Efficient Adjoint-Based Design Capability for Unsteady Conjugate Heat Transfer Problems

Thesis approved by
the Department of Computer Science
University of Kaiserslautern-Landau
for the award of the Doctoral Degree
Doctor of Engineering (Dr.-Ing.)

to

Tobias Kattmann

Date of Defense: 13.09.2024

Dean	Prof. Dr. Christoph Garth
Reviewer	Prof. Dr. Nicolas R. Gauger
Reviewer	Dr. Nijso Beishuizen

DE-386

Abstract

Shape optimization using gradients computed via the adjoint method has become a common feature and also, practically, a requirement among the major computational fluid dynamics (CFD) and multi-physics solvers. Accurate sensitivities are widespread available for single-zone steady-state problems, with unsteady or multi-physics adjoint solvers being less common. Yet, gradient availability is a desirable feature for all primal simulation capabilities. Many practical flows of industrial interest are unsteady in nature, and for the simulation of heating/cooling devices the coupling between a fluid and solid domain is essential to accurately capture the system's behavior.

Combining these two general observations of the demand for sensitivities and unsteady conjugate heat transfer applications, is the objective of the present work. Consequently, this dissertation presents the development and application of an unsteady discrete adjoint solver for conjugate heat transfer (CHT).

Based on the framework available in the open-source multi-physics and design solver SU2, an efficient method to compute shape sensitivities via the discrete adjoint method for transient CHT problems is presented. The handling of the algorithmic differentiation (AD) tool together with the orchestration of the numerous time-steps and various involved zones is crucial, to retain a code that is efficient with respect to memory and compute time. Therefore, a modular approach for the evaluation of the various derivative terms (diagonal, cross, dual-time-step) is used.

Beyond design for time-accurate simulations, capabilities for steady state optimization of repeating heat exchanger geometries, as pin-fin arrays, are developed.

For both tested CHT configurations (one unsteady and one steady), the gradient validation shows excellent accuracy against finite differences and successful constrained optimization highlights the practical applicability in a design chain.

The current development is not presented in isolation, but as a part of the SU2-project, for which detailed information is provided in the appropriate context.

Zusammenfassung

Die gradientenbasierte Formoptimierung mittels adjungierter Methoden, ist ein gängiges Feature und praktisch auch eine Anforderung an die gängigen Computational Fluid Dynamics (CFD) und Multiphysik-Löser geworden. Präzise Sensitivitäten sind weit verbreitet für stationäre Probleme mit einer einzelnen physikalischen Zone, während adjungierte Löser für instationäre oder multiphysik Probleme deutlich seltener vorhanden sind. Im Allgemeinen ist die Verfügbarkeit von Gradienten ein angestrebtes Feature für alle primären Simulationsmethoden. Viele industrierelevante Strömungen sind von Natur aus instationär, und für die Simulation von Heiz-/Kühlsystemen ist die Kopplung zwischen einer Strömungs- und einer Wärmeleitungsdomäne (CHT) unerlässlich, um das Verhalten des Systems simulieren zu können.

Das Ziel der vorliegenden Arbeit ist es, diese beiden allgemeinen Beobachtungen zu kombinieren: den Bedarf an Sensitivitäten und Anwendungen im Bereich des instationären, konjugierten Wärmetransports. Dementsprechend wird in dieser Dissertation die Entwicklung und Anwendung eines instationären, diskreten adjungierten Lösers für konjugierten Wärmetransport vorgestellt. Basierend auf dem Framework des Open-Source Multi-Physik und Design Lösers SU2 wurde eine effiziente Methode zur Berechnung der genannten Sensitivitäten entwickelt. Um einen effizienten Code bezüglich Speicher und Rechenzeit zu erhalten, ist die Handhabung des Tools zum Algorithmisches Differenzieren (AD) zusammen mit der Orchestrierung der Zeitschritte und der verschiedenen involvierten physikalischen Zonen entscheidend. Daher wird ein modularer Ansatz für die Auswertung der auftretenden Ableitungsterme verwendet.

Über das Design für instationäre Simulationen hinaus, wurden Methoden für die stationäre Optimierung von sich-wiederholenden Geometrien für Wärmetauscher entwickelt, wie zum Beispiel Pin-basierte Kühlkörper. Für beide simulierten CHT-Testfälle, stationär und instationär, zeigt die Gradientenvalidierung zwischen adjungierter und Finite-Differenzen Methode eine sehr gute Genauigkeit und die Optimierung unter Nebenbedingungen belegt die praktische Anwendbarkeit in einer automatisierten Formoptimierung.

Die in dieser Arbeit behandelten Methoden wurden nicht isoliert entwickelt, sondern sind als Teil des internationalen SU2-Projekts der Allgemeinheit zugänglich.

Acknowledgments

I would like to thank my supervisor Prof. Dr. Nicolas Gauger for guiding me through the PhD, while leaving the freedom to explore different problem-solving approaches and topics.

I thank Dr. Nijso Beishuizen not only for reviewing this work, but also for his support in various stages throughout this PhD time.

I thank Prof. Dr. Didier Stricker for his work as the head of my PhD committee.

The company Bosch created this opportunity in the first place, and I thank all the great colleagues I was able to work with, especially Dr. Peter Wassermann and Dr. Manuel Schmidt.

The SU2-community not only provided the platform for this research, but invaluable support and guidance. Specifically, I need to greatly thank (in chronological order of appearance), Dr. Thomas Economon, Tim Albring, Dr. Ole Burghardt, Dr. Pedro Gomes and Dr. Charanya Venkatesan-Crome. I thoroughly enjoyed being a part of this open-source community.

I am incredibly grateful to have been accompanied on this whole PhD journey by Dr. Jonas Holley. Your ability to understand and articulate the beauty in seemingly insignificant moments, as well as to recognize the privilege of the situation even when I could not, has been invaluable.

Finally, I want to thank my family for their unconditional love and support.

Thanks/Danke.
Tobias Kattmann

Contents

Abstract iii

Zusammenfassung v

Acknowledgements vii

Contents x

Nomenclature xi

1 Introduction 1
- 1.1 Motivation 1
- 1.2 State of the Art 3
- 1.3 Contributions of this Thesis 5
- 1.4 Thesis Layout 6

2 Open Source SU2 framework 9
- 2.1 The SU2 project 9
  - 2.1.1 Codebase 10
  - 2.1.2 Development Platform 15
  - 2.1.3 Community & Foundation 17
  - 2.1.4 Documentation 20
- 2.2 Physical and numerical modeling 21
  - 2.2.1 Unsteady Incompressible Solver 21
  - 2.2.2 Turbulence Modeling 30
  - 2.2.3 Solid Heat Conduction 32
  - 2.2.4 Conjugate Heat Transfer 33

3 Development of an efficient adjoint-based design capability 35
- 3.1 Modeling streamwise periodic flow 35
  - 3.1.1 Pressure Periodicity through the Momentum Equations 37
  - 3.1.2 Temperature Periodicity through the Energy Equation 39

3.1.3 Alternative Treatment in coupled CHT Applications 43
3.1.4 Implications on RANS Turbulence Modeling 44
3.2 Discrete Adjoint Method 45
3.2.1 Fundamentals of Algorithmic Differentiation 45
3.2.2 Derivation of Discrete Adjoint Equations 53
3.2.3 Projecting Sensitivities onto Design Variables 65
3.2.4 Handling Additional Fixed-Point Solutions 68
3.2.5 Memory-efficient implementation 69

4 Implementation, Sustainability and Maintainability Aspect 73
4.1 Contributed code 73
4.2 Efforts ensuring Maintainability 75
4.3 Result Reproducibility 80

5 Verification and Application 83
5.1 Steady coupled CHT Staggered Pin Array 83
5.1.1 Validation of the Streamwise Periodic Approach 87
5.1.2 Mesh Deformation Routine 90
5.1.3 Gradient Validation 92
5.1.4 Sensitivities with Additional Fixed-Point Equations 96
5.1.5 Constrained Shape-Optimization Results 97
5.2 Transient Coupled CHT Cylinder in Crossflow 103
5.2.1 Gradient Validation 107
5.2.2 Challenges in Objective Function Definition 111
5.2.3 Optimization Results 113

6 Conclusion and Outlook 119
6.1 Conclusion 119
6.2 Outlook 121

Bibliography 123

Curriculum Vitae 131

Nomenclature

This nomenclature separately lists Latin and Greek symbols, sub- and superscripts and abbreviations. Some entries serve multiple meanings in different contexts, which are all listed separately. Note that the **Unit**-column is left empty when no unique unit can be determined, and the entry $[\,-\,]$ signaling a dimensionless quantity. Few symbols are not listed in this nomenclature for the sake of brevity and are introduced without exception in the specific context.

