
$\boldsymbol{a}$	Design variable vector
$\boldsymbol{f}^{inv}$	Inviscid flux vector
$\boldsymbol{f}^{visc}$	Viscous flux vector
$\boldsymbol{g}$	Adjoint equation right hand side (piecewise)
$\boldsymbol{I}_H^h$	Multigrid prolongation operator
$\boldsymbol{I}_h^H$	Multigrid restriction operator
$\boldsymbol{P}$	Block-Jacobi preconditioner
$\boldsymbol{r}$	URANS residual
$\boldsymbol{r}_{adj}$	Adjoint residual
$\boldsymbol{R}_{forw}$	Forward differentiation residual
$\boldsymbol{s}$	Source terms
$\boldsymbol{u}$	Conservative flow variable vector
$\boldsymbol{w}$	Relative velocity vector
$\boldsymbol{x} = [x_1, x_2, x_3]^T$	Cartesian coordinates' vector
$\hat{\boldsymbol{n}}$	Unit normal vector
$\mathcal{K}$	Runge-Kutta & multigrid operator

Sub/Superscripts

H	Coarse grid index
h	Fine grid index
k	Pseudo-time-step index
m	Runge-Kutta stage index
n	Real time-step index
P	Central node of control volume
Q	Neighboring node of control volume
T	Transpose

Abbreviations

AD	Algorithmic Differentiation
AIAA	American Institute of Aeronautics & Astronautics
ASME	American Society of Mechanical Engineers
CFD	Computational Fluid Dynamics

ITN	Initial Training Network
LTT	Lab of Thermal Turbomachines
MPI	Message Passing Interface
NTUA	National Technical University of Athens
OPlus	Oxford Parallel library for unstructured solvers
POD	Proper Orthogonal Decomposition
RANS	Reynolds-Averaged Navier-Stokes
RRD	Rolls-Royce Deutschland
URANS	Unsteady Reynolds-Averaged Navier-Stokes

1 Introduction

The continuous evolution of high-performance computing empowered the long-lasting, ongoing transition from steady to unsteady Computational Fluid Dynamics (CFD) simulations and surely affects adjoint solvers too. The use of CFD-based, rather than exclusively experiment-based design, is nowadays established. The new demand is improving CFD optimization results by reducing the number of assumptions made in CFD which is also the case with adjoint solvers. This is the area which this thesis contributes to by extending the application of the unsteady adjoint method, formulated in the time-domain, to industrial, 3D, multi-row, turbomachinery applications. Before expanding the goals of this work, the fundamental background is set and the relevant literature review is presented. The chapter concludes with the outline of this thesis.

1.1 CFD & Optimization in Turbomachinery

CFD and optimization have played a decisive role in the evolution of turbomachinery design [1–5]. Over the years, the problems to be solved have increased in size and complexity. The growth of the available computing power has led to the decreasing usage of assumptions when modeling turbomachinery flows. Thus, modeling in 1D

and 2D was replaced by 3D computations, the Euler by the Navier-Stokes equations [6, 7], steady by unsteady computations and the use of turbulence models by LES (Large Eddy Simulation) [8–10]. However, in the industrial environment, some of the aforementioned simplifications are still being made in order to allow obtaining technical answers or performing designs within an acceptable timeframe.

Optimization [11, 12], more specifically shape optimization in the CFD context, aims at minimizing or maximizing a certain objective/cost function by iteratively modifying the initial geometry. Optimization methods can be broadly classified into two main approaches; stochastic and deterministic. *Stochastic methods* [13–15], mostly represented by evolutionary algorithms (EAs), mimic natural evolution operations, producing generations of designs until a convergence criterion is met. They are able to locate a global minimum within the design space and are suitable to capture the Pareto front in multi-objective optimization (MOO). On the other hand, they require a large number of objective function (CFD) evaluations. Thus, for complex design problems that involve a high computational cost per evaluation, they become prohibitive, at least in their standard form. Among the techniques that have been suggested to decrease this computational cost, contributions have been made by PhD theses completed (or running) in the Parallel CFD & Optimization Unit of the Laboratory of Thermal Turbomachines of the National Technical University of Athens (LTT/NTUA) [16, 17]

Deterministic methods [18–21] rely on the availability of the first- or even higher-order gradients of the objective function with respect to (w.r.t.) the design variables in order to perform optimization cycles [22, 23] according to methods such as steepest descent, conjugate gradients or Newton (exact or approximate) etc. Gradient-based methods could be trapped into local minima. This could be avoided by re-starting from a different initial design, which would naturally increase the optimization cost. Unlike EAs, they usually involve a smaller number of evaluations and addressing MOO problems requires transforming them to single objective optimization (SOO) problems using weights.

Each optimization method comes with its own advantages and disadvantages [24]. It is the analysis of the underlying problem and its properties that lead to the selection of a specific method to be used. In turbomachinery shape optimization, both stochastic [25, 26] and deterministic methods have extensively been used. When considering aerodynamic shape optimization of blades, baseline shapes are often considered to be near optimal and major shape changes are neither expected nor desired. Thus, an optimization method that searches for a local optimum is sufficient and gradient-based

approaches are quite popular in the field.

1.2 Gradient Computation via the Adjoint Method

A variety of different approaches may be used to compute the desired first-order gradient of the objective function w.r.t. the design variable vector. Using the *finite differences* method [27], each component of the design variable vector $\boldsymbol{a}$, which has N elements, is perturbed individually by an infinitesimally small value ϵ, defining a perturbed geometry that will be used to evaluate the objective function J. The corresponding element of the gradient vector, for a second-order scheme, is given by

$$\frac{dJ}{da_n} = \frac{J(a_1, a_2, \ldots, a_n + \epsilon, \ldots, a_{N-1}, a_N) - J(a_1, a_2, \ldots, a_n - \epsilon, \ldots, a_{N-1}, a_N)}{2\epsilon} \tag{1.1}$$

The process, which for CFD applications might be computationally demanding, needs to be performed for every element of the design variable vector and hence, the computational cost of the method is proportional to the number of design variables N. The method is inefficient for a large design vector and, at the same time, derivatives are sensitive to the size of the selected perturbation. Using a "too" small value for the perturbation can lead to computer round-off errors. On the other hand, using a large value for ϵ introduces a non-negligible truncation error. Practically, in most cases, more than one values of ϵ must be tried in order to find the most suitable.

Dependence on the perturbation step can be circumvented when the *complex variable method* [28, 29] is used instead. In this case, the gradient is computed as

$$\frac{dJ}{da_n} = \frac{Im[J(a_1, a_2, \ldots, a_n + i\epsilon, \ldots, a_{N-1}, a_N)]}{\epsilon} \tag{1.2}$$

where $i = \sqrt{-1}$ and Im is the imaginary part of any complex variable. However, the computational cost still scales with the number of design variables and the solver needs to be modified so as to support complex, instead of real variables.

The cost scales with the number of design variables also when using *forward/direct differentiation* [30, 31]. According to this, the flow equations are differentiated w.r.t. $\boldsymbol{a}$ and N linear systems need to be solved. This overcomes the dependency from ϵ but requires programming of software.

An alternative for efficiently computing the gradient is the *adjoint* method. Following this approach, the gradient is computed by solving an adjoint linear system

of equations just after the solution of the flow equations. Its fundamental property is that the computational cost is independent of the number of design variables, in contrast to all the aforementioned methods. The method is beneficial when the design variables significantly outnumber the objective functions. A vast majority of (turbomachinery) CFD applications share that characteristic. However, the method requires the development of a separate adjoint solver in accordance with the flow equations solver.

The adjoint method was introduced in fluid mechanics problems by Pironneau [32] and extended later by Jameson [33–36]. Over the last 25 years, it has been extensively used for initially external flow problems, such as airfoils and wings [37, 38]. The transition to internal flows and, in specific, turbomachinery came slightly later, when, for example, Yang et al. [39] applied it in 2D cascades considering an inviscid flow or Chung et al. [40] considered the shape optimization of the 3D Rotor 37. Applications of the adjoint method to turbomachinery cases followed by Wu et al. [41] and Papadimitriou and Giannakoglou [42–45]. The common element of these applications was that they were considering a single row only. Despite the fact that Denton [46] provided the mixing-plane technique in 1992 for flow prediction problems in multi-row configurations, adjoint solvers that supported multi-row applications appeared quite later. Prominent first examples are the work of Frey et al. [47], Wang and He [48, 49].

The adjoint method can be devised in two different ways. When the differentiation of the governing equations comes before the discretization, we refer to the continuous approach [50, 51]. When the flow equations are firstly discretized and then differentiated to produce the adjoint equations system directly in discrete form, we refer to the discrete approach [52–54]. Both approaches are used extensively in CFD and have their own advantages and disadvantages; comparisons can be found in [55–59]. In this work, the discrete adjoint approach is used.

Even if optimization cycles are not performed, the adjoint method can provide the designer with sensitivity maps over the existing design. Sensitivity maps express the derivatives of the objective function w.r.t. the normal displacement of all surface nodes. They are a useful tool for the designer since they highlight the areas of the geometry that contribute more towards a potential improvement of the objective function.

Apart from undertaking the gradient computation for shape optimization purposes, the adjoint method has also been used to tackle other CFD problems. Different usages are for error estimation and grid adaptation [60–62]. Error estimation refers to at-

tempting to quantify the difference between the discrete solution and the unknown exact solution. In grid adaptation, the adjoint-based estimation of the localized error is used as an indicator for the areas of the grid that need to be refined. Another popular application field of the adjoint method is uncertainty quantification [63–65], in which the propagation of uncertainties from the inputs (usually operating conditions) to the outputs (objective functions) is quantified. In order to compute the statistical moments of a quantity of interest, its derivatives are needed and because of its efficiency in computing derivatives, the adjoint method is used.

The benefits of using the adjoint method for CFD applications contributed to its increasing popularity. A clear view of this recognition in the scientific community over the years can be obtained by extracting data from the online databases/digital collections of the

- ASME (American Society of Mechanical Engineers) [1] and
- ARC (Aerospace Research Central) of AIAA (American Institute of Aeronautics and Astronautics) [2].

Published papers that refer to the adjoint method are retrieved. These papers are classified based on the year of publication and, also, on whether they refer to steady or unsteady flows (see section 1.3). In fig. 1.1, the per year number of papers that apply the adjoint method is plotted in a bar chart both for the entire ASME digital collection and the Journal of Turbomachinery, in specific. The same plot for the AIAA database can be found in fig. 1.2. The upward trend of the adjoint usage in scientific papers is apparent.

1.3 Motivation & Recent Advances in Unsteady Adjoint Solvers

Unsteadiness is an intrinsic characteristic of turbomachinery flows due to the interaction of rotating and stationary components. Other unsteady phenomena such as vortices and flow recirculations also dominate in turbomachinery flows and affect aerodynamic behavior. However, the steady state assumption along with the mixing plane

[1] https://asmedigitalcollection.asme.org
[2] https://arc.aiaa.org/search

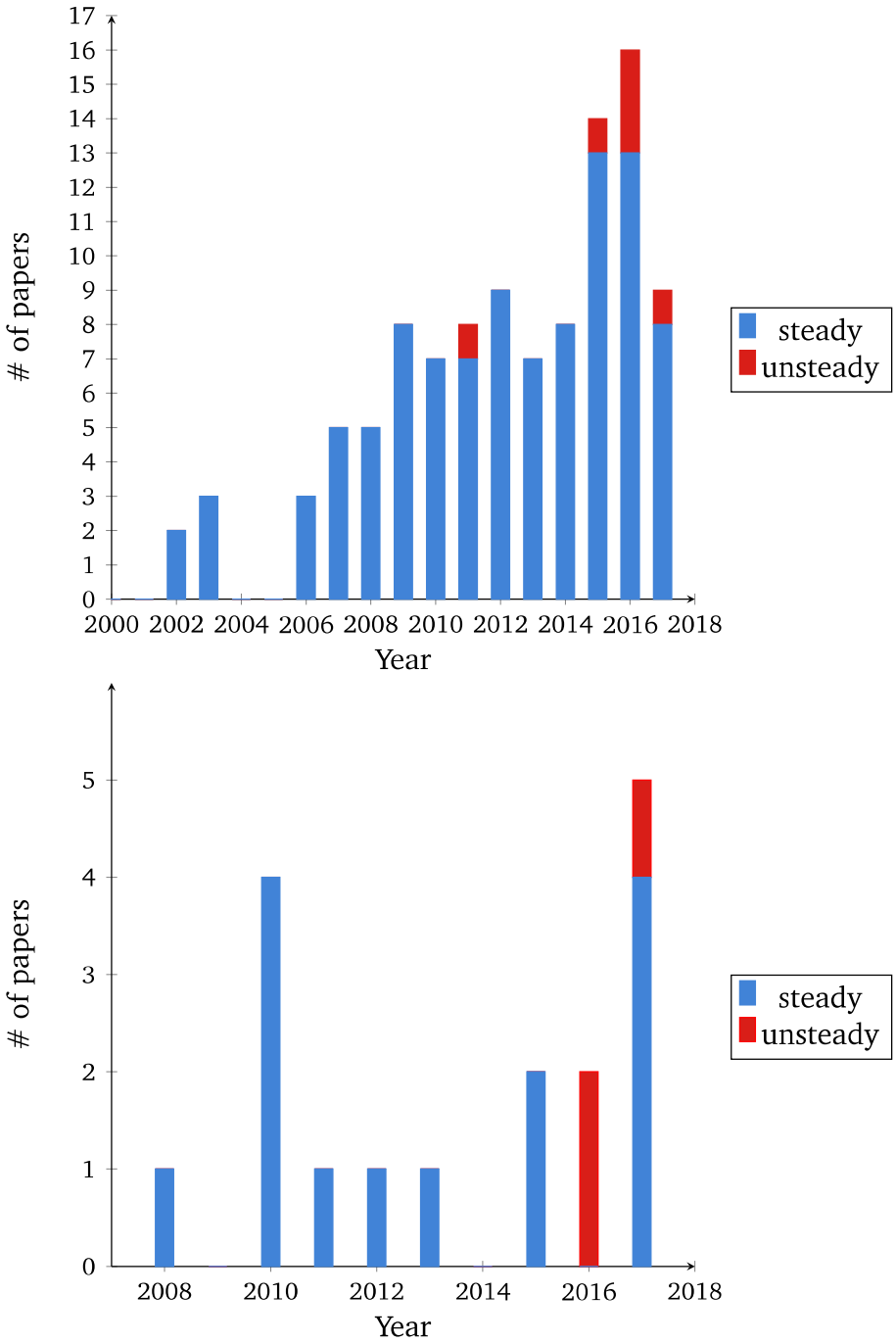


Figure 1.1: Research papers on the (steady and unsteady) adjoint method that appear in the ASME paper digital collection. Top: Entire ASME digital collection. Bottom: Journal of Turbomachinery.

technique [46] has been the dominant approach both for flow and adjoint flow computations. Restricted computational power and the large size of 3D turbomachinery

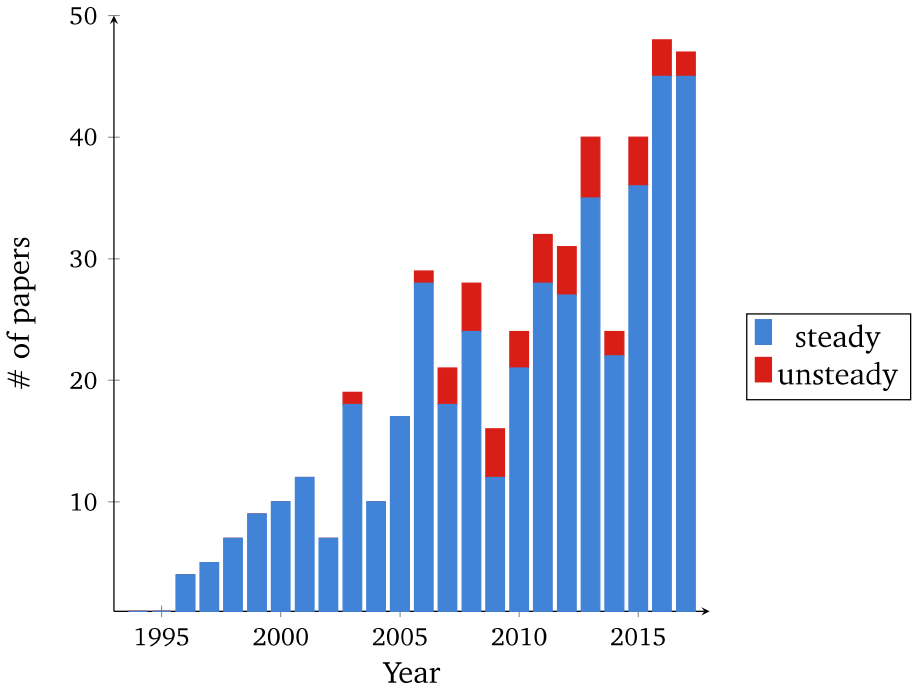


Figure 1.2: Research papers on the (steady and unsteady) adjoint method that appear in the AIAA paper digital collection.

cases calls for such a simplification in order to get solutions at a reasonable cost.

Nowadays, in an industrial environment, optimization problems demand modifying designs that are close to optimal and improvements are becoming increasingly hard to obtain. Improving, for example, the efficiency of a turbine or compressor stage by 0.1% is satisfactory for industrial applications. This is an indication that capturing flow phenomena more accurately is becoming a necessity that pushes towards the direction of considering unsteady flow computations for industrial turbomachinery optimization.

Another issue to face, when making the steady state assumption along with a discrete adjoint solver, is convergence difficulties. When solving an unsteady flow with a steady solver, the solver may not be able to converge to machine accuracy. If only the flow solution is needed and the objective function has converged to "engineering accuracy", the result is usually accepted by the engineer. However, the discrete adjoint solver is sensitive to the convergence of the flow equations and may diverge when the flow solver is not fully converged. For example, when a strong vortex shedding appears at the trailing edge of a blade, then steady flow convergence usually stagnates and the adjoint may diverge, as seen in fig. 1.3. As a remedy to this problem, the

GMRES [66] and RPM [67–69] algorithms were used to stabilize the flow solver. Another, less mathematical, solution would be to coarsen the grid close to the trailing edge to average out the vortex effect and, thus, achieve convergence while sacrificing accuracy.

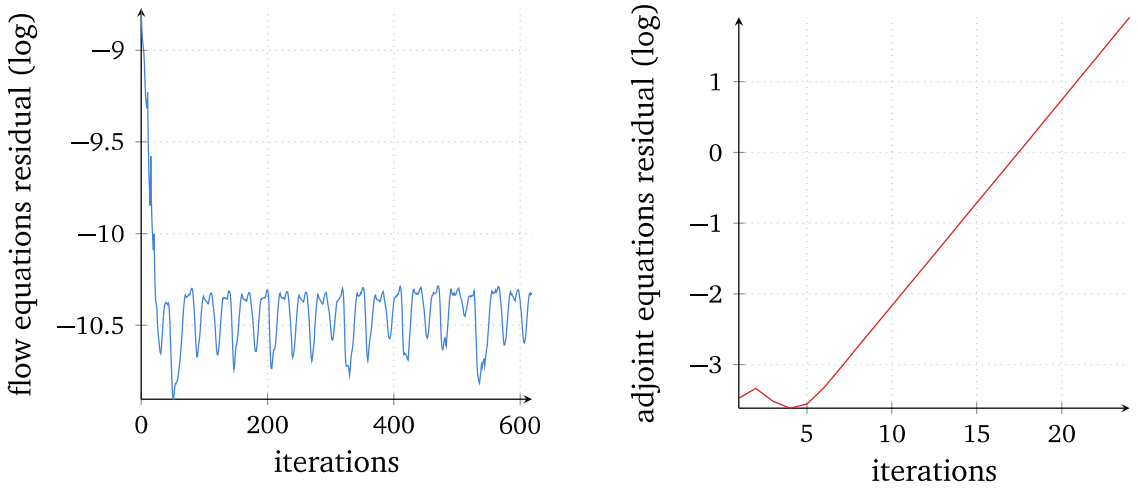


Figure 1.3: Flow solver stalling convergence leading to adjoint solver divergence. Left: Flow solver convergence. Right: Adjoint solver divergence.

The aforementioned, combined with the continuous advances in high-performance computing, enable the transition to unsteady solvers for the flow and adjoint equations for the design optimization of turbomachinery cases. Based on figs. 1.1 and 1.2, the tendency of using unsteady adjoint solvers over the last years is confirmed. The need for unsteady adjoint, especially for turbomachinery applications, is pronounced by the fact that, in the last years, the majority of papers on unsteady adjoint presented in the ASME Conference(s) are also published in the Journal of Turbomachinery.

The first papers on the application of the adjoint method to unsteady flows were published [70–74] at the beginning of the 2000s. Since then, a significant number of works has been published on unsteady adjoint. More specifically, it has been applied to a variety of external flow problems, 2D in the beginning and 3D later, mostly - but not exclusively - using the time-domain formulation. Applications involve flow past cylinders [75, 76], helicopter rotor blades [77–81], airfoils [82–102], wings [80, 103], full aircraft geometries [104] but also ducts [105, 106]. Most of them are concerned with fluid mechanics only while some other consider also the interaction with the structure [78, 104]. The computational grid is either static or moving. In the case of a moving grid, with or without considering overset grids [80], the Geometric Conserva-

tion Law [107] is incorporated into the equations to account for grid deformation. For turbomachinery flows, the vast majority of unsteady adjoint applications benefits from the periodic behavior of the flow in order to reduce the computational cost. The time-domain Fourier models or frequency domain methods, such as the harmonic balance method [108–114] are based on the Fourier representation of the unsteady solution, i.e. the unsteadiness is approximated using a mean value and the harmonics of user-prescribed frequencies. The space-time gradient [115] replaces the time derivative of the flow vector in the equations by a space derivative multiplied by the rotating speed. Such approaches neglect transient flow phenomena, such as the surge margin prediction, the transient flow behavior in cavities and the flow in the thrust reverser of a jet engine or periodic phenomena with frequencies different than the prescribed ones such as the vortex shedding of wakes downstream of trailing edges. Talnikar et al. [116] applied the adjoint method, formulated in the time-domain, using LES on a single-row turbine vane configuration. Preliminary work conducted within this PhD applied the unsteady adjoint method for URANS to a 3D stator blade [117] and a quasi-2D turbine stage [118]. To the author's knowledge, the time-domain unsteady adjoint method, with a time-domain formulation, is applied to 3D, multi-row turbomachinery applications for the first time in this thesis and in [119], which is a publication resulted by this PhD. To do so, the development of an interface that supports multi-row adjoint computations by coupling the grids of adjacent rows is required. This is a special variant of the overset grids [80].

It is well known, see also chapter 3, that the unsteady adjoint equations run backwards in time. That translates to having the unsteady flow available during the unsteady adjoint computations. For relatively large CFD cases, the required amount of RAM to keep the entire unsteady flow may quickly reach numbers that make unsteady adjoint computations impossible to run. In order to tackle this issue, the check-pointing technique or approximation models are used. In check-pointing [120–123], instead of storing the entire unsteady flow, only selected time-steps (checkpoints) are saved and the rest are recomputed when needed during the unsteady adjoint computation. Thus, run time increases in order to reduce storage requirements. In approximation models [124, 125], again the entire unsteady flow solution does not need to be stored. During the unsteady adjoint computation, the unsteady flow is reconstructed or approximated according to techniques such as the POD (Proper Orthogonal Decomposition). Approximation models reduce storage requirements and are faster than check-pointing algorithms, though less accurate. In [126], spatial and temporal

coarsening techniques were used to reduce the storage requirements of unsteady adjoint computations, which were also tested in the applications included in this thesis.

1.4 Research Objectives within the Research Framework of RR and LTT/NTUA

This work was conducted under the AboutFlow [127] Initial Training Network (ITN), funded by the European Commission, Marie Skłodowska-Curie FP7. The ITN funded 14 research positions focusing on robust adjoint solvers, the seamless integration of adjoint gradients into design chains and the application of unsteady adjoint in industrial design. This thesis evolves around the last axis. The research position combined the industrial placement in the Design Systems Engineering (DSE) group of Rolls-Royce Deutschland (RRD) and the academic placement in the LTT/NTUA.

The software used and expanded in this thesis is the Hydra CFD suite [128]. Hydra was originally developed by Prof. M. Giles and his research team in the period 1998-2004 at the Oxford University, using mainly Fortran 77, and was subsequently transferred to RR while receiving contributions from research groups at Cambridge, Imperial College, Loughborough, Surrey and Sussex Universities. It is extensively validated [129, 130] and has been used by RR since 2009 in almost every aspect of its aerodynamic design for turbomachinery components. RR products such as the Trent 1000 engine, fig. 1.4, which powers Boeing 787 airplanes, and the newer Trent XWB, which powers Airbus A350 aeroplanes, have made extensive use of Hydra's capabilities. Until the completion of this work, the Hydra suite included a steady and unsteady (time-domain formulation) flow equations solver and a steady adjoint solver.

LTT/NTUA has been developing and using both flow and adjoint solvers for over 25 years, as indicated by the completed PhD theses over that period. Papadimitriou [131] developed continuous and discrete adjoint methods for inviscid and viscous compressible flows. Asouti [132] extended the continuous and discrete adjoint methods for use with preconditioned flow equations for low-Mach number flows. Zymaris [133] formulated the continuous adjoint approach to the Spalart-Allmaras turbulence model and wall functions. Zervogiannis [134] developed a discrete adjoint approach to compute first- and second-order derivatives, using "hand differentiation", and performed a posteriori error analysis for the optimal adaptation of hybrid grids. Kontoleontos [16] developed the continuous adjoint approach to incompressible, turbu-

Figure 1.4: Rolls-Royce Trent1000 engine. © Rolls-Royce.

lent flows with the presence of heat transfer and to topology optimization problems. Papoutsis-Kiachagias [135] developed the continuous adjoint method to various turbulence models including wall functions, expanded the continuous adjoint for topology optimization to 3D flows using a truncated Newton algorithm to accelerate convergence and dealt with robust design problems. Finally, Kavvadias [136] developed the continuous adjoint to the k-ω SST turbulence model and a new approach to account for grid sensitivities in continuous adjoint, including the extension to unsteady flows.

The work of this thesis aims to contribute to the research activities of both parties, industrial and academic. Regarding the industrial party, in order to account for transient flow phenomena while computing gradients and complete the set of existing solvers in Hydra (steady and unsteady flow solvers and steady adjoint solver), an unsteady adjoint solver, formulated in the time-domain, was developed. The existing Hydra steady adjoint solver was naturally used as background tool for the unsteady adjoint solver. Regarding the academic party, this work can be viewed in parallel with the work presented in [136] and [137] despite the fact that, there, the unsteady adjoint solver was developed using the continuous (in contrast to the discrete) approach and the open source CFD software OpenFOAM [138] (in contrast to Hydra). In addition, the discrete adjoint developments presented in [131, 132, 134, 135] considered steady flows, while this work is extended to unsteady problems. There, the differential

terms were computed using "hand-differentiation" while, in this work, a combination of Algorithmic Differentiation (AD), Appendix A, and "hand-differentiation" is used to program the adjoint solver.

Summarizing, this thesis contributes to research efforts on CFD optimization for unsteady flows, by employing unsteady adjoint solvers. The work conducted within the thesis aims mainly to explore both the benefits and limits of using a discrete unsteady adjoint solver, which employs a time-domain formulation, to realistic, industrial, 3D turbomachinery applications. CFD computations for such cases is a daily routine for the engineers in the industrial environment.

The following points were considered during the development process of the solver:

- **Derivation of equations.** The discrete unsteady adjoint equations to be solved are derived so that the gradient of an objective function, which is defined over a specified time interval, be computed.

- **Iterative solving scheme.** In order to solve the unsteady adjoint equations in a consistent manner and correctly consider the extra unsteady terms, the adjoint version of the 5-stage Runge-Kutta scheme is derived and used.

- **Adjoint sliding interface.** An interface that ensures the communication between stationary and rotating grids is developed for the unsteady adjoint solver respecting the adjoint/reverse flow of information.

- **Handling increased size of data requirements.** In order to avoid RAM bottlenecks, a store-all-to-disk approach is selected. The implementation is faster than check-pointing algorithms and more accurate than approximation models but requires large disk capacities which, however, are relatively cheap comparing to RAM expansions. In addition, the temporal coarsening technique offers a reduction in time and storage space needed.

1.5 Thesis Outline

The present thesis consists of 5 chapters, including the introduction and conclusions chapters, which are summarized below.

In chapter 2, the Navier-Stokes equations with the Spalart-Allmaras turbulence model are presented as the governing flow equations. The equations are discretized and boundary conditions are imposed. A dual time-stepping technique which employs

5-stage Runge-Kutta is employed to solve them and geometric multigrid is used to accelerate convergence. The parameterization and grid generation are briefly described as the remaining steps of the simulation setup, which starts from the design variables and ends up with the objective function value.

In chapter 3, the unsteady adjoint equations are formulated starting from an objective function defined over a specific time interval and the corresponding boundary conditions are derived. A consistent numerical iterative scheme, adjoint to the 5-stage Runge Kutta, is derived in order to solve the adjoint equations and Algorithmic Differentiation is employed to compute the differential terms that appear in the adjoint equations. Moreover, the adjoint sliding interface is developed to enable the communication between rotating and stationary domains and the programming implementation is discussed. The temporal coarsening method is used to reduce the storage requirements of the adjoint solver and the specific case of periodic flows is examined by means of a simplified example. The chapter also presents the integration of the flow and adjoint flow solvers in the gradient computation process and an optimization setup. Lastly, the computational cost of the adjoint solver is discussed.

In chapter 4, two cases are initially used to validate the flow solver. Then, the unsteady adjoint solver is applied to a turbine vane with transient flow, a turbine stage and a three-row compressor case with periodic flows. The gradients computed via unsteady adjoint are compared against those computed by finite differences for the turbine vane and stage cases. Optimization cycles, with and without considering constraints, are performed using the gradients computed via unsteady adjoint to improve the geometries. Finally, the temporal coarsening technique is used for the multi-row cases and the gradients are compared with the reference gradients computed without it.

In Appendix A, the principles of algorithmic differentiation are presented. Two implementations are introduced and one of them, namely the source code transformation, which is used in this work, is further discussed using an example.

2

Governing Flow Equations: Formulation and Numerical Solution

In this chapter, the unsteady Navier-Stokes equations and the Spalart-Allmaras [139] turbulence model are presented as the flow governing equations of the CFD computation. The governing equations are discretized appropriately on the control volumes of an unstructured grid, according to the vertex-centered finite volume scheme. Boundary conditions are imposed to achieve the closure of the CFD problem. The solution of the system is reached iteratively by employing dual time-stepping, an outer real time loop and an inner pseudo-time loop, where the 5-stage Runge-Kutta scheme is combined with geometric multigrid for convergence acceleration. The pseudo-code of the flow solver is also given. Finally, the remaining steps of the flow problem set up are given by presenting how the objective function is computed for an unsteady flow problem and providing information about shape parameterization and grid generation.

2.1 Unsteady 3-D Navier-Stokes Equations

The principles of motion of a viscous compressible fluid through turbomachinery rows can be mathematically represented by coupling the unsteady 3D momentum conservation equations along with the continuity equation and the energy equation, forming the so-called Navier-Stokes equations' system [6, 7]. The equations are solved in the Cartesian coordinate system considering the relative velocity. A convention used throughout the thesis is that bold small letters indicate vectors while bold capital letters indicate matrices. In conservative vectorial form and using Einstein's notation (twice repeated indices imply summation over $i = 1, 2, 3$), the Navier-Stokes equations are expressed as

$$\frac{\partial \boldsymbol{u}}{\partial t} + \frac{\partial \boldsymbol{f}_i^{inv}}{\partial x_i} - \frac{\partial \boldsymbol{f}_i^{visc}}{\partial x_i} = \boldsymbol{s} \tag{2.1}$$

where $\boldsymbol{u}$ denotes the conservative variables vector, $\boldsymbol{f}_i^{inv}$ the inviscid flux vector, $\boldsymbol{f}_i^{visc}$ the viscous flux vector, t the time, $\boldsymbol{x} = [x_1, x_2, x_3]^T$ the Cartesian coordinates and $\boldsymbol{s}$ the source terms (if any). The involved vectors are defined as follows

$$\boldsymbol{u} = \begin{bmatrix} \varrho \\ \varrho w_1 \\ \varrho w_2 \\ \varrho w_3 \\ \varrho E \end{bmatrix}, \qquad \boldsymbol{f}_i^{inv} = \begin{bmatrix} \varrho w_i \\ \varrho w_1 w_i + p\delta_{i1} \\ \varrho w_2 w_i + p\delta_{i2} \\ \varrho w_3 w_i + p\delta_{i3} \\ \varrho E w_i + p w_i \end{bmatrix}, \qquad \boldsymbol{f}_i^{visc} = \begin{bmatrix} 0 \\ \tau_{1i} \\ \tau_{2i} \\ \tau_{3i} \\ w_j \tau_{ji} + q_i \end{bmatrix} \tag{2.2}$$

where ϱ is the density, $\boldsymbol{w} = [w_1, w_2, w_3]^T$ is the Cartesian relative velocity vector, p is the pressure, δ_{ij} is the Kronecker delta function, E is the relative total energy per unit mass and τ is the stress tensor.

The absolute Cartesian velocity vector $\boldsymbol{v} = [v_1, v_2, v_3]^T$ is defined as

$$\boldsymbol{v} = \boldsymbol{w} + \boldsymbol{\omega} \times \boldsymbol{x} \tag{2.3}$$

where $\boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3]^T$ is the angular velocity vector. In a rotating domain, solving for the relative velocity, the source term $\boldsymbol{s}$ accounts for the Coriolis and centrifugal

forces

$$\boldsymbol{s} = \begin{bmatrix} 0 \\ -\varrho\left(\omega_2 v_3 - \omega_3 v_2\right) \\ -\varrho\left(\omega_3 v_1 - \omega_1 v_3\right) \\ -\varrho\left(\omega_1 v_2 - \omega_2 v_1\right) \\ 0 \end{bmatrix} \tag{2.4}$$

In the present work, the rotor axis coincides with the x_1 axis, thus the angular velocity can be re-written as $\boldsymbol{\omega} = [\omega_1, 0, 0]^T$ or simply ω hereafter. The absolute velocity $\boldsymbol{v}$ can be expressed in terms of the relative velocity $\boldsymbol{w}$ as

$$\boldsymbol{v} = \begin{bmatrix} w_1 \\ w_2 - \omega x_3 \\ w_3 + \omega x_2 \end{bmatrix} \tag{2.5}$$

and the source terms reduce to

$$\boldsymbol{s} = \begin{bmatrix} 0 \\ 0 \\ \varrho\,\omega(w_3 + \omega x_2) \\ -\varrho\,\omega(w_2 - \omega x_3) \\ 0 \end{bmatrix} \tag{2.6}$$

The relative total energy per unit mass is given by

$$E = \frac{p}{\varrho}\frac{1}{\gamma - 1} + \frac{1}{2} w_i w_i - \frac{1}{2} r^2 \omega^2 \tag{2.7}$$

where $r^2 = x_2^2 + x_3^2$.

Assuming an isotropic Newtonian fluid and following Stokes' hypothesis for the viscosity [140], the stress tensor τ_{ij} is given by

$$\tau_{ij} = \mu\left(\frac{\partial w_i}{\partial x_j} + \frac{\partial w_j}{\partial x_i}\right) + \lambda \delta_{ij} \frac{\partial w_k}{\partial x_k}, \quad \lambda = -\frac{2}{3}\mu \tag{2.8}$$

where μ is the dynamic viscosity. The heat flow q_i is modeled using the Fourier law

$$q_i = k \frac{\partial T}{\partial x_i} \tag{2.9}$$

where $k = \frac{c_p \mu}{Pr}$ is the coefficient of thermal conductivity, c_p is the specific heat capacity under constant pressure and Pr is the Prandtl number with $Pr = 0.72$ for air.

The Navier-Stokes equations, along with the turbulence model equation presented below, are solved in non-dimensional form.

2.2 Turbulence Modeling

Turbulent flows are characterized by chaotic 3D fluctuations of the flow quantities throughout time and space. Turbulence should be taken into consideration in CFD computations since it is responsible for energy dissipation, mixing, momentum diffusion, high momentum convection etc., thus affecting the flow quantities to be predicted. Turbulence involves a wide range of length and time scales. Fully resolving them using extremely fine grids (Direct Numerical Simulation-DNS [141–145]) is computationally very expensive and almost prohibitive for complex geometries and industrial time-scales.

Alternatively, one may construct a model to predict the effects of turbulence without resolving it. Following the Boussinesq hypothesis [146], the averaged equations take the same form as the Navier-Stokes equations if the definition of viscosity is modified to incorporate both molecular and turbulent contributions as

$$\mu = \mu_l + \mu_t \tag{2.10}$$

where μ_t is the eddy viscosity and thermal conductivity k is transformed to

$$k = k_l + k_t = c_p \left(\frac{\mu_l}{Pr_l} + \frac{\mu_t}{Pr_t} \right) \tag{2.11}$$

with $Pr_t = 0.9$ for air. In order to close the problem, extra equation(s) which form the turbulence model must be solved. The Unsteady Reynolds-Averaged Navier Stokes equations coupled with the turbulence model equation(s) gives rise to the so-called URANS system.

Below, the unsteady one-equation Spalart-Allmaras turbulence model [139] used in this thesis is presented.

2.2.1 The Spalart-Allmaras Turbulence Model

The Spalart-Allmaras turbulence model [139] introduces one extra transport equation to solve for a kinematic viscosity like variable $\tilde{\nu}$, to be referred to as the Spalart-Allmaras variable, from which the turbulent eddy viscosity is computed as

$$\mu_t = \tilde{\mu} f_{v1} \tag{2.12}$$

where

$$\tilde{\mu} = \varrho\,\tilde{\nu}, \quad f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3}, \quad \chi = \frac{\tilde{\mu}}{\mu} \tag{2.13}$$

The equation of the model is

$$\begin{aligned} \frac{\partial \tilde{\nu}}{\partial t} + w_i \frac{\partial \tilde{\nu}}{\partial x_i} = \frac{1}{\sigma}\left(\frac{\partial}{\partial x_i}\left[(\nu + \tilde{\nu})\frac{\partial \tilde{\nu}}{\partial x_i}\right] + c_{b2}\left(\frac{\partial \tilde{\nu}}{\partial x_i}\right)^2\right) \\ + c_{b1}\tilde{S}\tilde{\nu} - \left(c_{w1} f_w - \frac{c_{b1}}{\kappa^2} f_{t2}\right)\left(\frac{\tilde{\nu}}{d}\right)^2 + f_{t1}\Delta w^2 \end{aligned} \tag{2.14}$$

where $\nu = \mu/\varrho$ is the kinematic viscosity. The last term in eq. 2.14, called trip term, provides a means for triggering transition at a specified location. However, during this work, the flow is fully turbulent and the trip term is set to zero. Coefficients are defined as follows

$$\tilde{S} = S + \frac{\tilde{\nu}}{\kappa^2 d^2} f_{v2}, \quad S = \sqrt{2\Omega_{ij}\Omega_{ij}}, \quad \Omega_{ij} = \frac{1}{2}\left(\frac{\partial w_i}{\partial x_j} - \frac{\partial w_j}{\partial x_i}\right),$$

$$f_{v2} = 1 - \frac{\chi}{1 + \chi f_{v1}}, \; f_w = g\left(\frac{1 + c_{w3}^6}{g^6 + c_{w3}^6}\right), \quad g = r + c_{w2}\left(r^6 - r\right), \quad r = \frac{\tilde{\nu}}{\tilde{S}\kappa^2 d^2}$$

where d denotes the distance to the nearest wall and constants are

$$c_{b1} = 0.1355, \qquad c_{b2} = 0.0622, \qquad \sigma = \frac{2}{3}, \qquad c_{v1} = 7.1,$$

$$c_{w1} = \frac{c_{b1}}{\kappa^2} + \frac{1 + c_{b2}}{\sigma}, \qquad c_{w2} = 0.3, \qquad c_{w3} = 2, \qquad \kappa = 0.41$$

Wall Functions

One of the challenges when modeling turbulence is how to treat the thin near-wall sublayer where the viscous effects become important. The most reliable way is to use a fine grid in the near-wall area which, however, may become expensive for 3D

geometries and might additionally lead to high aspect ratio cells. On the other side, the use of a coarser grid may lead to the incorrect computation of the wall shear stress. As a remedy for coarser grids, such as in the cases presented in this thesis (chapter 4), where the dimensionless distance y^+, defined in eq. 2.17, of the first node off the wall is high, wall-functions [147] are used. The original definition of the wall shear stress value, expressed in the discrete form is

$$\tau_w = \mu_w \frac{\Delta w}{y} \tag{2.15}$$

where y is the distance from the wall and Δw the velocity difference between the wall node and the first node off the wall. Since the relative velocity on the wall nodes is set to zero, section 2.4, Δw can be replaced simply by w at the first node off the wall. In this expression (eq. 2.15), considering that w varies linearly between the first node off the wall and the wall node leads to an incorrect estimation of τ_w. To avoid this, an appropriately modified dynamic viscosity μ_w is used in the momentum equations to produce the correct wall shear stress values. The equation that models the near-wall sublayer, firstly presented by Spalding [148], is written as

$$y^+ = w^+ + e^{-\kappa B}\left[e^{\kappa w^+} - 1 - \kappa w^+ - \frac{1}{2}\left(\kappa w^+\right)^2 - \frac{1}{6}\left(\kappa w^+\right)^3\right] \tag{2.16}$$

where $\kappa = 0.41$, $B = 5.3$, the dimensionless distance from the wall y^+ and the velocity w^+ are defined as

$$y^+ = \frac{y \varrho w_\tau}{\mu}, \quad w^+ = \frac{w}{w_\tau} \tag{2.17}$$

and the friction velocity w_τ is defined as

$$w_\tau = \sqrt{\frac{\tau_w}{\varrho}} \tag{2.18}$$

Initially, Spalding's equation 2.16 is solved iteratively to obtain the value of w^+. Then, to compute the modified μ_w value to be used in the momentum equations, equations 2.15, 2.17 and 2.18 are combined to yield

$$\mu_w = \frac{y^+}{w^+}\mu \tag{2.19}$$

2.3 Discretization of the Flow Equations

The discretization of the URANS equations presented here is performed on unstructured grids. The integration of eq. 2.1 is done for **node/vertex-centered** median dual control volumes. Each control volume V_P is defined by connecting the middle points of the edges around the central node P with the centroids of the grid cells this node belongs to, as in fig. 2.1. Integrating eq. 2.1 over the control volume V_P forms the residual $\boldsymbol{r}_{n,P}$ that corresponds to the control volume associated with node P for the time-step n,

$$\boldsymbol{r}_{n,P} = \underbrace{\iiint_{V_P} \left.\frac{\partial \boldsymbol{u}_P}{\partial t}\right|_n dV}_{temporal} + \underbrace{\iiint_{V_P} \left.\frac{\partial \boldsymbol{f}_{Pi}^{inv}}{\partial x_i}\right|_n dV}_{inviscid} - \underbrace{\iiint_{V_P} \left.\frac{\partial \boldsymbol{f}_{Pi}^{visc}}{\partial x_i}\right|_n dV}_{viscous} - \underbrace{\iiint_{V_P} \boldsymbol{s}_{n,P} dV}_{sources} \tag{2.20}$$

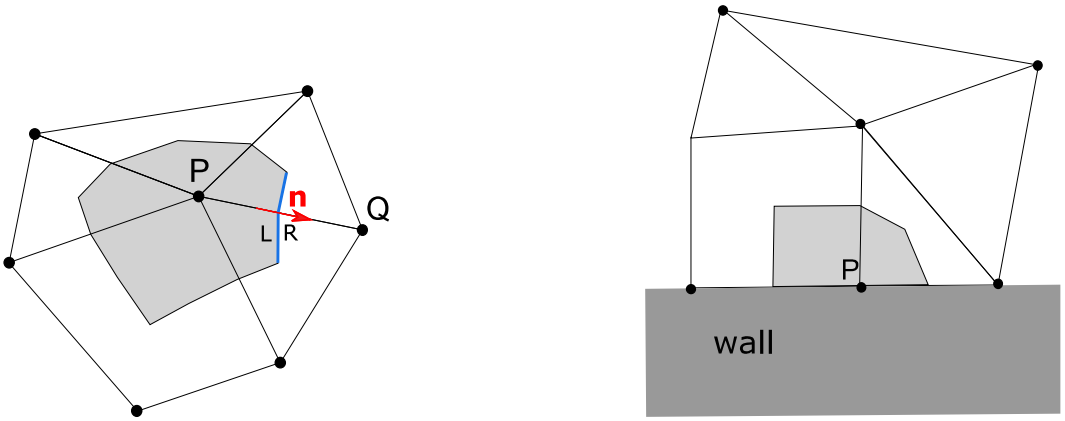


Figure 2.1: Vertex-centered control volumes (a 2D example). Left: Internal control volume. Right: Boundary control volume.

The next subsections describe how fluxes are assembled for internal and boundary nodes and, then, how the different terms in the residual are computed for each control volume. Subscript n is omitted for simplicity.

2.3.1 Discretization of Inviscid Fluxes

By applying the Green-Gauss theorem to the inviscid term, it occurs that

$$\boldsymbol{r}_P^{inv} = \iiint_{V_P} \frac{\partial \boldsymbol{f}_i^{inv}}{\partial x_i} dV = \oiint_{\partial V_P} \boldsymbol{f}_i^{inv} \hat{n}_i d(\partial V) \tag{2.21}$$

where $\hat{\boldsymbol{n}}$ is the outward unit normal vector to the control volume boundary ∂V.

Fluxes across the (inter)face associated with the edge between the central node P and its neighbor Q, at $\frac{1}{2}\left(\boldsymbol{x}_P + \boldsymbol{x}_Q\right)$, are computed as shown in the 2D case of fig. 2.1. There is obviously a one-to-one correspondence between faces and edges.

In the discrete sense, the inviscid nodal (node P) residual is approximated by the expression

$$\boldsymbol{r}_P^{inv} \simeq \sum_{Q \in E_P} \boldsymbol{\phi}_{PQ}^{inv} S_{PQ} \tag{2.22}$$

where $\boldsymbol{\phi}_{PQ}^{inv} = \boldsymbol{f}_{i,PQ}^{inv} \cdot \hat{n}_i$ is the inviscid flux vector and S_{PQ} represents the area of an internal face associated with edge (PQ) and S_b is the area of the boundary face if P is a boundary node with $\boldsymbol{\phi}_P^{inv}$ being the corresponding boundary inviscid flux. The set of the neighboring nodes of node P is denoted by E_P and the set of boundary faces by B_P.

Let $\boldsymbol{u}^L$ and $\boldsymbol{u}^R$ be the flow variable values' vectors on the left and right side of the face between volumes of P and Q. Based on the Roe's scheme [149], the flux from P to Q can be obtained by

$$\boldsymbol{\phi}_{PQ}^{inv} = \frac{1}{2}\left(\underbrace{\boldsymbol{\phi}^{inv}\left(\boldsymbol{u}^L\right) + \boldsymbol{\phi}^{inv}\left(\boldsymbol{u}^R\right)}_{convective\ term} - \underbrace{\left|\bar{\boldsymbol{A}}_{PQ}\right|\left(\boldsymbol{u}^R - \boldsymbol{u}^L\right)}_{dissipative\ term} \right) \tag{2.23}$$

where $\left|\bar{\boldsymbol{A}}_{PQ}\right|$ is the Jacobian matrix computed by the Roe-averaged flow variables at the face between nodes P and Q. To obtain the values of $\boldsymbol{u}^L$ and $\boldsymbol{u}^R$, a Taylor expansion is needed

$$\begin{aligned} \boldsymbol{u}^L &= \boldsymbol{u}_P + \frac{1}{2}(\boldsymbol{PQ}) \cdot \nabla \boldsymbol{u}_P \\ \boldsymbol{u}^R &= \boldsymbol{u}_Q - \frac{1}{2}(\boldsymbol{PQ}) \cdot \nabla \boldsymbol{u}_Q \end{aligned} \tag{2.24}$$

where $\boldsymbol{u}_P$ and $\boldsymbol{u}_Q$ are the flow variable vectors at P and Q respectively and $\nabla \boldsymbol{u}$ is the spatial gradient of the flow variable vector. However, to avoid the computation of $\boldsymbol{u}^L$,

$\boldsymbol{u}^R$ for the convective term, the following approximation is used

$$\begin{aligned}\boldsymbol{\phi}_{PQ}^{inv} &= \frac{1}{2}\left(\boldsymbol{\phi}^{inv}\left(\boldsymbol{u}_P\right)+\boldsymbol{\phi}^{inv}\left(\boldsymbol{u}_Q\right)-\left|\bar{\boldsymbol{A}}_{PQ}\right|\left(\boldsymbol{u}^R-\boldsymbol{u}^L\right)\right)\\ &= \frac{1}{2}\left(\boldsymbol{A}_P\boldsymbol{u}_P+\boldsymbol{A}_Q\boldsymbol{u}_Q-\left|\bar{\boldsymbol{A}}_{PQ}\right|\left(\boldsymbol{u}^R-\boldsymbol{u}^L\right)\right)\end{aligned} \tag{2.25}$$

To develop the dissipative term, inspiration is drawn from the corresponding approximation formula for a structured 1D stencil of equidistant consecutive points $\boldsymbol{x}_{j^+}$, $\boldsymbol{x}_j$, $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i^-}$ and the corresponding flow variable vectors $\boldsymbol{u}_{j^+}$, $\boldsymbol{u}_j$, $\boldsymbol{u}_i$ and $\boldsymbol{u}_{i^-}$ is considered at first. A family of four-point schemes [150] can be employed to express the dissipation term for this stencil which, by ignoring the application of limiters, can be expressed as

$$\left|\bar{\boldsymbol{A}}_{ij}\right|\left(\boldsymbol{u}^R-\boldsymbol{u}^L\right)=\frac{1}{2}(1-\kappa)\left|\bar{\boldsymbol{A}}_{ij}\right|\left[\left(\frac{1}{2}\boldsymbol{u}_{j^+}-\boldsymbol{u}_j+\frac{1}{2}\boldsymbol{u}_i\right)-\left(\frac{1}{2}\boldsymbol{u}_j-\boldsymbol{u}_i+\frac{1}{2}\boldsymbol{u}_{i^-}\right)\right] \tag{2.26}$$

where $\kappa\in[0,1]$ represents a one-parameter family of second-order schemes; herein, $\kappa=1/2$. Using the operator

$$\mathcal{L}_P(\boldsymbol{u})=\frac{1}{\#(E_P)}\sum_{Q\in E_P}\left(\boldsymbol{u}_Q-\boldsymbol{u}_P\right) \tag{2.27}$$

where $\#(E_P)$ is the number of faces of the element P and based on eq. 2.26, eq. 2.25 can be rewritten as

$$\boldsymbol{\phi}_{PQ}^{inv}=\frac{1}{2}\left\{\left[\boldsymbol{\phi}^{inv}\left(\boldsymbol{u}_P\right)+\boldsymbol{\phi}^{inv}\left(\boldsymbol{u}_Q\right)\right]-\frac{1}{2}(1-\kappa)\left|\bar{\boldsymbol{A}}_{PQ}\right|\left[\mathcal{L}_Q(\boldsymbol{u})-\mathcal{L}_P(\boldsymbol{u})\right]\right\} \tag{2.28}$$

The $\mathcal{L}_P$ operator, defined in eq. 2.27, was introduced in [151, 152]. However, a modified formula, proposed in [153] to ensure that the operator retains second order accuracy on stretched grids, is used here; the modified operator $\mathcal{L}_P^*$ is given by

$$\mathcal{L}_P^*(\boldsymbol{u})=\hat{\mathcal{L}}_P(\boldsymbol{u})-\nabla\boldsymbol{u}_P\;\hat{\mathcal{L}}_P(\boldsymbol{x}) \tag{2.29}$$

where the gradient $\nabla\boldsymbol{u}_P$ is approximated by

$$\nabla\boldsymbol{u}_P=\sum_{Q\in E_P}\frac{1}{2}\left(\boldsymbol{u}_Q+\boldsymbol{u}_P\right)\hat{\boldsymbol{n}}_{PQ}S_{PQ}+\sum_{Q_b\in B_P}\boldsymbol{u}_P\hat{\boldsymbol{n}}_bS_b \tag{2.30}$$

and $\hat{\mathcal{L}}_P(\boldsymbol{u})$, $\hat{\mathcal{L}}_P(\boldsymbol{x})$ are computed by, [154],

$$\hat{\mathcal{L}}_P(\zeta) = \left(\sum_{Q\in E_P} \frac{1}{|\boldsymbol{x}_Q - \boldsymbol{x}_P|}\right)^{-1} \sum_{Q\in E_P} \frac{\zeta_Q - \zeta_P}{|\boldsymbol{x}_Q - \boldsymbol{x}_P|} \tag{2.31}$$

In order to deal with discontinuities such as shock waves and ensure stability, similarly to what presented in [151, 155], the flux is re-written as

$$\begin{aligned}\boldsymbol{\phi}_{PQ}^{inv} =& \frac{1}{2}\left[\boldsymbol{\phi}^{inv}(\boldsymbol{u}_P) + \boldsymbol{\phi}^{inv}(\boldsymbol{u}_Q)\right] \\ &-\frac{1}{4}\left|\bar{\boldsymbol{A}}_{PQ}\right|\left[-\frac{1}{3}(1-\Omega)\left(\mathcal{L}_P^*(\boldsymbol{u}) - \mathcal{L}_Q^*(\boldsymbol{u})\right) + \Omega\left(\boldsymbol{u}^L - \boldsymbol{u}^R\right)\right]\end{aligned} \tag{2.32}$$

where

$$\Omega = min\left(\epsilon'\left|\frac{p_Q - p_P}{p_P + p_Q}\right|^2, 1\right) \tag{2.33}$$

and ϵ' is a user-defined constant, here $\epsilon' = 8$. The additional term offers stability by becoming dominant in discontinuities due to the way Ω is defined.

2.3.2 Discretization of Viscous Fluxes

Similarly to the inviscid term, the Green-Gauss theorem is applied to the viscous term

$$\boldsymbol{r}_P^{visc} = \iiint_{V_P} \frac{\partial \boldsymbol{f}_{Pi}^{visc}}{\partial x_i} dV = \iint_{\partial V_P} \boldsymbol{f}_{Pi}^{visc} \hat{n}_i d(\partial V) \tag{2.34}$$

or

$$\boldsymbol{r}_P^{visc} = \sum_{Q\in E_P} \boldsymbol{\phi}_{PQ}^{visc} S_{PQ} \tag{2.35}$$

where $\boldsymbol{\phi}_{PQ}^{visc}$ is the viscous flux per unit area vector in the normal to the finite volume boundary direction

$$\boldsymbol{\phi}_{PQ}^{visc} = \begin{bmatrix} 0 \\ \tau_{1i}^{PQ}\hat{n}_i \\ \tau_{2i}^{PQ}\hat{n}_i \\ \tau_{3i}^{PQ}\hat{n}_i \\ w_j\tau_{ji}^{PQ}\hat{n}_i + q_i\hat{n}_i \end{bmatrix} \tag{2.36}$$

where τ_{1i}^{PQ} is the stress tensor as defined by eq. 2.8. The computation of the viscous flux requires the approximation of $\nabla \boldsymbol{u}$ at the midpoint of each edge, which is computed by the following expression [156]

$$\nabla \boldsymbol{u}_{PQ} = \frac{1}{2}\left(\nabla \boldsymbol{u}_P + \nabla \boldsymbol{u}_Q\right) - \left(\frac{1}{2}\left(\nabla \boldsymbol{u}_P + \nabla \boldsymbol{u}_Q\right)(\hat{\boldsymbol{PQ}}) - \frac{\left(\boldsymbol{u}_Q - \boldsymbol{u}_P\right)}{\left|\boldsymbol{x}_Q - \boldsymbol{x}_P\right|}\right)(\hat{\boldsymbol{PQ}}) \tag{2.37}$$

where

$$(\hat{\boldsymbol{PQ}}) = \frac{\boldsymbol{x}_Q - \boldsymbol{x}_P}{\left|\boldsymbol{x}_Q - \boldsymbol{x}_P\right|}$$

2.3.3 Discretization of the Temporal Term

The time derivative is approximated by the second-order backward difference formula [157]

$$\left.\frac{\partial \boldsymbol{u}}{\partial t}\right|_n = \frac{3\boldsymbol{u}^n - 4\boldsymbol{u}^{n-1} + \boldsymbol{u}^{n-2}}{2\Delta t} \tag{2.38}$$

In the case of a stationary grid, where the control volume is constant, the temporal term in eq. 2.20 is computed by

$$\iiint_{V_P} \left.\frac{\partial \boldsymbol{u}_P}{\partial t}\right|_n dV = \frac{3\boldsymbol{u}_P^n - 4\boldsymbol{u}_P^{n-1} + \boldsymbol{u}_P^{n-2}}{2\Delta t} V_P \tag{2.39}$$

2.4 Boundary Conditions

Walls

Concerning the nodes that lay along solid walls, the components of the flow variable vector that correspond to the velocity and turbulence are set to zero. At the same time, the corresponding components of the residual vector are discarded so as not to update the initial flow variable vector for wall nodes at these components. If a 3D compressible flow is considered along with a one-equation turbulence mode and B is

a matrix that extracts the components of velocity and turbulence from wall nodes

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.40}$$

I the identity matrix and $\boldsymbol{r}_{wall}$ the residual vector of the wall nodes, the expressions that summarize the aforementioned can be written as follows

$$\begin{aligned} (I-B)\,\boldsymbol{r}_{wall} &= 0 \\ B\boldsymbol{u}_{wall} &= 0 \end{aligned} \tag{2.41}$$

In the cases studied in this thesis, the solid wall boundaries are considered adiabatic; so, the wall heat flux is set to zero.

Periodic

The grid is generated so that the nodes are matched in pairs across the periodic boundaries. The residual at the periodic nodes is formed by summing contributions from the finite volume integrals of the matching nodes. By ensuring the residual is identical at matching nodes along with the flow values and their spatial gradients, periodicity is enforced.

Inlet/Outlet

For the inlet/outlet, the boundary condition is imposed using the inviscid flux in eq. 2.23

$$\boldsymbol{\phi}_b^{inv} = \frac{1}{2}\left(\boldsymbol{\phi}\left(\boldsymbol{u}_{in/out}\right) + \boldsymbol{\phi}\left(\boldsymbol{u}_b\right) - \left|\bar{A}_b\right|\left(\boldsymbol{u}_b - \boldsymbol{u}_{in/out}\right)\right) \tag{2.42}$$

where subscript b denotes the boundary face and the definition of $\boldsymbol{u}_{in/out}$ depends on the case. For subsonic inflow and outflow boundary conditions, total pressure and temperature along with the flow angles are specified at the inlet together with the static pressure at the outlet.

2.4.1 Sliding Interface

In turbomachinery flows, CFD simulations often need to simultaneously account for adjacent rows of blades. Grids are though generated for each single blade row, separately. Thus, across the interface, flow quantities are exchanged between successive rotating and stationary rows (rotor-stator interface).

Before elaborating on the sliding interface, a brief overview of the mixing interface method, which is used in steady computations, is given. In steady computations, the domain of each row is separately solved in its relative frame of reference. Since the flow is considered steady, pseudo-time is employed to converge the flow equations similarly to what is described in section 2.5. At each pseudo-iteration, flow-field data from adjacent domains are communicated and operate as boundary conditions which are circumferentially averaged at the mixing interface.

For unsteady simulations, the sliding interface technique [158–160] is used which is a special type of overset grids [161]. Its parallelization is based on the OPlus library [162,163], section 2.7. The method requires that in each row an appropriate number

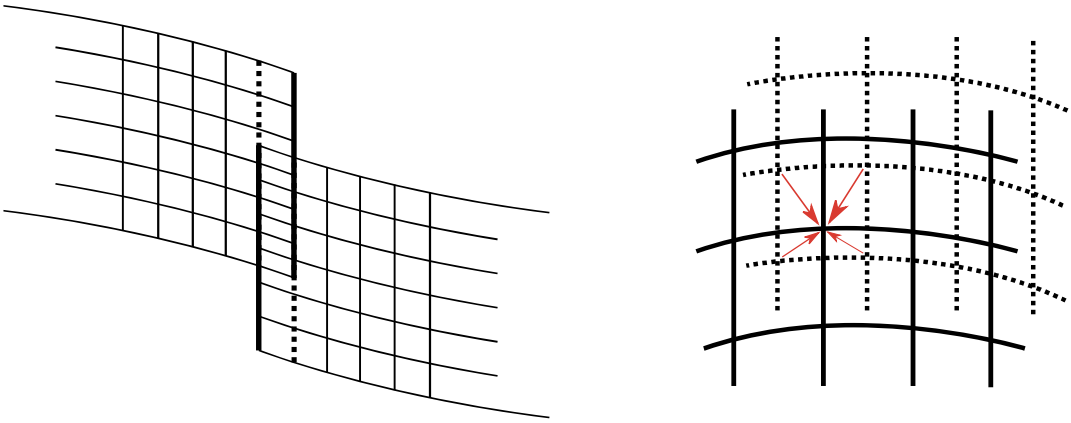


Figure 2.2: Simplified 2D sliding interface at radial and axial cut (thick dashed line: interior sliding plane (donors), thick line: exterior sliding plane (receivers), arrows: interpolation direction) Left: Radial cut. Right: Axial cut.

of blade-passages is considered for the simulation so that each row-domain is of equal circumferential pitch. The grids of two adjacent rows is pre-processed to create a one cell overlap between the two grids, fig. 2.3, in the sliding interface boundary. In other words, the exterior sliding plane of one row coincides with the interior plane of the other row and vice-versa. During the computation of fluxes, the nodes of the interior sliding plane (donors) are used to update the flow values at the nodes of the opposite

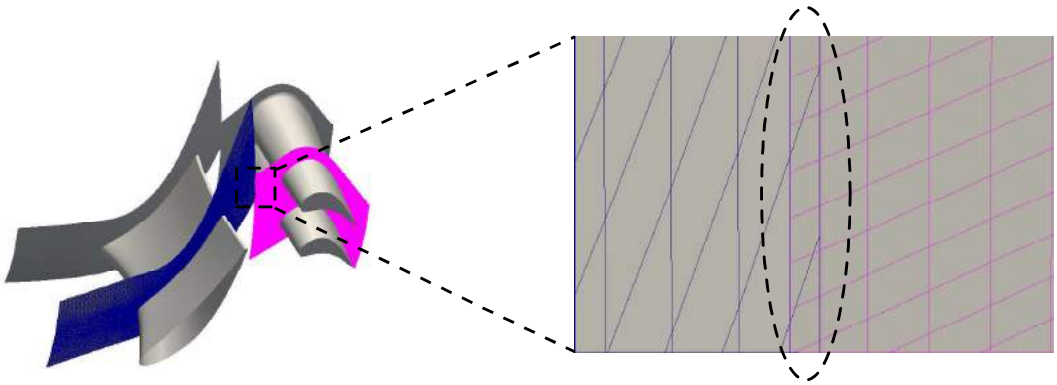


Figure 2.3: Radial cut of a 3D turbine stage grid. Zoom at the sliding interface area.

exterior sliding plane (receivers). At each time-step, rotor grids are moved relative to stator grids. It is reminded that, for each row, its relative frame of reference is used to solve the flow equations. Before updating them, the flow values (velocities) of the donors are being transformed from the relative Cartesian to the absolute cylindrical frame of reference[3] ($\boldsymbol{u} \rightarrow \boldsymbol{u}_{don}^{abs}$)

$$\begin{bmatrix} \varrho \\ v_1 \\ v_r \\ v_\theta \\ p \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & cos\theta & -sin\theta & 0 & -\omega sin\theta & -\omega cos\theta \\ 0 & 0 & sin\theta & cos\theta & 0 & \omega cos\theta & -\omega sin\theta \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \varrho \\ w_1 \\ w_2 \\ w_3 \\ p \\ x_2 \\ x_3 \end{bmatrix} \tag{2.43}$$

where ω is the angular speed and θ is the angular displacement of the row for the corresponding time-step of the CFD simulation. Then, when moving to the next time-step, a search is performed in order to locate the face of the exterior sliding plane, where each node of the opposite interior plane is located at the current time-step. Then, interpolation takes place using weights m_i computed based on the distance

[3] For stationary rows, where $\omega = \theta = 0$, the transformation matrix reduces to the identity matrix and, thus, no transformation takes place.

between the receiver node from the donors ($\boldsymbol{u}_{don}^{abs} \rightarrow \boldsymbol{u}_{rec}^{abs}$), fig. 2.2.

$$\boldsymbol{u}^{rec} = \sum_i m_i \boldsymbol{u}_i^{don}, \qquad \sum_i m_i = 1 \tag{2.44}$$

The interpolation takes place at every pseudo-time iteration while the search takes place only once for every real time-step (see section 2.5 for the description of physical and pseudo-time usage). Finally, the interpolated flow variables are transformed back from the absolute cylindrical to the relative Cartesian frame of reference[3] ($\boldsymbol{u}_{rec}^{abs} \rightarrow \boldsymbol{u}_{rec}^{rel}$).

$$\begin{bmatrix} \varrho \\ w_1 \\ w_2 \\ w_3 \\ p \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & cos\theta & sin\theta & 0 & 0 & \omega \\ 0 & 0 & -sin\theta & cos\theta & 0 & -\omega & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \varrho \\ v_1 \\ v_r \\ v_\theta \\ p \\ x_2 \\ x_3 \end{bmatrix} \tag{2.45}$$

These values are, then, used to compute the fluxes at the corresponding control volumes. At convergence of the real time-step, these values are consistent between the zones.

2.5 Iterative Solution Scheme

The system of discrete equations 2.1 is linearized and solved iteratively within each and every time-step using a dual time-stepping technique for the correction of the flow variables vector $\Delta \boldsymbol{u}$. To be more precise, the solution at each real time-step is considered as a steady problem with extra source terms for time variation. The iterative scheme that is used to converge the discrete residuals to zero is pseudo-time-stepping using the 5-stage Runge-Kutta method [164]. Let τ denote the pseudo-time and superscript k the pseudo-time-step counter within a real time step; then, a generic time marching scheme can be written as

$$\frac{\boldsymbol{u}_n^{k+1} - \boldsymbol{u}_n^k}{\Delta\tau} V_n = -(1-\beta)\boldsymbol{r}_n\left(\boldsymbol{u}^k\right) - \beta \boldsymbol{r}_n\left(\boldsymbol{u}^{k+1}\right) \tag{2.46}$$

where $\beta \in [0,1]$ determines whether a totally explicit or implicit scheme is used. Note that the residual $\boldsymbol{r}_n$ includes the temporal source terms introduced by eq. 2.38. If the residual at the $k+1$ pseudo-time-step is linearized using the Jacobian at k, it is obtained

$$\boldsymbol{r}_n\left(\boldsymbol{u}^{k+1}\right) \simeq \boldsymbol{r}_n\left(\boldsymbol{u}^{k}\right) + \frac{\partial \boldsymbol{r}_n}{\partial \boldsymbol{u}_n^k}\Delta \boldsymbol{u}_n^k \tag{2.47}$$

Combining eq. 2.46 and eq. 2.47 yields

$$\left(\frac{V_n}{\sigma \Delta \tau} + \beta \frac{\partial \boldsymbol{r}_n}{\partial \boldsymbol{u}_n^k}\right)\Delta \boldsymbol{u}_n^k = -\boldsymbol{r}_n\left(\boldsymbol{u}^k\right) \tag{2.48}$$

where σ is the CFL (Courant–Friedrichs–Lewy) number determined by the user. When $\sigma \to +\infty$ and the exact Jacobian matrix is computed, the equation translates to a Newton step, which gives quadratic convergence.

In practice, however, computing the exact Jacobian matrix comes at a prohibitive memory overhead and this is approximated instead. During this work, an approximation to the Jacobian, called block-Jacobi approximation, $\boldsymbol{P}_{b-J}$ is used in place of the full Jacobian matrix. The approximation is based on a local linearization of the Navier-Stokes equations using the central and neighboring control volumes and constructed by extracting the terms that correspond to the central node, thus producing a block-diagonal matrix [165]. The block-Jacobi approximation, augmented by the CFL weighted pseudo-time-stepping term, forms the solver's preconditioner $\boldsymbol{P}$,

$$\boldsymbol{P} = \frac{V_n}{\sigma \Delta \tau} + \boldsymbol{P}_{b-J} \tag{2.49}$$

transforming eq. 2.48 to

$$\boldsymbol{P}\Delta \boldsymbol{u}_n^k = -\boldsymbol{r}_n\left(\boldsymbol{u}^k\right) \tag{2.50}$$

The system is driven to iterative convergence using a 5-stage Runge-Kutta scheme [164] which, if the subscript is the real time-step counter, the first superscript is the pseudo-time-step counter and the second superscript the Runge-Kutta stage counter, can be expressed as

$$\begin{aligned} \boldsymbol{u}_n^{k,0} &= \boldsymbol{u}_n^k \\ \boldsymbol{u}_n^{k,m} &= \boldsymbol{u}_n^{k,m-1} - \alpha_m \boldsymbol{P}^{-1}\boldsymbol{r}_n^{k,m-1}, \qquad m = 1,\ldots,5 \\ \boldsymbol{u}_n^{k+1} &= \boldsymbol{u}_n^{k,5} \end{aligned} \tag{2.51}$$

where

$$\boldsymbol{r}_n^{k,m-1} = \boldsymbol{C}\left(\boldsymbol{u}_n^{k,m-1}\right) - \boldsymbol{B}^{k,m-1}$$
$$\boldsymbol{B}^{k,m-1} = \beta_m \boldsymbol{D}\left(\boldsymbol{u}_n^{k,m-1}\right) + (1-\beta_m)\,\boldsymbol{B}^{k,m-2}$$

Above, $\boldsymbol{C}$ is the convective operator arising from the discretization of the inviscid fluxes and $\boldsymbol{D}$ the dissipation, viscous and source term operator. The coefficients α_m and β_m are

$$\alpha_1 = \frac{1}{4}, \quad \alpha_2 = \frac{1}{6}, \quad \alpha_3 = \frac{3}{8}, \quad \alpha_4 = \frac{1}{2}, \quad \alpha_5 = 1$$
$$\beta_1 = 1, \quad \beta_2 = 0, \quad \beta_3 = \frac{14}{25}, \quad \beta_4 = 0, \quad \beta_5 = \frac{11}{25}$$

The scheme has a large stability region and a low computational cost due to the fact that β_2 and β_4 are zero, which does not require the computation of the dissipation and viscous terms $D_i\left(\boldsymbol{u}_n^{k,2}\right)$ and $D_i\left(\boldsymbol{u}_n^{k,4}\right)$.

The preconditioning matrix $\boldsymbol{P}$ is inverted by directly inverting each of the diagonal block matrices before the first Runge-Kutta stage. Then, it is stored in order to be multiplied with the residual vector $\boldsymbol{r}$ during the Runge-Kutta updates. Neglecting momentarily the turbulence model, the cost of storing the block-Jacobi preconditioner is five times that of the flow solution, i.e. a 5x5 matrix vs. the 5x1 flow variable vector needs to be additionally stored for each grid node.

The usual practice of considering the system as converged is applied when

$$\sum_{i \in [1, N_{nodes}]} \boldsymbol{r}_i \leq \epsilon \qquad or \qquad k \geq N_{iter} \tag{2.52}$$

where ϵ sufficiently small number and N_{iter} the maximum number of iterations, both provided by the user.

2.6 Multigrid

Multigrid [166–168] is a fundamental technique used in the majority of modern grid-based flow solvers. The concept behind the method is to have a sequence of successively coarser grids, which represent the smooth error modes of the finer grid along with an iterative "smoothing" procedure that eliminates the high frequency error modes on each grid. Thus, all error modes are eliminated and convergence is reached

faster.

Two approaches can be distinguished:

- Algebraic multigrid (AMG) [169, 170], where the hierarchy of operators is constructed directly from the system matrix. The levels of the hierarchy are subsets of unknowns without any geometric interpretation.
- Geometric multigrid (GMG) [171, 172], where the hierarchy of discretization is formulated on a series of successively coarsened grids based on the initial fine grid.

AMG methods can be applied as black-box for certain classes of sparse matrices and are considered advantageous where GMG is too difficult to apply. GMG, on the other hand, can be more efficient because the features of the flow, such as boundary layers, can be taken into account when coarsening the fine grid. During this work, GMG was used. GMG needs a grid coarsening algorithm to create the coarser grids and, also, needs to be properly constructed and embedded into the solver transfer operations. These are: *restriction* (from fine to coarse) and *prolongation* (from coarse to fine).

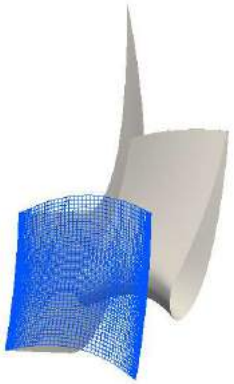
The consecutive coarse grids are produced by collapsing the grid edges of the finer level. The algorithm is sorting the cells of the grid in a list according to their volumes. Then, the shortest edges are collapsing and new cells are being formulated while trying to maintain a certain quality for the coarser grid and avoid negative volumes. More information can be found in [165]. An example of the fine grid and a sequence of coarser grids can be seen in fig. 2.4.

Once the coarser grids are generated, what remains is the definition of the transfer operations. During the coarsening procedure every grid node of the coarser grid i is associated with a set of nodes K_i on the finer grid from which it has been derived. In addition, for every fine grid point, the index of the coarse grid point which it has been collapsed to is available. The aforementioned is all the grid-to-grid connectivity information that is needed and contributes to a small addition to the storage footprint.

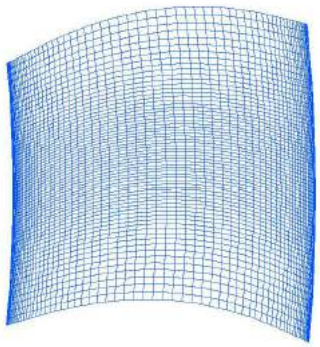
Prolongation

The correction $\Delta \boldsymbol{u}$ is transferred from the coarse to the fine grid. A linear interpolation is used, taking advantage of the spatial gradients of the corrections. If h denotes the fine and H the coarse grid values

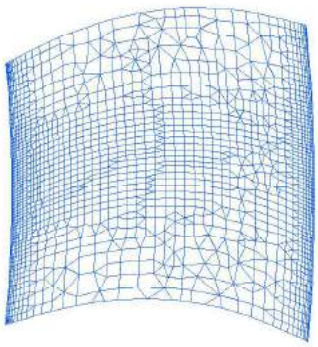
$$\Delta \boldsymbol{u}_i^h = \Delta \boldsymbol{u}_j^H + \left(x_i^h - x_j^H\right) \nabla \left(\Delta \boldsymbol{u}^H\right)_j, \qquad \forall i \in K_j \tag{2.53}$$



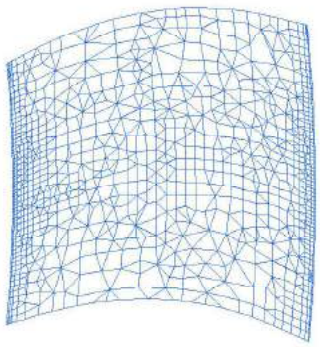
(a) Grid on the inlet face of a turbine stator domain.



(b) 1st grid level (finest).



(c) 2nd grid level.



(d) 3rd grid level (coarsest).

Figure 2.4: Multigrid levels at the inlet face of a turbine stator domain.

The gradient of the correction is computed using eq. 2.30.

Restriction

The residuals and the solution are transferred from the fine to the coarse grid. Volume weighting is used for the operation.

$$\boldsymbol{r}_j^H = \frac{\sum\limits_{i \in K_l} V_i^h \boldsymbol{r}_j^h}{\sum\limits_{i \in K_l} V_i^h} \qquad and \qquad \boldsymbol{u}_j^H = \frac{\sum\limits_{i \in K_l} V_i^h \boldsymbol{u}_j^h}{\sum\limits_{i \in K_l} V_i^h} \tag{2.54}$$

Multigrid Algorithm

The multigrid algorithm follows the Full Approximation Scheme (FAS) [163]. If $\boldsymbol{N}\left(\boldsymbol{u}^k\right) = \boldsymbol{f}$ is a non-linear system, whose solution is $\boldsymbol{u}$, the iterative scheme to solve it can be expressed as

$$\boldsymbol{u}^{k+1} \leftarrow \boldsymbol{u}^k + \mathcal{K}_{RK}\left(\boldsymbol{f} - \boldsymbol{N}\left(\boldsymbol{u}^k\right)\right) \tag{2.55}$$

where $\mathcal{K}_{RK}$ denotes the Runge-Kutta algorithm described in 2.51 and $\boldsymbol{f} - \boldsymbol{N}\left(\boldsymbol{u}^k\right) = -\boldsymbol{r}\left(\boldsymbol{u}^k\right)$ needs to be driven to zero. Initially, the solution is smoothed on the fine grid by performing a selected number of Runge-Kutta iterations.

$$\boldsymbol{u}^h \leftarrow \boldsymbol{u}^h + \mathcal{K}_{RK}\left(\boldsymbol{f}^h - \boldsymbol{N}^h\left(\boldsymbol{u}^h\right)\right) \tag{2.56}$$

At this point, the solution error, which exists if $\boldsymbol{N}^h\left(\boldsymbol{u}^h\right) \neq \boldsymbol{f}^h$, is defined as

$$\boldsymbol{E}^h = \hat{\boldsymbol{u}}^h - \boldsymbol{u}^h \tag{2.57}$$

where $\hat{\boldsymbol{u}}^h$ is the unknown exact solution. So by definition,

$$\boldsymbol{N}^h\left(\boldsymbol{u}^h + \boldsymbol{E}^h\right) = \boldsymbol{f}^h \tag{2.58}$$

Subtracting $\boldsymbol{N}^h\left(\boldsymbol{u}^h\right)$ from both sides yields

$$\boldsymbol{N}^h\left(\boldsymbol{u}^h + \boldsymbol{E}^h\right) - \boldsymbol{N}^h\left(\boldsymbol{u}^h\right) = \boldsymbol{f}^h - \boldsymbol{N}^h\left(\boldsymbol{u}^h\right) = -\boldsymbol{r}^h \tag{2.59}$$

The residual and the solution are restricted to the next coarser grid and this operation is denoted by $\boldsymbol{I}_h^H$. Eq. 2.59 can now be written for the coarser grid

$$\begin{aligned} \boldsymbol{N}^H\left(\boldsymbol{I}_h^H\boldsymbol{u}^h + \boldsymbol{E}^H\right) - \boldsymbol{N}^H\left(\boldsymbol{I}_h^H\boldsymbol{u}^h\right) &= -\boldsymbol{I}_h^H\boldsymbol{r}^h \\ \Rightarrow \boldsymbol{f}^H &= -\boldsymbol{I}_h^H\boldsymbol{r}^h + \boldsymbol{N}^H\left(\boldsymbol{I}_h^H\boldsymbol{u}^h\right) \end{aligned} \tag{2.60}$$

Thus, $\boldsymbol{f}^H$ is obtained and the smoother can be used for a selected number of iterations to converge to the solution on that grid in a way equivalent to eq. 2.56,

$$\boldsymbol{u}^H \leftarrow \boldsymbol{u}^H + \mathcal{K}_{RK}\left(\boldsymbol{f}^H - \boldsymbol{N}^H\left(\boldsymbol{u}^H\right)\right) \tag{2.61}$$

The procedure continues until the coarsest grid is reached. Then the correction is prolonged gradually from the coarsest to the finest grid which, if $\boldsymbol{I}_H^h$ stands for the

prolongation operator, is expressed as

$$\boldsymbol{u}^h \leftarrow \boldsymbol{u}^h + \boldsymbol{I}_H^h \left(\boldsymbol{u}^H - \boldsymbol{I}_H^h \boldsymbol{u}^h \right) \tag{2.62}$$

The sequence of operations described can be applied either in a V or W-cycle. A schematic can be seen in fig. 2.5. In this work, a V-cycle with 4 grid levels is used for the cases of chapter 4.

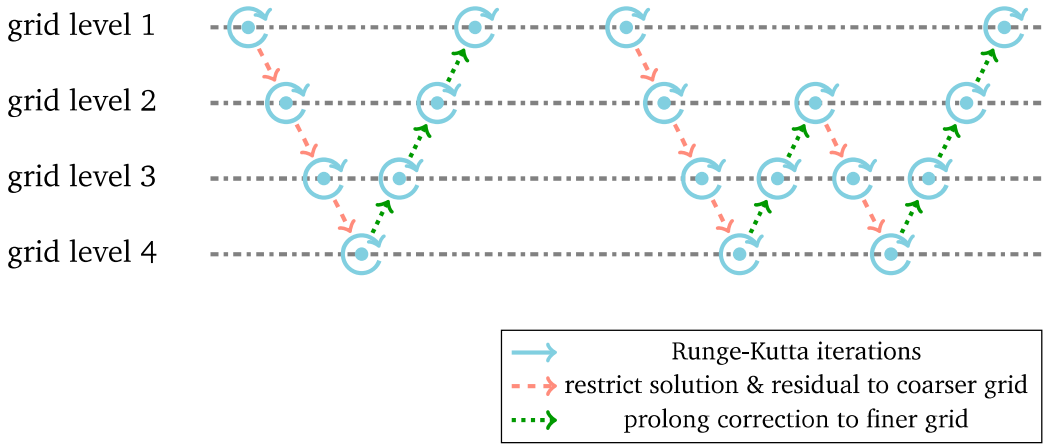


Figure 2.5: Schematic of V and W cycles of multigrid operations (1:finest, 4:coarsest grid).

The iterative process used to solve the unsteady flow equations per time-step, inside the outer physical time loop of the dual time-stepping scheme, can be summarized by

$$\boldsymbol{u}_n^k = \boldsymbol{u}_n^{k-1} - \Delta\tau \mathcal{K} \left(\frac{3\boldsymbol{u}_n^{k-1} - 4\boldsymbol{u}_{n-1} + \boldsymbol{u}_{n-2}}{2\Delta t} + \boldsymbol{r}_{st} \right) \tag{2.63}$$

where $\mathcal{K}$ is the operator incorporating both Runge-Kutta and multigrid.

2.7 Flow Solver's Algorithm & Parallelization

The solution method described in this chapter can be summarized in the following pseudo-code algorithm.

Algorithm 1: Unsteady flow solver.

```
initialize u;
t → 0;
while t < t_final do                                   // physical time loop
    iter → 0;
    while iter < iter_max and ∑r > ϵ do                // pseudo-time loop
        multigrid(to_fine, to_coarse);
        if to_fine then
            prolong;
        else if to_coarse then
            restrict;
        end
        iter^mg → 0;
        while iter^mg < iter^mg_max do                 // multigrid levels loop
            stage_RK → 1;
            while stage_RK ≤ 5 do                      // 5 stage Runge-Kutta loop
                impose BCs;
                compute fluxes;
                compute redidual r;
                update solution u;
                stage_RK → stage_RK + 1;
            end
            iter^mg → iter^mg + 1;
        end
        iter → iter + 1;
    end
    t → t + Δt;
end
```

The parallelization of the flow solver is based on the OPlus library [162, 163], initially developed at the Oxford University. It enables the parallelization of applications that involve unstructured grids through the straightforward insertion of simple subroutine calls. In other words, it creates an interface between the application code (i.e. Hydra) and the low level MPI routines. All MPI calls are within OPlus, without any direct message passing calls in Hydra. It is based on an underlying abstraction involving sets, pointers between sets and operations performed on sets. The set partitioning, computation of halo regions, and the exchange of halo data is performed automatically by OPlus after the user specifies the sets and pointers by Fortran subroutine calls. A single-source OPlus application code can be compiled for execution in either a parallel or sequential manner.

2.8 Objective Function & Practicalities

The previous sections of this chapter presented the details of the flow solver. This section attempts to place the flow solver within the process that is needed to set up the flow simulation and continue with an optimization.

The first step is to select a number of appropriate design parameters $\boldsymbol{a}$ to parameterize a given geometry. Modifying the design parameters translates to modifying the shape of the geometry (section 2.8.1). Then, using a grid generation software, the computational grid $\boldsymbol{x}$ is generated to discretize the space "close to" the geometry (section 2.8.2). The URANS solver that was presented is used to solve the flow problem. After converging the flow equations at each real time-step, the instantaneous value of a function of interest $j_{st}(\boldsymbol{u}, \boldsymbol{a}, t)$ is computed. In an unsteady problem, the objective function J is defined as the time-integral of j_{st} over a time interval starting from time-step n_0, corresponding to time t_0, and ending at the last time-step N, corresponding to the time-instant T. For equal time-steps Δt, J is given by

$$J(\boldsymbol{u}, \boldsymbol{a}) = \int_{t_0}^{T} j_{st}(\boldsymbol{u}, \boldsymbol{a}, t)\, dt = \Delta t \sum_{n=n_0}^{N} j_{st,n}(\boldsymbol{u}_n, \boldsymbol{a}) \tag{2.64}$$

Fig. 2.6 provides a schematic of the described process.

The URANS governing equations 2.1 can be rewritten, using eq. 2.38 and the quan-

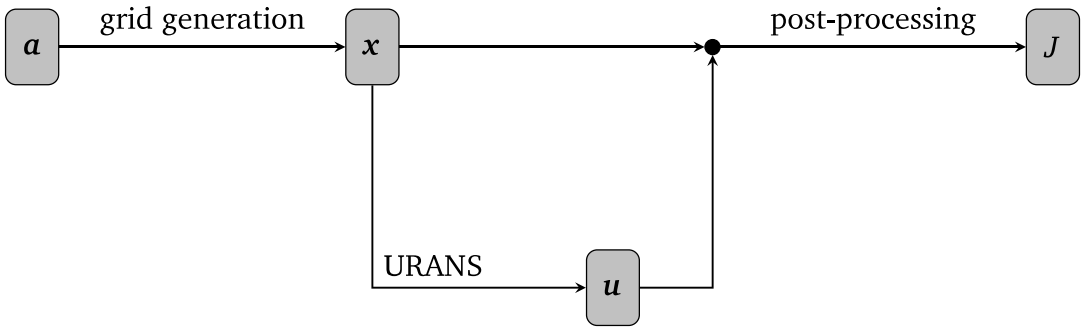


Figure 2.6: Simplified schematic of the flow problem's setup.

tities introduced in this section, in a semi-discrete form as

$$\begin{aligned} \boldsymbol{r}(\boldsymbol{u},\boldsymbol{a},t) =& \frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{r}_{st}(\boldsymbol{u},\boldsymbol{a}) = 0 \\ =& \frac{3\boldsymbol{u}_n - 4\boldsymbol{u}_{n-1} + \boldsymbol{u}_{n-2}}{2\Delta t} + \boldsymbol{r}_{st}(\boldsymbol{u},\boldsymbol{a}) = 0 \end{aligned} \tag{2.65}$$

where $\boldsymbol{r}_{st}$ is the steady residual, i.e. the URANS residual excluding the temporal term.

2.8.1 Parameterization

Both parameterization and grid generation are performed using PADRAM, which stands for PArametric Design and RApid Meshing [173, 174]. It is a CFD grid generation tool designed for turbomachinery components as well as a design system providing a rich set of parameters to alter a given configuration. The different ways to parametrically modify a geometry include the displacement of blade cut sections, bump functions, the free form deformation method, splines and NURBS.

For this work, the first method is selected and this is described below. Five sections are considered uniformly across the blade span. Each section can independently be:

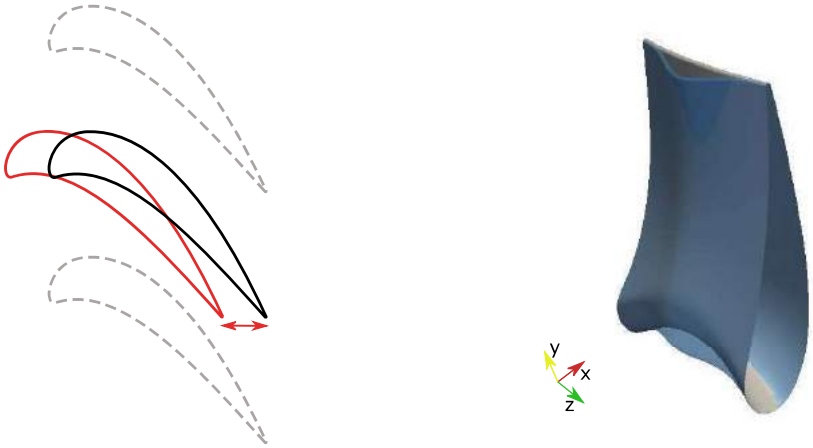
- axially shifted,
- rotated in the circumferential direction and
- rotated around a radial axis.

The blade surface is modified accordingly by a spanwise cubic spline interpolation of the five sections. In fig. 2.7, the position of the middle out of the five sections is changed and the effect on the geometry is displayed.

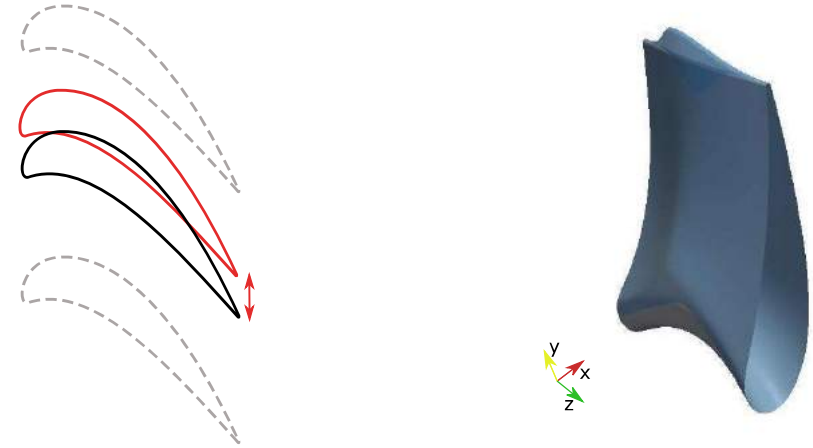
2.8.2 Grid Generation

Grid generation was performed by PADRAM which is capable of creating 2D, quasi-3D, 3D, single passage, multi-passage, structured, unstructured and hybrid (combination of structured and unstructured) grids. Grids are based on C-O-H multi-blocks. For the grids of the 3D blade passages of the applications in this thesis, initially, an O-grid is generated around the blade. An H-grid is then generated around it to link the O-grid to the periodic boundaries and for the upstream and downstream regions of the domain, as shown schematically in fig. 2.8. For a single blade passage, the generation of the grid is a matter of a single-digit number of seconds. Grid details are specified using an input file by the user. Options such as the number of O-grid layers, the number of H-grid layers can be set and this might also include the presence of realistic features such as fillets, gaps, cavities etc. By adjusting the appropriate parameters, the generated grid can be coarse, medium or fine.