Latin symbols

Symbol	**Unit**	**Description**
c_p	$[\,J/(kg \cdot K)\,]$	specific heat capacity at constant pressure
D	$[\,m\,]$	cylinder diameter in the unsteady testcase
f	$[\,1/s\,]$	(shedding) frequency
$\bar{F}^c$		convective fluxes
$\bar{F}^v$		viscous fluxes
G		fixed point iterator
h	$[\,W/(m^2 \cdot K)\,]$	heat transfer coefficient
$\bar{\bar{I}}$	$[\,-\,]$	identity matrix
J		scalar objective function
k	$[\,J/kg\,]$	turbulence kinetic energy
L		Lagrangian
$\dot{m}$	$[\,kg/s\,]$	mass flow rate
$\bar{n}$	$[\,-\,]$	outward facing unit normal vector
$\bar{n}_A$	$[\,-\,]$	outward facing area normal vector, i.e. 2 norm of vector is Area
p	$[\,Pa\,]$	physical pressure (or recovered pressure)
$\tilde{p}$	$[\,Pa\,]$	reduced/periodic pressure
p_o	$[\,Pa\,]$	operating pressure
P_Δ	$[\,Pa\,]$	pressure difference between periodic surfaces

Symbol	**Unit**	**Description**
Pr_{lam}	$[-]$	laminar Prandtl number
Pr_t	$[-]$	turbulent Prandtl number
$\dot{q}$	$[W/m^2]$	heatflux
Q		arbitrary source term
$\dot{Q}$	$[W]$	rate of heat flow
R	$[J/(mol \cdot kg)]$	universal gas constant
$\mathcal{R}$		primal residual operator
Re	$[-]$	Reynolds number
S		generic source term
St	$[-]$	Strouhal number
S_Δ	$[-]$	surface mesh deformation
t	$[s]$	physical time
$\bar{t}$	$[m]$	translation vector between periodic surfaces
$\bar{t}$	$[m]$	tangential vector to unit normal
t_p	$[m]$	distance to domain point from reference node in direction of the translation vector
T	$[K]$	physical temperature (or recovered temperature)
T	$[s]$	period length
$\tilde{T}$	$[K]$	reduced temperature
T_Δ	$[K]$	temperature difference between periodic surfaces
U		general-working/conservative variables
$\bar{v}$	$[m/s]$	velocity vector with components $(u, v, w)^\mathsf{T}$
V		primitive variables
$\bar{x}$	$[m]$	position vector of a point in the domain
$\bar{x}^*$	$[m]$	position vector of a reference point on the periodic surface which serves as inlet
X	$[m]$	vector of mesh node coordinates
X_Δ	$[m]$	update to the vector of mesh node coordinates
y^+	$[-]$	dimensionless wall distance

Greek symbols

Symbol	Unit	Description
α	$[\,m\,]$	design variables, FFD-box point displacements
β	$[\,-\,]$	pseudo speed of sound
δ_{ij}	$[\,-\,]$	Kronecker delta
ϵ	$[\,-\,]$	user-provided factor to pseudo speed of sound
κ	$[\,W/(m\cdot K)\,]$	thermal conductivity
λ		adjoint variables
μ	$[\,Pa\cdot s\,]$	dynamic viscosity
μ_{lam}	$[\,Pa\cdot s\,]$	laminar dynamic viscosity
μ_t	$[\,Pa\cdot s\,]$	turbulent dynamic viscosity
ρ	$[\,kg/m^3\,]$	density
τ	$[\,Pa\,]$	shear stress tensor in the Navier-Stokes equations
τ	$[\,s\,]$	pseudo time variable
ϕ	$[\,-\,]$	relaxation factor
ω	$[\,1/s\,]$	turbulence specific dissipation rate
Ω	$[\,m^3\,]$	computing domain
Ω_i	$[\,m^3\,]$	volume of mesh element

Sub- and superscripts

Symbol	Description
$\cdot_f$	fluid domain
$\cdot_i$	quantitiy associated with node i
$\cdot_j$	quantitiy associated with node j
$\cdot_{ij}$	quantitiy associated with edge connecting node i and j
$\cdot_m$	quantitiy associated with zone m
$\cdot^n$	quantitiy associated with timestep n
$\cdot_p$	quantitiy associated with pseudo-timestep p
$\cdot_s$	solid domain
$\lVert\cdot\rVert_2$	Euclidean 2 norm

Abbreviations

AD	Automatic Differentiation
CFD	Computational Fluid Dynamics
CHT	Conjugate Heat Transfer
CS	Computer Science
DA	Discrete Adjoint
(D)DES	(Delayed) Detached Eddy Simulation
DNS	Direct Numerical Simulation
Eq., Eqs.	equation, equations
FD	Finite Difference
FDS	Flux Difference Splitting
FFD	Free Form Deformation
GNU	recursive acronym for "GNU's Not Unix!"
IDE	Integrated Development Environment
LES	Large Eddy Simulation
(L)GPL	(Lesser) General Public License
PDE	Partial Differential Equation
PR	Pull Request
RANS	Reynolds Averaged Navier Stokes
RHRK	Regionales HochschulRechenzentrum Kaiserslautern
SA	Spallart-Allmaras
SST	Shear-Stress-Transport
SU2	Stanford University Unstructured
TU	Technical University
URANS	Unsteady Reynolds Averaged Navier Stokes

1 Introduction

1.1 Motivation

Numerical simulations are an indispensable tool for research and development in the field of Computational Fluid Dynamics (CFD) and are widely used across multiple industries such as aerospace, automotive, medical technology and many more. Optimizing the product design/shape with respect to specific objectives in the early stage of virtual prototyping can greatly improve the product's performance and thus fulfill regulators and market demands.

In order to determine optimized designs in a vast design space, gradient-based optimization methods are a valuable tool and this work focuses on efficiently providing accurate gradients for such methods. As the general optimization problem considered in this work is not assumed to be convex, gradient-based methods would have to be combined or replaced by suitable methods for global optimization. Gradient-based methods are suitable to find a local optimum efficiently, but generally are not able to find other, potentially more optimal points. Sampling the design space is a suitable approach to find the global optimum, and by employing a surrogate model, the choice of succeeding sampling points can be enhanced. A comprehensive discussion of gradient-free methods and surrogate modeling can be found in [MN21], chapter 7 and chapter 10, respectively.

As the name suggests, gradient-based optimization methods require the ability to determine the gradient of an objective function with respect to the designated set of design variables. The straight-forward approach of computing these sensitivities using finite differences scales linearly in computational time with the number of design variables (i.e. inputs), which quickly becomes computationally infeasible. Especially in an optimization cycle where repeated gradient computation with respect to the most recent design is required. Consider a three-dimensional free form deformation (FFD) box that encloses the to be deformed surface with 10 control points along each edge, which results in 3000 design variables, as each control point can be deformed in each Cartesian coordinate direction. And this example potentially only offers reduced design freedom, with higher FFD resolutions or other methods requiring even more design variables With a single primal evaluation in the realm of CFD often lasting a number of minutes up to multiple days on modern HPC hardware, the finite difference approach can be rendered computationally infeasible for practical use in industry.

A suitable alternative to compute the necessary gradients are adjoint methods, where the computational cost scales with the number of outputs (i.e. objectives) and not the number of inputs, which can be huge for geometry parametrizations. Adjoint methods can be generally separated in continuous and discrete adjoint methods. Independent on the type of adjoint method, the compute cost for an adjoint evaluation associated with one objective function is roughly considered to be of the same order of magnitude as one primal evaluation. This makes it the method of choice for gradient-based shape optimization approaches, where the amount of design variables almost always greatly exceeds the objective functions.

In the continuous adjoint approach, the adjoint model is derived directly from the governing equations and then discretized to be solved. Not only is this a meticulous and cumbersome task that needs to be repeated every time altering equations are considered, but one also encounters equations that are not analytically differentiable, e.g. some turbulence models.

The second adjoint method to compute the required gradients with adequate accuracy is the discrete adjoint (DA) method, where the adjoint model is derived based on the discretized partial differential equations (PDE's), i.e. the source code of the numerical solver. This process, when automatized by adequately employing algorithmic differentiation (AD), results in an efficient algorithm in compute time and memory, enabling the computation of sensitivities required for optimizing complex problems.