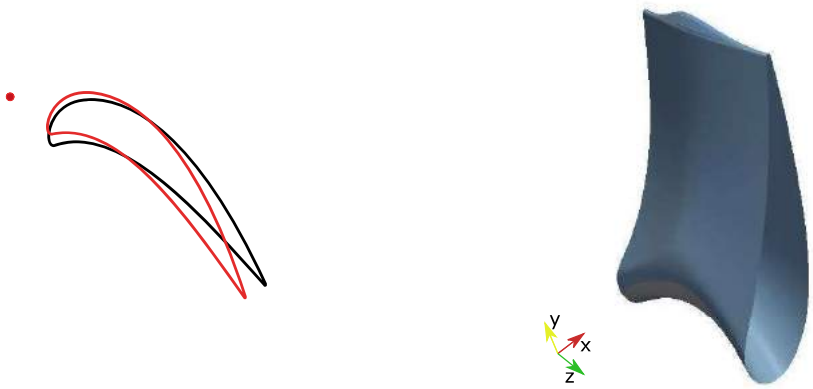
The so-generated grids are block structured and are preprocessed in order to be readable from Hydra, which is an unstructured grid solver. The entire computational domain is assembled by connecting the individual domains that correspond to each row (row-domains). For steady computations, the mixing plane technique is used for the exchange of information between different rows. For the unsteady computations, the sliding interface is needed, so the suitable number of blades to be used to form each row-domain needs to be found so that all of them are of equal circumferential pitch. The grid is generated for one of the blade passages belonging to the same row-domain and repeated for the others. At a post-processing step, a one-cell overlap between grids of adjacent rows is created (see section 2.4.1).



(a) Axial shift.



(b) Circumferential rotation.



(c) Rotation around radial axis.

Figure 2.7: Possible position changes of the blade sections (left) along with the modified blade geometries if the position change is applied only over the middle section (right).

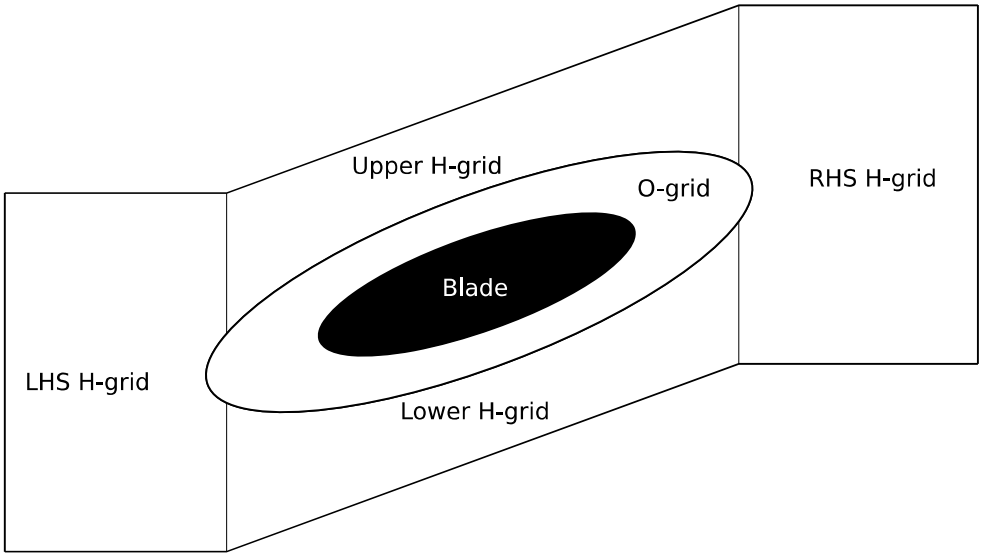


Figure 2.8: Single passage grid blocks.

3

Unsteady Discrete Adjoint Method: Time-Domain Formulation

In this chapter, the adjoint method is presented as an efficient means of computing the gradient of an objective function w.r.t. the design variables for unsteady flows.

The adjoint equations corresponding to the URANS flow equations are formulated in the real time-domain, contrasting with frequency-domain approaches, using the discrete approach. The adjoint method is applied also to the iterative scheme that employs Runge-Kutta and is used to solve them so that a consistent iterative scheme, that ensures the same convergence rate, is generated to solve the adjoint equations. The computation of some of the differential terms of the adjoint equations is aided by Algorithmic Differentiation, the usage of which and the combination with hand-differentiation in the adjoint solver is described. Furthermore, the imposed boundary conditions of the adjoint system are derived. In order to use the unsteady adjoint solver for multi-row cases, the communication of adjacent row-domains is needed. This is achieved by developing the adjoint to the sliding interface technique. In addition, the temporal coarsening technique is used as a way to reduce the storage space requirements and execution time footprint of the adjoint solver. The specific case of periodic

flows is also examined using an example in order to further reduce the computational cost of the adjoint solver when applicable.

Apart from solving the unsteady adjoint problem, the algorithm for obtaining the gradient of the objective function is given. The use of the gradient is described in a basic optimization loop by considering equality constraints, if needed. Finally, the time and storage space costs of an adjoint computation are discussed.

3.1 Unsteady Discrete Adjoint Equations

In order to employ gradient-based optimization methods, the gradient of an objective function w.r.t. the design variables' vector $\frac{dJ}{d\boldsymbol{a}}$ needs to be computed. Applying the chain rule to eq. 2.64, the gradient of the objective is given by

$$\frac{dJ}{d\boldsymbol{a}} = \Delta t \sum_{n=n_0}^{N} \frac{\partial j_{st,n}}{\partial \boldsymbol{a}} + \underbrace{\Delta t \sum_{n=n_0}^{N} \frac{\partial j_{st,n}}{\partial \boldsymbol{u}_n} \frac{d\boldsymbol{u}_n}{d\boldsymbol{a}}}_{\mathcal{B}} \tag{3.1}$$

To compute $\frac{d\boldsymbol{u}_n}{d\boldsymbol{a}}$ of term $\mathcal{B}$ in eq. 3.1, the equation that occurs by differentiating eq. 2.65 w.r.t. $\boldsymbol{a}$, to be referred as the forward differentiation system hereafter, needs to be solved

$$\boldsymbol{R}_{forw} = \frac{\partial \left(\frac{d\boldsymbol{u}_n}{d\boldsymbol{a}}\right)}{\partial t} + \frac{\partial \boldsymbol{r}_{st,n}}{\partial \boldsymbol{u}_n} \frac{d\boldsymbol{u}_n}{d\boldsymbol{a}} + \frac{\partial \boldsymbol{r}_{st,n}}{\partial \boldsymbol{a}} = 0 \tag{3.2}$$

with the initial condition $\frac{d\boldsymbol{u}_0}{d\boldsymbol{a}} = 0$. The cost of solving the forward differentiation equation is proportional to the number of design variables. To avoid solving it, the adjoint method is employed to replace term $\mathcal{B}$ with a term which is cheaper to compute. The augmented objective function is formulated as

$$J_{aug} = \Delta t \sum_{n=n_0}^{N} j_{st,n}(\boldsymbol{u}_n, \boldsymbol{a}) + \Delta t \sum_{n=0}^{N} \boldsymbol{\psi}_n^T \boldsymbol{r}_n \tag{3.3}$$

where $\boldsymbol{\psi}$ is the adjoint variable vector. The gradient of the augmented objective func-

tion w.r.t. $\boldsymbol{a}$ is

$$\frac{dJ_{aug}}{d\boldsymbol{a}} = \Delta t \sum_{n=n_0}^{N} \left(\frac{\partial j_{st,n}}{\partial \boldsymbol{a}} + \frac{\partial j_{st,n}}{\partial \boldsymbol{u}_n} \frac{d\boldsymbol{u}_n}{d\boldsymbol{a}} \right) + \Delta t \sum_{n=0}^{N} \boldsymbol{\psi}_n^T \left(\frac{\partial \left(\frac{d\boldsymbol{u}_n}{d\boldsymbol{a}} \right)}{\partial t} + \frac{\partial \boldsymbol{r}_{st,n}}{\partial \boldsymbol{u}_n} \frac{d\boldsymbol{u}_n}{d\boldsymbol{a}} + \frac{\partial \boldsymbol{r}_{st,n}}{\partial \boldsymbol{a}} \right)$$

$$= \Delta t \sum_{n=n_0}^{N} \left(\frac{\partial j_{st,n}}{\partial \boldsymbol{a}} + \frac{\partial j_{st,n}}{\partial \boldsymbol{u}_n} \frac{d\boldsymbol{u}_n}{d\boldsymbol{a}} \right) + \Delta t \left[\boldsymbol{\psi}_n^T \frac{d\boldsymbol{u}_n}{d\boldsymbol{a}} \right]_0^N - \Delta t \sum_{n=0}^{N} \frac{\partial \boldsymbol{\psi}_n^T}{\partial t} \frac{d\boldsymbol{u}_n}{d\boldsymbol{a}}$$
$$+ \Delta t \sum_{n=0}^{N} \boldsymbol{\psi}_n^T \left(\frac{\partial \boldsymbol{r}_{st,n}}{\partial \boldsymbol{a}} + \frac{\partial \boldsymbol{r}_{st,n}}{\partial \boldsymbol{u}_n} \frac{d\boldsymbol{u}_n}{d\boldsymbol{a}} \right)$$

$$= \Delta t \sum_{n=n_0}^{N} \frac{\partial j_{st,n}}{\partial \boldsymbol{a}} + \Delta t \sum_{n=0}^{N} \boldsymbol{\psi}_n^T \frac{\partial \boldsymbol{r}_{st,n}}{\partial \boldsymbol{a}} + \Delta t \left[\boldsymbol{\psi}_n^T \frac{d\boldsymbol{u}_n}{d\boldsymbol{a}} \right]_0^N$$
$$+ \Delta t \sum_{n=n_0}^{N} \left(-\frac{\partial \boldsymbol{\psi}_n^T}{\partial t} + \boldsymbol{\psi}_n^T \frac{\partial \boldsymbol{r}_{st,n}}{\partial \boldsymbol{u}_n} + \frac{\partial j_{st,n}}{\partial \boldsymbol{u}_n} \right) \frac{d\boldsymbol{u}_n}{d\boldsymbol{a}}$$
$$+ \Delta t \sum_{n=0}^{n_0-1} \left(-\frac{\partial \boldsymbol{\psi}_n^T}{\partial t} + \boldsymbol{\psi}_n^T \frac{\partial \boldsymbol{r}_{st,n}}{\partial \boldsymbol{u}_n} \right) \frac{d\boldsymbol{u}_n}{d\boldsymbol{a}} \tag{3.4}$$

To avoid computing $\frac{d\boldsymbol{u}_n}{d\boldsymbol{a}}$, the unsteady adjoint residual that must be zeroed while marching backwards in time, after applying the second-order time discretization formula, eq. 2.38, is

$$\boldsymbol{r}_{adj,n} = \frac{3}{2\Delta t} \boldsymbol{\psi}_n + \left[\frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}} \right]_n^T \boldsymbol{\psi}_n - \frac{2}{\Delta t} \boldsymbol{\psi}_{n+1} + \frac{1}{2\Delta t} \boldsymbol{\psi}_{n+2} + \boldsymbol{g}_n = 0 \tag{3.5}$$

where

$$\boldsymbol{g}_n = \begin{cases} \left[\frac{\partial j_{st}}{\partial \boldsymbol{u}} \right]_n^T, & n \in [n_0, N] \\ 0, & n \in [0, n_0) \end{cases} \tag{3.6}$$

and the initial condition to be used is $\boldsymbol{\psi}_N = 0$.

The gradient of J w.r.t. the design variable vector can now be computed by

$$\frac{dJ}{d\boldsymbol{a}} = \Delta t \sum_{n=n_0}^{N} \frac{\partial j_{st,n}}{\partial \boldsymbol{a}} + \Delta t \sum_{n=0}^{N} \boldsymbol{\psi}_n^T \frac{\partial \boldsymbol{r}_{st,n}}{\partial \boldsymbol{a}} \tag{3.7}$$

Eq. 3.5 implies that the unsteady adjoint system needs to be solved backwards

in time. The unknowns of the adjoint equations are the adjoint variable fields of the current time-step $\boldsymbol{\psi}^n$, while $\boldsymbol{\psi}^{n+1}$ and $\boldsymbol{\psi}^{n+2}$ have already been computed while solving for the time-steps which are next in terms of real time and previous in terms of order of solution. The flow solution of the current time-step $\boldsymbol{u}_n$ is also needed to form the unsteady adjoint equation. However, since it is obtained by marching forward in time, as implied by eq. 2.65, the flow and adjoint systems are solved separately. In order to have the flow solution fields available during the adjoint computation, flow-files are saved per time-step in the disk during the URANS solver run. They are read in, whenever needed, during the adjoint run. Even though RAM storage would provide faster access, it is avoided due to RAM size limitations. Moreover, the I/O operations from an SSD disk using the parallel library OPlus, is still relatively fast and offers far larger storage space capacity.

3.2 Solving the Discrete Unsteady Adjoint Equations

This section aims at formulating the iterative schemes that are used to solve the forward differentiation and the adjoint systems starting from the iterative scheme that is used to solve the flow equations. It is recalled that the flow solver is using a dual time-stepping scheme. The outer loop is marching forwards in real time and the inner loop performs pseudo-time-steps which employ the explicit Runge-Kutta scheme, eq. 2.51, and multigrid.

It is required that the iterative schemes have two properties. Firstly, it is desired that the forward differentiation and the adjoint solver converge with the same rate as the flow solver. The second requirement is consistency between the forward differentiation and the adjoint solver. This ensures that, by using the same number of convergence iterations for the corresponding real time-steps of the forward differentiation or the adjoint equations, the obtained gradient is the same. Giles [175] has produced the iterative scheme for steady adjoint problems but the analysis needs to be expanded to include the extra real time loop and unsteady terms.

For the sake of convenience, the equations are used in continuous form and index n for the real time-step is omitted. The forward differentiation equation, expressed for a single entry a_λ of the design variables' vector $\boldsymbol{a}$ becomes

$$\boldsymbol{r}_{forw} = \frac{\partial}{\partial t}\left(\frac{d\boldsymbol{u}}{da_\lambda}\right) + \frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}}\frac{d\boldsymbol{u}}{da_\lambda} + \frac{\partial \boldsymbol{r}_{st}}{\partial a_\lambda} = 0 \tag{3.8}$$

The adjoint equation becomes

$$\boldsymbol{r}_{adj} = -\frac{\partial \boldsymbol{\psi}}{\partial t} + \left[\frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}}\right]^T \boldsymbol{\psi} + \boldsymbol{g} = 0 \tag{3.9}$$

Differentiating eq. 2.63 w.r.t. a_λ yields the iterative scheme to solve eq. 3.8,

$$\boldsymbol{r}'_{forw} = \frac{\partial}{\partial \tau}\left(\frac{d\boldsymbol{u}}{da_\lambda}\right) + \mathcal{K}\left[\frac{\partial}{\partial t}\left(\frac{d\boldsymbol{u}}{da_\lambda}\right) + \frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}}\frac{d\boldsymbol{u}}{da_\lambda} + \frac{\partial \boldsymbol{r}_{st}}{\partial a_\lambda}\right] = 0 \tag{3.10}$$

where $\tau \in [0, \tau_\infty]$ and $t \in [0, T]$. τ_∞ may be different for each time-step but is expressed uniformly here to keep the expressions simpler. For the same reason, in the following equations, the notation $\frac{d\boldsymbol{u}}{da_\lambda}\Big|_\tau^t = \frac{d\boldsymbol{u}}{da_\lambda}(t,\tau)$, where superscript is for real-time and subscript for pseudo-time, is used. The initial conditions for eq. 3.10 are $\frac{d\boldsymbol{u}}{da_\lambda}\Big|_{\tau=0} = \frac{d\boldsymbol{u}}{da_\lambda}\Big|^{t=0} = 0$.
The adjoint method was used to avoid computing $\frac{d\boldsymbol{u}}{d\boldsymbol{a}}$ by appropriately replacing term $\mathcal{B}$ in eq. 3.1. In this section, function $\mathcal{B}_\lambda$ which corresponds to a_λ is selected to be computed; this is given by

$$\begin{aligned}
\mathcal{B}_\lambda &= \int_{t_0}^{T} \frac{\partial j_{st}}{\partial \boldsymbol{u}} \frac{d\boldsymbol{u}}{da_\lambda}\Big|_{\tau_\infty} dt \\
&= \int_{t_0}^{T} \frac{\partial j_{st}}{\partial \boldsymbol{u}} \frac{d\boldsymbol{u}}{da_\lambda}\Big|_{\tau_\infty} dt + \int_{0}^{t_0} 0 \frac{d\boldsymbol{u}}{da_\lambda}\Big|_{\tau_\infty} dt \\
&= \int_{0}^{T} \boldsymbol{g}^T \frac{d\boldsymbol{u}}{da_\lambda}\Big|_{\tau_\infty} dt
\end{aligned} \tag{3.11}$$

In eq. 3.11, index τ_∞ means that the values of $\frac{d\boldsymbol{u}}{da_\lambda}$ for the last pseudo-time step of each and every real-time instant are used. By introducing a Lagrange multiplier $\boldsymbol{w}$, instead of $\mathcal{B}_\lambda$, one may define the augmented function $\mathcal{B}_\lambda^{aug}$ which, through integration by parts, becomes

$$\mathcal{B}_\lambda^{aug} = \int_0^T \boldsymbol{g}^T \frac{d\boldsymbol{u}}{da_\lambda}\bigg|_{\tau_\infty} dt - \int_0^T \int_0^{\tau_\infty} \boldsymbol{w}^T \boldsymbol{r}'_{forw} d\tau dt$$

$$= \int_0^T \boldsymbol{g}^T \frac{d\boldsymbol{u}}{da_\lambda}\bigg|_{\tau_\infty} dt$$
$$- \int_0^T \int_0^{\tau_\infty} \boldsymbol{w}^T \left\{ \frac{\partial}{\partial \tau}\left(\frac{d\boldsymbol{u}}{da_\lambda}\right) + \mathcal{K}\left[\frac{\partial}{\partial t}\left(\frac{d\boldsymbol{u}}{da_\lambda}\right) + \frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}}\frac{d\boldsymbol{u}}{da_\lambda} + \frac{\partial \boldsymbol{r}_{st}}{\partial a_\lambda}\right]\right\} d\tau dt$$

$$= \int_0^T \boldsymbol{g}^T \frac{d\boldsymbol{u}}{da_\lambda}\bigg|_{\tau_\infty} dt - \int_0^T \left[\boldsymbol{w}^T \frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right]_0^{\tau_\infty} dt + \underbrace{\int_0^T \int_0^{\tau_\infty} \frac{\partial \boldsymbol{w}^T}{\partial \tau}\frac{d\boldsymbol{u}}{da_\lambda} d\tau dt}_{A1}$$
$$- \left[\int_0^{\tau_\infty} \boldsymbol{w}^T \mathcal{K} \frac{d\boldsymbol{u}}{da_\lambda} d\tau\right]_0^T + \underbrace{\int_0^T \int_0^{\tau_\infty} \frac{\partial}{\partial \tau}\left(\boldsymbol{w}^T \mathcal{K}\right)\frac{d\boldsymbol{u}}{da_\lambda} d\tau dt}_{A2}$$
$$\underbrace{- \int_0^T \int_0^{\tau_\infty} \boldsymbol{w}^T \mathcal{K} \frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}}\frac{d\boldsymbol{u}}{da_\lambda} d\tau dt}_{A3} - \int_0^T \int_0^{\tau_\infty} \boldsymbol{w}^T \mathcal{K} \frac{\partial \boldsymbol{r}_{st}}{\partial a_\lambda} d\tau dt \tag{3.12}$$

Terms $A1$, $A2$ and $A3$ are combined to form

$$\int_0^T \int_0^{\tau_\infty} \left[\frac{\partial \boldsymbol{w}^T}{\partial \tau} + \frac{\partial \boldsymbol{w}^T}{\partial t}\mathcal{K} - \boldsymbol{w}^T \mathcal{K}\frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}}\right]\frac{d\boldsymbol{u}}{da_\lambda} d\tau dt$$

which can be eliminated from eq. 3.12 if $\boldsymbol{w}$ satisfies the equation

$$\frac{\partial \boldsymbol{w}^T}{\partial \tau} + \frac{\partial \boldsymbol{w}^T}{\partial t}\mathcal{K} - \boldsymbol{w}^T \mathcal{K}\frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}} = 0$$

or, if transposed,

$$\frac{\partial \boldsymbol{w}}{\partial \tau} + \mathcal{K}^T \frac{\partial \boldsymbol{w}}{\partial t} - \left[\frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}}\right]^T \mathcal{K}^T \boldsymbol{w} = 0 \tag{3.13}$$

Eq. 3.12, after using the initial conditions for eq. 3.10, $\frac{d\boldsymbol{u}}{da_\lambda}\Big|_{\tau=0} = \frac{d\boldsymbol{u}}{da_\lambda}\Big|^{t=0} = 0$, can be

rewritten as

$$\mathcal{B}_\lambda^{aug} = \underbrace{\int_0^T \boldsymbol{g}^T \left.\frac{d\boldsymbol{u}}{da_\lambda}\right|_{\tau_\infty} dt}_{A4} - \underbrace{\int_0^T \boldsymbol{w}^T\big|_{\tau_\infty} \left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_{\tau_\infty} dt}_{A5} - \underbrace{\int_0^{\tau_\infty} \boldsymbol{w}^T \mathcal{K} \frac{d\boldsymbol{u}}{da_\lambda} d\tau \Big|^{t=T}}_{A6}$$
$$- \int_0^T \int_0^{\tau_\infty} \boldsymbol{w}^T \mathcal{K} \frac{\partial \boldsymbol{r}_{st}}{\partial a_\lambda} d\tau dt \tag{3.12'}$$

Terms $A4$ and $A5$ are eliminated if

$$\boldsymbol{w}|_{\tau_\infty} = \boldsymbol{g} \tag{3.14}$$

and term $A6$ is eliminated if

$$\boldsymbol{w}|^T = 0 \tag{3.15}$$

so that $\mathcal{B}_\lambda$ can be written as

$$\mathcal{B}_\lambda = \mathcal{B}_\lambda^{aug} = -\int_0^T \int_0^{\tau_\infty} \boldsymbol{w}^T \mathcal{K} \frac{\partial \boldsymbol{r}_{st}}{\partial a_\lambda} d\tau dt \tag{3.12''}$$

By defining

$$\boldsymbol{\psi}|_{\tau^*}^t = \int_{\tau^*}^{\tau_\infty} \mathcal{K}^T \boldsymbol{w}\, d\tau \tag{3.16}$$

where $\tau^* \in [0, \tau_\infty]$ is the pseudo-time used in the adjoint problem, $\mathcal{B}_\lambda$ can be computed as

$$\mathcal{B}_\lambda = \int_0^T \boldsymbol{\psi}^T\big|_0 \frac{\partial \boldsymbol{r}_{st}}{\partial a_\lambda} dt \tag{3.17}$$

Applying the Leibniz theorem to eq. 3.16 gives

$$\frac{\partial \boldsymbol{\psi}}{\partial \tau^*} = \mathcal{K}\, \boldsymbol{w}|_{\tau^*} \tag{3.18}$$

which because of 3.14 can be written as

$$\begin{aligned} \frac{\partial \boldsymbol{\psi}}{\partial \tau^*} &= \mathcal{K}\left(\boldsymbol{g} - \boldsymbol{w}|_{\tau_\infty} + \boldsymbol{w}|_{\tau^*}\right) \\ &= \mathcal{K}\left(\boldsymbol{g} - \int_{\tau^*}^{\tau_\infty} \frac{\partial \boldsymbol{w}}{\partial \tau} d\tau\right) \end{aligned} \tag{3.18'}$$

Because of eq. 3.13, this modified to

$$\frac{\partial \boldsymbol{\psi}}{\partial \tau^*} = \mathcal{K}\left[\boldsymbol{g} - \int_{\tau^*}^{\tau_\infty} \left(-\mathcal{K}^T \frac{\partial \boldsymbol{w}}{\partial t} + \left[\frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}}\right]^T \mathcal{K}^T \boldsymbol{w}\right) d\tau\right] \tag{3.18''}$$

which, finally, by using the definition of ψ, eq. 3.16, yields

$$\frac{\partial \boldsymbol{\psi}}{\partial \tau^*} = \mathcal{K}^T\left(\boldsymbol{g} - \frac{\partial \boldsymbol{\psi}}{\partial t} + \left[\frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}}\right]^T \boldsymbol{\psi}\right) \tag{3.18'''}$$

Eq. 3.18''' provides the iterative scheme that is used to solve the unsteady adjoint equations subject to the initial conditions

$$\boldsymbol{\psi}|_{\tau_\infty} = \int_{\tau_\infty}^{\tau_\infty} \mathcal{K}^T \boldsymbol{w}\, d\tau = 0 \tag{3.19}$$

(because of eq. 3.16) and

$$\boldsymbol{\psi}|^T = \int_{\tau^*}^{\tau_\infty} \mathcal{K}^T \boldsymbol{w}\, d\tau \Bigg|^T = 0 \tag{3.20}$$

(because of eq. 3.15). Since $\mathcal{K}$ is used to converge the flow and forward differentiation equations and $\mathcal{K}^T$ to converge the adjoint equations, all of them are expected to have the same convergence rate.

For completeness, the equations that summarize the iterative schemes used are written in semi-discrete form:
Forward differentiation:

$$\left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_n^{n_0} = \left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_n^{k-1} - \Delta\tau\mathcal{K}\left(\frac{3\left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_n^{k-1} - 4\left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_{n-1} + \left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_{n-2}}{2\Delta t} + \frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}} \left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_n^{k-1} + \left.\frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{a}}\right|_n\right) \tag{3.21}$$

Adjoint:

$$\boldsymbol{\psi}_n^{n_0} = \boldsymbol{\psi}_n^{k+1} - \Delta\tau\mathcal{K}^T\left(\frac{3\boldsymbol{\psi}_n^{k+1} - 4\boldsymbol{\psi}_{n+1} + \boldsymbol{\psi}_{n+2}}{2\Delta t} + \left[\frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}}\right]^T \boldsymbol{\psi}_n^{k+1} + \boldsymbol{g}_n\right) \tag{3.22}$$

Eq. 3.22 represents the outer loop of the pseudo-time-stepping scheme for the adjoint equations. Next step is to go one level deeper and modify the Runge-Kutta scheme from the flow solver to adjust it for the adjoint solver. For that purpose, it is considered that $\mathcal{K}$ involves only Runge-Kutta and not multigrid for the equations until the end of this section. Multigrid is addressed in the following section. The flow solver's explicit Runge-Kutta scheme introduced by eq. 2.51 is differentiated w.r.t. the design variable vector to produce the RK algorithm for solving the forward differentiation system

$$
\begin{aligned}
\boldsymbol{d}_n^{k,0} &= 0 \\
\left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_n^{k,0} &= \left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_n^{k} \\
\boldsymbol{d}_n^{k,m} &= \beta_m \boldsymbol{D} \left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_n^{k,m-1} + (1-\beta_m)\,\boldsymbol{d}_n^{k,m-1} \\
\left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_n^{k,m} &= \left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_n^{k,0} + a_m \boldsymbol{P}\left(\frac{\partial \boldsymbol{r}_{st}}{\partial a_\lambda} - \boldsymbol{C}\left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_n^{k,m-1} - \boldsymbol{d}_n^{k,m} - V\frac{3\boldsymbol{d}_n^{k,m-1} - 4\left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_{n-1} + \left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_{n-2}}{2\Delta t}\right) \\
\left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_n^{k+1} &= \left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_n^{k,5}
\end{aligned}
\tag{3.23}
$$

where $m = 1, 2, \cdots, 5$. For the 5-stage RK scheme, eq. 2.51, quantities

$$
\widetilde{\boldsymbol{d}}_n^{k,m} = \boldsymbol{d}_n^{k,m} - \boldsymbol{D}\left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_n^{n_0}, \quad \left.\widetilde{\frac{d\boldsymbol{u}}{d\boldsymbol{a}}}\right|_n^{k,m} = \left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_n^{k,m} - \left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_n^{k}
$$

are defined and the scheme can be reformulated for the solution of the forward differentiation system as follows

$$
\begin{aligned}
\widetilde{\boldsymbol{d}}_n^{k,0} &= -\boldsymbol{D}\left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_n^{n_0} \\
\left.\widetilde{\frac{d\boldsymbol{u}}{d\boldsymbol{a}}}\right|_n^{k,0} &= 0 \\
\widetilde{\boldsymbol{d}}_n^{k,m} &= \beta_m \boldsymbol{D}\left.\widetilde{\frac{d\boldsymbol{u}}{d\boldsymbol{a}}}\right|_n^{k,m-1} + (1-\beta_m)\,\widetilde{\boldsymbol{d}}_n^{k,m-1} \\
\left.\widetilde{\frac{d\boldsymbol{u}}{d\boldsymbol{a}}}\right|_n^{k,m} &= a_m \boldsymbol{P}\left(\boldsymbol{R}_{forw} - \boldsymbol{C}\left.\widetilde{\frac{d\boldsymbol{u}}{d\boldsymbol{a}}}\right|_n^{k,m-1} - \widetilde{\boldsymbol{d}}_n^{k,m} - \frac{3\left.\widetilde{\frac{d\boldsymbol{u}}{d\boldsymbol{a}}}\right|_n^{k,m-1}}{2\Delta t}\right) \\
\left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_n^{k+1} &= \left.\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right|_n^{k} + \left.\widetilde{\frac{d\boldsymbol{u}}{d\boldsymbol{a}}}\right|_n^{k,5}
\end{aligned}
\tag{3.24}
$$

where $m = 1, 2, \cdots, 5$. The RK operator $\mathcal{K}$ is defined implicitly by the previous system of equations as

$$
\mathcal{K} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & I \end{bmatrix} \Gamma^{-1} \begin{bmatrix} a_1\boldsymbol{P} & 0 & a_2\boldsymbol{P} & \cdots & 0 & a_5\boldsymbol{P} \end{bmatrix}^T \tag{3.25}
$$

where

$$
\Gamma = \begin{bmatrix}
I & & & & & \\
-\beta_2\boldsymbol{D} & I & & & & \\
a_2\boldsymbol{P}\left(\boldsymbol{C}+\frac{3}{2\Delta t}I\right) & a_2\boldsymbol{P} & I & & & \\
 & \ddots & \ddots & \ddots & & \\
 & & \ddots & \ddots & \ddots & \\
 & & & -(1-\beta_5) & -\beta_5\boldsymbol{D} & I \\
 & & & & a_5\boldsymbol{P}\left(\boldsymbol{C}+\frac{3}{2\Delta t}I\right) & a_5\boldsymbol{P} \quad I
\end{bmatrix}
$$

For the adjoint system, the transpose operation is needed, as proved by eq. 3.22

$$
\mathcal{K}^T = \begin{bmatrix} a_1\boldsymbol{P}^T & 0 & a_2\boldsymbol{P}^T & \cdots & 0 & a_5\boldsymbol{P}^T \end{bmatrix} \left(\Gamma^T\right)^{-1} \begin{bmatrix} 0 & 0 & \cdots & 0 & I \end{bmatrix}^T \tag{3.26}
$$

and the adjoint time-marching scheme of eq. 3.22 becomes

$$
\boldsymbol{\psi}^{n_0} = \boldsymbol{\psi}^{k+1} + \sum_{m=1}^{5} a_m \boldsymbol{P}^T \Gamma^T \widetilde{\boldsymbol{w}}_n^{k,m} \tag{3.27}
$$

where $\widetilde{\boldsymbol{w}}_n^k$ are defined by

$$\Gamma^T \begin{bmatrix} \widetilde{\boldsymbol{w}}_n^{k,1} & \widetilde{\boldsymbol{d}}_n^{k,2} & \widetilde{\boldsymbol{w}}_n^{k,2} & \cdots & \widetilde{\boldsymbol{d}}_n^{k,5} & \widetilde{\boldsymbol{w}}_n^{k,5} \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & \cdots & 0 & I \end{bmatrix}^T \boldsymbol{r}_{adj} \tag{3.28}$$

Finally, using $\widetilde{\boldsymbol{\psi}}^{k,m} = \boldsymbol{P}^T \widetilde{\boldsymbol{w}}^{k,m}$, the RK algorithm for solving the unsteady adjoint system becomes

$$\begin{aligned}
\widetilde{\boldsymbol{\psi}}_n^{k,5} &= \boldsymbol{P}^T \boldsymbol{r}_{adj} \\
\widetilde{\boldsymbol{d}}_n^{k,5} &= -a_5 \widetilde{\boldsymbol{\psi}}_n^{k,5} \\
\widetilde{\boldsymbol{\psi}}_n^{k,m} &= \boldsymbol{P}^T \left(-a_{m+1} \left(\boldsymbol{C}^T + \frac{3}{2\Delta t} I \right) \widetilde{\boldsymbol{\psi}}_n^{k,m+1} + \beta_{m+1} \boldsymbol{D}^T \widetilde{\boldsymbol{d}}_n^{k,m+1} \right) \\
\widetilde{\boldsymbol{d}}_n^{k,m} &= -a_m \widetilde{\boldsymbol{\psi}}_n^{k,m} + (1-\beta_{m+1}) \widetilde{\boldsymbol{d}}_n^{k,m+1} \\
\boldsymbol{\psi}_n^k &= \boldsymbol{\psi}_n^{k+1} + \widetilde{\boldsymbol{\psi}}_n^{k,1}
\end{aligned} \tag{3.29}$$

where $m = 5, 4, \cdots, 1$.

The important outcome of the analysis is that the unsteady terms referring to other real time-steps ($-\frac{2}{\Delta t}\boldsymbol{\psi}^{n+1}$, $\frac{1}{2\Delta t}\psi^{n+2}$) need to be added only once in the residual (stage 5 of the Runge-Kutta scheme), whereas the unsteady term that refer to the current real time-step ($\frac{3}{2\Delta t}\boldsymbol{\psi}^n$) is added at every Runge-Kutta stage since only $\boldsymbol{\psi}^n$ is being constantly updated.

3.2.1 Adjoint Multigrid

In eq. 2.51, operator $\mathcal{K}$ can be split into the following sub-operators

$$\mathcal{K} = \mathcal{K}_{re} \mathcal{K}_{RK} \mathcal{K}_{pr} \tag{3.30}$$

where $\mathcal{K}_{re}$ represents the restriction of the residual and the solution to a coarser grid, $\mathcal{K}_{RK}$ the Runge-Kutta scheme and $\mathcal{K}_{pr}$ the prolongation of solution's correction to a finer grid. For the adjoint solver, the transpose operation is needed, as shown in eq. 3.22, where

$$\mathcal{K}^T = \mathcal{K}_{pr}^T \mathcal{K}_{RK}^T \mathcal{K}_{re}^T \tag{3.31}$$

Eq. 3.31 shows that, apart for the adjoint version of the Runge-Kutta scheme that was derived, in order to use the multigrid technique, the transpose of the prolongation needs to be used as the adjoint restriction and vice-versa.

3.3 Algorithmic Differentiation in the Unsteady Adjoint Solver

Some of the differential terms of the unsteady adjoint equations are computed by employing algorithmic differentiation (AD) [176]. More specifically, the source code transformation technique is applied using the AD tool Tapenade [177] developed by INRIA. More information on the basic principles of AD can be found in Appendix A while the implementation of Tapenade on a steady CFD code is outlined in [178].

Revisiting fig. 2.6, it can be seen that the flow problem's setup to obtain the value of an objective function is a sequence of operations that starts with a number of inputs and results to a number of outputs similar to the one described in Appendix A. Since the number of output functions is smaller than the number of input design variables, one could assume that adjoint AD can be applied to the entire process to obtain the gradient $\frac{dJ}{d\boldsymbol{a}}$ or the entire flow solver. However, this is not the case due to a number of limitations. Applying AD to the entire chain is not possible because of the non-differentiability of certain steps such as parameterization and grid generation. For instance, term $\frac{d\boldsymbol{x}}{d\boldsymbol{a}}$ in eq. 3.37 is computed by means of finite differences by perturbing the design vector $\boldsymbol{a}$ by a small value ϵ for every element and getting the corresponding perturbed grid $\boldsymbol{x}$. Regarding the flow solver, AD tools are not yet very efficient handling iterative schemes, although progress has been made in this direction [179, 180], producing differentiated code with a large RAM footprint. To prevent this, the hand-differentiated analysis of section 3.2 is needed to solve the unsteady adjoint equations. Another reason is the fact that certain language features are not supported by the AD language parser. An example is the parallelization loops implemented by the external parallelization library OPlus [162, 163], despite the recent advances in the AD of the MPI (message passing interface) codes [181–183]. Because of the aforementioned, such a level of AD automation for the full chain is not possible and the analysis of sections 3.1, 3.2 and 3.2.1 that involves hand-differentiation is necessary in order to efficiently compute the gradient.

Nevertheless, AD is used selectively in order to aid the computation of two of the terms in eq. 3.5, namely $\frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}}^T \boldsymbol{\psi}$ and $\boldsymbol{g}$. It is assumed that a generic subroutine is given

```
subroutine res(→ u , ← r_st)
```

where the arrows indicate that $\boldsymbol{u}$ is an input and $\boldsymbol{r}_{st}$ an output. The subroutine takes the flow vector of the nodes of the computational domain and gives the corresponding

flow residual (without considering the unsteady terms). If Tapenade is applied to this subroutine in reverse mode, the result is a subroutine of the form

```
subroutine res_b(→ u, ← ū, ← r_st, → r̄_st)
```

where the computation $\overline{\boldsymbol{u}} = \left[\frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}}\right]^T \overline{\boldsymbol{r}_{st}} = \frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}}^T \overline{\boldsymbol{r}_{st}}$ takes place. So, if $\boldsymbol{\psi}$ is given as an input in place of $\overline{\boldsymbol{r}_{st}}$, then the result $\overline{\boldsymbol{u}}$ will contain the term $\frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}}^T \boldsymbol{\psi}$. Similarly, one may obtain $\boldsymbol{g}$ by applying adjoint mode AD to the subroutine

```
subroutine obj(→ u, ← j_st)
```

to obtain

```
subroutine obj_b(→ u, ← ū, ← j_st, → j̄_st)
```

So, in practice, the residual $\boldsymbol{r}_{st}$ is computed by assembling contributions of the various fluxes implemented by different subroutines. Tapenade is transforming, in reverse mode, only the part of the source code computing the fluxes over a single edge, i.e. the "leaves" of the "call-tree" in programming terms. The rest is hand-coded in order to wrap correctly the output code of Tapenade to maximize efficiency and reduce the memory overhead for the adjoint solver.

3.4 Adjoint Boundary Conditions

The boundary conditions which are implemented using inviscid fluxes can be directly translated to their adjoint variants of the boundary fluxes in the same fashion as for interior inviscid fluxes. Thus, AD can be used in a similar manner as for the interior inviscid flux subroutines.

For the solid walls, where boundary conditions were imposed by equations 2.41, the adjoint boundary conditions are derived by hand. The wall boundary conditions for the forward differentiation problem are formed by differentiating eq. 2.41 w.r.t. $\boldsymbol{a}$

$$\begin{aligned}(I-B)\left(\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{a}} + \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{u}}\frac{d\boldsymbol{u}}{d\boldsymbol{a}}\right) &= 0 \\ B\frac{d\boldsymbol{u}}{d\boldsymbol{a}} &= 0\end{aligned} \tag{3.32}$$

and adding by parts

$$\left[(I-B)\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{u}} + B\right]\frac{d\boldsymbol{u}}{d\boldsymbol{a}} + (I-B)\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{a}} = 0 \tag{3.33}$$

To obtain the solid wall boundary conditions for the adjoint problem, instead of using the mathematical analysis of the augmented objective function, for simplicity, the so-called adjoint equivalence is used. Starting from term $\mathcal{B}$ in eq. 3.1 and omitting time indices, we get

$$\frac{\partial j_{st}}{\partial \boldsymbol{u}}\frac{d\boldsymbol{u}}{d\boldsymbol{a}} = -\frac{\partial j_{st}}{\partial \boldsymbol{u}}\left[(I-B)\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{u}}+B\right]^{-1}(I-B)\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{a}} = \boldsymbol{\psi}^T(I-B)\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{a}} \tag{3.34}$$

where the adjoint vector $\boldsymbol{\psi}$ on the solid wall nodes is given by

$$B\boldsymbol{\psi}=0, \quad (I-B)\left(-\frac{\partial \boldsymbol{\psi}}{\partial t}+\left[\frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}}\right]^T\boldsymbol{\psi}\right) = \begin{cases} -\left[\frac{\partial j_{st}}{\partial \boldsymbol{u}}\right]^T, & t\in[t_0,T] \\ 0, & t\in[0,t_0) \end{cases} \tag{3.35}$$

Similarly to the flow boundary conditions, the adjoint vector is zeroed at the wall nodes along with the adjoint equations residual so that the adjoint vector is not updated during each pseudo-time iteration.

3.4.1 Adjoint Sliding Interface

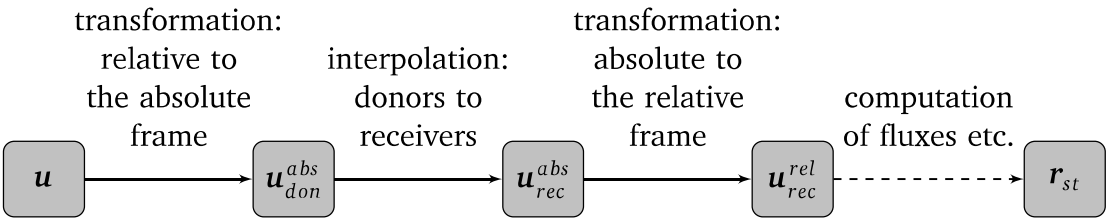


Figure 3.1: Sliding interface process for flow solver.

The sliding interface technique, described in section 2.4.1, is summarized schematically in fig. 3.1. To obtain $\frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}}^T\boldsymbol{\psi}$ for the sliding interface nodes, the chain rule is employed

$$\begin{aligned}\left[\frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}}\right]^T\boldsymbol{\psi} &= \left[\frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}^{rel}_{rec}}\frac{\partial \boldsymbol{u}^{rel}_{rec}}{\partial \boldsymbol{u}^{abs}_{rec}}\frac{\partial \boldsymbol{u}^{abs}_{rec}}{\partial \boldsymbol{u}^{abs}_{don}}\frac{\partial \boldsymbol{u}^{abs}_{don}}{\partial \boldsymbol{u}}\right]^T\boldsymbol{\psi} \\ &= \left[\frac{\partial \boldsymbol{u}^{abs}_{don}}{\partial \boldsymbol{u}}\right]^T\left[\frac{\partial \boldsymbol{u}^{abs}_{rec}}{\partial \boldsymbol{u}^{abs}_{don}}\right]^T\left[\frac{\partial \boldsymbol{u}^{rel}_{rec}}{\partial \boldsymbol{u}^{abs}_{rec}}\right]^T\left[\frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}^{rel}_{rec}}\right]^T\boldsymbol{\psi}\end{aligned} \tag{3.36}$$

In order to compute the final form of eq. 3.36, a combination of AD and "by hand"

programming is used. AD is used to compute the differential terms that correspond to low level operations or i.e. the "leaves" of the call-tree, such as the transformation from the absolute to relative frame of reference on a single node or the interpolation from the donors to a single receiver. These operations need to be performed in the correct, reverse order by the programmer. The parallelization of the execution of operations is implemented, similarly to the unsteady flow solver, by the OPlus library [162, 163]. Note that, because of the reverse flow of information, the interior sliding plane is now becoming the receiver and the exterior one the donor.

3.5 Adjoint Solver Algorithm

The structure of the code that is used to solve the unsteady adjoint equations can be summarized by the pseudo-code of algorithm 2.

Algorithm 2: Unsteady adjoint solver.

initialize $\boldsymbol{\psi}$;
$t \rightarrow t_{final}$;
while $t > 0$ **do** // real time loop
  read in $\boldsymbol{u}(t)$;
  compute objective_adj $\boldsymbol{g}$;
  $iter \rightarrow 0$;
  while $iter < iter_{max}$ *and* $\sum \boldsymbol{r}_{adj} > \epsilon$ **do** // pseudo-time loop
    multigrid(to_fine, to_coarse);
    if *to_fine* **then**
      restrict_adj;
    else if *to_coarse* **then**
      prolong_adj;
    end
    $iter^{mg} \rightarrow 0$;
    while $iter^{mg} < iter^{mg}_{max}$ **do** // multigrid levels loop
      $stage_{RK} \rightarrow 5$;
      while $stage_{RK} \geq 1$ **do** // 5 stage Runge-Kutta loop
        compute fluxes_adj $\frac{\partial \boldsymbol{r}_{st}}{\partial \boldsymbol{u}}^T \boldsymbol{\psi}$;
        impose BCs_adj;
        compute residual $\boldsymbol{r}_{adj}$;
        update solution $\boldsymbol{\psi}$;
        $stage_{RK} \rightarrow stage_{RK} - 1$;
      end
      $iter^{mg} \rightarrow iter^{mg} + 1$;
    end
    $iter \rightarrow iter + 1$;
  end
  $t \rightarrow t - \Delta t$;
end

3.6 Gradient Computation Algorithm

In order to compute the desired final gradient, eq. 3.7 is used which can be further developed as

$$\frac{dJ}{d\boldsymbol{a}} = \Delta t \sum_{n=n_0}^{N} \frac{\partial j_{st,n}}{\partial \boldsymbol{x}} \frac{d\boldsymbol{x}}{d\boldsymbol{a}} + \Delta t \sum_{n=0}^{N} \boldsymbol{\psi}_n^T \frac{\partial \boldsymbol{r}_{st,n}}{\partial \boldsymbol{x}} \frac{d\boldsymbol{x}}{d\boldsymbol{a}} \tag{3.37}$$

The computation methods for the different terms are listed below.

- $\boldsymbol{\psi}^T$ is computed by solving the unsteady adjoint equations,
- $\boldsymbol{\psi}^T \frac{\partial \boldsymbol{r}_{st,n}}{\partial \boldsymbol{x}}$ is computed by applying adjoint mode AD to the flux subroutines of the flow solver, differentiating the steady residual w.r.t. the grid nodes coordinates $\boldsymbol{x}$ while using $\boldsymbol{\psi}$ as an input,
- $\frac{\partial j_{st,n}}{\partial \boldsymbol{x}}$ is computed by applying adjoint mode AD to the post-processing step of computing the objective function, differentiating the objective function integrand $j_{st,n}$ w.r.t. the grid nodes coordinates $\boldsymbol{x}$ and
- $\frac{d\boldsymbol{x}}{d\boldsymbol{a}}$ is computed by means of finite differences using a fast parametric meshing tool [173].

Fig. 3.2 is a flow graph of how the terms in eq. 3.7 are constructed and assembled in order to compute the total derivative $\frac{\partial J}{\partial \boldsymbol{a}}$.

3.7 Unsteady Adjoint for Periodic Flows

The formulation of an unsteady adjoint solver in the time-domain allows applying it to transient flows. However, it is desired to take advantage of periodicity when the user knows beforehand that the flow is periodic. Usually, in the case of a periodic flow, the objective function is defined over the duration of a single period, which is the case for the periodic applications of chapter 4. As mentioned, the flow solution needs to be stored on the disk in order to be read in during the adjoint computations. In the case of a periodic flow, the flow of only a single period is stored on the disk, thus reducing significantly the storage space requirements of the computation.

Regarding the unsteady adjoint computations, there are two ways to compute the gradient of the objective function w.r.t. the design variables:

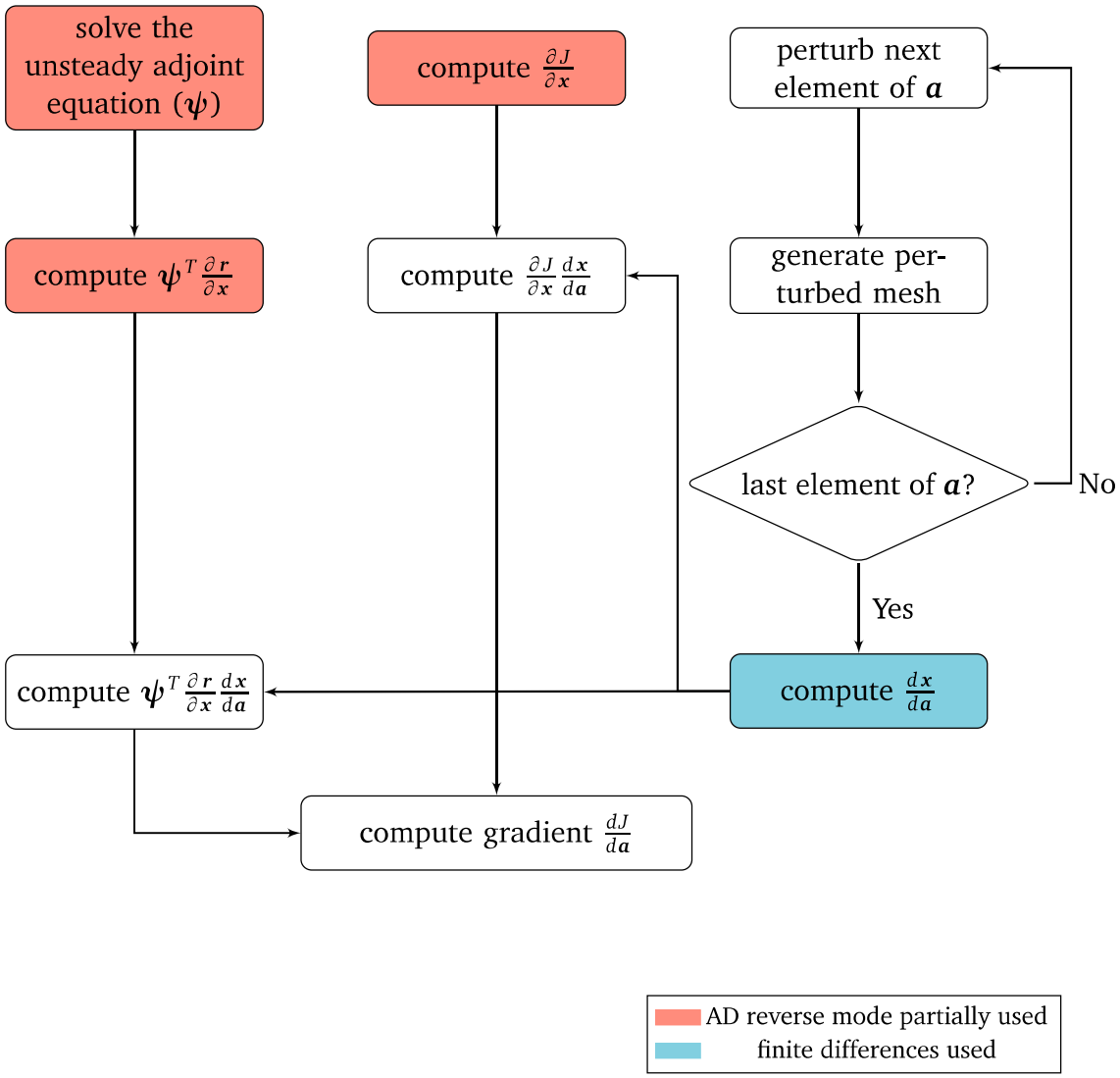


Figure 3.2: Unsteady gradient computation.

- The first one involves considering the objective function defined over only the last period of the flow solution. This means that term $\left[\frac{\partial J_{st}}{\partial u^n}\right]^T$ in eq. 3.5 is non-zero only for the first period of the adjoint computations. Zeroing the aforementioned term during the adjoint solution after the first period will eventually lead the adjoint solution to be zeroed too, as it will be shown below. The gradient is computed considering the adjoint solution from the beginning of the computations until the adjoint variables' fields decay to zero.

- The second one involves considering the objective function defined over each and every single period of the flow solution. The adjoint solution after a number

of initial periods will also get periodic. In order to compute the gradient, only one period of the adjoint variables' fields will be used.

To demonstrate the aforementioned, an 1D simple example is used.

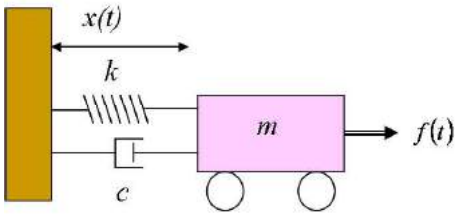


Figure 3.3: A spring-mass system with damping and external forcing.

A sping-mass system, fig. 3.3, is affected by a sinusoidal external force. The differential equation that governs this motion is

$$m\ddot{x} + c\dot{x} + kx = F_0 sin(\omega t) \tag{3.38}$$

where x denotes the position, m the mass, c the damping coefficient, k the spring constant and $F_0 sin(\omega t)$ the forcing with t denoting time and ω the angular frequency. The system can be rewritten in the following form

$$\begin{aligned} \boldsymbol{A}\dot{\boldsymbol{x}} + \boldsymbol{B}\boldsymbol{x} &= \boldsymbol{f} \\ \Leftrightarrow \dot{\boldsymbol{x}} &= \boldsymbol{A}^{-1}(\boldsymbol{f} - \boldsymbol{B}\boldsymbol{x}) \\ \Leftrightarrow \dot{\boldsymbol{x}} &= \boldsymbol{g}(\boldsymbol{x}, t) \end{aligned} \tag{3.39}$$

where $\boldsymbol{A} = \begin{bmatrix} 0 & m \\ -1 & 0 \end{bmatrix}$, $\boldsymbol{B} = \begin{bmatrix} k & c \\ 0 & 1 \end{bmatrix}$, $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$ and $\boldsymbol{f} = \begin{bmatrix} F_0 sin(\omega t) \\ 0 \end{bmatrix}$.
In order to solve eq. (3.39), a backward Euler, first order time discretization scheme is used

$$\begin{aligned} \frac{\boldsymbol{x}^{n+1} - \boldsymbol{x}^n}{\Delta t} &= \boldsymbol{g}^{n+1}(\boldsymbol{x}, t) \\ \Leftrightarrow \left(\frac{\boldsymbol{A}}{\Delta t} + \boldsymbol{B}\right)\boldsymbol{x}^{n+1} &= \boldsymbol{f}^{n+1} + \frac{\boldsymbol{A}}{\Delta t}\boldsymbol{x}^n \end{aligned} \tag{3.40}$$

where n denotes the time-step's number and Δt the time-step's duration. S.I. units are assumed and the following values are set: $m = 2.5$, $c = 75.0$, $k = 2000.0$, $F_0 = 30.0$ and $T_p = 0.2$. T_p denotes the period and $\omega = \frac{2\pi}{T_p}$. The initial condition for $\boldsymbol{x}$ is set to

$\boldsymbol{x}^1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. The simulation lasts from $T_0 = 0$ to $T_{tot} = 1$ time unit.

Using a sufficiently small time-step, the solution $\boldsymbol{x}$ is obtained for the entire time-domain plotted in fig. 3.4.

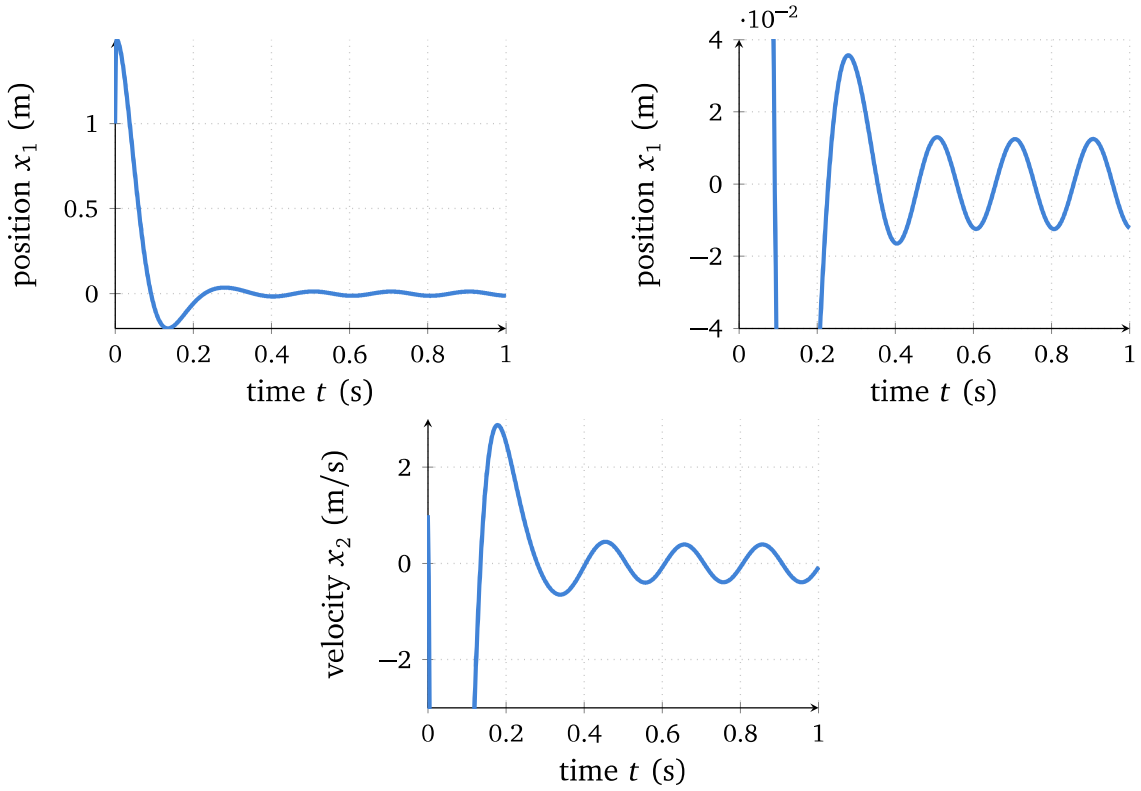


Figure 3.4: State equation's solution $\boldsymbol{x}$ plotted over time. Top left: Position x_1. Top right: Position x_1, close-up view. Bottom: Velocity x_2, close-up view.

The objective function J is defined only for the last period as follows

$$J = \frac{1}{T_p} \int_{T_{tot}-T_p}^{T_{tot}} \dot{x}^2 dt \tag{3.41}$$

The value of the objective function is $J = 7.6903E-002$.

The gradient of the objective function w.r.t. the design variables $\begin{bmatrix} \frac{dJ}{dm} & \frac{dJ}{dc} & \frac{dJ}{dk} \end{bmatrix}^T$ is computed using both of the ways mentioned in this section.

For the first way, the objective function is defined only for the last period ($0.8 \leq t \leq 1$). The adjoint solution is obtained by means of discrete adjoint and can be seen in fig. 3.5. As soon as the objective function in not defined ($t = 0.8$), the adjoint variables'

fields decay to zero. This observation can be explained by the following mathematical analysis. If the governing equation's residual $\boldsymbol{r}$ is defined as

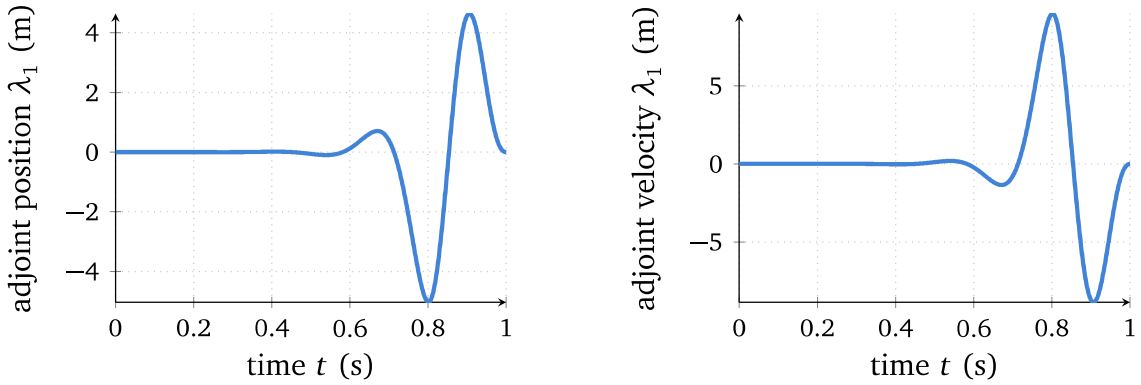


Figure 3.5: Adjoint solution $\boldsymbol{\lambda}$ plotted over time. Left: Adjoint position λ_1. Right: Adjoint velocity λ_2.

$$\boldsymbol{r} = \boldsymbol{A}\dot{\boldsymbol{x}} + \boldsymbol{B}\boldsymbol{x} - \boldsymbol{f} = 0 \tag{3.42}$$

then the adjoint equation becomes

$$\left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}}\right]^T \boldsymbol{\lambda} = \left[\frac{\partial J}{\partial \boldsymbol{x}}\right]^T \tag{3.43}$$

where $\boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}$ is the adjoint vector.

Using a backward-Euler scheme and by denoting the time-step where we start computing the objective function with n_0 (i.e. $t_{n_0} = t_{tot} - T_p$), eq. (3.43) is expanded as follows

$$\begin{bmatrix} \frac{\boldsymbol{A}}{dt} + \boldsymbol{B} & -\frac{\boldsymbol{A}}{dt} & 0 & \cdots & & 0 \\ 0 & \frac{\boldsymbol{A}}{dt} + \boldsymbol{B} & -\frac{\boldsymbol{A}}{dt} & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots & & \vdots \\ & & & & & \\ & & & & & \\ 0 & & & & & -\frac{\boldsymbol{A}}{dt} \\ 0 & 0 & \cdots & & 0 & \frac{\boldsymbol{A}}{dt} + \boldsymbol{B} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda}^1 \\ \boldsymbol{\lambda}^2 \\ \vdots \\ \\ \\ \\ \vdots \\ \boldsymbol{\lambda}^N \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \\ 0 \\ \frac{\partial J^{n_0}}{\partial \boldsymbol{x}^{n_0}} \\ \vdots \\ \\ \frac{\partial J^N}{\partial \boldsymbol{x}^N} \end{bmatrix} \tag{3.44}$$

The right hand side of eq. 3.44 is zero $\forall n \in [1, n_0)$ and the adjoint equation per time-step is transforming from

$$\left(\frac{\boldsymbol{A}}{dt} + \boldsymbol{B}\right)\boldsymbol{\lambda}^n = \frac{\partial J^n}{\partial \boldsymbol{\lambda}^n} + \frac{\boldsymbol{A}}{dt}\boldsymbol{\lambda}^{n+1} \tag{3.45}$$

to

$$\left(\frac{\boldsymbol{A}}{dt} + \boldsymbol{B}\right)\boldsymbol{\lambda}^n = \frac{\boldsymbol{A}}{dt}\boldsymbol{\lambda}^{n+1}, \qquad \forall n \in [1, k) \tag{3.46}$$

Eq. 3.46 can also be written as

$$\begin{aligned} \boldsymbol{A}\frac{d\boldsymbol{\lambda}}{dt} &= -\boldsymbol{B}\boldsymbol{\lambda}^4, && \forall n \in [1, k) \\ \Leftrightarrow \boldsymbol{A}\frac{1}{\boldsymbol{\lambda}}d\boldsymbol{\lambda} &= -\boldsymbol{B}dt, && \forall n \in [1, k) \\ \Leftrightarrow \boldsymbol{\lambda}(t) &= \boldsymbol{\lambda}^{n_0} e^{-\boldsymbol{A}^{-1}\boldsymbol{B}(t_{n_0} - t)}, && \forall n \in [1, k) \end{aligned} \tag{3.47}$$

Eq. 3.47 shows that the adjoint solution exponentially decays to zero if the eigenvalues of the $\left[\boldsymbol{A}^{-1}\boldsymbol{B}\right]$ matrix are located in the right half of the complex plane, i.e. if the real part of the eigenvalues is positive. In the spring-mass example, the two eigenvalues confirm that observation, as shown in fig. 3.6.

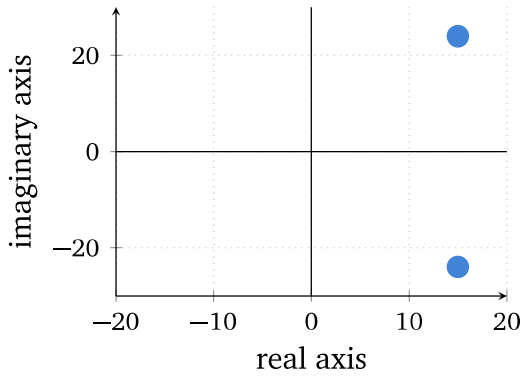


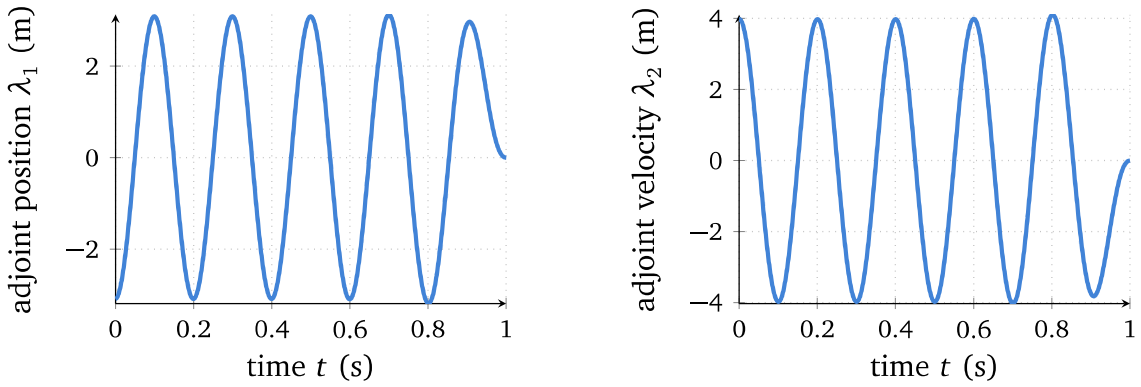
Figure 3.6: Plot of the eigenvalues of "$\boldsymbol{A}^{-1}\boldsymbol{B}$".

This can be generalized for CFD adjoint computations like the ones developed in this chapter with the equivalent expression

$$\boldsymbol{\psi}(t) = \boldsymbol{\psi}^{n_0} e^{-\left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{u}}\right]^T (t_{n_0} - t)} \tag{3.48}$$

The matrix $\left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{u}}\right]^T$ has eigenvalues with positive real parts for strongly stable numerical schemes and, thus, the adjoint solution will decay to zero after the objective function is not defined.

The second way of computing the gradient involves considering the objective function defined at each period of the governing problem. Thus, the adjoint solution has a periodic form as in fig. 3.7. In order to obtain the gradient of the objective function, the adjoint solution of the last (backwards) period is used ($0 \leq t \leq 0.2$).



(a) Adjoint position λ_1.

(b) Adjoint velocity λ_2.

Figure 3.7: Adjoint solution $\boldsymbol{\lambda}$ plotted over time.

Table 3.1 compares the derivatives obtained by:

- finite differences,
- the objective function being defined over a single period and using the entire adjoint solution trajectory (way A) and
- the objective function being defined over all periods and using the adjoint solution of a single period (way B) which are in total agreement.

	dJ/dm	dJ/dc	dJ/dk
Finite Differences	$-1.2400746E-002$	$-1.9687041E-003$	$1.2399984E-005$
Way A	$-1.2400087E-002$	$-1.9683723E-003$	$1.2398832E-005$
Way B	$-1.2400398E-002$	$-1.9684860E-003$	$1.2393586E-005$

Table 3.1: Comparison of derivatives calculated by discrete adjoint.

3.8 Temporal Coarsening

In order to reduce the overall run time of the adjoint computations and lower the disk storage footprint of the flow solution, temporal and spatial coarsening techniques were tested in [126]. It was found that the gradients' accuracy is significantly compromised when coarsening is applied to both the flow and adjoint solvers. However, if the coarsening is applied only to the adjoint solver, the gradients' accuracy is retained to the desired extent and the cost is remarkably reduced.

In this work, the temporal coarsening technique is optionally used to reduce the disk space requirements for storing the flow solution time-series. In order to apply the method, the unsteady flow solver runs using the baseline time-discretization. However, the unsteady adjoint solver is marching backwards in time considering every n_cth time-step, thus increasing the time-step from Δt to $n_c \Delta t$. So, the flow solution needs to be stored only every n_c time-steps. A schematic of the technique is seen in fig. 3.8. The required storage space for the flow solution and the running time of the adjoint solver are therefore reduced by a factor of n_c.

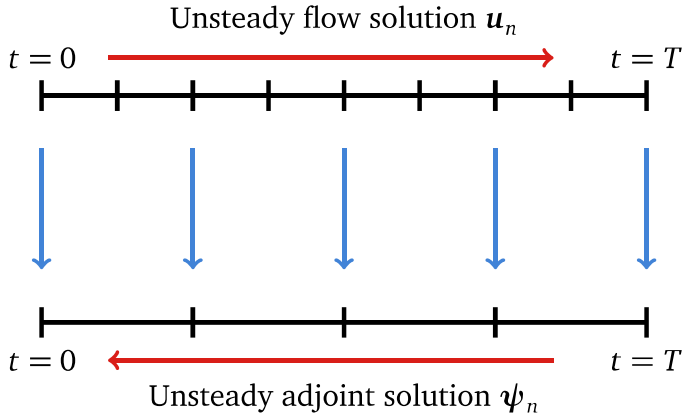


Figure 3.8: Schematic of temporal coarsening for unsteady adjoint solver.

A comparison of gradients after using different temporal coarsening set-ups and no temporal coarsening can be viewed in section 4.5.

3.9 Optimization Algorithm

In chapter 4, the gradients computed via the unsteady adjoint method are used for various optimization scenarios. In this section, the fundamental optimization algorithm

which is used is presented.

Initially, the baseline geometry is given and an appropriate parameterization is chosen in order to define the design space and the possible ways that the geometry can be modified. Then, the spatial domain is discretized by generating the computational grid. The flow equations are solved and the required objective functions are computed. If equality constraints are imposed, then their values are also obtained. The unsteady adjoint equations are solved to compute the gradients of each objective function and constraint. In the presence of equality constraints, then the perpendicular component of the gradient of the objective w.r.t. the gradients of the constraints, fig. 3.9, is computed as follows [184]

$$\left.\frac{dJ}{d\boldsymbol{a}}\right|_{\perp} = \left[I - \boldsymbol{M}^T\left(\boldsymbol{M}\boldsymbol{M}^T\right)^{-1}\boldsymbol{M}\right]\cdot\frac{dJ}{d\boldsymbol{a}} \tag{3.49}$$

where $\boldsymbol{M}$ is a matrix, formed by the gradients of the constraints as rows.

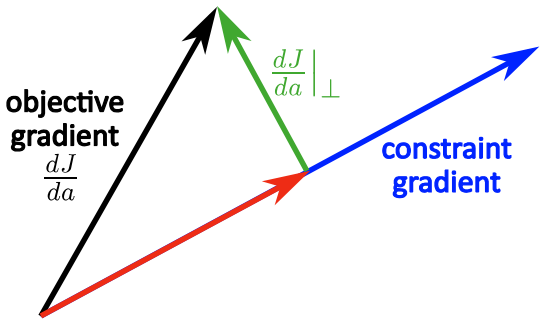


Figure 3.9: Schematic of perpendicular component of gradient of the objective function to the gradient of the constraint.

Finally, the design variables are modified using the gradient descent method [185] if no constraints are considered or the projected gradient descent, in the case of equality constraints,

$$\boldsymbol{a}_{new} = \boldsymbol{a}_{old} - s\left[\frac{dJ}{d\boldsymbol{a}_{old}}\right]^T \tag{3.50}$$

where s is the user-defined value of the step size. Thus, a new geometry is created before continuing to the next optimization loop.

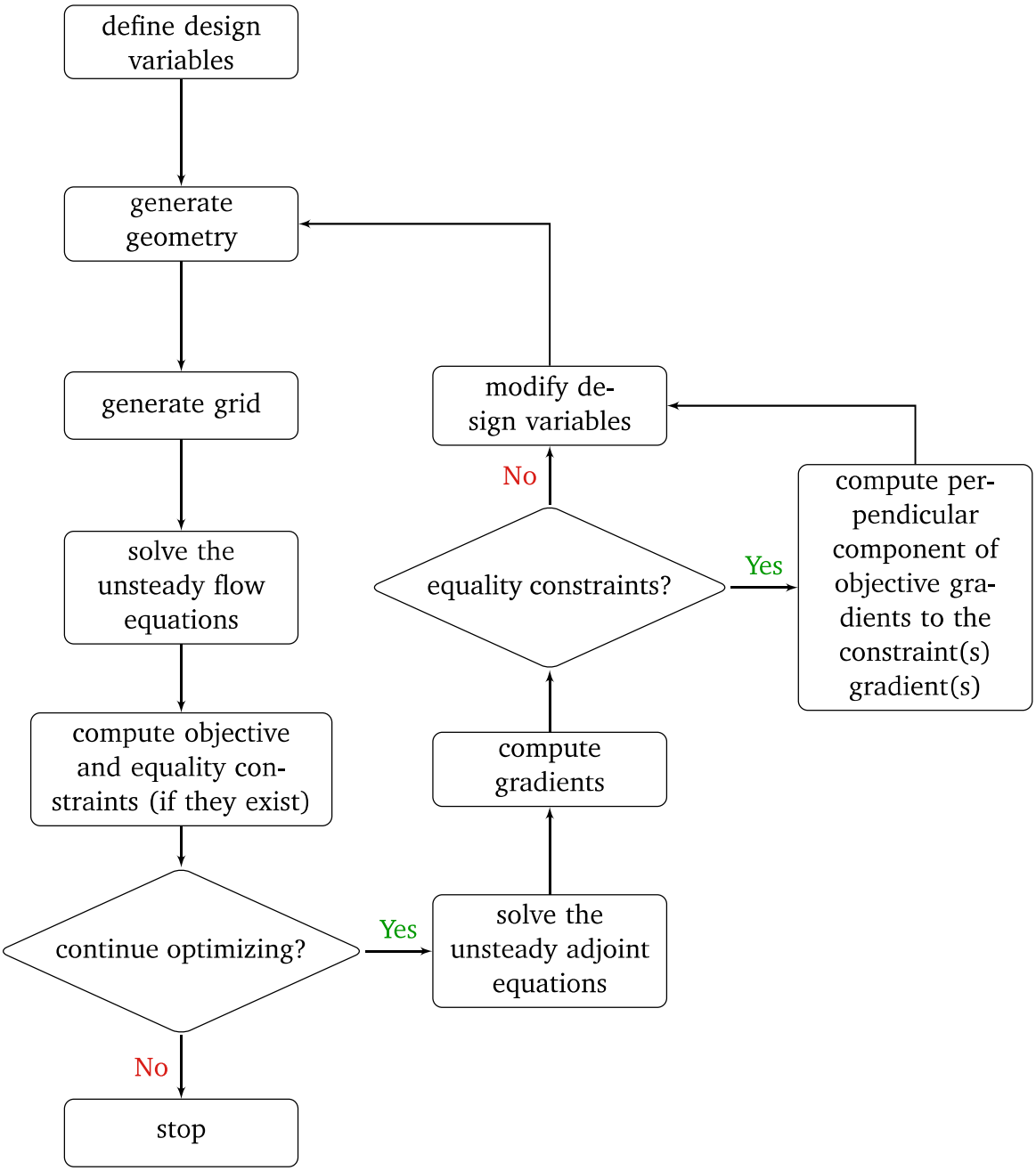


Figure 3.10: Optimization algorithm.

3.10 Cost of Unsteady Adjoint

3.10.1 Time Cost

The running time required for one pseudo-time iteration of the adjoint solver comes at a cost which is roughly three times the cost of one pseudo-time iteration of the flow solver. This observation is independent of whether the steady or unsteady variants are used and can be explained by a number of factors. As seen in Appendix A, AD in adjoint mode needs to have access to the flow solver code variables and a balance between the "recompute all" or "store all" approaches needs to be found. Both of these approaches add a cost overhead to the adjoint solver. The recomputation of intermediate variables is obviously adding calls to the flow solver subroutines during the adjoint computation, thus adding running time. Moreover, storing values in the memory also adds a (relatively smaller) time cost. That is partly because of the time needed for storage and retrieval of values in the memory stack by the "push" and "pop" commands. Another reason is that the additional memory needed for storing both the adjoint and intermediate variables makes the access to memory less efficient, increasing the overall time.

3.10.2 Storage Space Cost

The unsteady adjoint solver needs to have access to the corresponding unsteady flow solution. That increases the storage space cost linearly with the number of time-steps used. The use of temporal coarsening has been shown to reduce significantly the storage cost while maintaining a certain level of gradient accuracy. On the other hand, storing the unsteady adjoint solution is not necessary. During the adjoint computations, the per-time-step gradient ($\boldsymbol{\psi}_n^T \frac{\partial \boldsymbol{r}_n}{\partial \boldsymbol{x}} \frac{d\boldsymbol{x}}{d\boldsymbol{a}}$) is obtained and stored using the adjoint variables' fields of the corresponding time-step and then, the "old" adjoint variables' fields are disposed while marching backwards in time.

4

Applications

The main focus of this chapter is to present the application of the unsteady adjoint solver, formulated in the time-domain to 3D, multi-row, turbomachinery cases for the first time in the literature. The flow solver is additionally validated against two benchmark cases with available experimental data, namely a turbine vane and a compressor stator. Then, three cases with unsteady flows, namely the transient flow in a turbine vane and periodic flows in a turbine stage and a three-row compressor setup are studied. The gradients obtained by the unsteady adjoint method are compared with those obtained by finite-differences for the turbine vane and turbine stage cases. An unconstrained optimization process is performed for the turbine vane and turbine stage cases, while a constrained optimization setup is used for the compressor case. Moreover, for the turbine stage and the compressor case, the flow and adjoint variables' fields along with the sensitivity maps are compared with the corresponding steady flow results to shed light into the advantages of using unsteady flow and adjoint solvers. Finally, the temporal coarsening technique is used as a means of reducing storage space requirements and the gradients obtained via this method are compared with those of the standard method.

4.1 Flow Solver Validation

Hydra is a CFD solver that has been extensively used for over 10 years for the flow prediction in industrial turbomachinery cases. However, for completeness, two benchmark test cases are used to compare CFD and experimental data.

4.1.1 VKI LS89 Turbine Vane

The VKI LS89 test-case [186] has been frequently used to benchmark the aero-thermal predictive capabilities of CFD codes. The 2D LS89 blade profile is a high-pressure turbine nozzle guide vane developed by the von Karman Institute for Fluid Dynamics, and the case is representative of aero-engines. The geometry and geometric parameters of the LS89 blade airfoil can be viewed in fig. 4.1 and table 4.1.

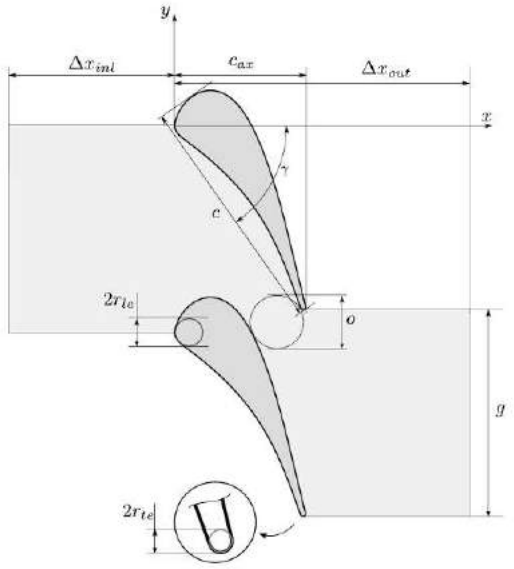


Figure 4.1: VKI LS89 turbine vane: Definition of the geometric parameters.

Parameter		Value
Chord	c	67.6 mm
Pitch	g	57.5 mm
Throat	o	14.9 mm
LE radius	r_{le}	4.1 mm
TE radius	r_{te}	0.7 mm
Axial chord	c_{ax}	36.9 mm
Inlet length	Δx_{inl}	55.0 mm
Outlet length	Δx_{out}	64.0 mm
Stagger angle	γ	55 °

Table 4.1: VKI LS89 turbine vane: Value of geometric parameters defined in fig. 4.1.

Hydra is a 3D CFD solver, so a quasi-3D grid is created out of the 2D geometry consisting of 57359 nodes per 2D slice, as seen in fig. 4.2. The maximum y^+ of the first nodes off the wall is 10 and wall functions are used. Flow conditions correspond to the transonic test number MUR47 as described in [187]. Uniform flow conditions are imposed at the inlet and outlet as listed in table 4.2. Zero velocity is imposed at the vane's surface nodes whereas only the normal component of the velocity is zeroed along the lateral boundaries of the domain. For the remaining two boundaries, peri-

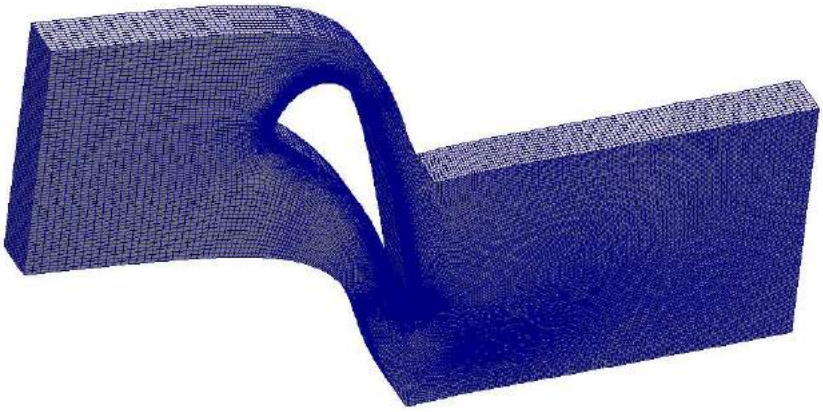


Figure 4.2: VKI LS89 turbine vane. Grid and computational domain.

Inlet total temperature	$T_{t,in}$	$420K$
Inlet total pressure	$P_{t,in}$	$1.596 \cdot 10^5 Pa$
Inlet flow angle	a_{in}	0.0^o
Inlet turbulence intensity	$T_{u,in}$	3%
Outlet static pressure	P_{out}	$0.8235 \cdot 10^5 Pa$

Table 4.2: VKI LS89 turbine vane. Boundary conditions at the inlet and outlet.

odicity is enforced. The convergence history of the flow equations is plotted in fig. 4.3, using the sum of the root mean square (RMS) of the residuals of all flow equations, defined as

$$r_{RMS} = \frac{1}{N_{PDEs} N_{nodes}} \left(\sum_{i=1}^{N_{PDEs}} \sum_{j=1}^{N_{nodes}} r_{ij}^2 \right)^{1/2} \tag{4.1}$$

where N_{PDEs} is the size of the flow vector (= 6 when using a one-equation turbulence model) and N_{nodes} is the total number of nodes. The resulting Mach number field is shown in fig. 4.4 where the presence of a shock wave over the suction side close to the trailing edge can be seen.

To validate the CFD solver, the isentropic Mach number distribution is compared with

the available experimental data. By definition, for a perfect gas,

$$M_{is} = \sqrt{\frac{2}{\gamma - 1}\left[\left(\frac{P_{t,in}}{P}\right)^{\frac{\gamma-1}{\gamma}} - 1\right]} \tag{4.2}$$

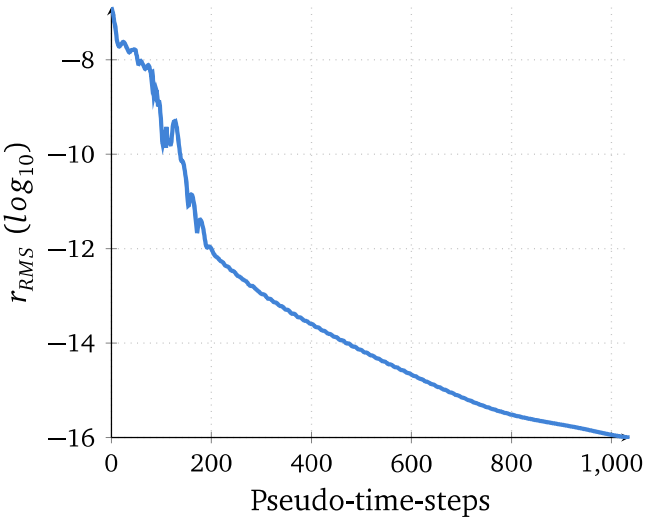


Figure 4.3: VKI LS89 turbine vane. Convergence history of the RMS of the flow equations' residuals.

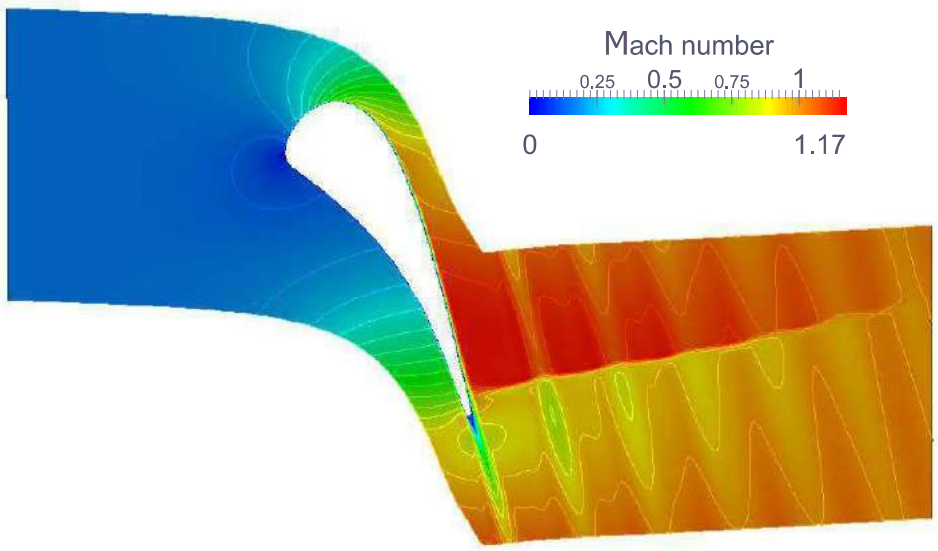


Figure 4.4: VKI LS89 turbine vane. Mach number iso-areas.

where P is the static pressure and γ the ratio of specific heats. The comparison between experimental measurements and CFD results is shown in fig. 4.5. The CFD results are in satisfactory agreement with the experimental data, despite the small differences observed on the pressure side close to the trailing edge, due to the challenging task of accurately capturing the shock wave by a RANS solver. Better results can be obtained using LES as shown in [188, 189].

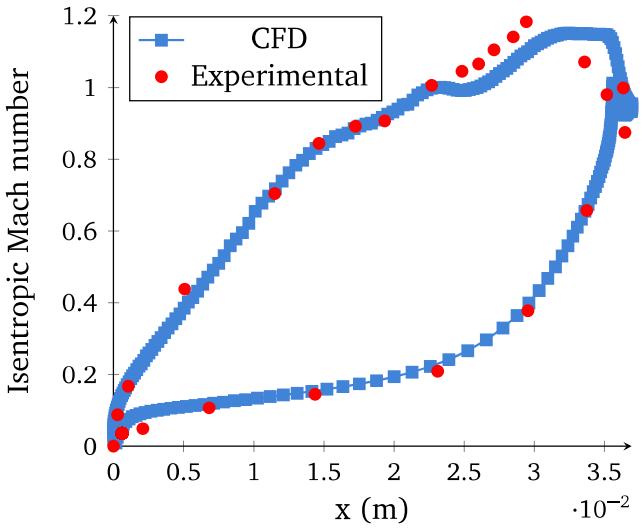


Figure 4.5: VKI LS89 turbine vane. Isentropic Mach number distribution along the x-coordinate of suction and pressure sides.

4.1.2 TurboLab Stator Blade

The TurboLab stator blade is used as a second validation case. This is a 3D compressor stator blade in a measurement rig, fig. 4.6, of the TU Berlin which was used for benchmarking different flow solvers and comparing optimization results [190, 191]. The basic geometric parameters are listed in table 4.3 and the boundary condition profiles are plotted in fig. 4.10.

Parameter	Value
Hub radius	147.5 mm
Tip radius	297.5 mm
Throat	14.9 mm
LE radius	1.9 mm
TE radius	1.3 mm
Axial chord	182.2 mm

Table 4.3: TurboLab stator. Geometric data.

Figure 4.8 shows the geometry, the CFD domain and a cut of the grid at midspan. The grid has 1975380 nodes and is a combination of an O-type grid in the area close and around the blade along with an H-type grid for the rest of the computational domain. The convergence history of the flow equations is shown in fig. 4.9. Figure 4.10 shows a radial cut of the Mach number iso-areas and the streamlines formed on the suction side of the stator blade, where flow separation occurs.

Figure 4.6: TurboLab stator. Photo of the rig. *Image from:* `http://aboutflow.sems.qmul.ac.uk/events/munich2016/benchmark/testcase3/`.

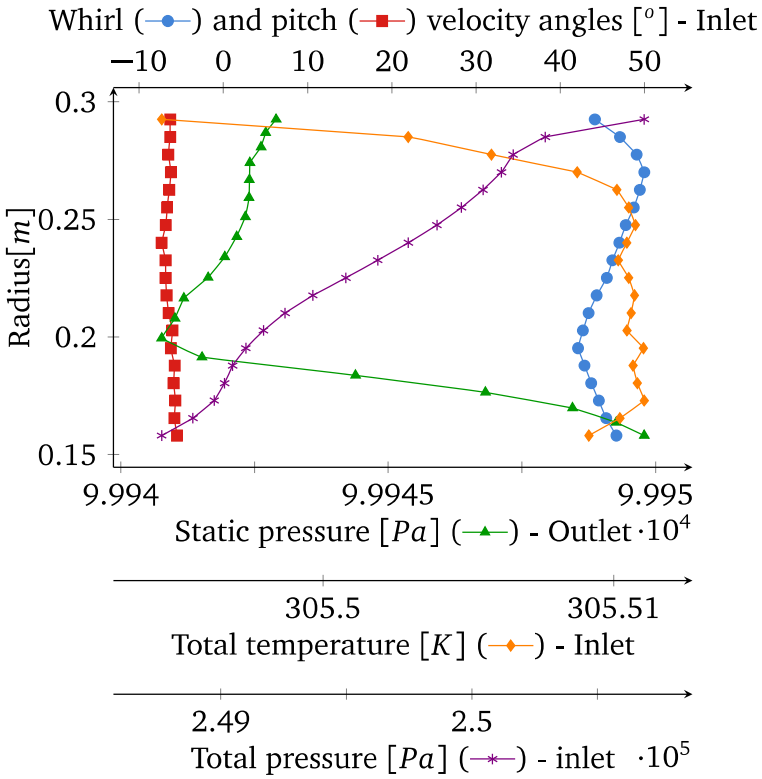


Figure 4.7: TurboLab stator. Inlet and outlet flow profiles.