In Conjugate Heat Transfer (CHT), the fluid zone is coupled with heat conduction in solid zones, which is crucial to accurately simulate cooling/heating applications. The CHT application considered in this work are pin-fin heat exchangers, where tens to thousands of pins are arranged in a pin array to dissipate heat generated by e.g. power electronics, into a cooling liquid that flows through the array. With advancing electrification and simultaneous miniaturization in especially the automotive field, the need for customized high-performance cooling is of ever-growing importance. Residential heating is another wide-spread example where often natural gas boilers, which potentially are replaced by hydrogen boilers in the future, have to efficiently exchange heat between a high-temperature air stream and a cooler water stream, while simultaneously respecting the regulator's emission requirements.

Due to the high computational cost, these CHT simulations and optimizations are often performed in a steady-state manner, i.e. the physical process is assumed to be time-independent. Employing unsteady Reynolds averaged Navier-Stokes (URANS) simulations represents a suitable middle ground between prohibitively expensive scale-resolving (e.g. (D)DES, LES, DNS) and steady-state simulations to model large-scale transient effects that are neglected in steady-state simulations.

The available code framework SU2, which this thesis builds upon, is a key enabler for obtaining industrially relevant results. SU2 is a global software project with a community including multiple institutions in academia and industry, working with industry-standard collaborative tools. The code is able to solve multiphysics problems and is written in hybrid-parallel (MPI+OpenMP), and thus features the necessary requirements to make use of modern high-performance computing infrastructure.

1.2 State of the Art

This section explores the research status of the fields that are associated to this thesis. It thus constitutes the foundation for the subsequent section 1.3, which covers the contributions to the research fields made by this work. The state of the art is chronologically structured based on the contents of this thesis. Note that throughout that main body of this thesis, extended explanations on the research status are given where appropriate.

The open-source computer code SU2 is used as the research platform and extended with own implementations. The SU2 code was initially developed in Stanford by Palacios et al. [Pal+13; Pal+14], hence the acronym SU2 which stands for Stanford University Unstructured. At the early stage in the SU2 project, the focus in sensitivity computation was solely on the continuous adjoint method, with applications ranging from hypersonic inlet design [KA17], over adjoint-based goal-oriented mesh adaptation [Cop+] to optimization of a pitching wing in turbulent flow using URANS [Eco14]. The last-mentioned contribution constitutes an early application of adjoint methods to unsteady CFD phenomena in the context of a widely available multi-purpose implementation. It also features exclusively forced motions by a user-defined pitching or rotation frequency, which simplifies the objective function definition to an a priori known and fixed time interval.

The book of Blazek [Bla15] provides an excellent resource for a general introduction to a multitude of aspects concerning CFD codes, complementing the research literature written by the various authors in the SU2 community. The choice to utilize SU2 as a code framework for sensitivity computation is not arbitrary, but rather because SU2 was written with sensitivity computation in mind and not an afterthought. The most notable alternative open-source code framework is OpenFOAM [Wel+98; OI01], which also offers adjoint-based design capabilities [STN14; Tow18].

The flow equations considered in this thesis are the incompressible Navier-Stokes equations in both, laminar and turbulent regime. An implementation in SU2 for steady state flows was contributed by Economon [Eco20], which in turn was based on earlier work by Weiss and Smith [WS95; WMS97].

Handling of fully developed flow via additional source terms in the momentum and energy equation, that arise from a decomposition of pressure and temperature in a truly periodic and linear component, was pioneered by Patankar [PLS77]. In addition to the original paper, a discussion of the implementation for finite volume codes was done by Beale [Bea05; Bea06], who also presents an early application to an offset-fin plate-fin heat exchanger.

As mentioned above, the continuous adjoint initially was the predominant method to compute sensitivities within SU2. The application of the adjoint method to fluid mechanics was pioneered by Giles [GP97; GP00] and Jameson [Jam03]. In this thesis, solely the discrete adjoint method, constructed via algorithmic differentiation, is employed. The initial implementation in SU2 was contributed by Albring et al. [ASG15; ASG16], who applied the operator overloading AD-tool CoDiPack developed by Sagebaum et al. [SAG19]. Early fundamental research on the discrete adjoint method based on algorithmic differentiation applied to attractive fixed points was conducted by Christianson [Chr94].

A framework for discrete adjoint computations handling unsteady flows was developed by Zhou et al. [Zho+16] who later added aeroacoustic capabilities [Zho+17] based on the Ffowcs Williams-Hawkings (FW-H) method. Specialized work on reducing the computational cost in wall time and memory for the unsteady discrete adjoint comprises spatial and temporal coarsening techniques by Nimmagadda et al. [Nim+18], investigations on time-averaging steady vs. full unsteady adjoint by Hückelheim [Hue+15] or harmonic balance by Rubino et al. [Rub+18], to name a selection.

With the release of primal multi-physics solvers (FSI, CHT) in SU2, the development of adjoint-based design capabilities for these coupled problems naturally followed. A discrete adjoint implementation for FSI problems was presented by Sanchez [San17]. A generalized multi-physics discrete adjoint solver was presented by Burghardt [BGE19; Bur+22], which independently treats solvers in separate zones which are coupled via communication over arbitrary interfaces. The framework offers the flexibility to accurately compute sensitivities for combined steady state CHT and FSI cases, as presented in [Bur+22] at the example of a cooled turbine under an aerodynamic load. A discrete adjoint solver for CHT applications based on a black-box approach was also shown in

OpenFOAM by Gkaragkounis et al. [GPG19].

For FSI, the multi-physics discrete adjoint was extended to unsteady flows with a limitation to URANS cases by Crome [Ven20]. Applications range from forced motion of rotating or plunging airfoils to vortex shedding behind stationary objects as cylinders or airfoils. The current thesis extends the applicability to unsteady CHT cases, building on the existing generalized framework.

With regard to the primary application considered in this thesis, pin-design in heat-exchangers, a broad discussion exists in literature. From work exploring general design primitives as staggered or in-line pin formations [Pat+18; Kha04], over various pin-shape designs (round vs. elliptical vs. drop-shaped, straight vs. tapered, etc.) in [CB16], towards adjoint-optimization on a steady state fluid domain in [Vid20].

1.3 Contributions of this Thesis

This section first defines two major research goals and subsequently discusses the contributions made by this thesis towards their achievement. Motivated by the previous sections the two research goals in a brief summary are:

- Providing accurate gradients for unsteady CHT simulations via a computationally efficient discrete adjoint method, and thus enabling shape optimization.
- Identification and realization of suitable methods to efficiently evaluate a pin-shape's performance in an extended pin-array, as well as evaluation of validated gradients for a shape optimization workflow.

An indispensable initial groundwork in the first research goal for unsteady CHT simulations in the context of the here considered heat exchanger applications, is the involvement in the implementation of the unsteady incompressible Navier-Stokes solver. Beyond the validation of the single-zone primal implementation, the validity of the sensitivities computed via the discrete adjoint method was ensured. With respect to the transition to multi-physics systems, the author enabled an operational unsteady CHT discrete adjoint solver. The implementation yields very accurate sensitivities compared to finite differences, including a complete shape design chain with geometry parameterization based on FFD boxes and a subsequent volume mesh deformation. This proves the consistency of the discrete adjoint gradient to the exact discrete primal problem formulation. Within the bounds of an ordinary application case of a hollowed cylinder in crossflow, the framework showcased its robustness over a complete optimization cycle including a secondary constraint function and simple geometry constraints. This thesis contributes a suitable approach, including the derivation thereof, to compute unsteady CHT gradients and their practical usability in a design chain. To the best of the author's knowledge, this constitutes the first unsteady discrete adjoint method in the field of coupled CHT simulations, that yields high-precision sensitivities.

The second research goal is principally concerned with steady state simulations, but in its general findings equally applicable to time-accurate simulations. The concept of streamwise periodic flow was implemented, in order to perform simulations on an elementary unit cell that is representative for a fully extended array. This naturally involves an approximation to the concrete application, but enables the performance evaluation of a pin-shape in computationally feasible cost, and thus wall time and memory. Notable extended features are the possibility of massflow prescription, inclusion of eddy viscosity into the derivation and applicability to CHT simulations by replacing the energy equation's volume heat sink by a periodic outlet heat sink. In order to achieve an accurate gradient validation for streamwise periodic simulations with prescribed massflow, the occurring fixed point iteration in the applied pressure drop needs to be treated equivalently to the solution variables. The identification of this circumstance and subsequent specialized treatment in the implementation constitute a limited but crucial contribution to the applicability to the here considered application. With the fully validated gradient, the consequent progression towards free-form shape optimization using FFD-boxes was made to highlight the stability in a fully automated design chain. For this purpose, the periodicity-preserving geometry and mesh deformation setup to retain a single pin-shape for the whole array in the simulation presents a sophisticated use-case for otherwise limited usability of FFD-boxes.