Using the flow field, the total pressure and whirl angle at the outlet and the total pressure difference between the outlet and the inlet are circumferentially averaged and plotted span-wise versus the experimental data in fig. 4.11. For this configuration, the measurements were taken at different time instants for the inlet and outlet and, so, the measured ambient pressures at the inlet and the outlet were different. In order to compare the measurements with the CFD results, a correction is applied for the experimental total pressure at the outlet of the following form

$$P_t^{corr} = P_t \frac{P_{amb}^{in}}{P_{amb}^{out}} \tag{4.3}$$

where the index amb stands for ambient values. The CFD results match the experimental data with adequate accuracy.

Figure 4.8: TurboLab stator. Geometry and radial grid cut at midspan.

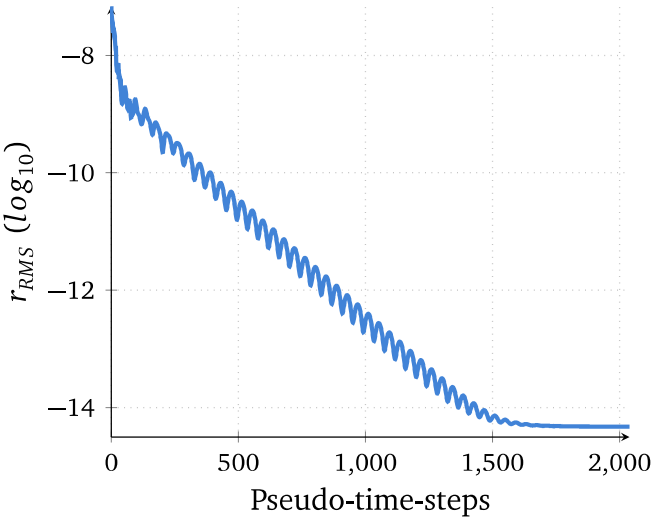


Figure 4.9: TurboLab stator. Convergence history of the flow equations residual.

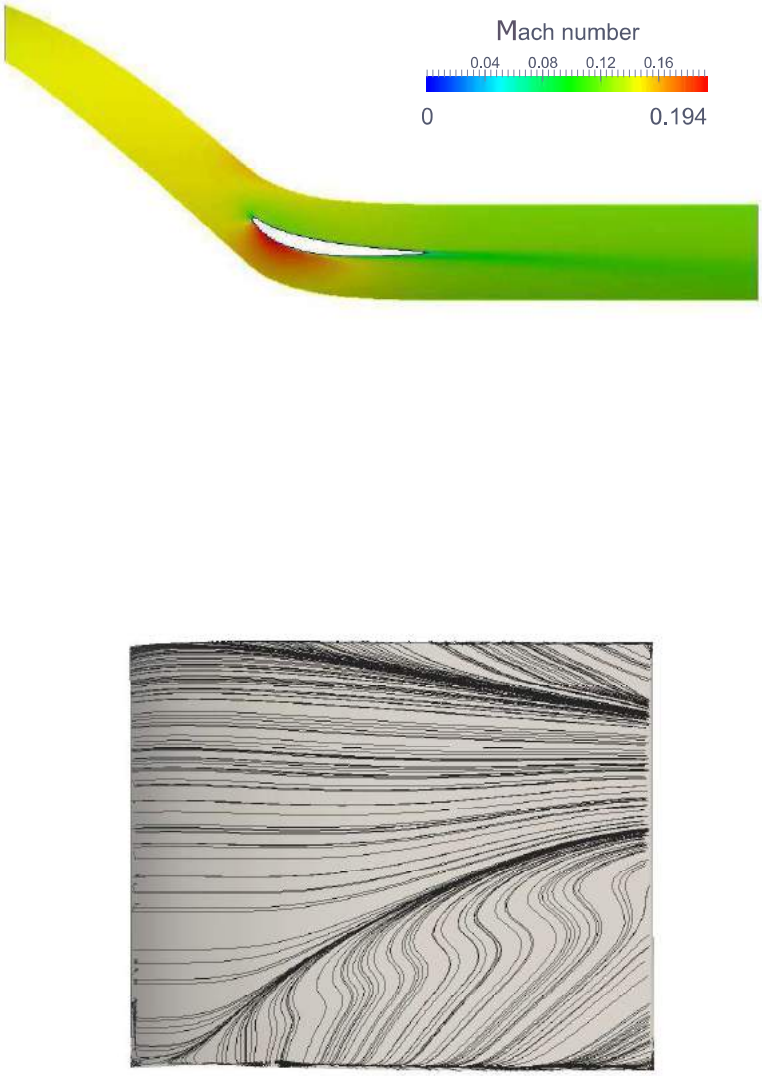


Figure 4.10: TurboLab stator. Top: 3D radial cut of the Mach number iso-areas at midspan. Bottom: Streamlines on the suction side, flow direction from left to right.

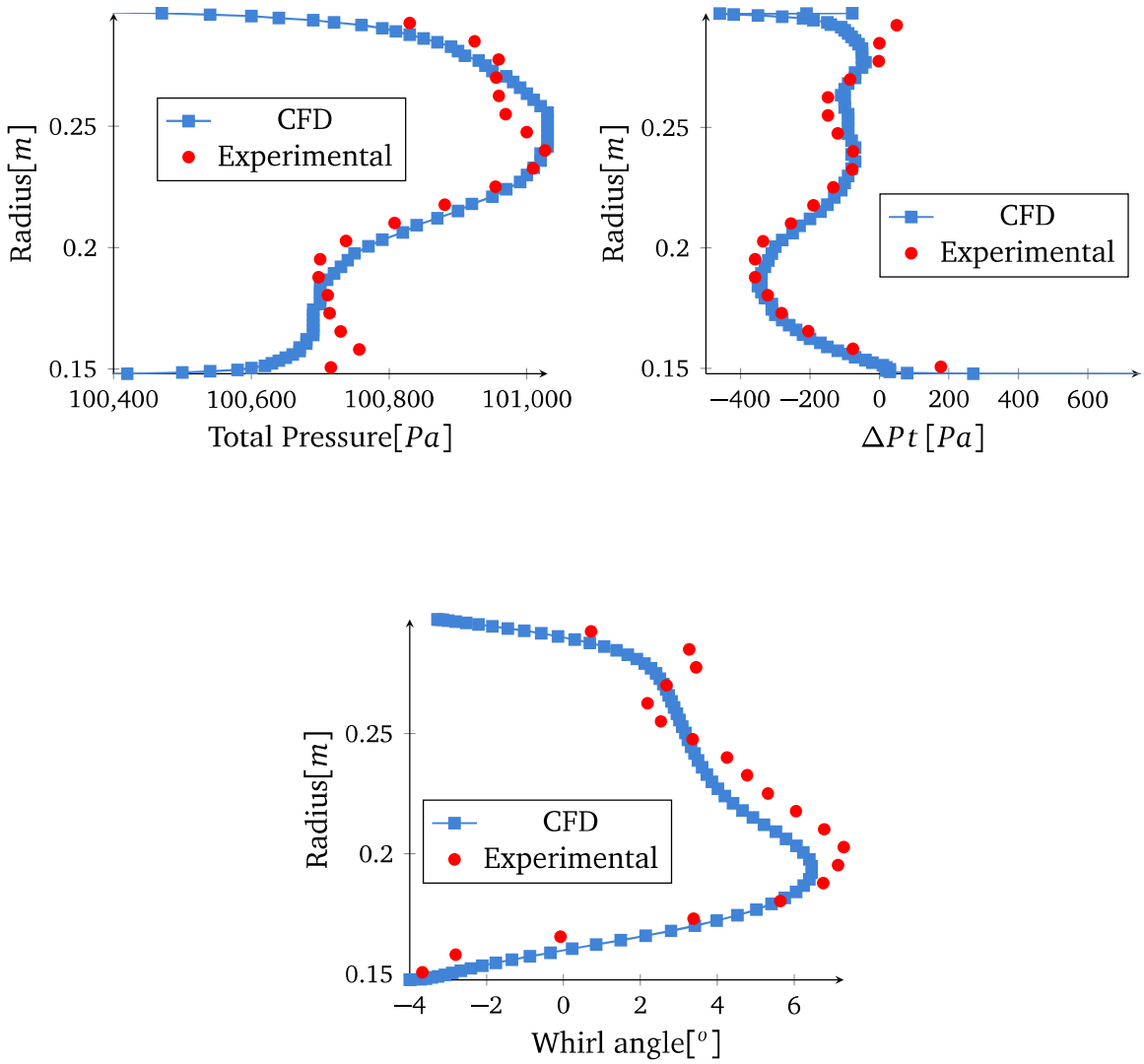


Figure 4.11: TurboLab stator. Comparison of circumferentially averaged values at the CFD domain's outlet between CFD results and experimental measurements. Top left: Total pressure at the outlet. Top right: Total pressure difference between outlet and inlet. Bottom: Whirl angle at the outlet.

4.2 High-Pressure Turbine Vane

As a first industrial case, where the unsteady adjoint solver is applied, the transient flow in an annular 3D High-Pressure Turbine (HPT) vane is studied. The basic geometric parameters of the vane are defined in fig. 4.12. For the section at the vane's midspan, the chord is $c = 59mm$, the stagger angle is $\gamma = -68°$ and the throat is $o = 9.5mm$. The geometry along with a blade-to-blade radial cut of the grid at midspan and details of the surface grid are shown in fig. 4.13. The grid has 955290 nodes and comprises an O-type grid close and around the vane and an H-type grid for the rest of the domain.

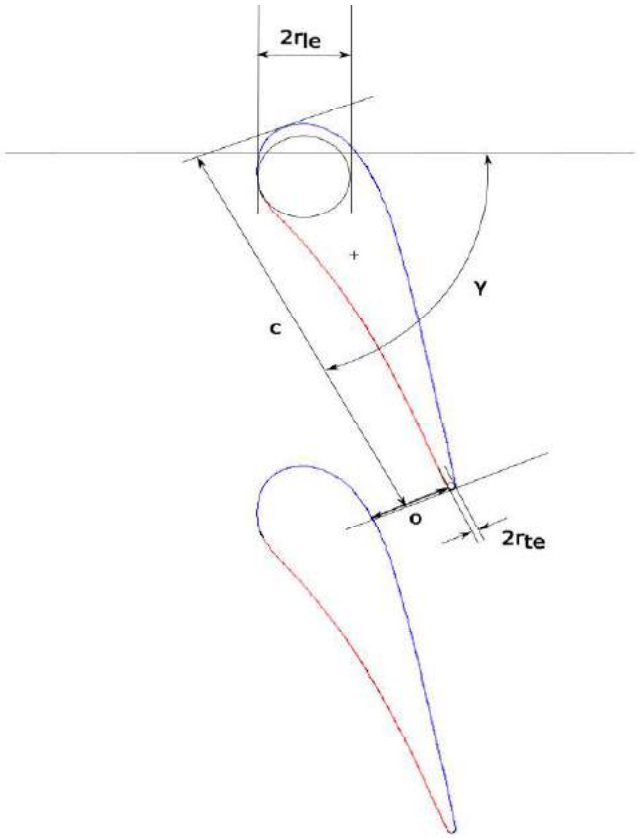


Figure 4.12: HPT vane. Definition of basic geometric parameters.

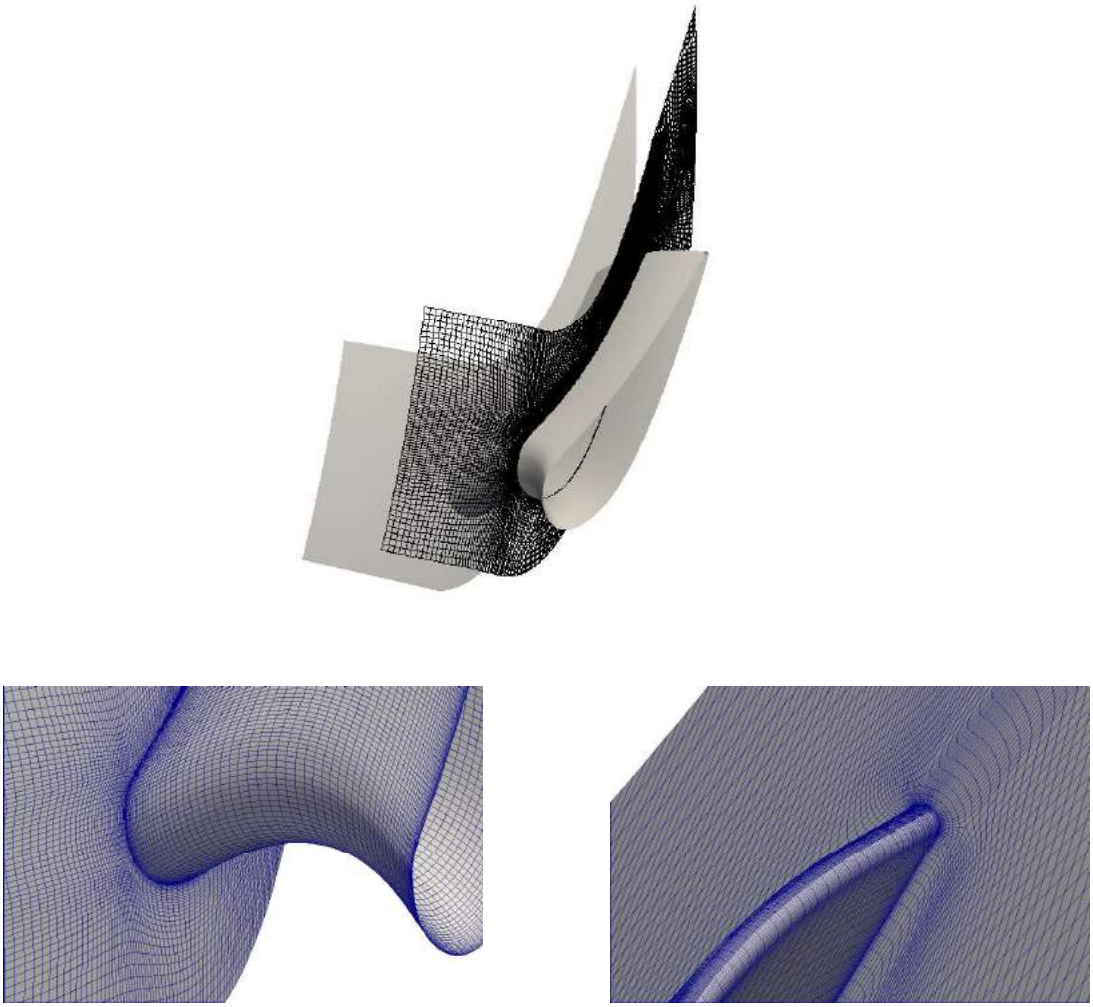


Figure 4.13: HPT vane. Top: Vane geometry and blade-to-blade grid at midspan. Bottom left: Close-up view of the hub and the leading edge surface grids. Bottom right: Close-up view of the hub and the trailing edge surface grids.

At the inlet, the total pressure ($P_t = 1.9 \cdot 10^5 Pa$) and total temperature ($T = 565K$) are imposed along with a span-wise profile for the whirl and pitch flow angles[5], fig. 4.14, and a uniform eddy viscosity ($1.76 \cdot 10^{-4} \frac{m^2}{s}$). At the outlet, a prescribed static pressure span-wise profile is given, as plotted in fig. 4.14.

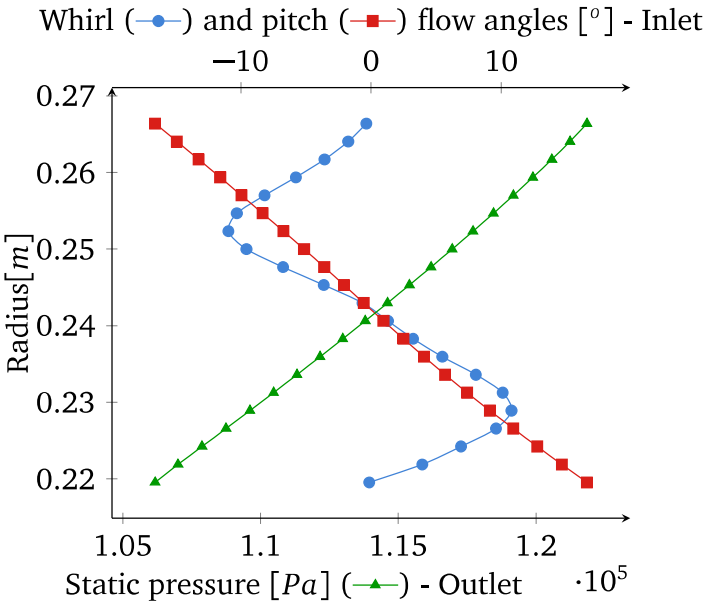


Figure 4.14: HPT vane. Inlet and outlet flow condition profiles.

The unsteady flow solver is initialized by the solution of the steady flow equations computed by imposing the aforementioned boundary conditions with a time-step

[5]The inlet flow angles are given in the cylindrical coordinate system $(x - r - \theta)$ for annular cases and, then, converted to the cartesian $(x - y - z)$ that is used by Hydra (x is the axial direction). The whirl and pitch angles are given by

$$a_w = tan^{-1}(V_\theta / V_x), \qquad a_p = tan^{-1}(V_r / V_x)$$

while the velocity magnitude is

$$|V| = V_x \sqrt{1 + tan^2 a_w + tan^2 a_p}$$

and the direction cosines for the cylindrical coordinates are

$$c_x = \frac{1}{\sqrt{1 + tan^2 a_w + tan^2 a_p}}, \; c_\theta = \frac{tana_w}{\sqrt{1 + tan^2 a_w + tan^2 a_p}}, \; c_r = \frac{tana_p}{\sqrt{1 + tan^2 a_w + tan^2 a_p}}$$

which are converted to the Cartesian system by

$$c_y = \frac{y c_r + z c_\theta}{r}, \; c_z = \frac{y c_\theta + z c_r}{r}$$

equal to $\Delta t = 2 \cdot 10^{-5} s$. A transient flow is caused by gradually decreasing the inlet whirl angle profile by 3^o and this change takes place within a 50 time-step window, using the following sigmoid function

$$a_w = a_{w-init} - 3^o \frac{e^{(n-25)/5}}{e^{(n-25)/5} + 1}, \tag{4.4}$$

where n is the time-step counter, starting after the sigmoid function is activated. As soon as the change in the whirl angle starts, the solver is run for 50 time-steps and, then, stopped. The inlet whirl angle profiles are plotted in fig. 4.15, both before activating the sigmoid function and, also, 50 time-steps after the activation.

The total pressure ratio P_t^{out}/P_t^{in} is considered as the function of interest. The unsteady objective function is defined by integrating the quantity of interest only over the time interval where the sigmoid function is active. The values of the total pressure ratio per time-step and the time interval over which the unsteady objective function is defined are shown in fig. 4.17. In total, the flow solution of 800 time-steps that requires $46.5GB$ is stored on the disk.

The convergence of the flow and adjoint equations at an arbitrary real time-step is plotted in fig. 4.16 and shows that the two computations practically converge with the same rate. To visualize the adjoint variables' fields in a line plot, the L2 norm is used; this is defined as the square root of the sum of all the volume-averaged squared adjoint values for all grid points

$$E(t) = \left(\sum_{i=1}^{N_{PDEs}} \sum_{j=1}^{N_{nodes}} \psi_{ij}^2 \frac{V_j}{V_{tot}} \right)^{1/2} \tag{4.5}$$

where V_j is the volume of the control volume and V_{tot} is the total volume of the computational domain. The L2 norm is plotted using logarithmic scale in the y-axis, fig. 4.17, and its value decays exponentially to zero within the time interval for which the unsteady objective function is not defined, as shown in eq. 3.48.

The vane is parameterized using the setup of section 2.8.1; thus, 15 design variables are defined. Before using the gradients of the objective function w.r.t. the design variables for an optimization process, they are compared against finite differences, eq. 1.1. Each design variable is perturbed for $\epsilon = \pm 0.05$. The comparison of the gradients obtained using the unsteady adjoint and finite differences are shown in fig. 4.18. The outcome of finite differences is in agreement with the unsteady adjoint

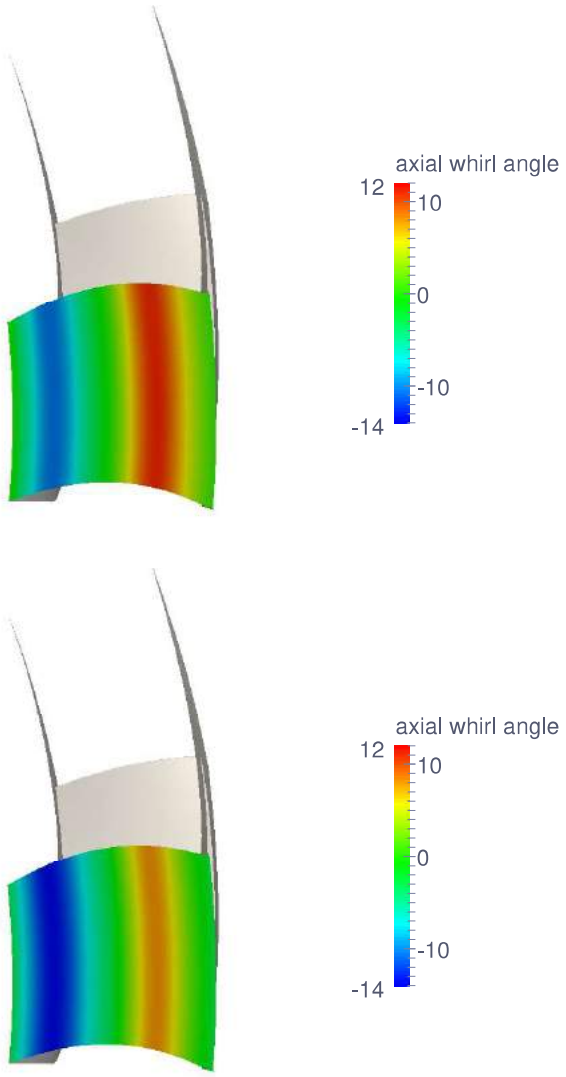


Figure 4.15: HPT vane. Whirl angle at the inlet before activating the sigmoid function (top) and 50 time-steps after the activation (bottom).

gradients. Small differences are due to the fact that a discrete adjoint of the entire chain (parameterization-grid generation-flow solver) is not available. The gradient is obtained by using the adjoint method to compute $\frac{dJ}{d\boldsymbol{x}}$ and finite differences to obtain $\frac{d\boldsymbol{x}}{d\boldsymbol{a}}$ before computing the full gradient $\frac{dJ}{d\boldsymbol{a}}$ by the chain rule. Another reason is the use of finite differences to compute the reference $\frac{dJ}{d\boldsymbol{a}}$ values.

Furthermore, optimization cycles are performed to increase the total pressure ratio using the adjoint-based gradients and steepest ascent; the change in the objective function value is plotted in fig. 4.19. Since no constraints are imposed, the run is

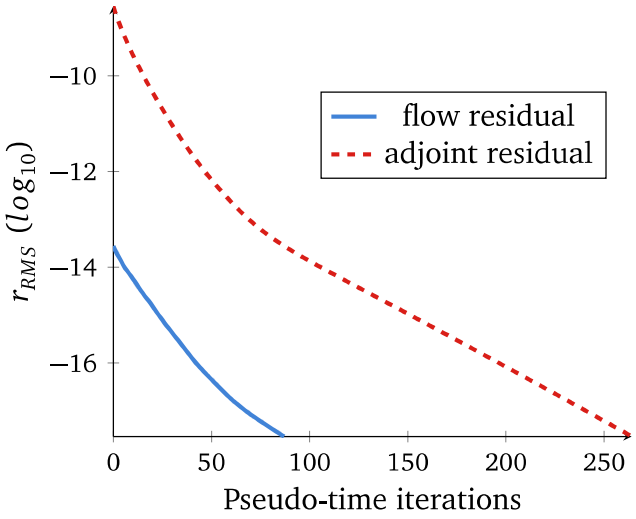


Figure 4.16: HPT vane. Convergence plot of the flow and adjoint equations in pseudo-time at an arbitrarily selected real time-step.

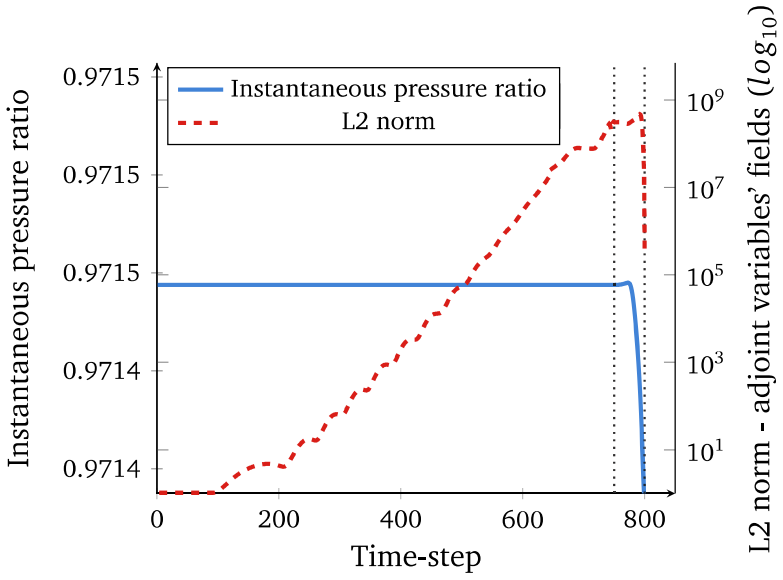


Figure 4.17: HPT vane. Instantaneous pressure ratio values and L2 norm of all the adjoint variables' fields. The dotted lines define the time interval over which the unsteady objective function is defined (from the 750th to the 800th time-step).

terminated after the first 30 optimization cycles. The improved vane is compared to the baseline geometry in fig. 4.20. The new geometry increases the time-integrated total pressure ratio by 0.61%. The main areas of the blade that are modified by the optimization process are the leading and trailing edges. It is only the outlet total pressure that can be changed in order to increase the objective function because the inlet

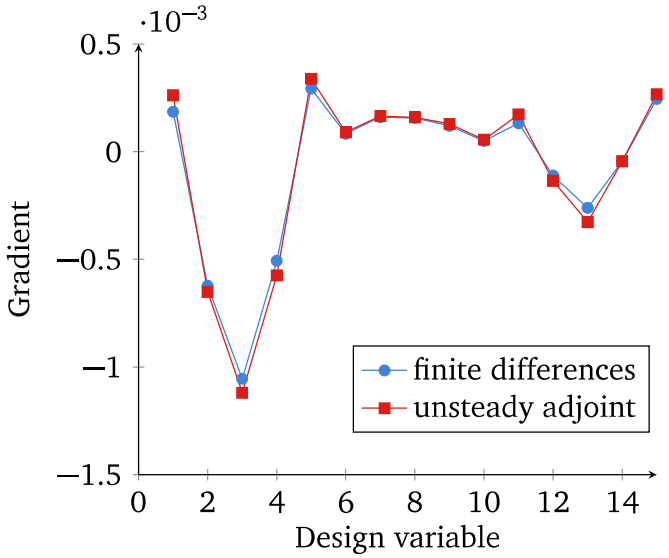


Figure 4.18: HPT vane. Comparison of gradients computed using the unsteady adjoint method and finite differences.

total pressure is imposed by the boundary conditions. In fig. 4.21, the time-averaged outlet total pressure is plotted for the 50 time-step window used in the definition of the objective function, for both the baseline and improved geometries. The optimization process manages to increase the outlet total pressure by minimizing the areas of the outlet plane that correspond to low total pressure; these areas are marked in white.

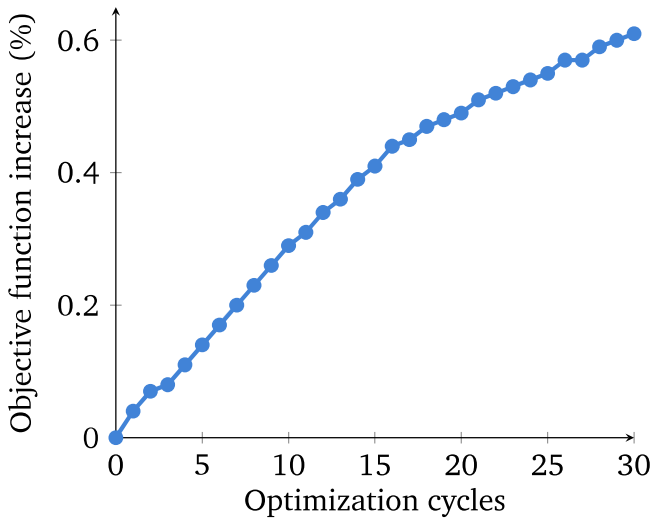


Figure 4.19: HPT vane. Progress of the optimization aiming at maximum total pressure ratio.

Figure 4.20: HPT vane. Baseline (gray) vs. improved (green) vane geometries. Left: Pressure side. Right: Suction side.

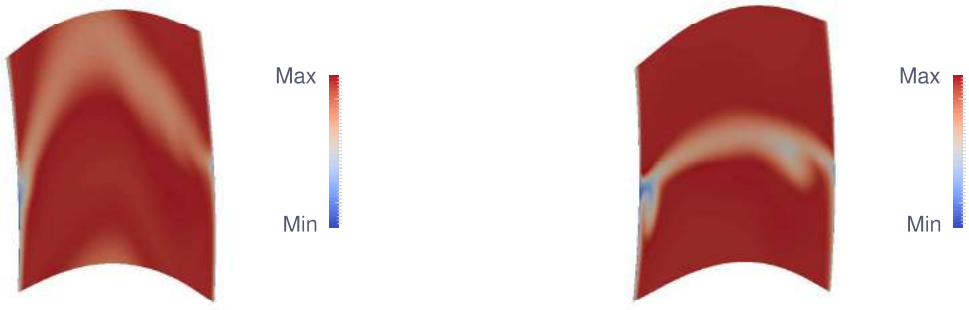


Figure 4.21: HPT vane. Time-averaged total pressure for the baseline (left) and improved (right) geometries at the outlet boundary.

4.3 High-Pressure Turbine Stage

The first stage (S1-R1) of a high-pressure turbine is formed by considering a rotor blade after the vane of the previous section, 4.2. The meridional view of the configuration is shown in fig. 4.22. For the section at the rotor blade's midspan, the chord is $c = 23mm$, the stagger angle is $\gamma = -39^o$, the throat is $o = 8mm$. The size of the tip clearance is $0,9\%$ of the total rotor span. For the steady computations, a grid of

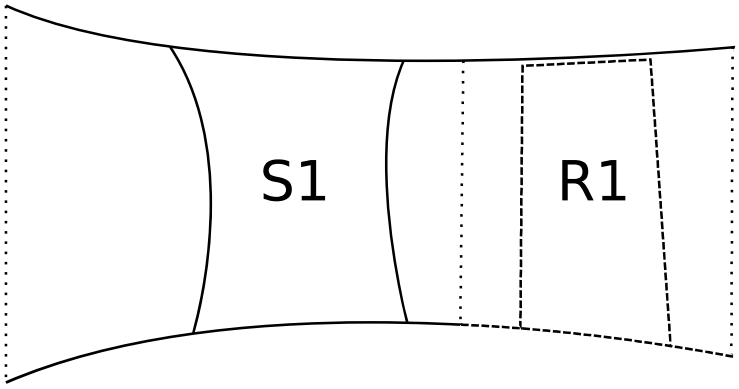


Figure 4.22: HPT stage. Meridional view and boundaries.
Plain line: stationary wall. Dashed line: rotating wall. Dotted line: inflow/outflow/mixing/sliding.

1965430 nodes is generated. Similarly to the vane's grid, the grid of the rotor blades is a combination of an O-type grid around the blade along with an H-type grid for the remaining domain. To form the computational domain for the unsteady computations, it is considered that the number of stator blades per row is 34 and that of the rotor blades 68. Thus, a convenient 1:2 ratio exists, where one stator blade and two rotor blades are needed to form two domains of equal circumferential angular width. The final grid size is 2979630 nodes. The geometry used for the unsteady computations is shown in fig. 4.23 along with details of the rotor blade's surface grid. Wall functions are used and the maximum y^+ at the first nodes off the wall is 70; higher y^+ values appear at the rotors' tip clearance area.

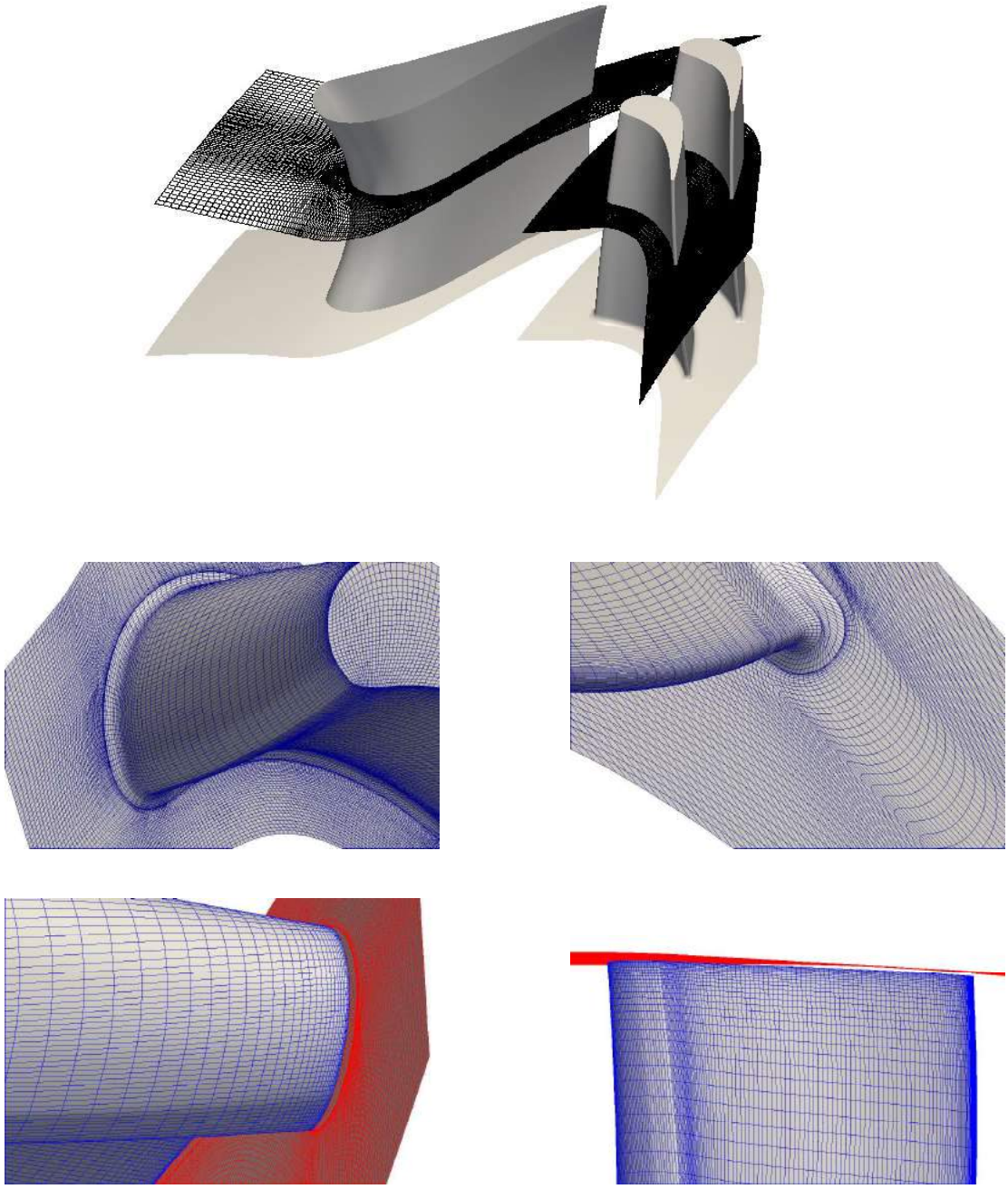


Figure 4.23: HPT stage. Top: Geometry and grid at midspan. Middle left: Close-up view of the rotor blade's hub, fillet and leading edge. Middle right: Close-up view of the rotor blade's hub, fillet and trailing edge. Bottom left: Close-up view of the rotor blade's leading edge and tip. Bottom right: Close-up view of the rotor blade's suction side, blade tip and tip clearance.

Unlike the transient flow in the HPT vane case, the flow in this case is periodic and the time interval over which the objective function is defined coincides with the period of the flow. Results for just a single period of the flow solution are stored on the disk.

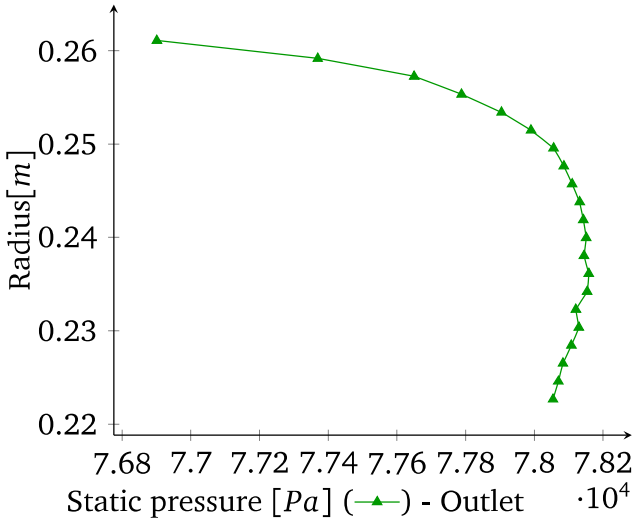


Figure 4.24: HPT stage. Radial districation of the static pressure profile at the rotor blade's outlet.

The same boundary conditions (apart from the mixing and sliding interfaces) are used both for steady and unsteady computations and applied as described in section 2.4. The domain outlet is close to the rotor blade trailing edge since the case is taken from a multi-stage turbine setup, where the rotor blade is followed by another stator blade. Using non-uniform profiles for some quantities as boundary conditions was found to reduce unwanted reflections from the boundaries. The inlet boundary conditions are identical (type and values) to those of the HPT vane case, fig. 4.14. A prescribed static pressure span-wise profile is given at the rotor's outlet, fig. 4.24.

The rotating speed is $1023.53 rad/s$ and, based on the number of blades per row for the unsteady computation, the simulation period is equal to $1.92 \cdot 10^{-4} s$. This period is discretized using 40 equidistant time-steps. The number of time-steps is smaller compared to the next case and is used to keep the computational cost of the initial application of the unsteady adjoint solver low. The unsteady primal solver runs until a periodic solution is obtained for the physical time of 12 periods. The convergence of the flow equations for an arbitrarily selected real time-step is shown in fig. 4.27.

Only the last period is stored on the disk, i.e. 40 files x 137 MB per file = 5.48

GB. Obviously, for the steady flow solver[6], only a single file needs to be stored, with a 90 MB footprint. The size is smaller than that of the file of a single instant of the unsteady simulation since the grid for the steady run is also smaller. Figure 4.25 shows a comparison between the steady and different snapshots of the unsteady flow fields. The most apparent difference is in the wakes from the stator blade's trailing edge that are passing through the interface between the two rows and shows how the use of the sliding interface technique affects the resulting flow-fields.

Two quantities of interest are considered:

- the axial force on the rotor blade and
- the capacity of the inlet boundary of the stator blade domain defined as $\dot{m}\sqrt{T_t}/P_t$, where $\dot{m}$ is mass flow.

The corresponding unsteady objective functions are formed by integrating the instantaneous values over one period. The axial force is selected because its value is sensitive to the accurate prediction of the unsteady wakes coming from the vanes to the rotor blades. The wakes are passing through the sliding interface, where these are averaged out when using the mixing interface technique. A comparison between the steady and unsteady values of the axial force objective function is shown in fig. 4.26. Unlike the next compressor case, the steady solver gives a result very close to the time-averaged axial force.

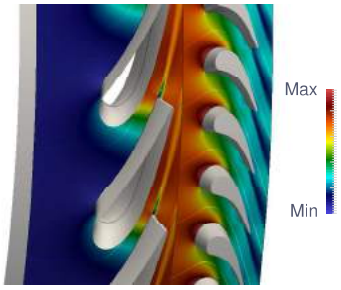
Moving to the unsteady adjoint computations, the convergence history of the adjoint equations for an arbitrarily selected real time-step is shown in fig. 4.27. The rate of convergence is practically equal to that of the flow solver. The unsteady adjoint computation runs until the adjoint solution becomes absolutely periodic. An indicator of this is the L2 norm of all the adjoint variables' fields per time-step, plotted in fig. 4.28.

The cost of a single adjoint pseudo-time iteration is roughly three times more than that of the flow solver. Section 3.10 gives more information on the cost overhead of the adjoint solver.

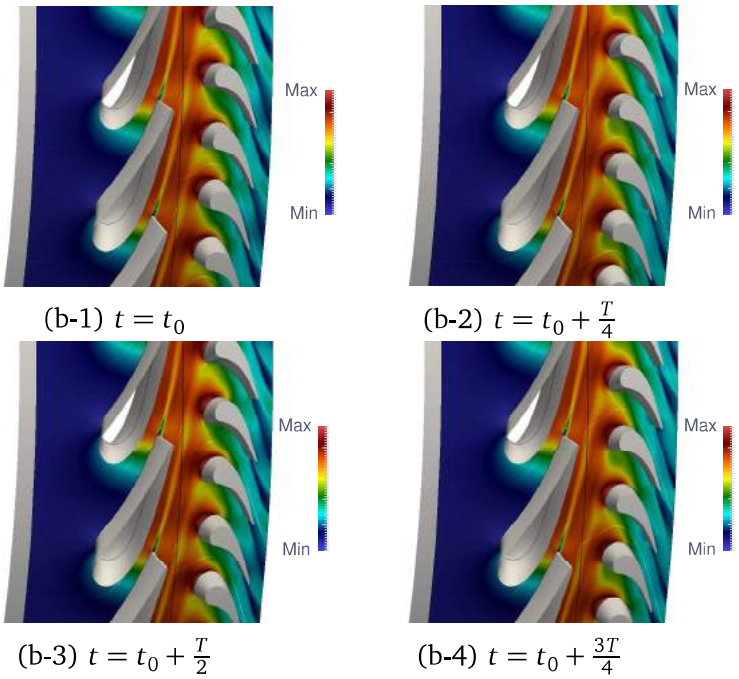
In fig. 4.29, a radial cut at the blade's midspan of the converged steady and unsteady adjoint variables' fields for the inlet capacity objective function is shown.

Similarly to what was observed in the flow solution, the backward propagating wakes from the rotor blades are passing through only if the sliding interface is used.

[6]The steady-state solution is obtained using the mixing interface technique (and not the frozen-rotor assumption).



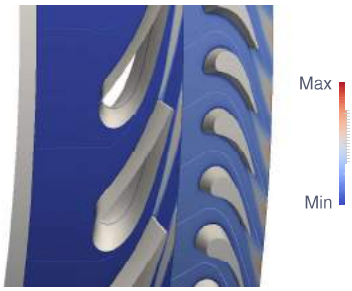
(a) Steady flow - Mach number.



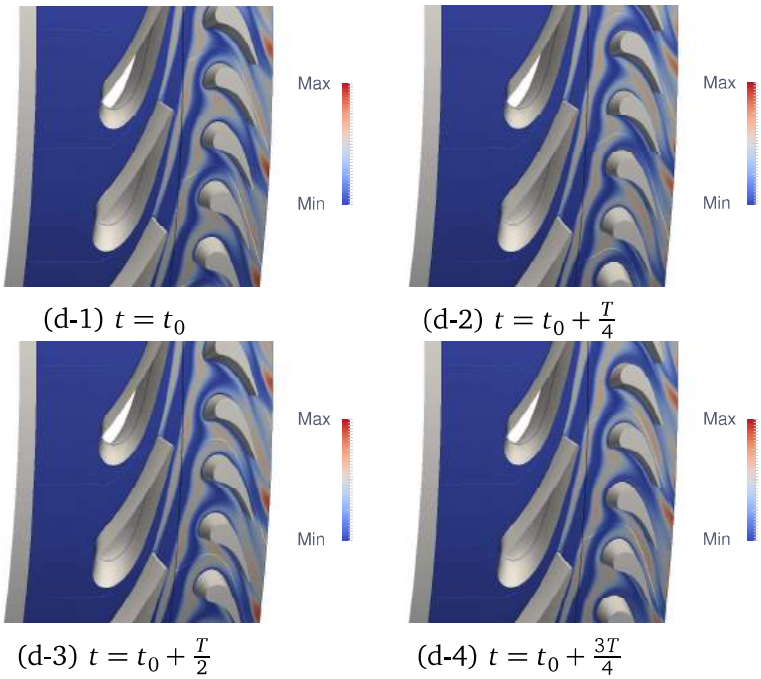
(b-1) $t = t_0$ (b-2) $t = t_0 + \frac{T}{4}$

(b-3) $t = t_0 + \frac{T}{2}$ (b-4) $t = t_0 + \frac{3T}{4}$

(b) Unsteady flow snapshot - Mach number.



(c) Steady flow - Spalart-Allmaras variable.



(d-1) $t = t_0$ (d-2) $t = t_0 + \frac{T}{4}$

(d-3) $t = t_0 + \frac{T}{2}$ (d-4) $t = t_0 + \frac{3T}{4}$

(d) Unsteady flow snapshot - Spalart-Allmaras variable.

Figure 4.25: HPT stage. Steady and unsteady flow fields at midspan.

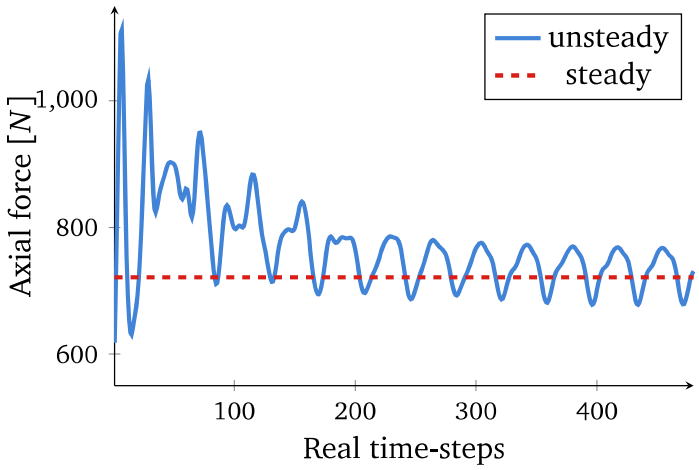


Figure 4.26: HPT stage. Axial force on the rotor blade for steady and unsteady flow computations.

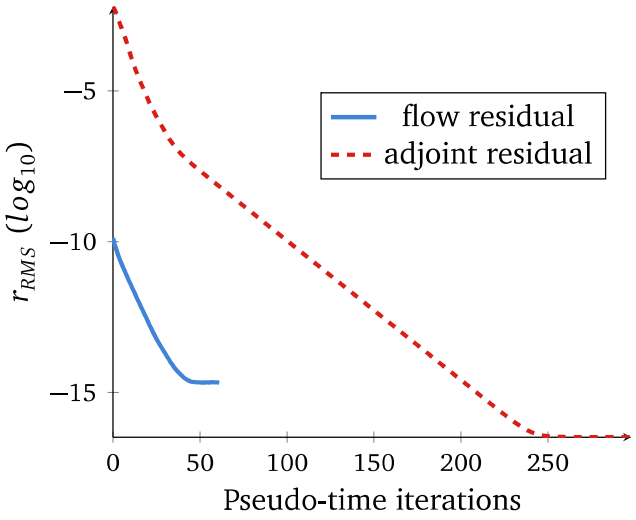


Figure 4.27: HPT stage. Convergence plot of the flow and adjoint equations in pseudo-time at an arbitrarily selected, real time-step.

The unsteady adjoint variables' fields have a more "wavy" form comparing to the steady adjoint variables' fields and the averaging-out logic of the adjoint mixing plane.

A very useful tool for the designers is sensitivity maps which can be obtained using the adjoint method. These correspond to the iso-areas of the objective function derivative w.r.t. to the normal displacement of each point (node), plotted on the surface of the shape to be optimized. Practically, they reveal which areas of the geometry could potentially affect the value of the objective function and what are the changes these should undergo. According to the notation of this thesis, in order to increase the

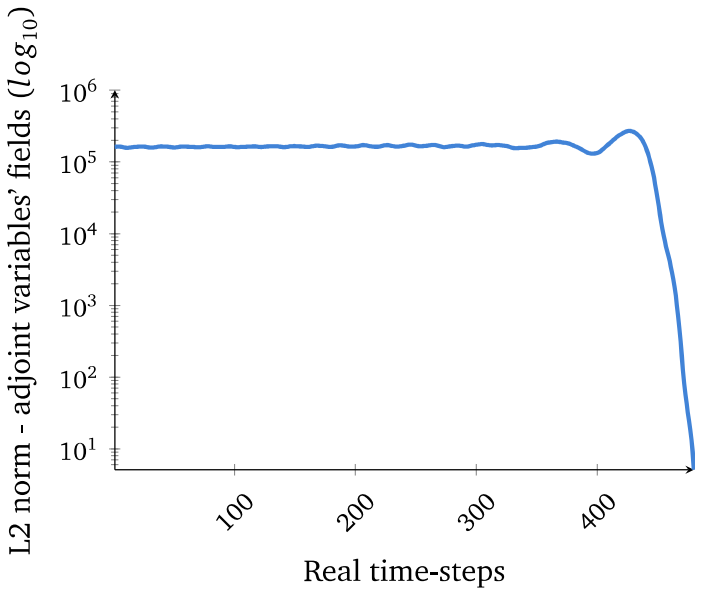
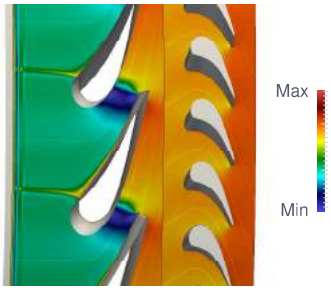


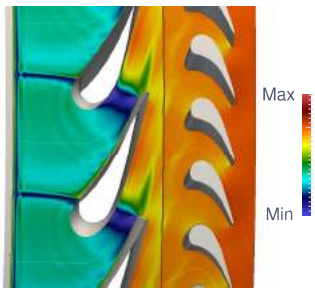
Figure 4.28: HPT stage. Objective function: Inlet capacity. L2 norm of all the unsteady adjoint variables' fields per real time-step.

value of the objective function one needs to push inwards the blue areas and/or pull outwards the red areas. During an optimization loop, the modified geometry is given based on the gradient of the value of interest w.r.t. the design variables.

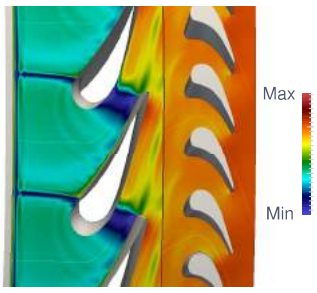
In fig. 4.30, the sensitivity maps of the stator blade are given for the axial force on the rotor as the objective function. The sensitivity maps are generated using the steady and unsteady adjoint solutions at a post-processing step. In the case of the unsteady adjoint solver, both the time-averaged and four instantaneous sensitivity maps are given. The information that can be extracted by these sensitivity maps is that, in order to improve the axial force on the rotor blade, two areas on the stator blade are important. The first one is the trailing edge which affects the wakes moving downstream towards the rotor blade. The second area is on the suction side close to the leading edge. Modifying the blade in this area has an impact on the available space between two neighboring vanes. The comparison is more straightforward between steady and time-averaged sensitivity maps. The gradient of the unsteady objective function is computed by integrating, over a single period, the product of the instantaneous sensitivities ($\frac{dJ}{d\boldsymbol{x}} = \frac{\partial J}{\partial \boldsymbol{x}} + \boldsymbol{\psi}_T \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}}$) with $\frac{d\boldsymbol{x}}{d\boldsymbol{a}}$.



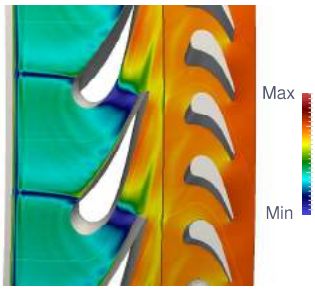
(a) Steady adjoint - adjoint density.



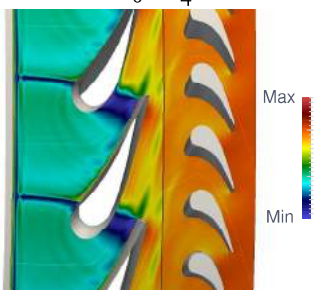
(b-1) $t = t_0$



(b-2) $t = t_0 + \frac{T}{4}$

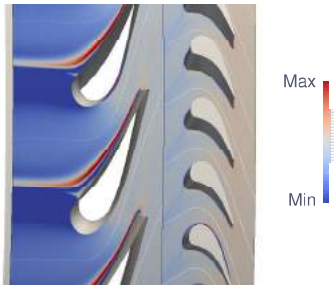


(b-3) $t = t_0 + \frac{T}{2}$

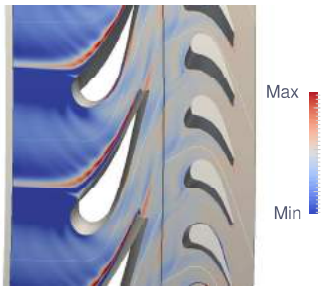


(b-4) $t = t_0 + \frac{3T}{4}$

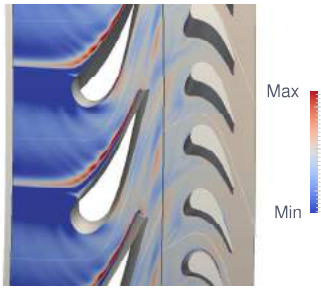
(b) Unsteady adjoint - adjoint density.



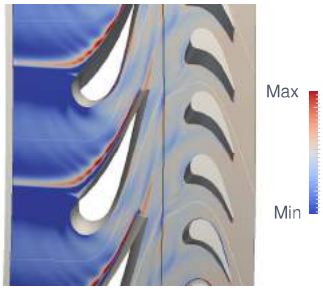
(c) Steady adjoint - adjoint Spalart-Allmaras variable.



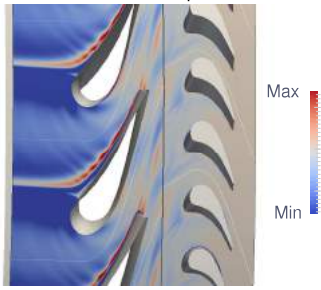
(d-1) $t = t_0$



(d-2) $t = t_0 + \frac{T}{4}$



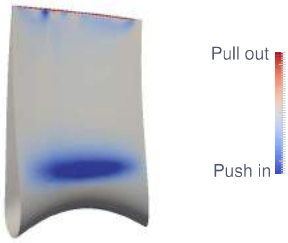
(d-3) $t = t_0 + \frac{T}{2}$



(d-4) $t = t_0 + \frac{3T}{4}$

(d) Unsteady adjoint - adjoint Spalart-Allmaras variable.

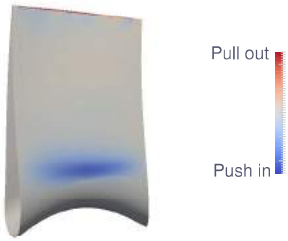
Figure 4.29: HPT stage. Objective function: inlet capacity. Steady and unsteady adjoint variables' fields at midspan.



(a) Steady - suction side.

(c-1) $t = t_0$

(c-2) $t = t_0 + \frac{T}{4}$



(b) Time-averaged - suction side.

(c-3) $t = t_0 + \frac{T}{2}$

(c-4) $t = t_0 + \frac{3T}{4}$

(c) Instantaneous - suction side.

(d) Steady - pressure side.

(f-1) $t = t_0$

(f-2) $t = t_0 + \frac{T}{4}$

(e) Time-averaged - pressure side

(f-3) $t = t_0 + \frac{T}{2}$

(f-4) $t = t_0 + \frac{3T}{4}$

(f) Instantaneous - pressure side.

Figure 4.30: HPT stage. Objective function: axial force on rotor blade. Steady, time-averaged and instantaneous snapshots of the sensitivity maps on the stator blade.

4.3.1 Comparing Adjoint Gradients with Finite Differences

Using the unsteady adjoint variables' fields ψ, the gradient of the objective function can be computed w.r.t. the design variables' vector according to section 3.6 and fig. 3.2.

Before using the gradients within an optimization loop, these are compared with the gradients computed by finite differences. For this comparison, the vane is parameterized in a less rich manner than the one described in section 2.8.1, in order to reduce the cost required by finite differences, which scales with to the number of design variables. More specifically, instead of considering 5 sections across the vane's span, the perturbed geometries for the finite differences are created by:

- skewing (rotating) the entire stator blade for $\epsilon_1 = \pm 0.05^o$ (name: skew) and
- shifting the entire blade for $\epsilon_2 = \pm 0.5mm$ in the axial direction (name: x-shift).

Thus, two parameters are solely used.

The gradients of the two objective functions are compared in table 4.4. The results are in close agreement.

function	design variable	finite differences	unsteady adjoint	relative difference (%)
axial force $[Ns]$	skew	-1.418	-1.439	1.45
	x-shift	6.988e-03	6.923e-03	0.93
capacity $\left[kgK^{1/2}Pa^{-1}\right]$	skew	-2.273e-07	-2.283e-07	0.43
	x-shift	2.557e-09	2.539e-09	0.72

Table 4.4: HPT stage. Comparison between objective function gradients obtained by unsteady adjoint and finite differences.

4.3.2 Reduction of the Axial Force

The unconstrained reduction of the axial force exerted on the rotor blade is considered as an optimization example. The parameterization of the stator blade is performed according to section 2.8.1, using 5 spanwise sections which can be displaced in 3 ways, resulting to 15 design variables. The new design is found using the steepest descent method with a step-size equal to $s = 4$. Figure 4.31 shows the improved geometry of the stator blade compared to the initial one and fig. 4.32 the 2D profiles of the blade at cuts near the hub and tip. The new geometry reduces the objective function value

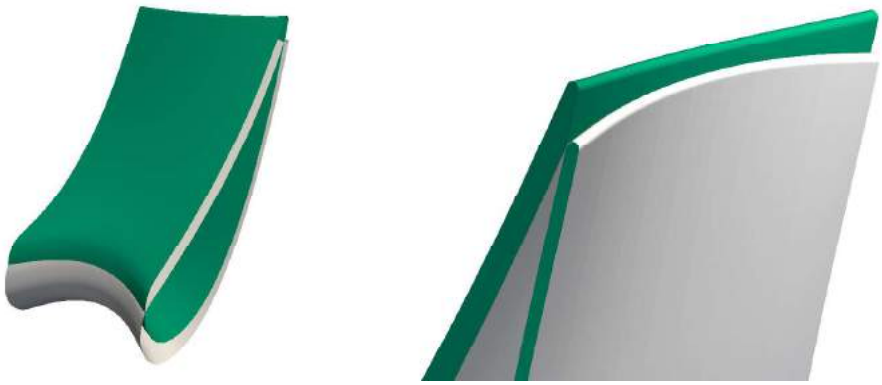


Figure 4.31: HPT stage. Objective function: axial force on rotor blade. Baseline (gray) compared to improved (green) stator blade geometry. Left: Pressure side. Right: Trailing edge close-up view.

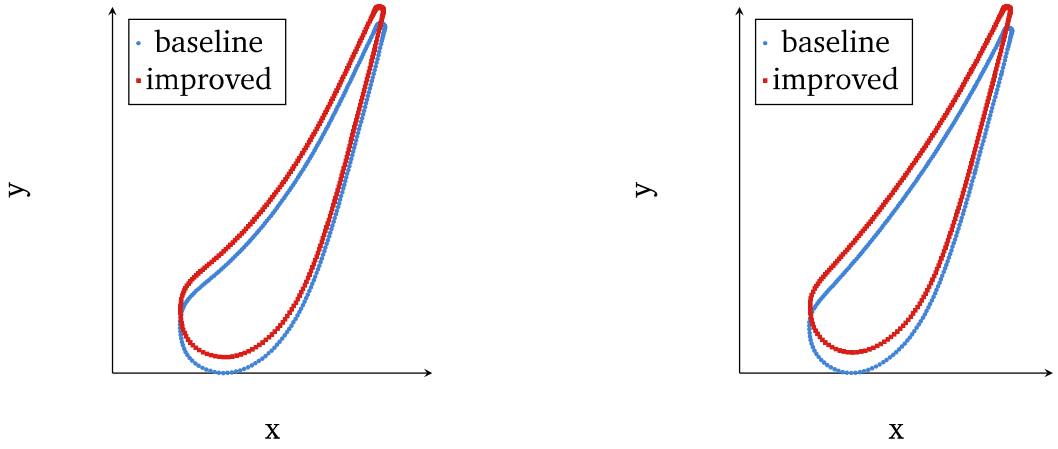


Figure 4.32: HPT stage. Baseline vs. improved vane's profiles near the hub, at 5% blade's span (left) and close to the tip, at 95% blade's span (right).

from 5.488Ns to 5.005Ns, i.e. by 8.81%, by modifying the shape of blade areas that are important for the objective function as seen in the sensitivity maps of fig. 4.30. In fig. 4.33, the axial force on the rotor blade is plotted as a function of time, for the initial and the improved stator blade setup.

The total force on a blade is the sum of the pressure and viscous forces. However, the viscous forces contribute to a very small percentage of the overall force. This is apparent in fig. 4.34 where the total force is plotted against the force only due to pressure forces, by ignoring the viscous forces. Thus, improvements to the total axial force objective function can be spotted by looking at the pressure distribution at the midspan of the rotor blade.

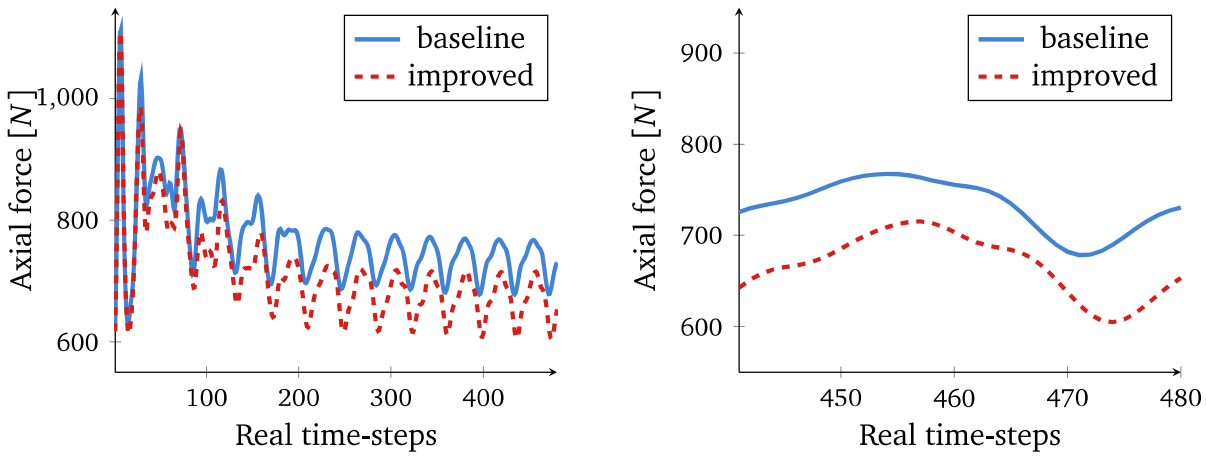


Figure 4.33: HPT stage. Instantaneous axial force on rotor blade for the baseline and improved stator blade geometry. Left: Total running time. Right: Last period.

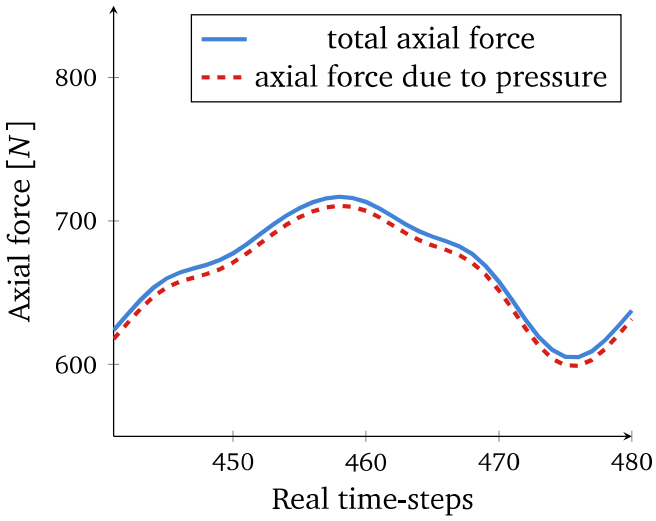


Figure 4.34: HPT stage. Total axial force and axial force only due to pressure forces on the rotor blade with the improved vane geometry.

Figure 4.35 shows the time-averaged pressure contours on the rotor blade before and after the optimization. A sound difference is the decrease in pressure on the pressure side close to the tip and leading edge areas. The pressure distribution at midspan is plotted for the baseline and improved geometries in fig. 4.36.

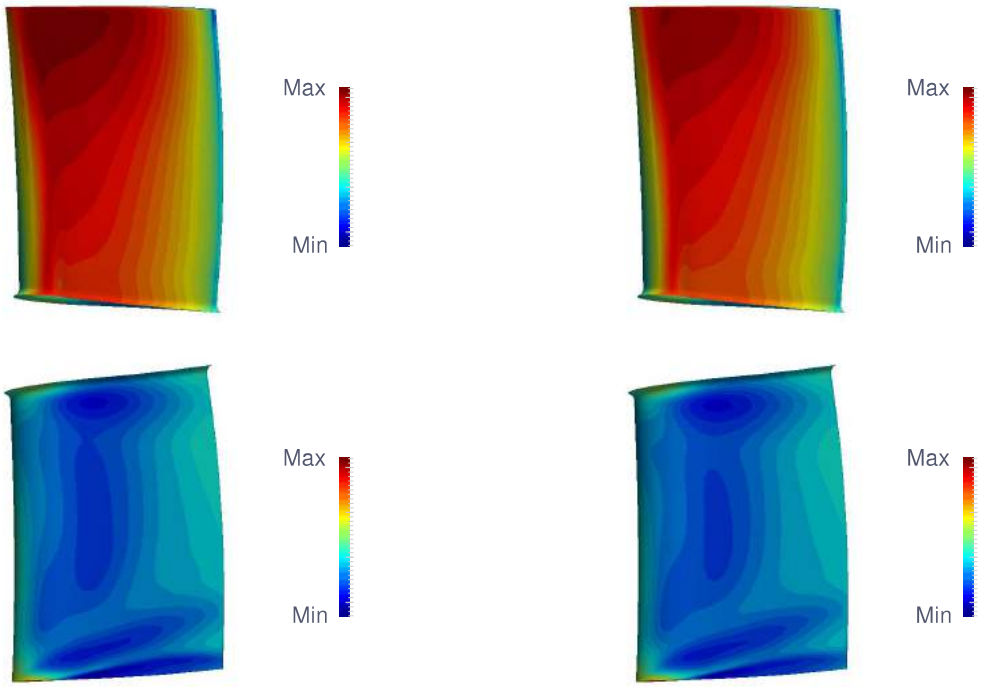


Figure 4.35: HPT stage. Time-averaged pressure contours on rotor blade. Top: Pressure side. Bottom: Suction side. Left: Baseline geometry. Right: Improved geometry.

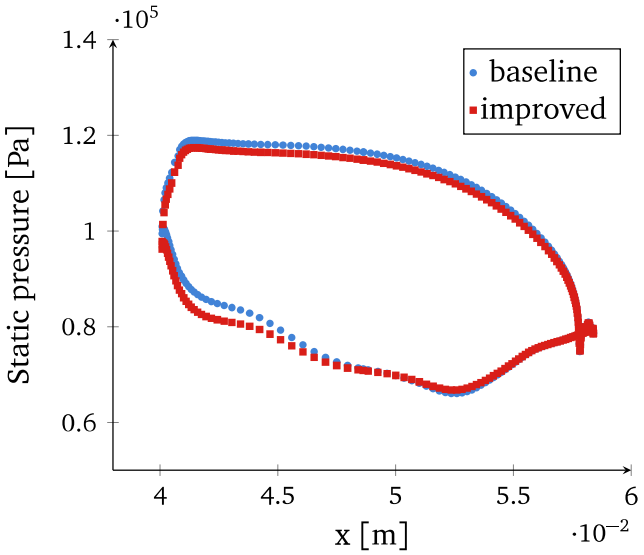


Figure 4.36: HPT stage. Pressure distribution at midspan of the rotor for the baseline and improved vane geometries.

4.4 Three-Row Compressor

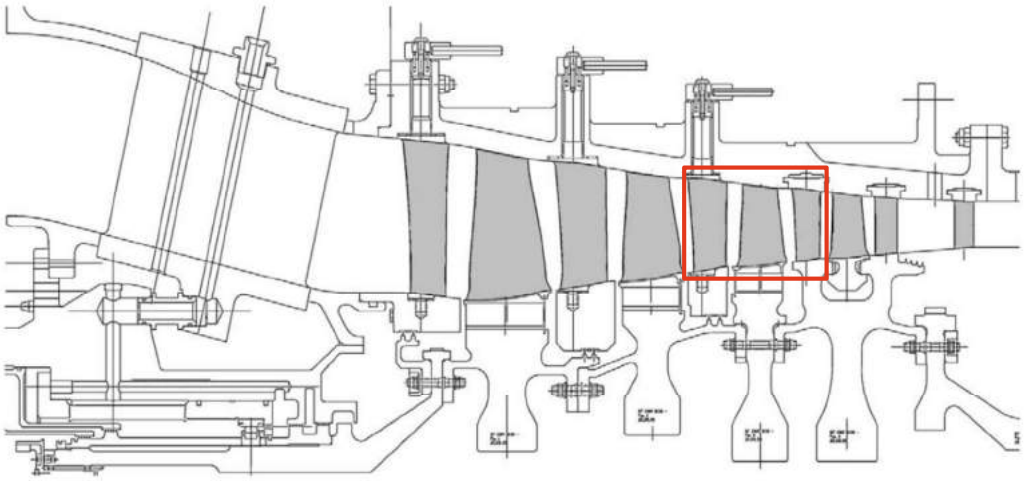


Figure 4.37: Three-row compressor. Cross section of the compressor and selected three row configuration; from [192].

The next case is a three row compressor configuration; a stator blade row followed by a stage. The setup is based on the Rig250 compressor, shown in fig. 4.37, a four stage axial compressor with inlet guide vane (total pressure ratio 5:1 at the design point) of the German Aerospace Center (DLR) Institute of Propulsion Technol-

ogy [192–194]. The stator of the second stage and the entire third stage are considered, by changing slightly the blade-count to reduce the size of the grid for the unsteady computations. Below, the abbreviation S2-R3-S3, with evident interpretation, is used. The blade-count used in this setup is 48:36:72 respectively and, thus, the computational domain for the unsteady runs consists of four S2 stator blades, three R3 rotor blades and six S3 stator blades. The computational grid is generated using Padram similarly to for the turbine case. The grid for the steady runs has 2849966 nodes while that for the unsteady runs has 12369648 nodes in total. Wall functions are used with the maximum y^+ of the first nodes off the wall being 15. The boundary surfaces are indicated in the meridional view in fig. 4.39 and the profiles of the boundary conditions for the inlet and outlet are given in fig. 4.40.

The rotating speed is $1357.17 rad/s$ which, along with the number of blades used per row for the computational domain, gives the simulation period, which is $3.86 \cdot 10^{-4} s$. 100 equidistant time-steps per period are used; thus, in order to store the flow fields during a full period to the disk, 100 x 570 MB = 57 GB are required. The solver runs for 8 periods until periodicity is established. The convergence history of the flow equations for an arbitrarily selected real time-step is shown in fig. 4.41. A comparison between the steady and unsteady flow fields is shown in fig. 4.42.

For this case, three functions of interest are considered:

- the axial force on the rotor blade R3,
- the exit capacity ($= \dot{m}\sqrt{T_t}/P_t$) at the outlet of S3 and
- the total pressure coefficient between the inlet and outlet of the stage defined as $\frac{P_{t,in}-P_{t,out}}{P_{t,in}-P_{in}}$ (where the inlet is the interface between S2 and R3 and the exit is the outlet plane of S3).

The objective function for the unsteady run is computed by integrating the axial force on R3 over one period while the remaining two functions are also integrated over one period and used as constraints. The plot of the total pressure coefficient in fig. 4.43 shows differences between the steady and unsteady flow solvers. Three adjoint computations are needed to compute the gradients for the objective function and the constraints. Figure 4.44 shows the L2 norm of all the instantaneous adjoint variables' fields (computed by the time-dependent run) for the total pressure coefficient constraint. Figure 4.45 shows a comparison between the steady and unsteady adjoint

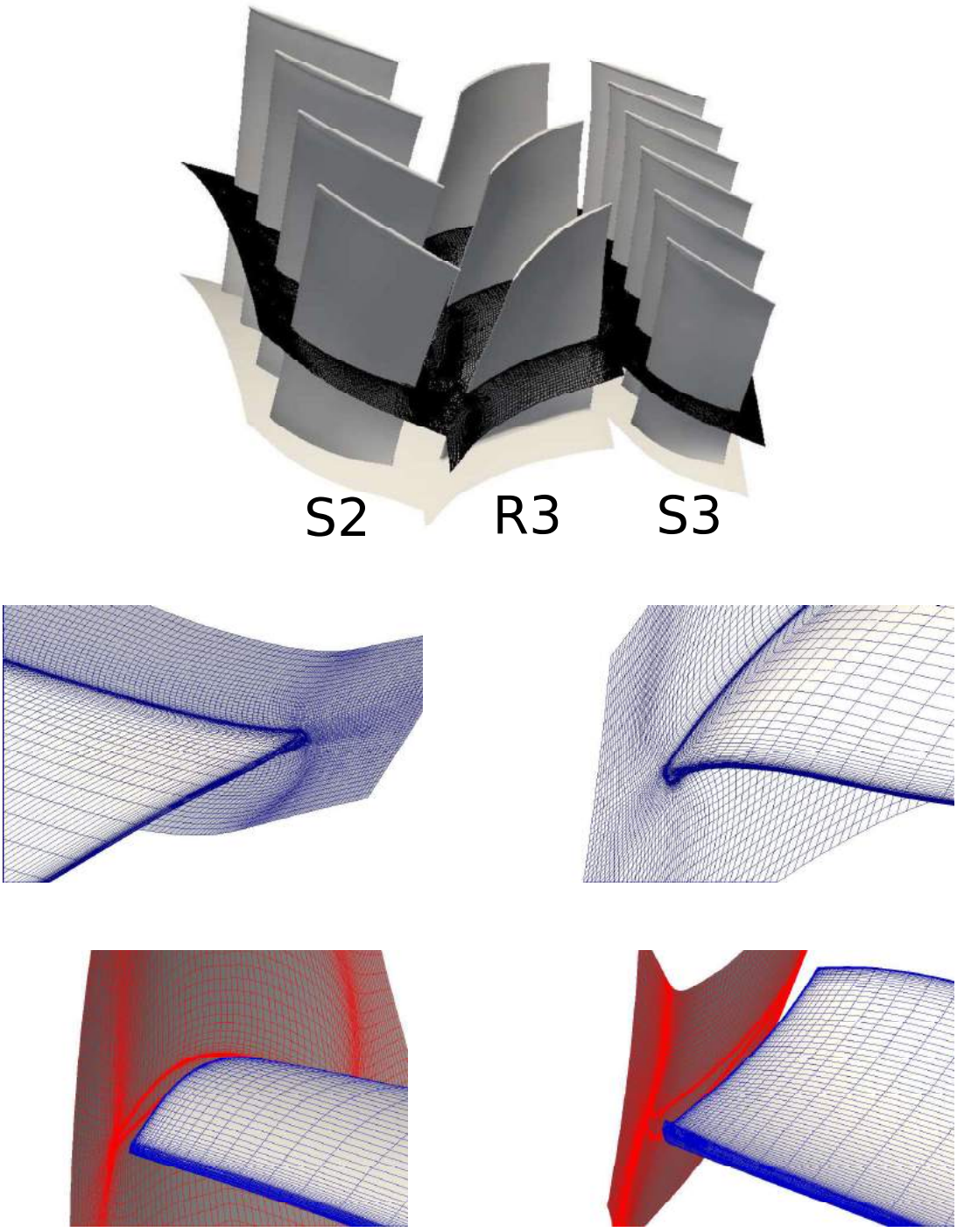


Figure 4.38: Three-row compressor. Top: Geometry and grid at midspan. Middle left: Stator blade S2 tip and leading edge. Middle right: Rotor blade R3 hub and leading edge. Bottom left: Rotor blade R3 trailing edge, tip and tip clearance. Bottom right: Stator blade S3 leading edge, hub and hub clearance.

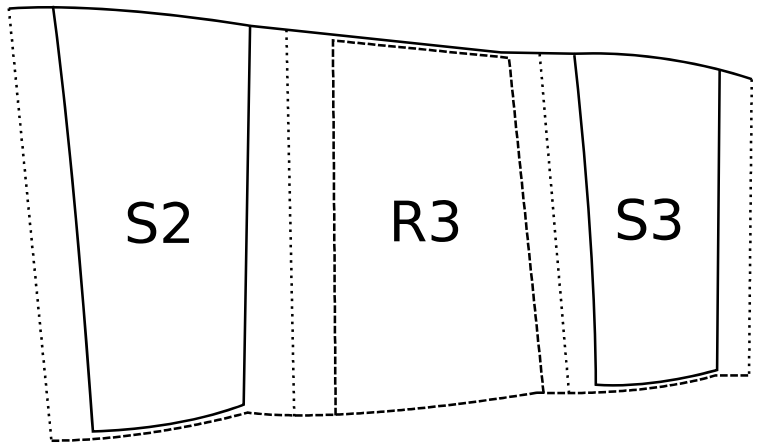


Figure 4.39: Three-row compressor. Meridional view and boundaries.
Plain line: stationary wall. Dashed line: rotating wall. Dotted line: inflow/outflow/mixing/sliding.

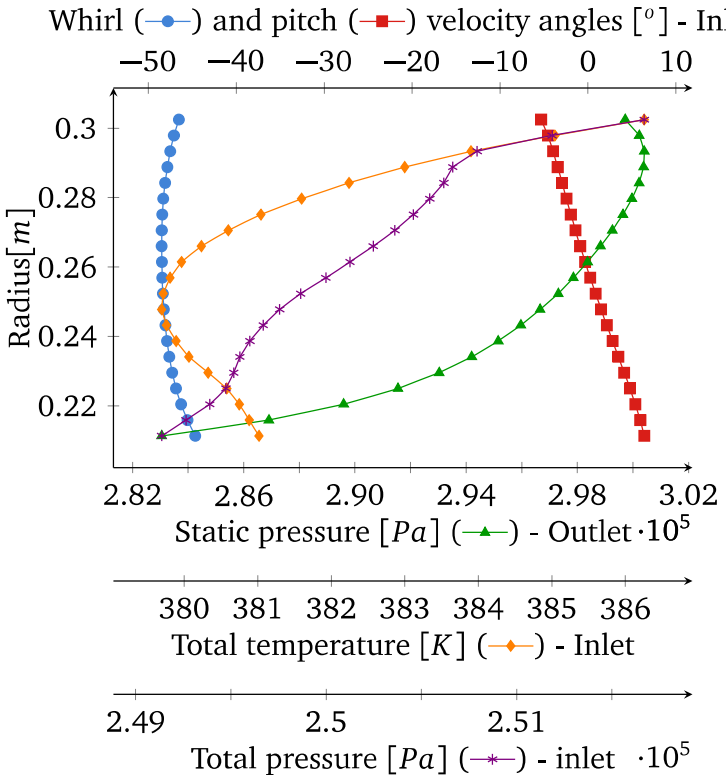


Figure 4.40: Three-row compressor. Inlet and outlet boundary condition profiles.

variables' fields when the axial force on the rotor blade is the objective function. Similarly to the previous case, an important difference that is apparent is the fact that the backward propagating wakes are passing through the sliding interface, unlike the ad-

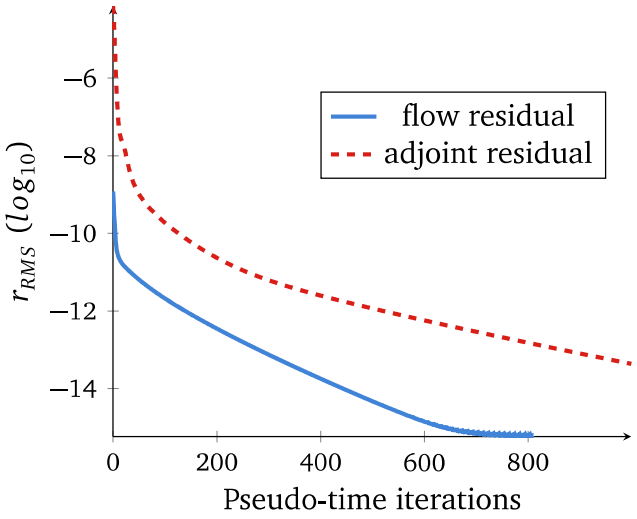
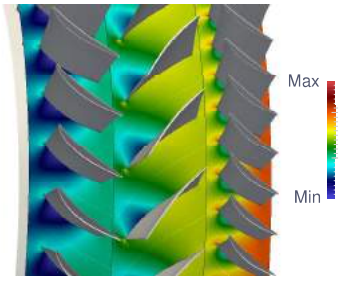
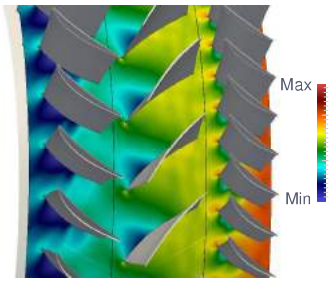


Figure 4.41: Three-row compressor. Convergence plot of the flow and adjoint equations in pseudo-time at an arbitrarily selected, real time-step.

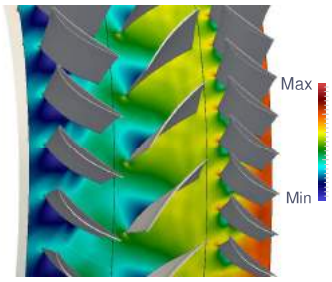
joint mixing interface method. Using the adjoint variables' fields, sensitivity maps can be extracted for the objective function and constraints. Figure 4.46 shows the sensitivity map on the S2 blade for the axial force objective function while fig. 4.47 shows the sensitivity map on the R3 blade for the exit capacity constraint. Figure 4.46 confirms the importance of the trailing edge area of the upstream stator when the force on the downstream rotor is considered. In the case of a strip-like sensitivity map as in fig. 4.47, the selected parameterization behaves as an implicit smoother when translating the surface sensitivities to gradients w.r.t. the design variables. As a consequence, smooth improved blades are obtained at each optimization cycle.



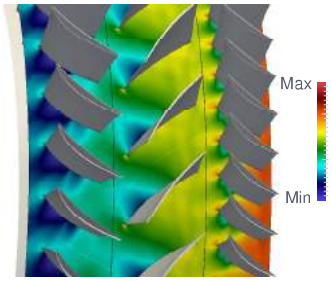
(a) Steady flow - static pressure.



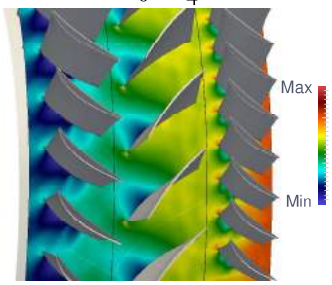
(b-1) $t = t_0$



(b-2) $t = t_0 + \frac{T}{4}$

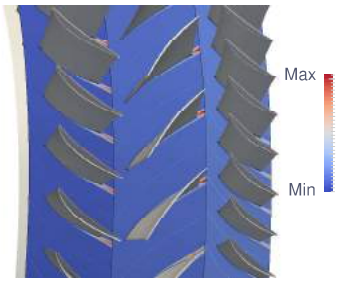


(b-3) $t = t_0 + \frac{T}{2}$

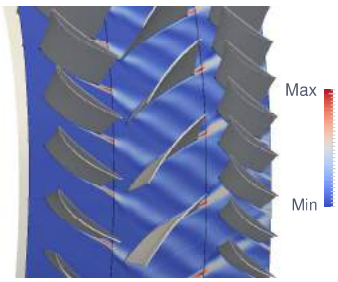


(b-4) $t = t_0 + \frac{3T}{4}$

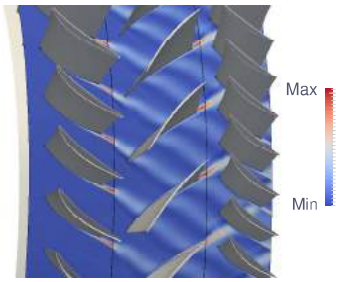
(b) Unsteady flow snapshot - static pressure.



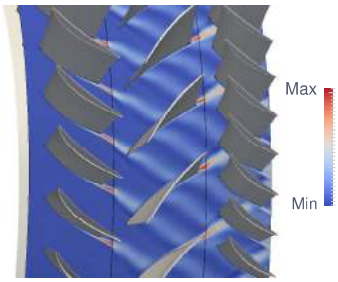
(c) Steady Spalart-Allmaras variable.



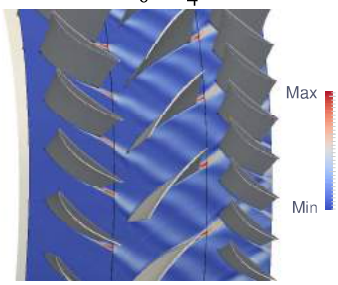
(d-1) $t = t_0$



(d-2) $t = t_0 + \frac{T}{4}$



(d-3) $t = t_0 + \frac{T}{2}$



(d-4) $t = t_0 + \frac{3T}{4}$

(d) Unsteady snapshot Spalart-Allmaras variable.

Figure 4.42: Three-row compressor. Steady and unsteady flow fields at midspan.

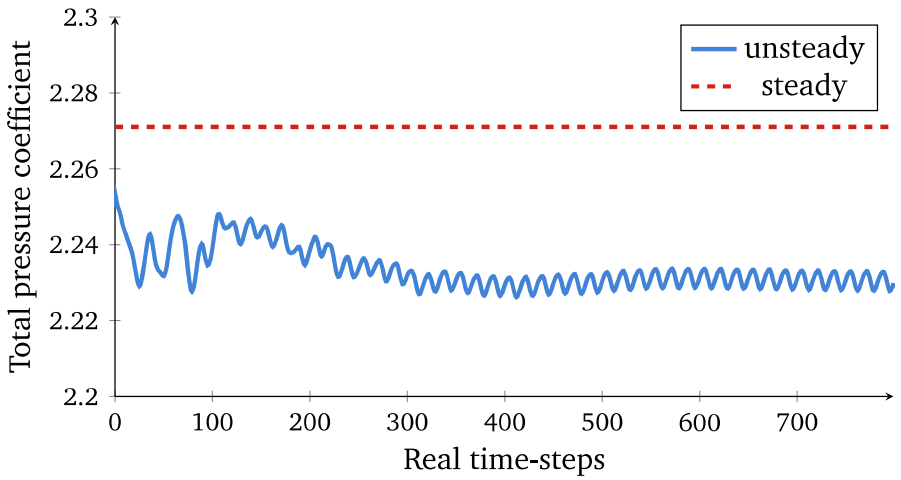


Figure 4.43: Three-row compressor. Total pressure coefficient of the stage formed by R3-S3 using the unsteady and steady solver.

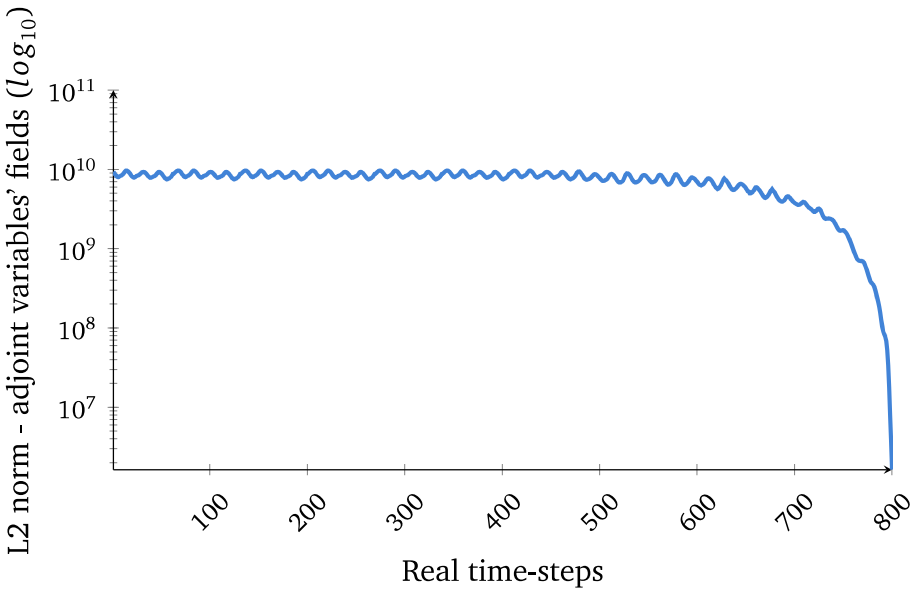
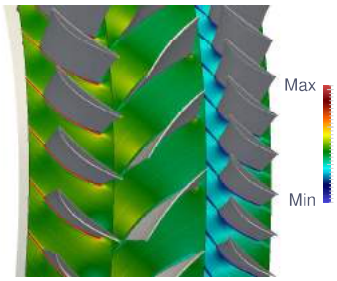
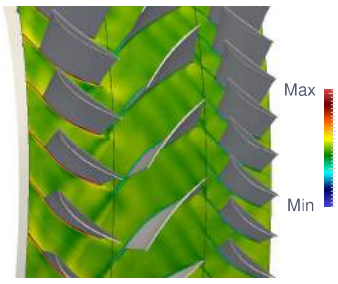


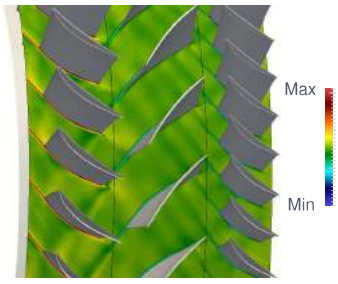
Figure 4.44: Three-row compressor. Objective function: Total pressure coefficient. L2 norm of all the unsteady adjoint variables' fields per real time-step.



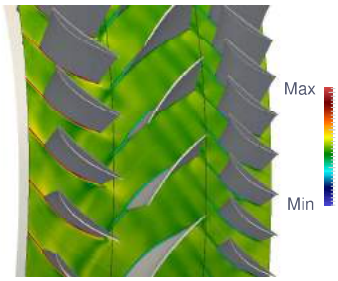
(a) Steady adjoint - adjoint static pressure.