All mentioned implementations and contributions are fully implemented and openly available in the main version of the open-source PDE solver SU2. Within the scope of the listed main contributions, numerous less integral code modifications and especially corrections were made by the author. The involved rigorous orientation towards sustainable and maintainable code allows and ensures the practical usability and modifiability of the contributions by a broader scientific audience.

1.4 Thesis Layout

Following this introduction which motivated the conducted research, presented the state of the art and highlighted the contributions to the research field made by this thesis, the main body provides detailed insights and explanation to the relevant contents. The second chapter 2 first covers an overview of the open-source SU2 project in its entirety, focusing on a broad description including many essential non-technical aspects. In the second section of chapter 2, the majority of primal modeling required for the simulations conducted in this thesis is presented. The development of an efficient adjoint-based design capability is the subject of chapter 3. Therefore, the concept of streamwise periodic flow with the implemented extensions is detailed, before covering the discrete adjoint method constructed via AD. Considerations with respect to implementation, sustainability and maintainability of the code development process are presented in chapter 4. A particular focus is put on the reproducibility of all results shown in this thesis. The developed methods from the previous chapters are applied to selected testcases in chapter 5. Validations of the gradient accuracy are conducted for the developed methods. Two CHT applications are featured in chapter 5, first a steady-state case on an elementary unit cell of a pin-fin heat exchanger, and second, a time accurate simulation of a cylinder in crossflow. The thesis closes with conclusions and outlook in chapter 6.

2 Open Source SU2 framework

The first Sec. 2.1 of this chapter gives an overview of the SU2 project in its entirety, covering general information about the codebase, general development principles, community organization and various other topics. In the second Sec. 2.2 the primal solution procedure of the time-accurate incompressible Navier-Stokes solver and the extension to coupled CHT simulation is presented.

2.1 The SU2 project

First and foremost, SU2 represents a piece of software solidifying itself in a codebase, which will be explored in Sec. 2.1.1 in various aspects including file structure, license, build system, version control and testing framework. But the creation of said codebase is and was highly influenced, and even made possible in parts, by other factors discussed in this section. These include the employed development platform in Sec. 2.1.2, the role of the community and foundation in Sec. 2.1.3, as well as documentation aspects in Sec. 2.1.4. Figure 2.1 presents a mind-map featuring the most relevant keywords in the context of SU2, following this section's structure.

The code is predominantly used as a research platform for academia and industry by providing state-of-the-art numerical methods in a hybrid-parallel code with an extensive testcase library. This alleviates the need to write from scratch, thoroughly test and to maintain standard solver features within each organization or even for each individual PhD student. This dramatically reduces the time necessary for a project to present first results as the general framework including parallel file-IO, linear solver infrastructure, domain decomposition, etc. can be taken off-shelf.

Above all, SU2 distinguishes itself from other available Open-Source CFD and multiphysics codes with the straight-forward availability of sensitivities via the discrete adjoint method constructed with AD. The sensitivities determined by the discrete adjoint method are predominantly used for shape optimization in the context of SU2, but general applications in the field of topology optimization [GP20], optimal control [Car+13] or uncertainty quantification [Wan09] are possible as well.

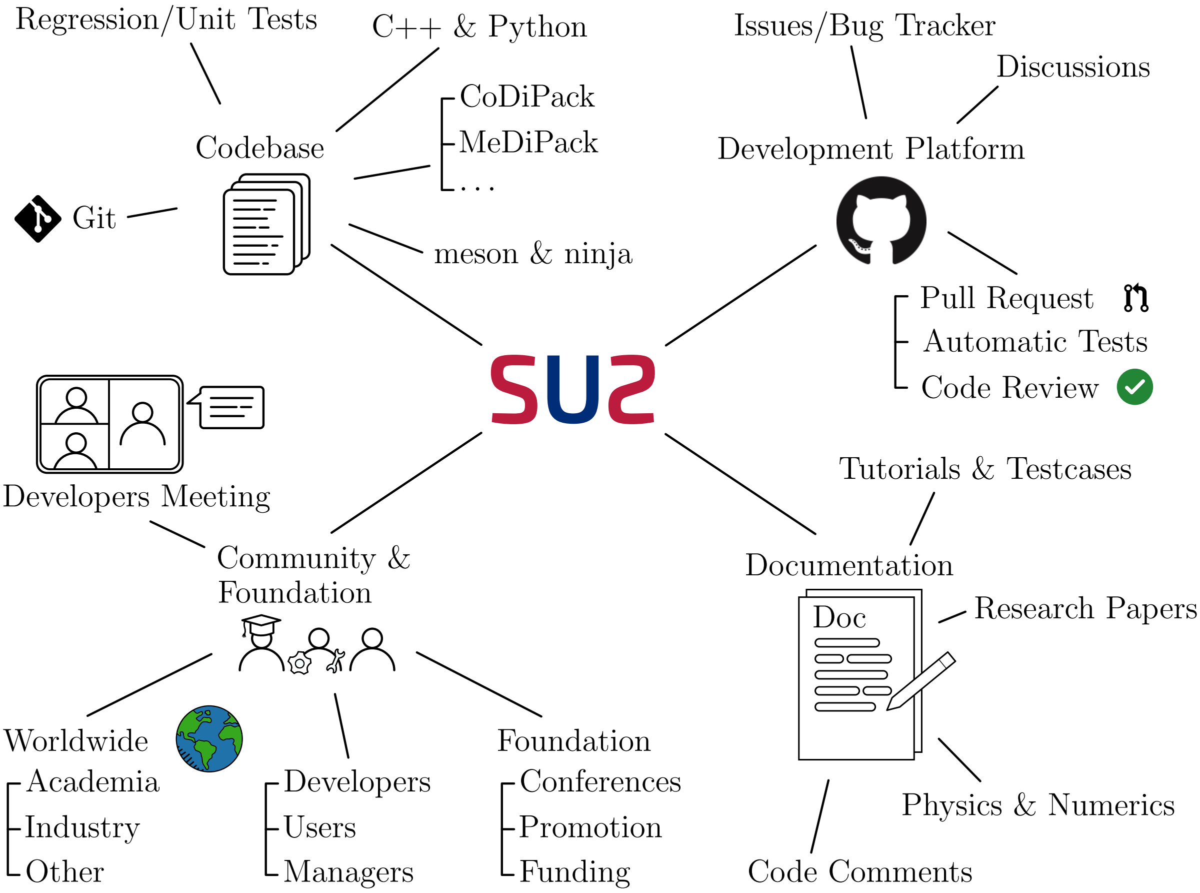


Figure 2.1: Mind-map of the SU2 project, representing the four parts of this section.

2.1.1 Codebase

SU2 is an open-source collection of software tools written in C++ and Python for the analysis of partial differential equations (PDE's) and PDE-constrained optimization problems on unstructured meshes with state-of-the-art numerical methods. The primary applications are computational fluid dynamics and aerodynamic shape optimization, but has been extended to treat a great variety of equations such as structural dynamics, heat equation and species transport.

The general-purpose programming language C++ offers considerable performance benefits and a large, active community, such that many performance critical libraries are optimized for most frequently used hardware. Especially with respect to scalability over multiple cores and nodes in a high-performance computing system, C++ is a popular choice, partly due to the extensive experience of the high-performance community. On the downside C++ requires detailed knowledge of the language and also sometimes the specific hardware to produce fast, correct and stable code. This leads to extended training times for new, inexperienced developers. For many not performance critical tasks that wrap the main solver in e.g. an optimization loop, the python programming language is used. It is more straight-forward to learn, especially compared to C++ as it simplifies certain concepts e.g. does not require strict type specification for variables or automated garbage collection. These simplifications are of course paid for with decreased performance, but Python makes up for this by extensive, well-documented packages for numerical computation, data-processing, visualization and many more. This allows for quick implementation and automation of otherwise cumbersome tasks. This thesis for example, successfully uses the SLSQP implementation of the SciPy python package off-shelf for the optimization results presented in Sec. 5.1.5 and Sec. 5.2.3. Ultimately, it is the developer's task to carefully weigh the benefits and disadvantages of the potential programming languages and their infrastructure for the specific task at hand.

SU2 consists, as nearly all larger software projects, of the code written specifically for the project and included libraries as well as other software packages that take over certain specific tasks. Using well-established libraries often greatly reduces development effort by not having to write, test and maintain the code. Examples in SU2 include the widely used domain decomposition tool ParMETIS [KSK97] or the C++ unit testing framework Catch2[1].

The compiled code features multiple modules, i.e. different binaries, that serve special purposes. For example, the binary `SU2_CFD` evokes the primal solution capabilities while `SU2_DEF` is used for all mesh deformation related aspects. The modules share a structure of header and implementation files, e.g. the functionalities to read and interpret a configuration file are used by most modules while the specific code for source term residual contributions is merely used by the main solver module. As the file- and class structure is subject to constant change, a detailed layout here would only represent one snapshot in time.