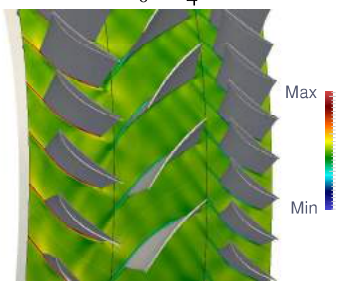
(b-1) $t = t_0$



(b-2) $t = t_0 + \frac{T}{4}$

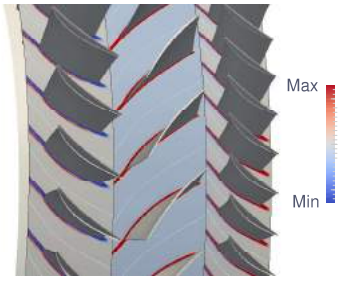


(b-3) $t = t_0 + \frac{T}{2}$

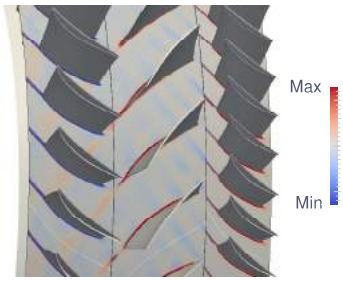


(b-4) $t = t_0 + \frac{3T}{4}$

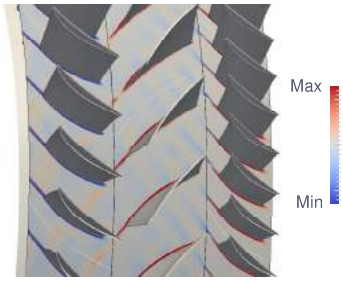
(b) Unsteady adjoint snapshot - adjoint static pressure.



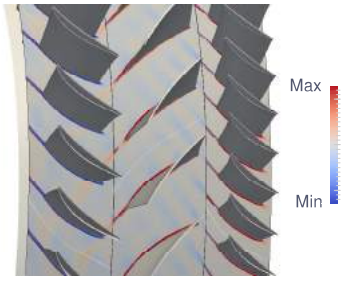
(c) Steady adjoint Spalart-Allmaras variable.



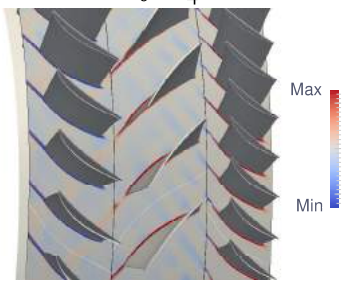
(d-1) $t = t_0$



(d-2) $t = t_0 + \frac{T}{4}$



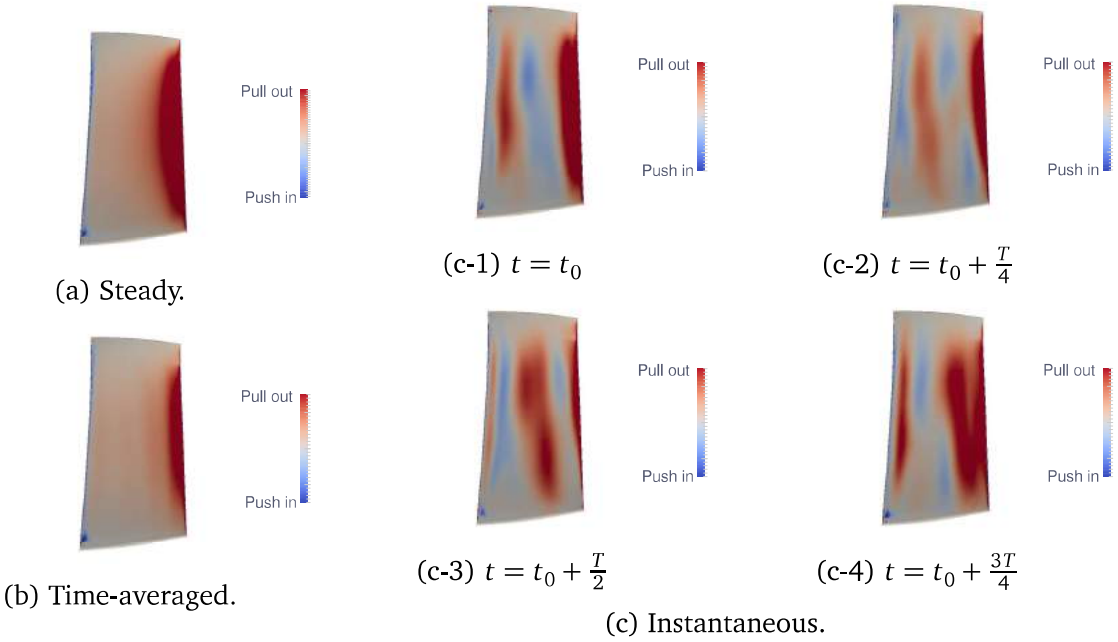
(d-3) $t = t_0 + \frac{T}{2}$



(d-4) $t = t_0 + \frac{3T}{4}$

(d) Unsteady adjoint snapshot Spalart-Allmaras variable.

Figure 4.45: Three-row compressor. Objective function: Axial force on R3. Steady and unsteady adjoint variables' fields at midspan.



(a) Steady.

(c-1) $t = t_0$

(c-2) $t = t_0 + \frac{T}{4}$

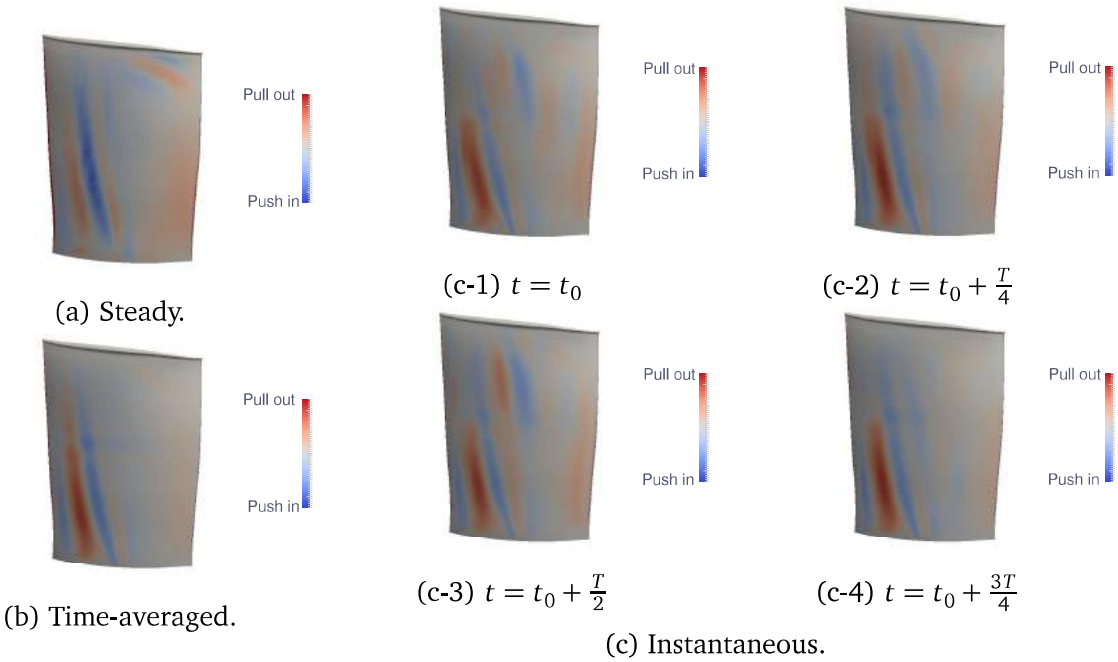
(b) Time-averaged.

(c-3) $t = t_0 + \frac{T}{2}$

(c-4) $t = t_0 + \frac{3T}{4}$

(c) Instantaneous.

Figure 4.46: Three-row compressor. Objective function: Axial force on the R3 blade. Comparison of steady, time-averaged and instantaneous snapshots of sensitivity maps on the pressure side of the S2 blade.



(a) Steady.

(c-1) $t = t_0$

(c-2) $t = t_0 + \frac{T}{4}$

(b) Time-averaged.

(c-3) $t = t_0 + \frac{T}{2}$

(c-4) $t = t_0 + \frac{3T}{4}$

(c) Instantaneous.

Figure 4.47: Three-row compressor. Steady, time-averaged and instantaneous snapshots of sensitivity maps on the suction side of the R3 blade. Constraint: Exit capacity.

4.4.1 Minimization of Axial Force, Constrained by Exit Capacity & Total Pressure Coefficient

Using the parameterization setup of section 2.8.1, the S2 and R3 blades are parameterized. This gives rise to 30 design variables in total.

To avoid violating the constraints, a smaller step-size ($s = 0.1$) than in the previous case (same objective function for both cases) is used at each step for the projected gradient descent method, section 3.9, and after four optimization cycles, new geometries are obtained for the S2 and R3 blades. Changes in the S2 and R3 blades are shown in fig. 4.48, in 3D, and their profiles are compared at 5% and 95% of the blade span in fig. 4.49.

(a) Stator blade (S2) (view from tip/leading edge side).

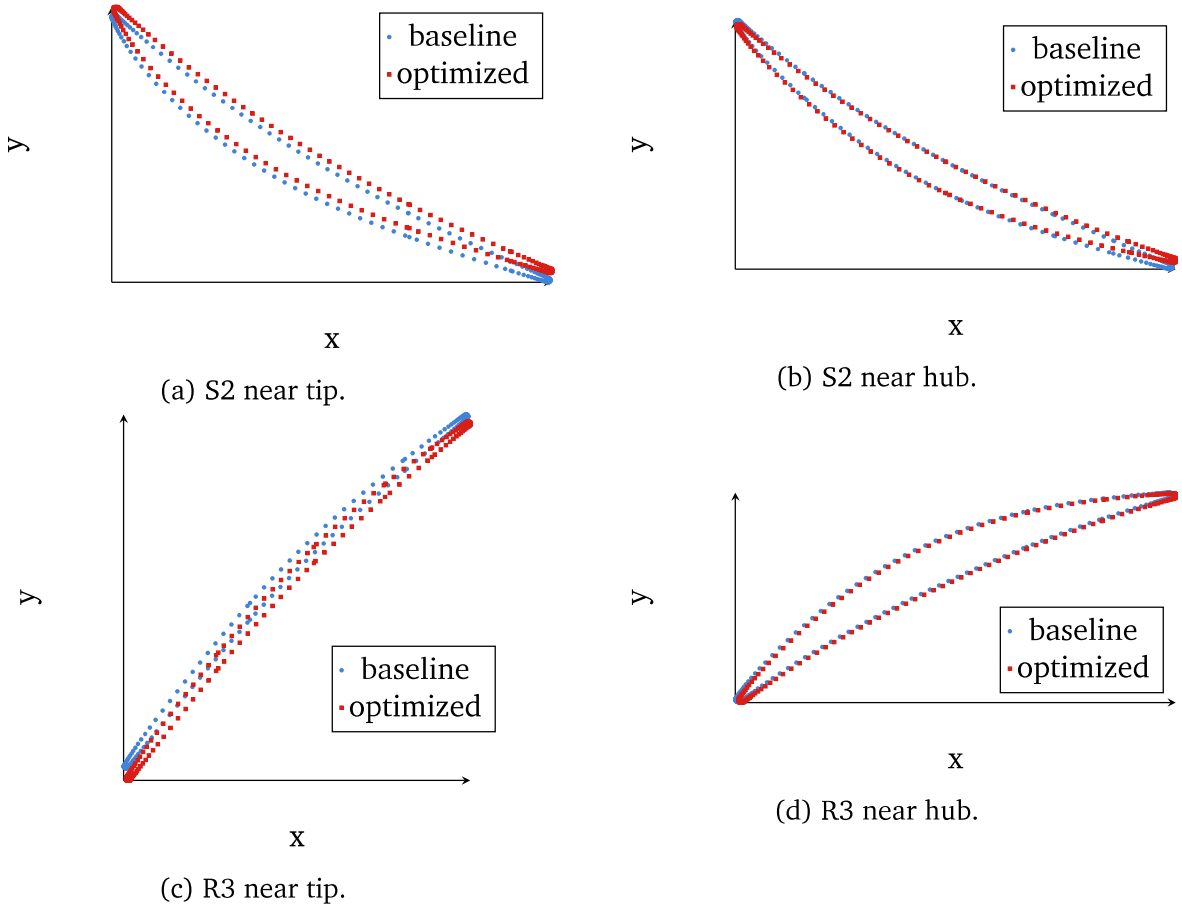
(b) Stator blade (S2) (view from trailing edge side).

(c) Rotor blade (R3) (view from leading edge side).

(d) Rotor blade (R3) (view from trailing edge side).

Figure 4.48: Three-row compressor. Objective function: axial force on the R3 blade. Constraints: Exit capacity and total pressure coefficient. Baseline (gray) vs. improved (green) S2 and R3 blades' geometries.

The new geometry of the configuration reduces the time-integrated objective func-



(a) S2 near tip.

(b) S2 near hub.

(c) R3 near tip.

(d) R3 near hub.

Figure 4.49: Three-row compressor. Baseline vs. improved blade profiles near hub (5% blade's span) and tip (95% blade's span).

tion by 0.76% (from $5.604 \cdot 10^{-1} Ns$ to $5.562 \cdot 10^{-1} Ns$), while violating the time-integrated outlet capacity constraint by 0.06% (from $7.916 \cdot 10^{-7}\, kgK^{1/2}Pa^{-1}$ to $7.912 \cdot 10^{-7}\, kgK^{1/2}Pa^{-1}$) and the time-integrated total pressure coefficient constraint by 0.04% (from $6.318 \cdot 10^{-4} s$ to $6.316 \cdot 10^{-4} s$), fig. 4.50. The static pressure distribution at the midspan of the R3 blade for the baseline and improved designs are given in fig. 4.51 while the instantaneous values of the axial force for the baseline and improved geometries in fig. 4.52.

Since a small step-size is used for 4 optimization cycles, the new geometry does not significantly differ from the baseline. Thus, the spotted differences in fig. 4.52 are marginal. However, the reduction in the objective function is apparent, see fig. 4.52.

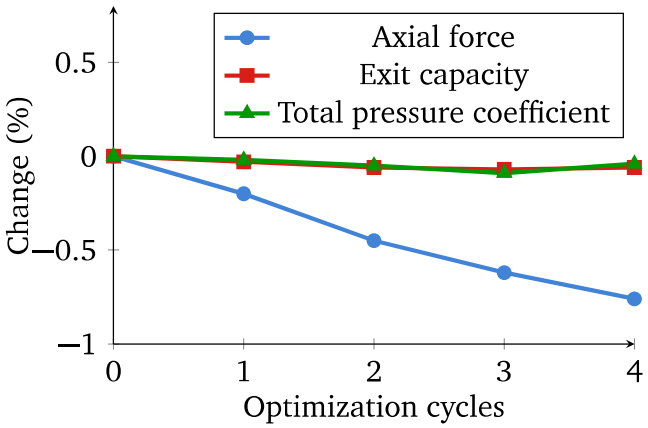


Figure 4.50: Three-row compressor. Objective function: axial force on the R3 blade. Constraints: Exit capacity and total pressure coefficient. Change (%) of the objective function and constraints after four optimization cycles.

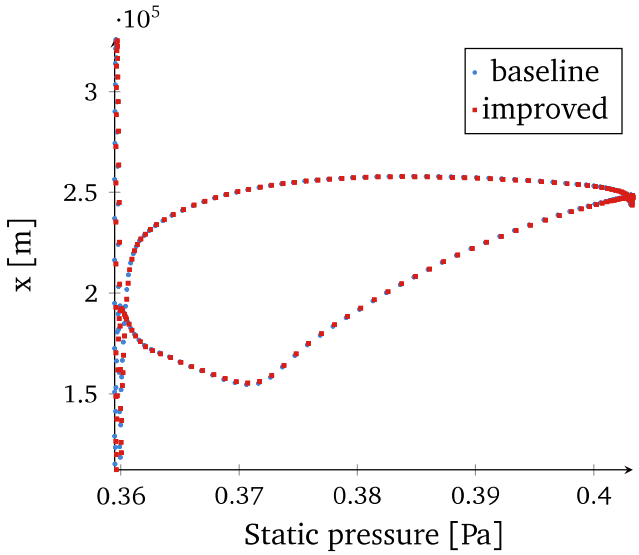


Figure 4.51: Three-row compressor. Pressure distribution at midspan of the R3 blade for baseline and improved design configurations.

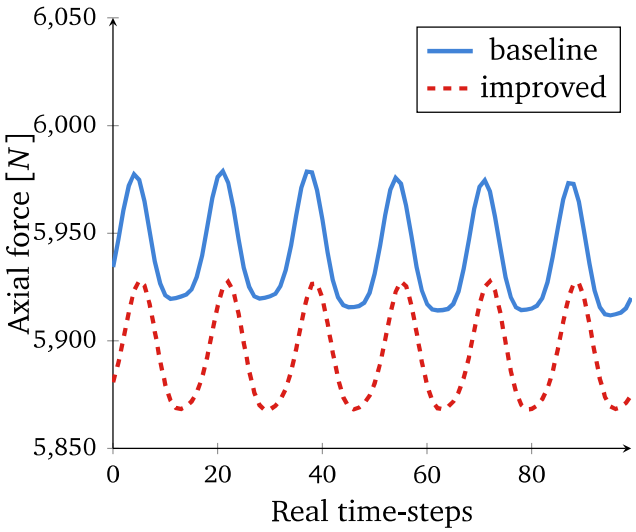


Figure 4.52: Three-row compressor. Instantaneous axial force on the R3 blade for the baseline and improved configuration geometries.

4.5 Temporal Coarsening

Using both the turbine stage and the three-row compressor cases, two alternative coarsening setups will be used for the adjoint solver considering the axial force as the (time-integrated) objective function, the gradient of which needs to be computed. On the first one, the real time-step is considered to be twice as high as that of the reference case and, on the second one, quadruple. The treatment leads to a reduction in both the storage space needed for the primal flow and the running time of the adjoint to half and a quarter of the initial setup, respectively. The comparison of the obtained gradients is plotted using a line graph for the turbine stage in fig. 4.53 and a bar graph for the three-row compressor in fig. 4.54.

The gradients obtained after the temporal coarsening technique are relatively close to the reference values, especially when using a double time-step. In large cases, in which the storage of the entire flow solution time-series becomes impossible due to hardware limitations, the method can be used to keep the cost at affordable levels, without though significantly compromising accuracy.

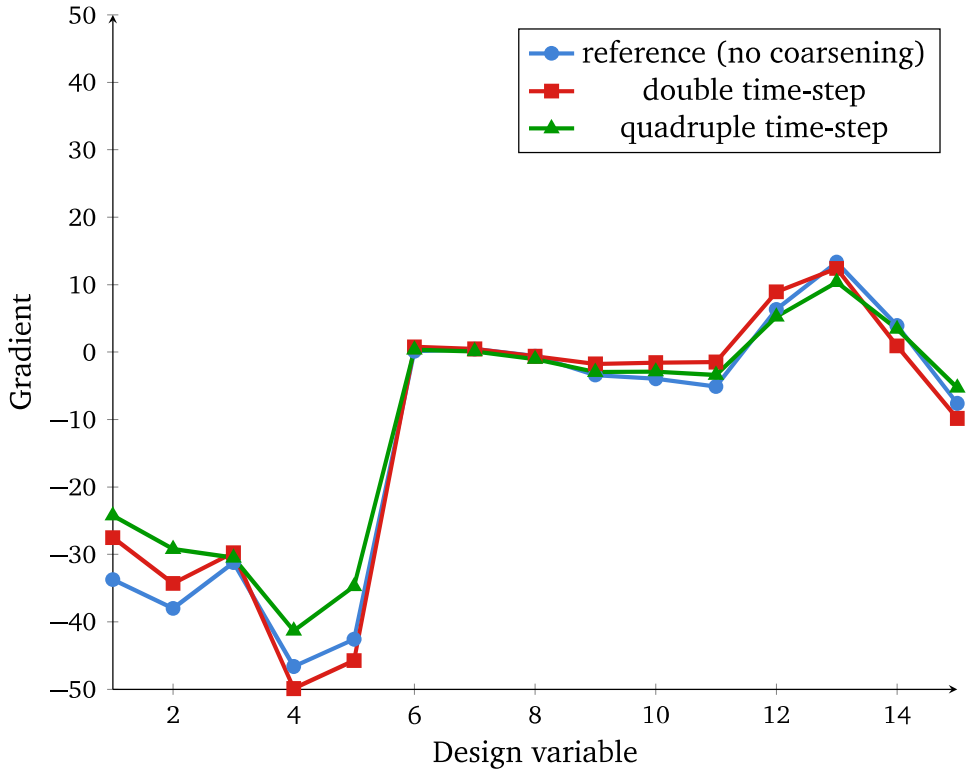


Figure 4.53: HPT stage. Objective function: Axial force on the rotor. Original gradients compared with gradients obtained via the temporal coarsening method.

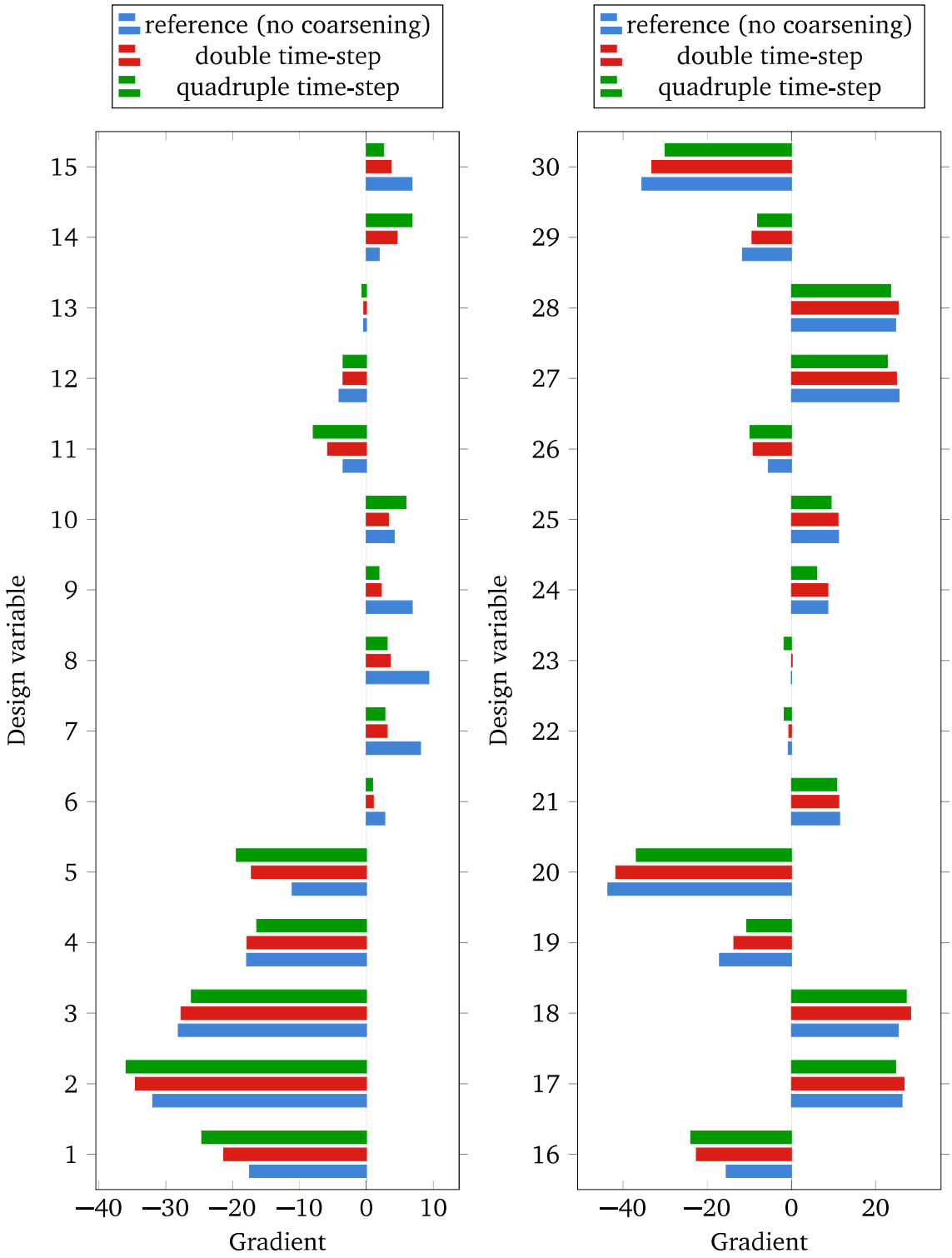


Figure 4.54: Three-row compressor. Objective function: Axial force on the R3 blade. Original gradients compared with gradients obtained via the temporal coarsening method. Left: Gradients computed for the S2 blade. Right: Gradients for the R3 blade.

5

Closure - Conclusions

This PhD thesis aimed at the development, programming, application and assessment of the unsteady discrete adjoint method, formulated in the time-domain, for the exact computation of sensitivity derivatives to be used in shape optimization of 3D, multi-row turbomachines. The development was carried out in the CFD suite Hydra, the standard CFD tool in RR, written in Fortran 77, using a combination of hand-differentiation and algorithmic differentiation by implementing Tapenade, INRIA. The method was applied to periodic and transient compressible flows governed by the URANS equations. The computed derivatives were validated against finite differences and used in an optimization loop to improve turbine and compressor blades while considering equality constraints. In the following paragraphs, some remarks are made and conclusions are drawn concerning the work conducted in this thesis.

The unsteady adjoint solver made use of parallel storage to SSD disks instead of the RAM to retrieve the flow fields during the reverse time integration of the adjoint computations. Thus, RAM limitations regarding the storage of flow fields can be eliminated without creating a significant time overhead. To quantify it, reading in the flow fields comes at a cost of less than 5% compared to the cost of solving the adjoint equations. The size of available SSD capacity may pose a limitation for running very large

cases (larger than the ones considered in this thesis). Nevertheless, the cost-capacity ratio is considerably lower for SSD disks compared to RAM. In addition, in order to reduce the disk storage space and time cost of the adjoint solver for such large cases, the temporal coarsening method was used, which proved to be able to provide gradients with adequate accuracy.

The solution scheme of the flow equations was differentiated by hand to derive an adjoint to the Runge-Kutta method used to solve the unsteady discrete adjoint equations. The implementation of algorithmic differentiation (Appendix A) in the adjoint solver was employed for the computation of selected differential terms in the adjoint equations. Thus, a combination of hand and algorithmic differentiation was used and proved to take advantage of the strengths of both methods. Algorithmic differentiation allowed the quick differentiation of subroutines of the flow solver, avoiding programming errors while eliminating the need to reprogram by hand the adjoint code to incorporate changes made in certain parts of the flow solver. On the other hand, hand differentiation helped to reduce the RAM footprint of the adjoint solver while maintaining solving efficiency.

Formulating an unsteady adjoint solver in the time-domain appears to be the only way to solve optimization problems which involve non-periodic (transient) phenomena. In such cases, the use of existing adjoint solvers formulated in the frequency-domain is not possible. For the specific case of periodic flows, adjoint solvers in both the time-domain and frequency-domain can be used to compute gradients. The cost of frequency-domain solvers is proportional to the number of frequencies used. Quantifying the cost difference between time- and frequency-domain computations (for periodic flows) is case specific and depends on the number of used frequencies for the frequency-domain solver, the number of time-steps for the time-domain solver etc. In summary, the availability of an adjoint time-domain solver is the only possible option for transient flows, which can also deal with periodic flows, at a higher cost though than frequency-domain methods.

The developed unsteady adjoint solver was applied to three turbomachinery cases, successfully optimizing the selected objective function while, in one case, considering equality constraints. The turbine vane case was used as a demonstration of a transient flow case and the time-integrated total pressure ratio was increased by 0.6% after 30 optimization loops. In the turbine stage case, the time-integrated axial force on the rotor was reduced by 8.81% after a single optimization loop; this is a relatively large value justified by the lack of constraints. In the three-row compressor case, the time-

integrated axial force is reduced by 0.76% while violating the constraints by 0.04% and 0.06% after 4 optimization loops; the reduction is smaller in this case because of the smaller optimization step size used to avoid violating the constraints. Since the focus of the thesis is the development of the unsteady adjoint solver, emphasis is given to the validation of the computed gradients of the objective function w.r.t. the design variables using the adjoint method against finite differences.

5.1 Novelties in this Thesis

The novel contributions of this PhD thesis are summarized below:

- The unsteady adjoint method, formulated in the time-domain was applied for the first time in the literature to 3D, multi-row turbomachinery cases. This allows the use of the developed method and software in turbomachinery shape optimization with transient flow phenomena.
- The iterative scheme adjoint to the 5-stage Runge-Kutta method was derived and used to solve the time-accurate adjoint equations so as to ensure the same convergence rate with the URANS solver and obtain the same gradient value whether solving the equations that occur from forward differentiation or using the adjoint method.
- An adjoint sliding interface method was developed to achieve the coupling of the domains of adjacent rows during the adjoint computations by combining hand and algorithmic differentiation.

Publications

The publications that resulted from the research carried out in this thesis are listed below:
Journal publication:

- Ntanakas, G., Meyer, M., and Giannakoglou, K.C. Employing the time domain unsteady discrete adjoint method for shape optimization of 3D multi-row turbomachinery configurations. Journal of Turbomachinery, 140(8):081006, 2018

Conference publications (with full paper):

- Ntanakas, G. and Meyer, M. Towards unsteady adjoint analysis for turbomachinery applications. Paper No. IS14-S8-3, 6th European Conference on Computational Fluid Dynamics-ECFD VI, Barcelona, Spain, 2014.
- Ntanakas, G. and Meyer, M. The unsteady discrete adjoint method for turbomachinery applications. 14th International Symposium on Unsteady Aerodynamics, Aeroacoustics and Aeroelasticity of Turbomachines, Stockholm, Sweden, pp. 5071-5081, 2015.

5.2 Future Work

A brief overview of the points that could be the continuation and expansion of this work follows:

- In the flow solver, OPlus is employed for the parallel execution of the sliding plane implementation. Flow variables are transferred (interpolated) from the internal sliding plane nodes to the corresponding external sliding plane nodes. For the adjoint solver, the process is reversed; data are transferred from the external sliding plane to the corresponding internal. This new operation that needs to be performed is not yet supported by OPlus for multiple partitions. As a remedy, when splitting the grid into different partitions, the user needs to manually assign the nodes of each single sliding interface (internal and external planes) to the same partition for the operation to be performed. This restriction may pose an upper bound to the scalability of the parallel execution of the unsteady adjoint solver for (very) big cases on a large number of computer nodes. It is, thus, essential for future versions of OPlus to fulfill this reversal of data transfers for sliding interface nodes that lay on different partitions.
- For the reduction of the cost of the adjoint computations the temporal coarsening technique was used. Cost reduction is obvious but this might reduce the accuracy of the computed gradients. To maintain the accuracy of the reference gradients, while reducing the storage space cost, a check-pointing scheme can be used. However, this will increase the running time of the flow solver due to recomputation of the flow of intermediate time-steps. A compromise that, if implemented, maintains an acceptable level of gradients' accuracy while not adding an execution time overhead, as big as check-pointing does, is reduced-order modeling methods, such as POD, which will allow the unsteady adjoint

solver to be applied to big cases where storing the flow for every time-step exceeds the available disk's storage limits.

- The unsteady adjoint solver can be extended for problems that involve grid deformation [80, 88], e.g. to account for blade vibration. In this case, the Geometric Conservation Law needs to be incorporated into the flow and adjoint equations to consider the displacement of grid nodes per time-step.

Appendix A

Algorithmic Differentiation Principles

The unsteady adjoint solver is using partially algorithmic differentiation (AD) [176] in order to compute some of the differential terms that appear in the unsteady adjoint equations. In this appendix, the basic principles of AD are given in order to provide the required background that is needed before going into the details of the AD implementation in the unsteady adjoint solver of this work.

It is assumed that there is a computer program $\boldsymbol{P}$ that takes a vector of inputs $\boldsymbol{x} = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix}$ and produces a vector of outputs $\boldsymbol{y} = \begin{bmatrix} y_1 & \cdots & y_m \end{bmatrix}$. There is a wide number of ways to compute the derivative of the outputs w.r.t. the inputs. A natural way is to differentiate by hand the mathematics that led to the computer program and write another program that computes the derivative. This, for large and complex programs that involve solving differential equations, is not an efficient and viable option because it is error-prone and implies discretizing new equations, recoding etc. Another way is to use the original program and finite differences or the complex variable method, as described in section 1.2, after carefully considering their disadvantages.

Using AD software tools, one is able to compute the exact analytical derivatives, and not an approximation to them, by constructing a new augmented program $\boldsymbol{P}'$, called the differentiated program.

Initially, a simple example is set up [195–197]. A computer program is used to compute the following function

$$y_1 = cos(x_1) + lnx_2^2 \tag{A.1}$$

In order to compute the gradient $\frac{dy}{d\boldsymbol{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \end{bmatrix}^T$ the following fundamental steps

are performed:

- the original function is decomposed into intrinsic functions,
- the intrinsic functions are differentiated and
- the differentiated terms are appropriately multiplied according to the chain rule.

If the input vector is $\boldsymbol{x} = \begin{bmatrix} 2 & 3 \end{bmatrix}^T$, the program implements the following intrinsic functions by using the intermediate variables w_i

Original program		
x_1		$= 2$
x_2		$= 3$
w_1	$= cos(x_1)$	$= -0.416$
w_2	$= x_2^2$	$= 9$
w_3	$= lnw_2$	$= 2.197$
w_4	$= w_1 + w_3$	$= 1.781$
y_1	$= w_4$	$= 1.781$

Table A.1: Implementation of the function calculation by a program.

Two distinct modes of AD are presented; forward mode and reverse/adjoint mode. Applying AD in **forward mode** is the conceptually most simple type. Every intermediate variable is associated with a derivative

$$\dot{w}_i = \frac{\partial w_i}{\partial x_j}, \qquad j = 1, 2 \tag{A.2}$$

and the chain rule is applied following the forward trace of the original program.

Forward Differentiation 1st Pass			
$\downarrow$	$\dot{x}_1$		$= \mathbf{1}$
$\downarrow$	$\dot{x}_2$		$= \mathbf{0}$
$\downarrow$	$\dot{w}_1$	$= -\dot{x}_1 sin(x_1)$	$= -0.909$
$\downarrow$	$\dot{w}_2$	$= 2x_2\dot{x}_2$	$= 0$
$\downarrow$	$\dot{w}_3$	$= \dot{w}_2/w_2$	$= 0$
$\downarrow$	$\dot{w}_4$	$= \dot{w}_1 + \dot{w}_2$	$= -0.909$
$\downarrow$	$\dot{y}_1$	$= \dot{w}_4$	$= \mathbf{-0.909}$

Forward Differentiation 2nd Pass			
$\downarrow$	$\dot{x}_1$		$= \mathbf{0}$
$\downarrow$	$\dot{x}_2$		$= \mathbf{1}$
$\downarrow$	$\dot{w}_1$	$= -\dot{x}_1 sin(x_1)$	$= 0$
$\downarrow$	$\dot{w}_2$	$= 2x_2\dot{x}_2$	$= 6$
$\downarrow$	$\dot{w}_3$	$= \dot{w}_2/w_2$	$= 0.667$
$\downarrow$	$\dot{w}_4$	$= \dot{w}_1 + \dot{w}_2$	$= 0.667$
$\downarrow$	$\dot{y}_1$	$= \dot{w}_4$	$= \mathbf{0.667}$

Table A.2: Operations performed from the forward differentiated program.

Using the **reverse/adjoint mode**, every intermediate variable is associated with an adjoint variable

$$\bar{w}_i = \frac{\partial y_1}{\partial w_i} \tag{A.3}$$

and derivatives are computed by propagating the adjoint variables backwards from the outputs to the inputs in Table A.3

Reverse/Adjoint Differentiation				
↑	$\bar{x}_1$	$= \bar{w}_1 \frac{\partial w_1}{\partial x_1}$	$= -\bar{w}_1 sin(x_1)$	$= \mathbf{-0.909}$
↑	$\bar{x}_2$	$= \bar{w}_2 \frac{\partial w_2}{\partial x_2}$	$= \bar{w}_2 2x_2$	$= \mathbf{0.667}$
↑				
↑	$\bar{w}_2$	$= \bar{w}_3 \frac{\partial w_3}{\partial w_2}$	$= \bar{w}_3 \frac{1}{w_2}$	$= 0.111$
↑	$\bar{w}_1$	$= \bar{w}_4 \frac{\partial w_4}{\partial w_3}$	$= \bar{w}_4$	$= 1$
↑	$\bar{w}_3$	$= \bar{w}_4 \frac{\partial w_4}{\partial w_3}$	$= \bar{w}_4$	$= 1$
↑	$\bar{w}_4$	$= \bar{y}_1$		$= \mathbf{1}$

Table A.3: Operations performed from the reverse differentiated program.

Often, to compute an adjoint intermediate variable, an intermediate variable from the original program is needed. For example, to compute $\bar{w}_2$, w_2 is required. Thus, the reverse mode often has two phases. The first one is a forward sweep of the original program to compute any needed intermediate variables, which is omitted in this example. The second phase involves the reverse sweep of the differentiated code.

It is observed that in order to compute the full gradient using forward differentiation, two passes using a different initial seed are needed while using reverse differentiation only one. I.e. the cost of forward differentiation is proportional to the size of the input vector whereas reverse differentiation to the size of the output vector.

In the general case of a function $f : \boldsymbol{R}^n \rightarrow \boldsymbol{R}^m$ so that $\boldsymbol{y} = f(\boldsymbol{x})$, the Jacobian is given by

$$\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

If a single-entry input vector of the form

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x}_1 \cdots \dot{x}_i \cdots \dot{x}_n \end{bmatrix}^T = \begin{bmatrix} 0 \cdots 1 \cdots 0 \end{bmatrix}^T \tag{A.4}$$

(zeros everywhere except the entry $\dot{x}_i$) is given, a forward pass yields the i_{th} column of the Jacobian, indicated by the plain border-line. In the general case, where $\dot{\boldsymbol{x}}$ is not simply a single-entry vector, the forward pass offers a matrix free way of computing

$$\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} \begin{bmatrix} \dot{x}_1 \\ \vdots \\ \dot{x}_n \end{bmatrix} \tag{A.5}$$

On the other hand, if a transposed single-entry input vector of the form

$$\bar{\boldsymbol{y}} = \begin{bmatrix} \bar{y}_1 \cdots \bar{y}_i \cdots \bar{y}_m \end{bmatrix}^T = \begin{bmatrix} 0 \cdots 1 \cdots 0 \end{bmatrix}^T \tag{A.6}$$

is given, a reverse pass yields the i_{th} row of the Jacobian, indicated by the dashed border-line. In the general case, where $\bar{\boldsymbol{y}}$ is not simply a single-entry vector, the reverse pass offers a matrix free way of computing

$$\begin{bmatrix} \bar{y}_1 & \cdots & \bar{y}_m \end{bmatrix}^T \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} \tag{A.7}$$

If the Jacobian needs to be computed, choosing the forward or reverse mode translates to whether the columns are more than the rows or equivalently whether the inputs n are more than the outputs m.

Usually, a computer program implements a composition of functions and not a single function. To examine this case, it is assumed that a computer program reads in an input vector $\zeta_0 \in \mathbb{R}^n$ and produces an output vector $\zeta_N \in \mathbb{R}^n$ through a series of operations of the form

$$\zeta_n = f_n(\zeta_{n-1}) \tag{A.8}$$

so that

$$\zeta_N = f_N \circ f_{N-1} \circ \ldots \circ f_2 \circ f_1(\zeta_0) = \boldsymbol{F}(\zeta_0) \tag{A.9}$$

Since $\boldsymbol{F}$ is a composition of functions, the first-order derivative is given by the chain

rule as

$$\boldsymbol{F}'(\zeta_0) = f'_N(\zeta_{N-1}) \cdot f'_{N-1}(\zeta_{N-2}) \cdot \ldots \cdot f'_1(\zeta_0) \tag{A.10}$$

Similarly to the application of AD to a single function program, the differentiated code computes the products

$$\boldsymbol{F}'(\zeta_0) \cdot \dot{\zeta}_0 \quad \text{or} \quad \overline{\zeta}_N \cdot \boldsymbol{F}'(\zeta_0)$$

where $\dot{\zeta}_0 \in \mathbb{R}^n$ and $\overline{\zeta}_N \in \mathbb{R}^m$, when using the forward or the reverse mode, respectively.

Forward Mode	Reverse Mode
$\zeta_1 = f_1(\zeta_0)$	$\zeta_1 = f_1(\zeta_0)$
$\dot{\zeta}_1 = f'_1(\zeta_0) \cdot \dot{\zeta}_0$	$\zeta_2 = f_2(\zeta_1)$
$\zeta_2 = f_2(\zeta_1)$	$\vdots$
$\dot{\zeta}_2 = f'_2(\zeta_1) \cdot \dot{\zeta}_1$	$\zeta_N = f_N(\zeta_{N-1})$
$\vdots$	$\overline{\zeta}_{N-1} = \overline{\zeta}_N \cdot f'_N(\zeta_{N-1})$
$\zeta_N = f_N(\zeta_{N-1})$	$\overline{\zeta}_{N-2} = \overline{\zeta}_{N-1} \cdot f'_{N-1}(\zeta_{N-2})$
$\dot{\zeta}_N = f'_N(\zeta_{N-1}) \cdot \dot{\zeta}_{N-1}$	$\vdots$
	$\overline{\zeta}_0 = \overline{\zeta}_1 \cdot f'_1(\zeta_0)$

Table A.4: Forward and reverse mode for program of composite functions.

The resulting vector from the forward mode is

$$\dot{\zeta}_N = f'_N(\zeta_{N-1}) \cdot f'_{N-1}(\zeta_{N-2}) \cdot \ldots \cdot f'_1(\zeta_0) \cdot \dot{\zeta}_0 \tag{A.11}$$

and from the reverse mode

$$\overline{\zeta}_0 = \overline{\zeta}_N \cdot f'_N(\zeta_{N-1}) \cdot f'_{N-1}(\zeta_{N-2}) \cdot \ldots \cdot f'_1(\zeta_0) \tag{A.12}$$

The Jacobian of the outputs w.r.t. the inputs can be computed by running either the program that resulted by applying tangent mode on the original program m times or the program that resulted by applying reverse mode to the original program n times, using the appropriate input vectors.

Forward or reverse differentiation are two extreme ways of traversing the chain rule. The problem of computing a full Jacobian with the minimum number of operations is known as the Optimal Jacobian Accumulation Problem, which is NP-complete [198] and is beyond the scope of this appendix. Simplifying, it can be said that forward

mode is best suited when $n < m$ while reverse mode is best suited when $n > m$.

AD is implemented using two strategies: source code transformation and operation overloading.

Source code transformation [177, 199, 200]: The original source code that implements the function composition of which the derivative needs to be computed is replaced by an automatically generated augmented source code, fig. A.1. An AD tool takes over the transformation of the original source code. The AD tool parses the original source code similarly to a compiler but instead of producing objective files and executables, produces new source code. Source code transformation is possible in most programming languages including legacy Fortran or C codes. Moreover, it allows compile-time optimizations that reduce running time. On the other side, creating a source code transformation AD tool is more development-intensive than operation overloading.

Operation overloading [201–204]: If the programming language permits it, one can replace the types of floating-point variables with a new type that contains additional derivative information and overload the arithmetic operations for this new type to propagate the derivative information. This is performed by a library that defines the overloaded type and arithmetic operations, fig. A.2. There are no substantial changes that need to be done in the original code and coding the AD tool is easier comparing to source code transformation. However, operation overloading is limited to selected programming languages that support it. In addition, current compilers lag behind in optimizing code that uses operation overloading libraries.

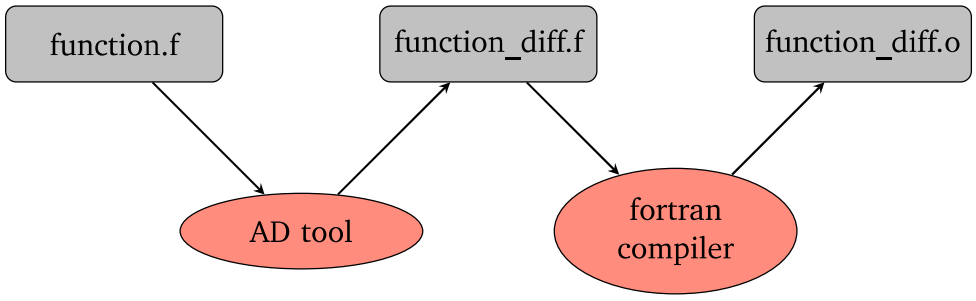


Figure A.1: Simplified schematic of source code transformation.

An exhaustive list of AD tools, on-going research and applications can be found in [205].

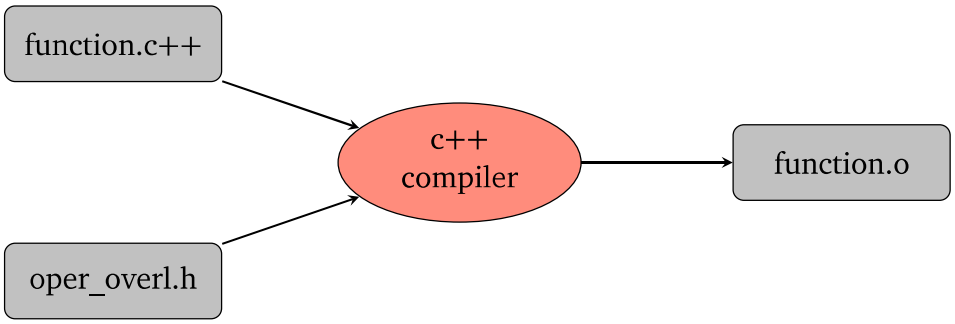


Figure A.2: Simplified schematic of operation overloading.