The most recent code structure can be easily explored by compiling the Doxygen[2] documentation to the available codebase as it provides a class hierarchy overview, extracts function prototype explanation, lists class variables and much more in a developer-friendly fashion. In order to create such documentation, the code must contain specially formatted comments that Doxygen can extract. Largely, the code follows a one-class-per-file policy, but this rule is disregarded for certain exceptions.

For a project with the size of SU2, the author heavily recommends the usage of an Integrated Development Environment (IDE) that provides proper syntax highlighting,

---

[1] `https://github.com/catchorg/Catch2` (Accessed 2022-08-15)

[2] `https://www.doxygen.nl` Doxygen is a documentation generation tool for source trees, that generates .html, .pdf, etc. files containing the output.

code completion, visual debugging, etc. IDE's like Visual-Studio(-Code), Qt-creator or Eclipse to name a few, support the developer to achieve his tasks faster and with fewer mistakes.

License

The code of SU2 is distributed under the GNU Lesser General Public License (LGPL) in the version 2.1. The LGPL is a compromise between the GPL license that enforces a strong copyleft and permissive licenses such as MIT. LGPL code can be included in a software (even proprietary) as a shared library without having to distribute the own software under LGPL, so the strong copyleft of the GPL does not apply for LGPL. Modifications to LGPL code have to published under the same license though.

As stated above, SU2 includes a variety of third-party software packages and libraries that all have their own license that needs to be respected individually. The most notable for the present work are CoDiPack and MeDiPack as well as ParMETIS. CoDiPack and MeDiPack are distributed under the GPL license, which enforces a strong copyleft. They are not shipped with SU2 itself and are only an integral component when the code is compiled with discrete adjoint capabilities. Therefore the distribution of the SU2 codebase which does not include the source code to the mentioned packages, does not trigger the GPL copyleft. In the case of ParMETIS, the original author gave the permission to use the software under some condition, which is proper referencing and inclusion of the original code and documentation.

The choice of license for an open-source software has to balance multiple interests and can have a significant influence on its adoption in the community. For one thing, a permissive license allows for the greatest possible distribution in that every programmer can include the piece of software without having to worry about incompatible licenses, on the other hand, a strong copyleft license potentially leads to a developer adopting the license instead of a proprietary one for a new project, resulting in more free software that others can benefit of. Highlighting the downsides, software with a permissive software might be vastly modified/extended and distributed in closed-source, proprietary code, but never shared with the community. A strong copyleft might discourage potential users to include the software due to license incompatibilities, this is especially true for projects with commercial interests. Ultimately, to some extent, open-source license choice at a project start is also down to personal preference.

Version Control

The used version control system Git is the de facto standard tool to track file-changes in a source tree, with its application ranging from small projects created by one person to large-scale international collaborative software development. The notable alternatives, svn and Mercurial, are no longer widely spread although legacy codes still use version control systems other than Git.

Git is a distributed version control system, meaning that the repository downloaded by a user contains all the source code of a project, which includes the history of all changes to all files in all branches. A branch represents an independent line of development within the existing code. Most notably, the `master` and `develop` branch are core branches in the GitFlow branching model explained below. The common workflow in SU2 for a developer is to create a separate branch of the code, giving it a specific name, e.g. `feature_streamwise_periodicity`, and when the development is finished the code is merged back into the `develop` branch. File changes are tracked on the scale of *commits* and each commit represents a snapshot of the codebase at that moment in time. Commits are identified by a unique commit-ID that for example can be shared to allow another person to use the exact same code for result reproduction. Users have to create commits, and based on personal preference this can be in small chunks or large-scale contributions. Due to its popularity and widespread use, vast resources to learn and master Git exist for all experience levels. The book of Straub and Chacon [CS14] provides an extensive introduction.

As already mentioned, the usage of Git within the project needs to be introduced. Namely, the popular GitFlow[3] concept is used by and large, which generally includes one main branch, one development branch and arbitrary many feature branches. The feature branches get merged into the development branch once their development is finished and in freely chosen intervals, the accumulated changes in the development version get merged into the main branch. The accumulated changes that get merged into the main branch represent official releases of the software that get tagged with a three-part version number, e.g. version 7.3.1 where all code contributions of this thesis are available in.

---

[3] See `https://nvie.com/posts/a-successful-git-branching-model/` (Accessed 2022-08-14) for the original blog post by Vincent Driessen, who first published and popularized GitFlow.

Build system

Build systems are used to create executable binaries from a collection of C++ source code files in a simple, user-friendly fashion. Most build systems for C++ code can be generally split into two sections: first a configuration step where exact build instructions are created including user-chosen options and the second step where those instructions are executed to create the binaries which ultimately constitute the executable program. SU2 uses a combination of meson[4] and ninja[5], where the former is the higher level build system that creates the input files for the latter, which in turn ultimately makes the appropriate calls to the specified compiler and linker.

It is not uncommon for developers to trigger incremental builds every other minute over the course of hours, especially when actively debugging. For inexperienced developers or new users of the software, a build system should allow a seamless adaption to their system, including their compilers and possible MPI-implementations. Therefore, an easy to use and fast build system can make all the difference for users and in the developers workflow.

Regression and Unit test

The code includes an extended set of integration/regression and unit tests that ship with the codebase. These can be run locally by each developer to verify that the own contribution does not invalidate existing code.

Unit tests are meant to test limited code sections (e.g. a function) without running the entire code, which includes a multitude of possible error sources. Input values are prescribed and the output values are compared to reference values that are also defined within the unit test. A failing unit test isolates the cause to a very narrow code region that is easier to debug compared to the whole code. Usually, some sort of mocking framework is required, that provides all necessary values and data structures to let the code run. In SU2 the unit testing framework Catch2[6] is used, with a notable alternative in the C++ community, being Google Test[7].

Integration or also called regression tests, on the other hand, apply the complete compiled software binary on a problem that it is originally intended to solve. Within the context of SU2, this means a regular simulation including a mesh and configuration file is run for a fixed number of iterations before comparing the output with stored values. The

---

[4] `https://mesonbuild.com/` (Accessed 2022-08-15)

[5] `https://ninja-build.org/` (Accessed 2022-08-15)

[6] `https://github.com/catchorg/Catch2` (Accessed 2022-08-15)

[7] `https://github.com/google/googletest` (Accessed 2022-08-15)

integration tests framework for SU2 is self-written and mainly stores/compares screen output values, e.g. residuals, objective functions, solver parameters, etc. Again, the test fails if the reference values are not matched within a small margin to account for minimal floating point arithmetic effects.

Integration tests are by and large easy to set up, as the framework can externally operate the software and just has to extract screen output values, whereas creating a mocking structure for unit tests to allow the specific code part to run requires an increased effort in an inheritance-heavy code like SU2. An integration test, by definition, tests the whole simulation run and can therefore detect a greatly increased number of possible errors. On the downside, a failing regression test by itself does not give a good indication what code part(s) broke the test. Interestingly, in the author's experience, due to the vast amount of integration test[8] it is possible in some cases to pinpoint the error quickly by the combination of integration tests that fail and checking what these have in common.

2.1.2 Development Platform

This subsection highlights the role of the used code hosting platform for collaborative development in the SU2 project: GitHub. GitHub is owned by Microsoft since 2018, and although SU2 uses GitHub specifically, all relevant features are available on similar platforms like GitLab or Bitbucket, or in combination with a further 3rd party tool to take over e.g. the regression testing with Travis CI.

As the name suggest, GitHub's primary task is to host Git repositories and therefore provides a central place to upload and share the repository with its various branches and commits. Although Git is a distributed version control system, all developers synchronize their repositories with the one hosted on GitHub if the aim is to merge their work into the official develop-branch. These merges are the final part of a Pull Request (PR) on the platform that are examined in the following, but there are more comfort features that facilitate the development process across the community, which are introduced thereafter.

A Pull Request on GitHub signals in the majority of cases that the development of a feature branch is finished, and the contributor wants to merge the changes into the development branch of the repository. The Pull Request into develop represents the major quality gate for developed code in the SU2 project, where multiple checks (in-

---

[8] 462 to be precise (in version 7.3.1), although it should be noted that there are major overlaps, e.g. the same configuration in a serial and parallel test. For the sake of completeness, there are 249 assertions tested in 28 unit test setups.

person and automatic) have to be fulfilled before the merge can take place. Merges in between feature branches or from the develop branch into the main branch do not have the same significance in terms of community engagement and required checks, therefore the process described below mainly relates to merges into the develop branch. In a Pull Request, the code author presents the code changes and provides an explanation of what could be relevant to the community to get a good overview without having to dig through the code: purpose, methods used, coding techniques, showcasing results in form of e.g. plots, etc. Other community members now have the possibility to generally comment on the work or add specific comments to selected lines of the code changes. In software development it became good practice to review code contributions by a different community member to spot possible mistakes, streamline efforts of parallel developments and to ensure an overall good quality of code. Ultimately, one member of the community has to provide a review that specifically approves the changes, otherwise the code cannot be merged by a regular member. In the SU2 project, one approving review is required for merging. The contrary to an approval is also possible by requesting changes, which consequently blocks the merge, even if another member approves the changes. This signals the PR author to implement the requested changes or to discuss its necessity until the person who blocked the PR lifts that restriction.

In addition to this in-person check, the regression and unit tests are automatically run and their successful passing is required for the PR to be merged. To be precise, not only the regression and unit tests are run, but also various build configurations of the code are compiled to check their integrity. These tests run in a docker container and the necessary computing resources are provided by GitHub itself. In the development process of a new feature it is helpful, in the experience of the author, to monitor early on if/which tests are failing to fix the introduced errors. Therefore, it is simplest to open a so-called Draft Pull Request on GitHub in an early stage of the development process as this will trigger the automatic builds and tests with every new commit, and additionally, allows community members to already review the work and spot possible issues or conflicts with other developments.

As a third pillar of automatic testing, multiple code quality tools are used that scan the code for obvious weaknesses and mistakes, or just bad style. The tools used at the time of writing are LGTM and CodeFactor, but there exists a multitude of tools and are the ones used by SU2 at the moment are most likely subject to change in the future. These tools are often called static code analysis tools, which in the case of SU2 are not explicitly geared towards security concerns, but e.g. unsafe handling of memory is reasonable to fix in any software. In modern software development, an increasing number of automatic tools are and will be deployed to support the developers in producing well-written and bug-free code. That is also the case for SU2. Ultimately, in spite of best efforts to thoroughly check every code contribution, it is impossible to prevent new bugs, issues and incompatibilities to occur with every new Pull Request.

The GitHub platform includes a bug or issue tracker which allows anyone to write a bug report or just describe their issue. In case someone wants to ask just a question that does not qualify for an issue, the discussion tab can be used, also just to showcase produced results. The benefit of having Pull Requests, issues and discussion all on the same platform is the ease with which certain details can be cross-referenced and the increased visibility of the variety of development efforts and problems potentially avoids time-consuming work down the road.

Apart from the main code repository for SU2, multiple complementary repositories are hosted on GitHub. Most notably are the Testcases repository that contains meshes, restart files or any other larger supplementary file that is required to run the regression tests, and the SU2 project's website[9].

2.1.3 Community & Foundation

Originally, the SU2 code was developed solely in Stanford with the first release in 2012, hence the acronym *Stanford University Unstructured*. Today, there are developers and users across multiple institutions from academia and industry that are organized without an obvious hierarchical structure. The following outlines how the SU2 community is composed and presents virtual and in-person communication spaces and formats. The SU2 foundation that formed in 2019 is an integral part of the community, but it is given its own subsection in this discourse due to its distinctive role.

Community

Three essential roles can be identified within the SU2 community: users, developers and managers. These are not mutually exclusive such that one person can represent all three roles to some extent. Think for example of a senior researcher that develops novel methods for a specific use-case and concurrently supervises a master's or PhD-student that works on an associated subproject.

As already mentioned, no single institution determines the course of SU2. This is ultimately done via the contributions from the involved organizations from academia and industry, mainly led in the form of professors and industry managers. Resources

---

[9] `https://su2code.github.io/` (Accessed 2022-08-14)

are allocated, personnel is acquired, and research focuses are set based on the boundary conditions in these specific institutions. By and large, that process is not widely communicated until countable facts can be presented. In order to make informed decision in these processes, these managers coordinate the realizability of the envisioned project with (mostly their) developers and users.

Most publicly known projects including SU2 involve some form of code development, which naturally requires developers. The largest group of developers historically and still nowadays, are PhD students from universities and also industry. This circumstance leads to a constant developer rotation as some PhD's finish their project and do not continue to work with the code and others that are just starting their project. Therefore it is crucial for the community to provide a welcoming and inclusive atmosphere to support each new member as good as possible in their learning phase and beyond. New developers always provide a new perspective and ideas which can be beneficial, but for a successful project some longer standing members that act as mentors are definitely an advantage. The main research task is set by their respective manager, but within the work it is natural that side projects arise that have positive side effects on their and other people's project. The introduced development platform GitHub provides a focal point for developers, as here their features are merged into the code, issues and discussions directly concerning the code arise and other developer's work is presented and should be followed as it potentially has direct impact on their own course of the project. The developers currently[10] schedule a weekly video conference meeting with a loose format in which everyone is invited to chat about their projects, problems, ideas, etc., and all participants try to discuss the topic and to find solutions and guidance if necessary. Although other formats were already used and probably will evolve over time, the direct verbal contact in addition to text-based communication in forums represents a crucial component to a productive community. No external determined hierarchy exists among developers, albeit a form of meritocracy naturally evolved, in which experienced developers take over a leading role.

Lastly, the group of users is difficult to capture in its entirety as communication often only occurs in form the of bug or issue reports or publicly shared success stories, whereas the former is much more common. As a more user-focused platform, a cfd-online Forum[11] was created, which covers mostly practical aspects of setting up configuration files, installation problems, etc., but also can provide valuable bug reports for developers.

Beyond that, there are some other notable platform and channels, where the commu-

---

[10] As of 2022-08-16

[11] https://www.cfd-online.com/Forums/su2/ (Accessed 2022-08-14)

nity manifests itself. A slack channel[12] with some subgroups is usually best to reach out easily to the majority of the community. Here, questions are posed, collaborators are searched and announcements from the foundation or the developers are made. Additional announcement channels are LinkedIn[13] and Twitter[14] to reach as many people as possible that are usually outside of the community, e.g. information about an upcoming conference. Ultimately, it does not matter on which platform a topic is started, but how detailed and complete it is, e.g. a bug report with detailed information and the possibility of reproduction for an outsider, is more likely to receive fast and helpful advice.

Through the initiative of the community including users, developers and managers around the world, SU2 is now a well established tool with wide applicability to aeronautical, automotive, ship, and renewable energy industries, to name a few.

Foundation

In response to an ever-growing community, the need for a foundation arose, that acts as a central point of contact for various topics around the SU2 project that can not naturally be answered or taken over by individual organizations or persons. The self-imposed principal purpose of the SU2 foundation is to *'promote global software development and education to increase the pace of innovation in the engineering sciences for the public benefit of all society'* and to *'provide a neutral forum for community collaboration by offering efficient infrastructure and technical governance'*. Both statements are taken from the foundation's bylaws that can be found in its entirety on their website[15].

The foundations board[16] is composed of selected community members from various institutions. The aim is to have a representative for all stakeholder groups, such as academia and industry, as well as developers and users.

A major, publicly visible event with the SU2 conference spanning 3 days is organized by the SU2 foundation in 2022 for the 3rd time. In-person developer's meetings were held before, but the increased organizational and financial effort that come with a larger community are easier to coordinate for a foundation, that considers that very task as a central aspect.

---

[12] Messaging platform that allows direct messaging to individual people and multiple open feeds with various focuses where public discussions or announcments can be made.

[13] `https://www.linkedin.com/company/su2code/` (Accessed 2022-08-14)

[14] `https://twitter.com/su2code` (Accessed 2022-08-14)

[15] `https://su2foundation.org/` (Accessed 2022-08-16)

[16] Comprised of 7 persons, as of 2022-08-16. The most recent cast can be reviewed on the foundation's website given above.

The foundation is also able to collect funding to finance certain activities within the scope of its bylaws, e.g. supporting the SU2 conference. A long-term vision is to gather sufficient funds to finance developers to work on code aspects that are partly neglected by the remaining community. It is natural that general code maintenance tasks are difficult to justify within the scope of a PhD student's work, such that it would be reasonable to take over these tasks through the foundation.

2.1.4 Documentation

For a numerical solver as SU2, which is embedded in a scientific community, multiple types and levels of documentation in a broad sense have to be considered: research papers, tutorials and testcases, explanation of physical and numerical models and finally implementation aspects. Despite the various community efforts, it is not possible to document all aspects in all detail, and some features even remain undocumented.