A.1 Source Code Transformation using Tapenade

Tapenade, INRIA, is the source code transformation AD tool that was used to compute some of the differential terms during the unsteady adjoint computations of this work. In this section, the functionality of Tapenade is demonstrated in the simple function example that was introduced in the previous section. The programming language is Fortran 90.

Initially, the code that mimics the operations that were performed in table A.1 is written as follows

```
subroutine function(x1,x2,y1)
!---------------------------
implicit none
real*8 :: x1,x2,y1
real*8 :: w1,w2,w3,w4
!---------------------------
w1=cos(x1)
w2=x2**2
w3=log(w2)
w4=w1+w3
y1=w4
end subroutine
```

This source code is given as an input to Tapenade along with the arguments that flag forward differentiation and identify x_1 and x_2 as inputs and y_1 as output. Tapenade produces the following source code.

```
!       Generated by TAPENADE   (INRIA, Tropics team)
!  Tapenade 3.4 (r3375) - 10 Feb 2010 15:08
!
!  Differentiation of function in forward (tangent) mode:
```

```
!   variations  of useful results: y1
!   with respect to varying inputs: x1 x2
!   RW status of diff variables: y1:out x1:in x2:in
!-----------------------------
SUBROUTINE FUNCTION_D(x1, x1d, x2, x2d, y1, y1d)
  IMPLICIT NONE
  REAL*8 :: x1, x2, y1
  REAL*8 :: x1d, x2d, y1d
  REAL*8 :: w1, w2, w3, w4
  REAL*8 :: w1d, w2d, w3d, w4d
  INTRINSIC COS
  INTRINSIC LOG
!-----------------------------
  w1d = -(x1d*SIN(x1))
  w1 = COS(x1)
  w2d = 2*x2*x2d
  w2 = x2**2
  w3d = w2d/w2
  w3 = LOG(w2)
  w4d = w1d + w3d
  w4 = w1 + w3
  y1d = w4d
  y1 = w4
END SUBROUTINE FUNCTION_D
```

It is observed that along with the original operations there are additional program lines that compute derivatives. The variables that end with a "d" correspond to the variables with an overdot of table A.2.

Simply by changing the flag that indicates the differentiation mode from "forward" to "reverse", Tapenade provides the adjoint code.

```
!       Generated by TAPENADE   (INRIA, Tropics team)
!  Tapenade 3.4 (r3375) - 10 Feb 2010 15:08
!
!  Differentiation of function in reverse (adjoint) mode:
!   gradient    of useful results: y1
!   with respect to varying inputs: y1 x1 x2
!   RW status of diff variables: y1:in-zero x1:out x2:out
!-----------------------------
SUBROUTINE FUNCTION_B(x1, x1b, x2, x2b, y1, y1b)
  IMPLICIT NONE
  REAL*8 :: x1, x2, y1
```

```
  REAL*8 :: x1b, x2b, y1b
  REAL*8 :: w1, w2, w3, w4
  REAL*8 :: w1b, w2b, w3b, w4b
  INTRINSIC COS
  INTRINSIC LOG
!----------------------------
  w2 = x2**2
  w4b = y1b
  w1b = w4b
  w3b = w4b
  w2b = w3b/w2
  x2b = 2*x2*w2b
  x1b = -(SIN(x1)*w1b)
  y1b = 0.0_8
END SUBROUTINE FUNCTION_B
```

In this case, variables that end with a "b" correspond to the overbar variables of table A.3. The resulting code can be divided into two sub-parts. In the first part, a forward sweep computes the intermediate w variables needed for the reverse differentiation. For the current example, that translates only to w_2 and only to one line (the first line after the variable declaration) of the reverse differentiated source code. Then, a reversal of the trace of operations of the original source code takes place.

In the case of a subroutine where some of the intermediate w values are overwritten during the execution of the original code, a problem arises because of the data-flow reversal of the adjoint code. There are two extreme options:

- Recompute-all: The part of the code that is needed to retrieve an overwritten value is re-executed every time an overwritten value is needed. This option adds execution time.
- Store-all: During the forward sweep of the original code, whenever an intermediate value that will be needed during the backward sweep is overwritten, its old value is stored on a stack with a "push" command. During the backward sweep, when the intermediate value is needed, this is retrieved using a "pop" command. This option increases the memory requirements and, indirectly, the running time by slowing down the memory access.

Often, a trade-off between the two methods is used. This translates to storing some of the overwritten intermediate values in the stack and using them as a starting point to recompute the remaining needed intermediate values.

Bibliography

[1] Gorla, R.S.R. and Khan, A.A. *Turbomachinery: design and theory*. Mechanical Engineering. CRC Press, 2003.

[2] Logan, E. *Handbook of turbomachinery*. Mechanical Engineering. CRC Press, 2003.

[3] Korpela, S.A. *Principles of turbomachinery*. John Wiley & Sons, 2012.

[4] Rolls-Royce. *The jet engine*. John Wiley & Sons, 2015.

[5] Papailiou, K.D., Mathioudakis, K.M., and Giannakoglou, K.C. *Introduction to thermal turbomachines*. NTUA, 2000. (in Greek).

[6] Navier, C. L. M. H. Mémoire sur les lois du mouvement des fluids. *Memémoires de l' Académie des sciences de l' Institut de France*, 6:389–416, 1823.

[7] Stokes, G. G. On the theories of the internal friction of fluids in motion, and of the equilibrium and motion of elastic solids. *Transactions of the Cambridge Philosophical Society*, 8:287–305, 1845.

[8] Smagorinsky, J. General circulation experiments with the primitive equations. *Monthly Weather Review*, 91(3):99–164, 1963.

[9] Deardorff, J. W. A numerical study of three-dimensional turbulent channel flow at large Reynolds numbers. *Journal of Fluid Mechanics*, 41(2):453–480, 1970.

[10] Sullivan, P. P., McWilliams, J. C., and Moeng, C.H. A subgrid-scale model for Large-Eddy Simulation of planetary boundary-layer flows. *Boundary-Layer Meteorology*, 71(3):247–276, 1994.

[11] Forst, W. and Hoffmann, D. *Optimization—theory and practice*. Springer Science & Business Media, 2010.

[12] Chong, E. K.P. and Zak, S. H. *An introduction to optimization*. John Wiley & Sons, 2013.

[13] Bäck, T. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Inc., 1996.

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

[15] Winter, G., Periaux, J., Galan, M., and Cuesta, P. *Genetic Algorithms in Engineering and Computer Science*. John Wiley & Sons, Inc., 1996.

[16] Kontoleontos, E. Designing thermo-fluid systems using gradient-based optimization methods and evolutionary algorithms. PhD thesis, Lab. of Thermal Turbomachines, NTUA, Athens, Greece, 2012. (in Greek).

[17] Kyriacou, S. Evolutionary algorithm-based design-optimization methods in turbomachinery. PhD thesis, Lab. of Thermal Turbomachines, NTUA, Athens, Greece, 2013. (in Greek).

[18] Thevenin, D. and Janiga, G. *Constrained optimization and Lagrange multiplier methods*. Athena Scientific, 1996.

[19] Mohammadi, D. and Pironneau, O. *Applied shape optimization for fluids*. Oxford University Press, 2001.

[20] Luenberger, D. G. *Linear and nonlinear programming*. Kluwer Academic Publishers, 2003.

[21] Thevenin, D. and Janiga, G. *Optimization and computational fluid dynamics*. Springer, 2008.

[22] Nocedal, J. and Wright, S. *Numerical optimization*. Springer Series in Operations Research, 2006.

[23] Giannakoglou, K.C. *Optimization methods in aerodynamics*. NTUA, 2005. (in Greek).

[24] Zhihui Li, Z. and Xinqian Zheng, X. Review of design optimization methods for turbomachinery aerodynamics. *Progress in Aerospace Sciences*, 93:1 – 23, 2017.

[25] Pierret, S. and Braembussche, R. Turbomachinery blade design using a Navie-Stokes solver and artificial neural network. *Journal of Turbomachinery*, 121(2):326–332, 1999.

[26] Yongsheng, L. and Meng-Sing, L. Multiobjective optimization using coupled response surface model and evolutionary algorithm. *AIAA Journal*, 43(6):1316–1325, 2005.

[27] Vanderplaats, G. N., Hicks, R. M., and Murman, E. M. Application of numerical optimization techniques to airfoil design. *Proc. Aerodynamic Analyses Requiring Advanced Computers, Part II, Langley Research Center, Hampton, USA*, 1975.

[28] Newmann, J. C., Anderson, W. K., and Whitfield, D. L. Multidisciplinary sensitivity derivatives using complex variables. MSSU-COE-ERC-98-08, Mississippi State University, Mississippy, USA, 1998.

[29] Nielsen, E. J. and Kleb, W. L. Efficient construction of discrete adjoint operators on unstructured grids using complex variables. *AIAA Journal*, 44(4):827–836, 2006.

[30] Baysal, O. and Eleshaky, M. E. Aerodynamic sensitivity analysis methods for the compressible euler equations. *Journal of Fluids Engineering*, 113(4):681–688, 1991.

[31] Turgeon, É., Pelletier, D., Borggaard, J., and Etienne, S. Application of a sensitivity equation method to the k–ϵ model of turbulence. *Optimization and Engineering*, 8(4):341–372, Dec 2007.

[32] Pironneau, O. *Optimal shape design for elliptic systems*. Springer Series in Computational Physics, 1984.

[33] Jameson, A., Pierce, N., and L., Martinelli. Aerodynamic design via control theory. *Theoretical and Computational Fluid Dynamics*, 3(3):233–260, 1988.

[34] Jameson, A. and Reuther, J. Control theory based airfoil design using the euler equations. AIAA Paper 1994-4272, 5th Symposium on Multidisciplinary Analysis and Optimization, Florida, USA, 1994.

[35] Jameson, A. Optimum aerodynamic design using CFD and control theory. AIAA Paper 1995-1729, 12th Computational Fluid Dynamics Conference, San Diego, USA, 1995.

[36] Jameson, A. Optimum aerodynamic design using the Navier–Stokes equations. *Journal of scientific computing*, 10(1-4):213–237, 1998.

[37] Anderson, W. K. and V., Venkatakrishnan. Aerodynamic design optimization on unstructured grids with a continuous adjoint formulation. AIAA Paper 1997-0643, 35th AIAA Aerospace Sciences Meeting and Exhibit. Reno, USA, 1997.

[38] Anderson, W. K. and Bonhaus, D. L. Airfoil design on unstructured grids for turbulent flows. *AIAA Journal*, 37(2):185–191, 1999.

[39] Yang, S., Wu, H., Liu, F., and Tsai, H. Aerodynamic design of cascades by using an adjoint equation method. AIAA Paper 2003-1068, 41st Aerospace Sciences Meeting and Exhibit, Aerospace Sciences Meetings, Reno, USA, 2003.

[40] Chung, J., Lee, K., and Martin, G. Aerodynamic design of 3D compressor blade using an adjoint method. AIAA Paper 2004-27, 42nd AIAA Aerospace Sciences Meeting and Exhibit, Reno, USA, 2004.

[41] Wu, H., Liu, F., and Tsai, H. Aerodynamic design of turbine blades using an adjoint equation method. AIAA paper 2005-1006, 43rd AIAA Aerospace Sciences Meeting and Exhibit, Aerospace Sciences Meetings, Reno, USA, 2005.

[42] Papadimitriou, D. I. and Giannakoglou, K. C. A continuous adjoint method for the minimization of losses in cascade viscous flows. AIAA paper 2006-49, 44th AIAA Aerospace Sciences Meeting and Exhibit, Reno, USA, 2006.

[43] Papadimitriou, D. I. and Giannakoglou, K. C. Compressor blade optimization using a continuous adjoint formulation. ASME Paper GT2006-90466, ASME Gas Turbine and Aeroengine Technical Congress and Exposition, Barcelona, Spain, 2006.

[44] Papadimitriou, D. I. and Giannakoglou, K. C. A continuous adjoint method with objective function derivatives based on boundary integrals, for inviscid and viscous flows. *Computers & Fluids*, 36(2):325–341, 2007.

[45] Papadimitriou, D. I. and Giannakoglou, K. C. Total pressure loss minimization in turbomachinery cascades using a new continuous adjoint formulation. *Proceedings of the Institution of Mechanical Engineers, Part A: Journal of Power and Energy*, 221(6):865–872, 2007.

[46] Denton, J. D. The calculation of three-dimensional viscous flow through multistage turbomachines. *Journal of Turbomachinery*, 114(1):18–26, 1992.

[47] Frey, C., Kersken, H. K., and Nürnberger, D. The discrete adjoint of a turbomachinery rans solver. ASME Paper GT2009-59062, ASME Turbo Expo, Orlando, USA, 2009.

[48] Wang, D. X. and He, L. Adjoint aerodynamic design optimization for blades in multistage turbomachines—Part I: Methodology and verification. *Journal of Turbomachinery*, 132(2):021011, 2010.

[49] Wang, D. X., He, L., Li, Y. S., and Wells, R. G. Adjoint aerodynamic design optimization for blades in multistage turbomachines—Part II: Validation and application. *Journal of Turbomachinery*, 132(2):021012, 2010.

[50] Jameson, A. Aerodynamic shape optimization using the adjoint method. VKI Lecture Series on Aerodynamic Drag Prediction and Reduction, von Karman Institute of Fluid Dynamics, Brussels, Belgium, 2003.

[51] Jameson, A., Shankaran, S., and Martinelli, L. Continuous adjoint method for unstructured grids. *AIAA Journal*, 46(5):1226–1239, 2008.

[52] Giles, M. B. and Pierce, N. Adjoint equations in CFD-duality, boundary conditions and solution behavior. AIAA Paper 1997-1850, 13th AIAA Computational Fluid Dynamics Conference, Snowmass Village, USA, 1997.

[53] Giles, M. B., Duta, M. C., Müller, J.-D., and Pierce, N. A. Algorithm developments for discrete adjoint methods. *AIAA Journal*, 41(2):198–205, 2003.

[54] Mavriplis, D. J. Discrete adjoint-based approach for optimization problems on three-dimensional unstructured meshes. *AIAA Journal*, 45(4):741–750, 2007.

[55] Giles, M. B. and Pierce, Niles A. An introduction to the adjoint approach to design. *Flow, Turbulence and Combustion*, 65(3):393–415, 2000.

[56] Nadarajah, S. and Jameson, A. A comparison of the continuous and discrete adjoint approach to automatic aerodynamic optimization. AIAA 2000-667, 38th AIAA Aerospace Sciences Meeting, Reno, USA, 2000.

[57] Nadarajah, S. and Jameson, A. Studies of the continuous and discrete adjoint approaches to viscous automatic aerodynamic shape optimization. AIAA Anaheim, USA, 2001.

[58] Papadimitriou, D. I., Zymaris, A. C., and Giannakoglou, K. C. Discrete and continuous adjoint formulations for turbomachinery applications. EUROGEN, Munich, Germany, 2005.

[59] Peter, J. E. V. and Dwight, R. P. Numerical sensitivity analysis for aerodynamic optimization: A survey of approaches. *Computers & Fluids*, 39(3):373 – 391, 2010.

[60] Giles, M. B. On adjoint equations for error analysis and optimal grid adaptation. Technical report, Computing Laboratory, Oxford University, UK, 1997.

[61] Houston, P. Adjoint error estimation and adaptivity for hyperbolic problems. In *Handbook of Numerical Methods for Hyperbolic Problems*, volume 18, pages 233 – 261. Elsevier, 2017.

[62] Fidkowski, K. J. and Darmofal, D. L. Review of output-based error estimation and mesh adaptation in computational fluid dynamics. *AIAA Journal*, 49(4):673–694, 2011.

[63] Wang, Q. *Uncertainty quantification for unsteady fluid flow using adjoint-based approaches*. PhD thesis, Stanford University, USA, 2008.

[64] Shankaran, S. and Jameson, A. Robust optimal control using polynomial chaos and adjoints for systems with uncertain inputs. AIAA Paper 2011-3069, 20th AIAA Computational Fluid Dynamics Conference, Fluid Dynamics and Co-located Conferences, Honolulu, Hawaii, 2011.

[65] Palacios, F., Duraisamy, K., Alonso, J. J., and Zuazua, E. Robust grid adaptation for efficient uncertainty quantification. *AIAA Journal*, 50(7):1538–1546, 2012.

[66] Saad, Y and Schultz, M. H. GMRES: A generalized minimal residual algorithm for solving non-symmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[67] Shroff, G. M. and Keller, H. B. Stabilization of unstable procedures: The recursive projection method. *SIAM Journal on Numerical Analysis*, 30(4):1099–1120, 1993.

[68] Keller, H. *A remedy for instability*, chapter 10, pages 185–106. SIAM, 2002.

[69] Campobasso, M. S. and Giles, M. B. Stabilization of linear flow solver for turbomachinery aeroelasticity using recursive projection method. *AIAA Journal*, 42(9):1765–1774, 2004.

[70] Campobasso, M. S., Duta, M. C., and B., Giles M. Adjoint methods for turbomachinery design. ISABE Conference Paper 2001-1055, Bangalore, India, 2001.

[71] Duta, M. C., B., Giles M., and Campobasso, M. S. The harmonic adjoint approach to unsteady turbomachinery design. *International Journal for Numerical Methods in Fluids*, 20:323–332, 2002.

[72] Campobasso, M. S., Duta, M. C., and B., Giles M. Adjoint calculation of sensitivities of turbomachinery objective functions. *Journal of Propulsion and Power*, 14(4):693–703, 2003.

[73] Nadarajah, S. K., McMullen, M., and Jameson, A. Optimum shape design for unsteady flows using time accurate and non-linear frequency domain methods. AIAA Paper 2003-3875, 33rd AIAA Fluid Dynamics Conference and Exhibit, Fluid Dynamics and Co-located Conferences, Orlando, USA, 2003.

[74] Nadarajah, S. K. and Jameson, A. Optimal control of unsteady flows using a time accurate method. AIAA Paper 2002-5436, 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Atlanta, USA, 2002.

[75] Wang, Q., Ham, F., Iaccarino, G., and Moin, P. Risk quantification in unsteady flow simulations using adjoint-based approaches. AIAA Paper 2009-2277, 50th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Palm Springs, USA, 2009.

[76] Flynt, B. T. and Mavriplis, D. J. Discrete adjoint based adaptive error control in unsteady flow problems. AIAA Paper 2012, 50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition, Aerospace Sciences Meetings, Nashville, USA, 2012.

[77] Mani, K. and Mavriplis, D. Geometry optimization in three-dimensional unsteady flow problems using the discrete adjoint. AIAA Paper 2013-0662, 51st AIAA Aerospace Sciences Meeting, Texas, USA, 2013.

[78] Mishra, A., Mavriplis, D., and Sitaraman, J. Time-dependent aeroelastic adjoint-based aerodynamic shape optimization of helicopter rotors in forward flight. *AIAA Journal*, 54(12):3813–3827, 2016.

[79] Nielsen, E. J., Diskin, B., and Yamaleev, N. K. Discrete adjoint-based design optimization of unsteady turbulent flows on dynamic unstructured grids. *AIAA Journal*, 48(6):1195–1206, 2010.

[80] Nielsen, E. J. and Diskin, B. Discrete adjoint-based design for unsteady turbulent flows on dynamic overset unstructured grids. *AIAA Journal*, 51(6):1355–1373, 2013.

[81] Fabiano, E., Mishra, A., Mavriplis, D., and Mani, K. Time-dependent aeroacoustic adjoint-based shape optimization of helicopter rotors in forward flight. AIAA Paper 2016-1910, 57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, San Diego, USA, 2016.

[82] Thomas, J. P., Hall, K. C., and Dowell, E. H. Discrete adjoint approach for modeling unsteady aerodynamic design sensitivities. *AIAA Journal*, 43(9):1931–1936, 2005.

[83] Lee, K. H., Alonso, J. J., and van der Weide, E. Mesh adaptation criteria for unsteady periodic flows using a discrete adjoint time-spectral formulation. AIAA Paper 2006-692, 44th AIAA Aerospace Sciences Meeting and Exhibit, Reno, USA, 2006.

[84] Nadarajah, S. K. and Jameson, A. Optimum shape design for unsteady flows with time-accurate continuous and discrete adjoint methods. *AIAA Journal*, 45(7):1478–1491, 2007.

[85] Mani, K. and Mavriplis, D. J. Discrete adjoint based time-step adaptation and error reduction in unsteady flow problems. AIAA Paper 2007-3944, 18th AIAA Computational Fluid Dynamics Conference, Fluid Dynamics and Co-located Conferences, Miami, USA, 2007.

[86] Mavriplis, D. J. Solution of the unsteady discrete adjoint for three-dimensional problems on dynamically deforming unstructured meshes. AIAA Paper 2008-727, 46th AIAA Aerospace Sciences Meeting and Exhibit, Reno, USA, 2008.

[87] Rumpfkeil, M. P. and Zingg, D. W. Unsteady optimization using a discrete adjoint approach applied to aeroacoustic shape design. AIAA Paper 2008-18, 46th AIAA Aerospace Sciences Meeting and Exhibit, Aerospace Sciences Meetings, Reno, USA, 2008.

[88] Mani, K. and Mavriplis, D. J. Unsteady discrete adjoint formulation for two-dimensional flow problems with deforming meshes. *AIAA Journal*, 46(6):1351–1364, 2008.

[89] Mani, K. and Mavriplis, D. J. Adjoint-based sensitivity formulation for fully coupled unsteady aeroelasticity problems. *AIAA Journal*, 47(8):1902–1915, 2009.

[90] Krakos, J. A. and Darmofal, D. L. Effect of small-scale output unsteadiness on adjoint-based sensitivity. *AIAA Journal*, 48(11):2611–2623, 2010.

[91] Wang, L., Mavriplis, D. J., and Anderson, W. K. Adjoint sensitivity formulation for discontinuous Galerkin discretizations in unsteady inviscid flow problems. *AIAA Journal*, 48(12):2867–2883, 2010.

[92] Wang, L., Mavriplis, D. J., and Anderson, K. Unsteady discrete adjoint formulation for high-order discontinuous Galerkin discretizations in time-dependent flow problems. AIAA Paper 2010-367, 48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, Orlando, USA, 2010.

[93] Nemili, A., Özkaya, E., Gauger, N. R., Carnarius, A., and Thiele, F. Optimal control of unsteady flows using discrete adjoints. AIAA-Paper 2011-3720, 41st AIAA Fluid Dynamics Conference and Exhibit, Fluid Dynamics and Co-located Conferences, Honolulu, Hawaii, 2011.

[94] Lee, B. J., Padulo, M., and Liou, M. S. Non-sinusoidal trajectory optimization of flapping airfoil using unsteady adjoint approach. AIAA Paper 2011-1312, 49th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, Orlando, USA, 2011.

[95] Lee, B. J. and L., M. S. Unsteady adjoint approach for design optimization of flapping airfoils. *AIAA Journal*, 50(11):2460–2475, 2012.

[96] Flynt, B. T. and Mavriplis, D. J. Efficient adaptation using discrete adjoint error estimates in unsteady flow problems. AIAA Paper 2013-520, 51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition, Texas, USA, 2013.

[97] Flynt, B. T. and Mavriplis, D. J. Optimal error control using discrete adjoint error estimates in unsteady flow problems. AIAA Paper 2014-1434, 52nd Aerospace Sciences Meeting, Maryland, USA, 2014.

[98] Zhang, Z., Chen, P. C., Yang, S., Wang, Z., and Wang, Q. Unsteady aerostructure coupled adjoint method for flutter suppression. *AIAA Journal*, 53(8):2121–2129, 2015.

[99] Economon, T. D., Palacios, F., and Alonso, J. J. Unsteady continuous adjoint approach for aerodynamic design on dynamic meshes. *AIAA Journal*, 53(9):2437–2453, 2015.

[100] Hückelheim, J., Xu, S., Gugała, M., and Müller, J. D. Time-averaged steady vs. unsteady adjoint: a comparison for cases with mild unsteadiness. AIAA Paper 2015-1953, 53rd AIAA Aerospace Sciences Meeting, Kassimmee, USA, 2015.

[101] Zhou, B. Y., Albring, T., Gauger, N. R., Economon, T. D., Palacios, F., and Alonso, J. J. A discrete adjoint framework for unsteady aerodynamic and aeroacoustic optimization, 2015.

[102] Zhou, B. Y., Albring, T. A., Gauger, N. R., Economon, T. D., and Alonso, J. J. An efficient unsteady aerodynamic and aeroacoustic design framework using discrete adjoint. AIAA Paper 2016-3369, 17th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Washington, USA, 2016.

[103] Nadarajah, S. K., S., McMullen M., and Jameson, A. Non-linear frequency domain based optimum shape design for unsteady three-dimensional flows. AIAA Paper 2006-1052, 44th AIAA Aerospace Sciences Meeting and Exhibit, Reno, USA, 2006.

[104] Kennedy, G. J. and Martins, J. An adjoint-based derivative evaluation method for time-dependent aeroelastic optimization of flexible aircraft. AIAA Paper 2013-1530, 54th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Boston, USA, 2013.

[105] Yamaleev, N. K., Diskin, B., and Nielsen, E. J. Adjoint-based methodology for time-dependent optimization. AIAA Paper 2008-5857, 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Victoria, Canada, 2008.

[106] Yamaleev, N. K., Diskin, B., and Nielsen, E. J. Local-in-time adjoint-based method for design optimization of unsteady compressible flows. AIAA Paper, 2009-1169, 47th AIAA Aerospace Sciences Meeting and Exhibition, Orlando, USA, 2009.

[107] Thomas, P. D. and Lombard, C. K. Geometric Conservation Law and its application to flow computations on moving grids. *AIAA Journal*, 17(10):1030–1037, 1979.

[108] He, L. and Wang, D. X. Concurrent blade aerodynamic-aero-elastic design optimization using adjoint method. *Journal of Turbomachinery*, 133(1):011021, 2010.

[109] Valero, O., He, L., and Li, Y. S. Concurrent blade aerodynamic-aeroelastic design optimization with re-scaled response surface. ASME Paper GT2014-25854, ASME Turbo Expo, Düsseldorf, Germany, 2014.

[110] Valero, O., He, L., Wang, D., Li, Y., and Wells, R. Concurrent multi-bladerow aerodynamic-aeroelastic design optimization using adjoint approach. 14th International Symposium on Unsteady Aerodynamics, Aeroacoustics and Aeroelasticity of Turbomachines ISUAAAT14, Stockholm, Sweden, 2015.

[111] Ma, C., Su, X., and Yuan, X. Discrete adjoint solution of unsteady turbulent flow in compressor. ASME Paper GT2015-42948, ASME Turbo Expo, Montreal, Canada, 2015.

[112] Ma, C., Su, X., and Yuan, X. An efficient unsteady adjoint optimization system for multistage turbomachinery. *Journal of Turbomachinery*, 139(1):011003, 2016.

[113] Ma, C., Su, X., and Yuan, X. Adjoint-based unsteady aerodynamic optimization of a transonic turbine stage. ASME Paper GT2016-56885, ASME Turbo Expo, Seoul, South Korea, 2016.

[114] Engels-Putzka, A. and Frey, C. Adjoint harmonic balance method for forced response analysis in turbomachinery. VI International Conference on Coupled Problems in Science and Engineering, Venice, Italy, 2015.

[115] Yi, J. and Capone, L. Adjoint-based sensitivity analysis for unsteady bladerow interaction using space–time gradient method. *Journal of Turbomachinery*, 139(11):111008, 2017.

[116] Talnikar, C., Wang, Q., and Laskowski, G. M. Unsteady adjoint of pressure loss for a fundamental transonic turbine vane. *Journal of Turbomachinery*, 139(3):031001, 2016.

[117] Ntanakas, G. and Meyer, M. Towards unsteady adjoint analysis for turbomachinery applications. 6th European Conference on Computational Fluid Dynamics-ECFD VI, Barcelona, Spain, 2014.

[118] Ntanakas, G. and Meyer, M. The unsteady discrete adjoint method for turbomachinery applications. 14th International Symposium on Unsteady Aerodynamics, Aeroacoustics and Aeroelasticity of Turbomachines, Stockholm, Sweden, 2015.

[119] Ntanakas, G., Meyer, M., and Giannakoglou, K.C. Employing the time domain unsteady discrete adjoint method for shape optimization of 3D multi-row turbomachinery configurations. *Journal of Turbomachinery*, 140(8):081006, 2018.

[120] Griewank, A. and Walther, A. Algorithm 799: Revolve: an implementation of check-pointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software*, 26(1):19–45, 2000.

[121] Wang, Q., Moin, P., and Iaccarino, G. Minimal repetition dynamic checkpointing algorithm for unsteady adjoint calculation. *SIAM Journal on Scientific Computing*, 31(4):2549–2567, 2009.

[122] Nemili, A., Özkaya, E., Gauger, N. R., Carnarius, A., and Thiele, F. A discrete adjoint approach for unsteady optimal flow control. New Results in Numerical and Experimental Fluid Mechanics VIII. Notes on Numerical Fluid Mechanics and Multidisciplinary Design, vol 121. Springer, 2013.

[123] Torbert, S. and Löhner, R. check-pointing schemes for adjoint methods and strongly unsteady flow. AIAA Paper 2012-572, 50th AIAA Aerospace Sciences Meeting, Nashville, USA, 2012.

[124] Brand, M. Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra and its Applications*, 415(1):20–30, 2006.

[125] Vezyris, C., Kavvadias, I., Papoutsis-Kiachagias, E., and Giannakoglou, K. C. Unsteady continuous adjoint method using POD for jet-based flow control. EC-COMAS conference, Barcelona, Spain, 2014.

[126] Nimmagadda, S., Economon, T. D., Alonso, J. J., Silva, C., Zhou, B. Y., and Albring, T. Low-cost unsteady discrete adjoints for aeroacoustic optimization using temporal and spatial coarsening techniques. AIAA Paper 2018-1911, AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference. Kissimmee, USA, 2018.

[127] European Commission. AboutFlow Website. `http://aboutflow.sems.qmul.ac.uk/`, Online accessed: 23-May-2018.

[128] Lapworth, L. Hydra-CFD: a framework for collaborative CFD development. International Conference on Scientific and Engineering Computation , Singapore, 2004.

[129] Lapworth, L., Shahpar, S., and Radford, D. Simulation and design of gas turbine components. Second European Workshop on Automatic Differentiation, Swindon, UK, 2005.

[130] Beard, P. F., Smith, A. D., and Povey, T. Experimental and computational fluid dynamics investigation of the efficiency of an unshrouded transonic high-pressure turbine. *Proceedings of the Institution of Mechanical Engineers, Part A: Journal of Power and Energy*, 225(8):1166–1179, 2011.

[131] Papadimitriou, D. Adjoint formulations for the analysis and design of turbomachinery cascades and optimal grid adaptation using a posteriori error analysis. PhD thesis, Lab. of Thermal Turbomachines, NTUA, Athens, Greece, 2006. (in Greek).

[132] Asouti, V. Aerodynamic analysis and design methods at high and low speed flows, on multiprocessor platforms. PhD thesis, Lab. of Thermal Turbomachines, NTUA, Athens, Greece, 2009. (in Greek).

[133] Zymaris, A. Adjoint methods for the design of shapes with optimal aerodynamic performance in laminar and turbulent flows. PhD thesis, Lab. of Thermal Turbomachines, NTUA, Athens, Greece, 2010. (in Greek).

[134] Zervogiannis, T. Optimization methods in aerodynamics and turbomachinery based on the adjoint technique, hybrid grids and the exact hessian matrix. PhD thesis, Lab. of Thermal Turbomachines, NTUA, Athens, Greece, 2011. (in Greek).

[135] Papoutsis-Kiachagias, E. M. Adjoint methods for turbulent flows, applied to shape or topology optimisation and robust design. PhD thesis, Lab. of Thermal Turbomachines, NTUA, Athens, Greece, 2013.

[136] Kavvadias, I. Continuous adjoint methods for steady and unsteady turbulent flows with emphasis on the accuracy of sensitivity derivatives. PhD thesis, Lab. of Thermal Turbomachines, NTUA, Athens, Greece, 2016.

[137] Vezyris, C. OpenFOAM-based continuous adjoint method for aerodynamic optimization of unsteady turbulent flows. PhD thesis, Lab. of Thermal Turbomachines, NTUA, Athens, Greece. (in progress).

[138] OpenFOAM Website. `www.openfoam.com`, Online accessed: 1-Juni-2018.

[139] Spalart, P. R. and Allmaras, S. A one-equation turbulence model for aerodynamic flows. AIAA Paper, 30th Aerospace Sciences meeting and Exhibit, Reno, USA, 1992.

[140] White, F. M. *Viscous fluid flow*. McGraw-Hill series in mechanical engineering. New York, USA, 1991.

[141] Orszag, S. A. Analytical theories of turbulence. *Journal of Fluid Mechanics*, 41:363–386, 1970.

[142] Moin, P. and Mahesh, K. Direct Numerical Simulation: A tool in turbulence research. *Annual Review of Fluid Mechanics*, 30(1):539–578, 1998.

[143] Wagner, C., Shishkin, A., and Shishkina, O. The use of direct numerical simulations for solving industrial flow problems. In *Direct and Large-Eddy Simulation VIII*, pages 397–404. Springer Netherlands, 2011.

[144] Rai, M. A direct numerical simulation of stator-rotor interaction in an axial compressor. AIAA Paper 2010-6533, 46th AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit, Nashville, USA, 2010.

[145] Michelassi, V., Wissink, J. G., and Rodi, W. Direct Numerical Simulation, large eddy simulation and unsteady Reynolds-averaged Navier—Stokes simulations of periodic unsteady flow in a low-pressure turbine cascade: A comparison. *Journal of Power and Energy*, 217(4):403–411, 2003.

[146] Boussinesq, J. Theorie de l' ecoulement tourbillonant. *Comptes-Rendux de l' Academie des Sciences*, 23:46–50, 1877.

[147] Mohammadi, B. and Puigt, G. Wall functions in computational fluid mechanics. *Computers & Fluids*, 35(10):1108 – 1115, 2006.

[148] Spalding, D. B. A single formula for the "law of the wall". *Journal of Applied Mechanics*, 28(3):455–458, 1961.

[149] Roe, P. L. Approximate Riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, 135(2):250–258, 1997.

[150] Hirsch, C. Chapter 4 - the finite difference method for structured grids. In *Numerical Computation of Internal and External Flows)*, pages 145 – 201. Butterworth-Heinemann, Oxford, second edition, 2007.

[151] Jameson, A. Transonic airfoil calculations using the euler equations. In *Proceedings of Conference on Numerical Methods in Aeronautical Fluid Dynamics, London, UK*, pages 289–308, 1982.

[152] Jameson, A. and Mavriplis, D. Finite volume solution of the two-dimensional euler equations on a regular triangular mesh. *AIAA Journal*, 24(4):611–618, 1986.

[153] Crumpton, P. I. A cell vertex method for 3D Navier-Stokes solutions. Technical Report NA93/09, University of Oxford, UK, 1993.

[154] Crumpton, P. I., Moinier, P., and Giles, M. B. An unstructured algorithm for high Reynolds number flows on highly stretched grids. 10th International Conference Numerical Methods in Laminar and Turbulent Flow, Swansea, UK, 1997.

[155] Crumpton, P.I., Mackenzie, J.A., and Morton, K.W. Cell vertex algorithms for the compressible Navier-Stokes equations. *Journal of Computational Physics*, 109(1):1 – 15, 1993.

[156] Barth, T. and Jespersen, D. The design and application of upwind schemes on unstructured meshes. AIAA Paper 1989-0366, 27th Aerospace Sciences Meeting, Reno, USA, 1989.

[157] Ascher, U. M. and Petzold, L. R. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Society for Industrial and Applied Mathematics, Philadelphia, USA, 1998.

[158] Hills, N. Achieving high parallel performance for an unstructured unsteady turbomachinery CFD code. *The Aeronautical Journal*, 111(1117):185–193, 2007.

[159] Ganine, V., Amirante, D., and Hills, N. Enhancing performance and scalability of data transfer across sliding grid interfaces for time-accurate unsteady simulations of multistage turbomachinery flows. *Computers & Fluids*, 115:140–153, 2015.

[160] Amirante, D., Hills, N. J., and Barnes, C. J. A moving mesh algorithm for aero-thermo-mechanical modelling in turbomachinery. *International Journal for Numerical Methods in Fluids*, 70(9):1118–1138, 2012.

[161] Rogers, S. E., Suhs, N. E., and Dietz, W. E. PEGASUS 5: an automated preprocessor for overset-grid computational fluid dynamics. *AIAA Journal*, 41(6):1037–1045, 2003.

[162] Burgess, D. A., Crumpton, P. I., and Giles, M. B. A parallel framework for unstructured grid solvers. Programming environments for massively parallel distributed systems, Monte Verita, Switzerland, 1994.

[163] Crumpton, P. I. and Giles, M. B. Multigrid aircraft computations using the OPlus parallel library. *Parallel Computational Fluid Dynamics: Implementation and Results using Parallel Computers*, 95:339–346, 1996.

[164] Martinelli, L. *Calculations of viscous flows with a multigrid method*. PhD thesis, Princeton University, USA, 1987.

[165] Moinier, P. *Algorithm Developments for an Unstructured Viscous Flow Solver*. PhD thesis, University of Oxford, UK, 1999.

[166] Briggs, W. L., Henson, V. E., and McCormick, S. F. *A multigrid tutorial*. Society for Industrial and Applied Mathematics, 2000.

[167] Trottenberg, U., Oosterlee, C. W., and Schuller, A. *Multigrid*. Academic press, 2000.

[168] Wesseling, P. *An introduction to multigrid methods*. Pure and Applied Mathematics. John Wiley & Sons, 1992.

[169] Ruge, J. W. and Stüben, K. *Algebraic multigrid*, chapter 4, pages 73–130. Multigrid methods, Society for Industrial and Applied Mathematics, 1987.

[170] Stüben, K. A review of algebraic multigrid. *Journal of Computational and Applied Mathematics*, 128(1):281–309, 2001.

[171] Wesseling, P. and Oosterlee, C. W. Geometric multigrid with applications to computational fluid dynamics. *Journal of Computational and Applied Mathematics*, 128(1):311–334, 2001.

[172] Hülsemann, F., Kowarschik, M., Mohr, M., and Rüde, U. Parallel geometric multigrid. In *Numerical Solution of Partial Differential Equations on Parallel Computers*, pages 165–208. Springer, 2006.

[173] Milli, A. and Shahpar, S. PADRAM: Parametric design and rapid meshing system for complex turbomachinery configurations. ASME Paper GT2012-69030, ASME Turbo Expo. Copenhagen, Denmark, 2012.

[174] Shahpar, S. and Lapworth, L. PADRAM: Parametric design and rapid meshing system for turbomachinery optimisation. ASME Paper GT2003-38698, ASME Turbo Expo. Atlanta, USA, 2003.

[175] Giles, M. B. On the use of Runge-Kutta time-marching and multigrid for the solution of steady adjoint equations. Technical Report NA00/10, Computing Laboratory, Oxford University, UK, 2000.

[176] Griewank, A. and Walther, A. *Evaluating Derivatives*. Society for Industrial and Applied Mathematics, 2008.

[177] Hascoët, L. and Pascual, V. The Tapenade automatic differentiation tool: principles, model, and specification. *ACM Transactions Mathematical Software*, 39(3):1–43, 2013.

[178] Giles, M.B., Ghate, D., and Duta, M.C. Using automatic differentiation for adjoint CFD code development. Post-SAROD Indo-French Workshop on Recent Developments in Tools for Aerodynamics and Multidisciplinary Optimization. Bangalore, India, 2005.

[179] Taftaf, A., Hascoet, L., and Pascual, V.. Implementation and measurements of an efficient fixed point adjoint. International Conference on Evolutionary and Deterministic Methods for Design, Optimization and Control with Applications to Industrial and Societal Problems, EUROGEN, 2015.

[180] Akbarzadeh, S., Wang, Y., and Mueller, J. D. Fixed point discrete adjoint of SIMPLE-like solvers. AIAA Paper 2015-2750, 22nd AIAA Computational Fluid Dynamics Conference, Dallas, USA, 2015.

[181] Schanen, M., Naumann, U., Hascoët, L., and Utke, J. Interpretative adjoints for numerical simulation codes using MPI. *Procedia Computer Science*, 1(1):1825–1833, 2010.

[182] Towara, M., Schanen, M., and Naumann, U. MPI-parallel discrete adjoint openfoam. *Procedia Computer Science*, 51:19–28, 2015.

[183] Hascoët, L. and Taftaf, A. On the correct application of ad check-pointing to adjoint mpi-parallel programs. Technical Report RR-8864, Inria Sophia Antipolis, France, 2016.

[184] Meyer, C. D. *Matrix analysis and applied linear algebra*, pages 429–430. Society for Industrial and Applied Mathematics„ 2000.

[185] Freund, R. M. The steepest descent algorithm for unconstrained optimization and a bisection line-search method. *Massachusetts Institute of Technology, USA*, 2004.

[186] Arts, T. and Lambert de Rouvroit, M. Aero-thermal performance of a two-dimensional highly loaded transonic turbine nozzle guide vane: A test case for inviscid and viscous flow computations. *ASME Journal of Turbomachinery*, 114(1):147–154, 1992.

[187] Arts, T., Lambert de Rouvroit, M., and Rutherford, A.W. Aero-thermal investigation of a highly loaded transonic linear turbine guide vane cascade: A test case for inviscid and viscous flow computations. Technical Report TN174, Von Karman Institute for Fluid Dynamics, 1990.

[188] Collado-Morata, E., Gourdain, N., Duchaine, F., and Gicquel, L.Y.M. Effects of free-stream turbulence on high-pressure turbine blade heat transfer predicted by structured and unstructured LES. *International Journal of Heat and Mass Transfer*, 55(21):5754 – 5768, 2012.

[189] Miguel Segui, L., Gicquel, L.Y.M., Duchaine, F., and de Laborderie, J. LES of the LS89 cascade: influence of inflow turbulence on the flow predictions. 12th European Conference on Turbomachinery Fluid Dynamics and Thermodynamics, Stockholm, Sweden, 2017.

[190] AboutFlow Website. TU Berlin TurboLab Stator. `http://aboutflow.sems.qmul.ac.uk/events/munich2016/benchmark/testcase3`, Online accessed: 20-March-2018.

[191] Vasilopoulos, I., Flassig, P., and Meyer, M. CAD-based aerodynamic optimization of a compressor stator using conventional and adjoint-driven approaches. ASME Paper GT2017-63199, ASME Turbo Expo. Charlotte, USA, 2017.

[192] Schmitz, A., Aulich, M., Schönweitz, D., and Nicke, E. Novel performance prediction of a transonic 4.5 stage compressor. ASME Paper GT2012-69003, ASME Turbo Expo. Copenhagen, Denmark, 2012.

[193] Schönweitz, D., Voges, M., Goinis, G., Enders, G., and Johann, E. Experimental and numerical examinations of a transonic compressor-stage with casing treatment. ASME Paper GT2013-95550, ASME Turbo Expo. San Antonio, USA, 2013.

[194] Schönweitz, D., Goinis, G., and Voges, M. *Numerical and Experimental Investigations of the Near Wall Flow in a Multi-stage Axial-flow Compressor with Casing Treatment Using TRACE and Particle Image Velocimetry*. PhD thesis, Technische Universität Darmstadt, Germany, 2011.

[195] Hogan, R. J. Fast reverse-mode Automatic Differentiation using expression templates in C++. *ACM Transactions on Mathematical Software*, 40(4):26:1–26:16, 2014.

[196] Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018.

[197] Homescu, C. Adjoints and automatic (algorithmic) differentiation in computational finance. Technical Report 1107.1831, arXiv.org, 2011.

[198] Naumann, U. Optimal Jacobian accumulation is NP-complete. *Mathematical Programming*, 112(2):427–441, 2008.

[199] Utke, J., Naumann, U., Fagan, M., Tallent, N., Strout, M., Heimbach, P., Hill, C., and Wunsch, C. OpenAD/F: A modular, open-source tool for automatic differentiation of Fortran codes. *ACM Transactions on Mathematical Software*, 34(4):18:1–18:36, 2008.

[200] Carle, A. and Fagan, M. ADIFOR 3.0 overview. Technical Report CAAM-TR-00-02, Department of Computational and Applied Mathematics, Rice University, 2000.

[201] Forth, S. An efficient overloaded implementation of forward mode automatic differentiation in MATLAB. *ACM Transactions on Mathematical Software*, 32(2):195–222, 2006.

[202] Griewank, A., Juedes, D., and Utke, J. ADOL-C: A package for the automatic differentiation of algorithms written in C/C++. *ACM Transactions on Mathematical Software*, 22(2):131–167, 1996.

[203] Walther, A. and Griewank, A. Getting started with ADOL-C. In U. Naumann und O. Schenk, Combinatorial Scientific Computing, Chapman-Hall CRC Computational Science, pp. 181–202, 2012.

[204] Lotz, J., Leppkes, K., and Naumann, U. dco/c++ - Derivative code by overloading in C++. Technical Report AIB-2011-06, RWTH Aachen, Department of Computer Science, Aachen, Germany, 2011.

[205] Autodiff website. http://www.autodiff.org/, Online accessed: 12-April-2018.