On the highest level, research papers present recent advances in a specific scientific field and document the employed methods and achieved results. As such, research papers do not only act as a documentation but also advertisement for the SU2 project. It is not guaranteed though, that the code utilized to achieve the presented results in a paper, is merged into the official SU2 version, or even publicly available at all in a feature branch.

Arguably the best way to get familiar with the software SU2 is to use it. For this purpose, 33 tutorials are available on the SU2 website, ranging from the well known 2D NACA0012 airfoil in crossflow, over unsteady conjugate heat transfer to constrained shape design of an ONERA M6 wing. The tutorials explain various aspects about usage of involved code options, running the code in multiple configurations and discuss result interpretation. The testcase suite used for the regression tests contains a great number of simulation setups, i.e. mesh and configuration files, that provide a starting point getting to know an option that is not featured in one of the tutorials. As a necessary prerequisite to run the code, a detailed setup and installation guide for the code is available on the website.

Physical and numerical models available in SU2 are, to the most part, either documented on the SU2-projects website, or are widely known in the broader scientific community and their specific documentation is not strictly necessary as other resources take over that task. For example, the classic Roe convective scheme is not explicitly documented and can be found in literature as in e.g. the book of Blazek [Bla15] which was used frequently in the development of standard CFD solver features.

In the last category, there is the code documentation itself. On the one hand side, the code itself provides the true implementation and thus documents all methods and models in a sense, on the other hand code documentation includes all intermediate comments that explain isolated code pieces. The comments in the code largely follow the syntax that can be parsed by the automatic code documentation software Doxygen, that was presented in the above Sec. 2.1.1. Note that the amount and verbosity of the code comments are based on the individual programmer's preference and in spite of review checks during each Pull Request, the comments can be of inconsistent quantity and quality throughout the code. While the three prior documentation aspects are geared towards users and developers alike, the code comments are naturally developer-focused. Lastly it should be noted, that due to the open-source nature of the code, anyone can check specific simulation methods or attempt result reproduction if the original creator provides a commit ID, that uniquely identifies the used code snapshot, and the utilized configuration files. Section 4.3 details result reproducible for the results presented in this thesis.

As an additional aspect, the accumulated content on the various platforms in cfd-online threads, resolved issues, discussions and Pull Request on GitHub or SU2-conference contents provide a vast knowledge resource with the downside of limited searchability.

2.2 Physical and numerical modeling

In this section, the governing equations for unsteady incompressible flow are described, and a density-based solution method for solving the used low-Mach form of the Navier-Stokes equations is derived[17], Sec. 2.2.1. After a brief presentation of the turbulence model employed in Sec. 2.2.2, the heat equation for solid zones, Sec. 2.2.3, and the coupling between fluid and solid for CHT computations is introduced, Sec. 2.2.4.

2.2.1 Unsteady Incompressible Solver

Low-Mach formulations of the Navier-Stokes equations have been shown to be well suited for incompressible flow while avoiding the complexity of the fully compressible form of the Navier-Stokes equations [Eco20]. Importantly, it augments the range of validity of compressible solvers for the analysis of low-speed applications across a range of flow regimes using a single code base. Here they are presented for time-accurate, fluid equations in the domain $\Omega \subset \mathbb{R}^3$:

$$\mathcal{R}\left(U\right) = \frac{\partial U}{\partial t} + \nabla \cdot \bar{F}^c\left(U\right) - \nabla \cdot \bar{F}^v\left(U, \nabla U\right) - S = 0, \tag{2.1}$$

---

[17] This part is a to a large extent joint work in cooperation with Crome, such that passages were taken and adapted from the joint publication [Ven+19].

where the conservative variables are $U = \{\rho, \rho\bar{v}, \rho c_p T\}^{\mathsf{T}}$ and S is a generic source term. In the scope of this thesis, S will hold contributions in the momentum equations due to the streamwise periodic approach presented in Sec. 3.1, but a further example would be asymmetric source terms. The sum of the terms in Eq. (2.1) is represented by the Residual operator $\mathcal{R}$. The relevant conditions to be fulfilled on the boundary (no-slip wall, symmetry, periodicity and farfield) are introduced later in this Sec. 2.2.1.

The convective and viscous fluxes can be written as:

$$\bar{F}^c(U) = \left\{ \begin{array}{c} \rho\bar{v} \\ \rho\bar{v} \otimes \bar{v} + \bar{\bar{I}}p \\ \rho c_p T\bar{v} \end{array} \right\}, \quad \bar{F}^v(U, \nabla U) = \left\{ \begin{array}{c} \cdot \\ \bar{\bar{\tau}} \\ \kappa\nabla T \end{array} \right\}, \tag{2.2}$$

where ρ is the fluid density, $\bar{v} = \{u, v, w\}^{\mathsf{T}} \in \mathbb{R}^3$ is the flow speed in Cartesian system of reference. Dynamic pressure is given by p while the thermodynamic pressure, p_o, is assumed constant throughout the domain and in time. The decoupled pressure-density system of equations are closed using an equation of state for density as a function of temperature alone. Assuming an ideal gas with gas constant R, density is given by $\rho = p_o/RT$.

The energy equation is also included in Eq. (2.2) where c_p is the specific heat at constant pressure, T is the temperature, κ is the thermal conductivity. κ is defined in SU2 via the laminar Prandtl number Pr_{lam} via:

$$Pr_{lam} = \frac{c_p\mu}{\kappa}. \tag{2.3}$$

Viscosity is given by μ and $\bar{\bar{\tau}}$ is the viscous stress tensor, which can be expressed as

$$\bar{\bar{\tau}} = \mu\left(\nabla\bar{v} + \nabla\bar{v}^T\right) - \mu\frac{2}{3}\bar{\bar{I}}\left(\nabla\cdot\bar{v}\right). \tag{2.4}$$

For laminar flows, viscosity is purely given by the fluid's dynamic viscosity, μ, which can be constant or assumed to vary with temperature according to Sutherland's law [Whi17]. Note that in this work, no temperature-dependent material conditions are considered for the application testcases. For turbulent flows, Reynolds-averaged Navier-Stokes (RANS) equations are solved with either the Shear Stress Transport (SST) model of Menter [Men94] or the Spalart-Allmaras (SA) [SA92] model. The coupling to the mean flow is achieved via an increase to the laminar dynamic viscosity by a turbulent contribution noted with subscript t, such that the total dynamic viscosity computes to:

$$\mu = \mu_{lam} + \mu_t. \tag{2.5}$$

Consequently, the thermal conductivity in turbulent flows is:

$$\kappa = \frac{c_p\mu_{lam}}{Pr_{lam}} + \frac{c_p\mu_t}{Pr_t}, \tag{2.6}$$

where the turbulent Prandtl number Pr_t is considered a constant of 0.9. The SST-model variant utilized in this work is specified in Sec. 2.2.2.

If Euler flows are investigated, then the viscous terms are removed by setting $\bar{F}^v = 0$, in the presented governing Eqs. (2.1) and the no-slip boundary condition is replaced with parallel flow conditions that are equivalent to the symmetry wall conditions in their implementation.

A Density-Based Solution Method

The approach of Weiss and Smith [WS95] is applied to develop a time-accurate density-based strategy for the low-Mach equations in Eq. (2.1). In this work, only incompressible flows are considered rather than a unified approach for compressible and incompressible flows. Identical to the density-based approach for steady incompressible flow given by Economon [Eco20], this derivation for unsteady incompressible flow starts with the assumption, that the equation of state for density is a function of both pressure and temperature, $\rho = \rho\,(p, T)$. In the derivation, the dependence on the pressure p will be eliminated as a consequence of infinite pressure wave speeds, which will leave $\rho = \rho\,(T)$.

For time-accurate solutions using an implicit solver scheme, a pseudo-time derivative in τ that needs to be marched in each pseudo time-step is introduced to the governing equations to give:

$$\frac{\partial U}{\partial t} + \frac{\partial U}{\partial \tau} + \nabla \cdot \bar{F}^c\,(U) - \nabla \cdot \bar{F}^v\,(U, \nabla U) - S = 0. \tag{2.7}$$

The working variables for the incompressible solver will be set to the primitive variables, $V = \{p, \bar{v}, T\}^\mathsf{T}$. This is not only a natural choice for incompressible flows, but for higher order spatial reconstruction, the gradients with respect to V rather than U are required. However, the final governing equations remain in conservative form, and the time derivative with respect to the physical time in the dual time-time formulation will be retained in terms of conservative variables, U.

The derivation of the preconditioning matrix starts by transforming the initial Eqs. (2.1) from conserved quantities U to primitive variables V as follows:

$$\frac{\partial U}{\partial t} + \frac{\partial U}{\partial V}\frac{\partial V}{\partial \tau} + \nabla \cdot \bar{F}^c\,(V) - \nabla \cdot \bar{F}^v\,(V, \nabla V) - S = 0, \tag{2.8}$$

where the Jacobian of the transformation between the conservative variable vector and the primitive vector was introduced:

$$\frac{\partial U}{\partial V} = \begin{bmatrix} \rho_p & 0 & 0 & 0 & \rho_T \\ \rho_p u & \rho & 0 & 0 & \rho_T u \\ \rho_p v & 0 & \rho & 0 & \rho_T v \\ \rho_p w & 0 & 0 & \rho & \rho_T w \\ \rho_p c_p T & 0 & 0 & 0 & \rho_T c_p T + \rho c_p \end{bmatrix}. \tag{2.9}$$

The partial derivatives of the equation of state with respect to pressure and temperature are given by $\rho_p = \partial\rho/\partial p$ and $\rho_T = \partial\rho/\partial T$, respectively. Note that c_p still represents the specific heat capacity at constant pressure and not a partial derivative. The equations are converted to non-conservative form through multiplication of an additional transformation matrix K, given by

$$K = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -u & 1 & 0 & 0 & 0 \\ -v & 0 & 1 & 0 & 0 \\ -w & 0 & 0 & 1 & 0 \\ -c_p T & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.10}$$

to give

$$K\frac{\partial U}{\partial t} + K\frac{\partial U}{\partial V}\frac{\partial V}{\partial \tau} + K\,\nabla \cdot \bar{F}^c - K\,\nabla \cdot \bar{F}^v - KS = 0. \tag{2.11}$$

The choice of K is motivated by the resulting matrix that premultilpies the pseudo-time derivative term:

$$K\frac{\partial U}{\partial V} = \begin{bmatrix} \rho_p & 0 & 0 & 0 & \rho_T \\ 0 & \rho & 0 & 0 & 0 \\ 0 & 0 & \rho & 0 & 0 \\ 0 & 0 & 0 & \rho & 0 \\ 0 & 0 & 0 & 0 & \rho c_p \end{bmatrix}, \tag{2.12}$$

insofar that upon inspection it can be noted, that the term ρ_p that multiplies the pressure time derivative in the continuity equation controls the speed of propagation of acoustic waves in the system. Under convergence the pseudo-time derivative tends to zero, therefore the matrix in Eq. (2.12) can be manipulated without affecting the final solution to V. For an incompressible flow, one has $\rho_p = 0$. Now to help with numerical convergence, this term is replaced by a value inversely proportional to the magnitude of the velocity in the domain, instead of just zeroing the contributions:

$$\Gamma_{nc} = \begin{bmatrix} \frac{1}{\beta^2} & 0 & 0 & 0 & \rho_T \\ 0 & \rho & 0 & 0 & 0 \\ 0 & 0 & \rho & 0 & 0 \\ 0 & 0 & 0 & \rho & 0 \\ 0 & 0 & 0 & 0 & \rho c_p \end{bmatrix}, \tag{2.13}$$

where β is a factor related to the maximum velocity in the domain and the subscript nc stands for non-conservative. This effectively controls the magnitudes of eigenvalues of the system. The variable name choice of β is made to draw a link to the artificial compressibility method, where β is typically used. A problem-independent scaling definition is adopted here as:

$$\beta^2 = \epsilon^2 \left(\bar{v} \cdot \bar{v}\right)_{max}, \tag{2.14}$$

where ϵ is a user-provided value and $\left(\bar{v} \cdot \bar{v}\right)_{max}$ is the maximum squared velocity in the domain, which is updated with each time step for an adaptive β value. In the experience of the author, the default value in the SU2 implementation for ϵ of 2.0 was not necessary to be changed for flow convergence. After the preconditioning of the system by replacing the matrix (2.12) with (2.13), the governing equations are transformed back into conservative form by pre-multiplying with K^{-1}. This effectively reverts the prior multiplication with K in Eq. (2.11) with the exceptions being the modified matrix that multiplies the pseudo-time derivative. The final form of the density-based approach in conservative form, where the incompressible governing equations are recovered when the pseudo-time derivative term is relaxed to zero, consequently writes:

$$\mathcal{R}\left(V\right) = \frac{\partial U}{\partial t} + \Gamma \frac{\partial V}{\partial \tau} + \nabla \cdot \bar{F}^c - \nabla \cdot \bar{F}^v - S = 0. \tag{2.15}$$

The final preconditioner takes the form:

$$\Gamma = \begin{bmatrix} \frac{1}{\beta^2} & 0 & 0 & 0 & \rho_T \\ \frac{u}{\beta^2} & \rho & 0 & 0 & \rho_T u \\ \frac{v}{\beta^2} & 0 & \rho & 0 & \rho_T v \\ \frac{w}{\beta^2} & 0 & 0 & \rho & \rho_T w \\ \frac{c_p T}{\beta^2} & 0 & 0 & 0 & \rho c_p + \rho_T c_p T \end{bmatrix}. \tag{2.16}$$

Treating the system with matrix Γ achieves pressure-velocity coupling and enables a coupled solution approach for incompressible problems. The matrix Γ treats all equations simultaneously with a pressure time derivative term, rather than only the continuity equation treatment seen in the classic artificial compressibility approach.

This approach results in a generic formulation for steady (by ignoring the physical time derivative) and unsteady flows, which has the added benefit of simplifying the implementation. Additional incorporation of grid movement into the incompressible flow equations does not alter the preconditioner [Ven20] which allows a straight-forward addition for unsteady flows.

Spatial Integration

Spatial integration methods are inherited from the steady solver implementation[Eco20]. Discretization in space is performed using a finite volume method (FVM) with edge-based data structure on a vertex-centered dual grid with control volumes. Semi-discretized, integral form of the governing equations are obtained by integrating Eq. (2.15) over a control volume $|\Omega_i|$ and applying the divergence theorem to give:

$$\int_{\Omega_i} \frac{\partial U_i}{\partial t}\, d\Omega + \int_{\Omega_i} \Gamma_i \frac{\partial V_i}{\partial \tau}\, d\Omega + \sum_{j\in\mathcal{N}(i)} \left(\tilde{F}^c_{ij} - \tilde{F}^v_{ij}\right) \Delta S_{ij} - S_i \left|\Omega_i\right| = 0, \tag{2.17}$$

where $\tilde{F}^c_{ij}$ and $\tilde{F}^v_{ij}$ are the numerical approximations of the convective and viscous fluxes projected along an edge ij, that connects points i and j. The fluxes are evaluated at the midpoint of each edge. ΔS_{ij} is the area of the face associated with the edge ij, $|\Omega_i|$ is the volume of the dual control volume, and $\mathcal{N}(i)$ is the set of neighboring vertices to vertex i.

The convective fluxes have been discretized using both an upwind flux difference splitting scheme (FDS) and a centered scheme based on a modified version of the Jameson, Schmidt, and Turkel (JST) scheme [JST81]. For the upwind scheme, second-order accuracy can be achieved via reconstruction of the primitive variables on the cell interfaces by a MUSCL approach [van79]. A detailed description of the convective schemes can be found in the steady solver description [Eco20].

The viscous fluxes are computed using a corrected average-gradient method [WMS97] by firstly obtaining the spatial gradients of the flow variables using a Green-Gauss or weighted least-squares approach [Bla15] which are then averaged to obtain these gradients at the control volume faces. Source terms are approximated at each vertex using piece-wise constant reconstruction within each of the dual control volumes.

For turbulent flows, the convective terms from a turbulence model are discretized using a scalar upwind scheme. The viscous and source terms from the turbulence model are handled similarly to the mean flow using a corrected average-gradient method and piece-wise constant reconstruction, respectively.

Time Integration

For time-accurate simulations, a dual-time stepping scheme is used as mentioned above. The simulation is marched in physical time with an internal pseudo-time marching at each physical time-step. Upon convergence of the inner iterations, the preconditioned pseudo time derivative vanishes, but the physical time derivative remains. The coupled system of partial differential equations represented by Eq. (2.17), can be rewritten as:

$$\frac{\partial}{\partial t}\left(U_i|\Omega_i|\right) + \frac{\partial}{\partial \tau}\left(\Gamma_i V_i|\Omega_i|\right) + \mathcal{R}_i\left(V^n\right) = 0, \tag{2.18}$$

where $\mathcal{R}_i\left(V^n\right)$ is the numerical residual that represents the integrated sum of all spatial terms, i.e. convective and viscous terms plus source terms, for the control volume surrounding vertex i with cell volume $|\Omega_i|$. The superscript n indicates the current physical time step.

The physical time-dependent term is evaluated through an implicit discretization scheme that can be chosen either first or second order accurate within the scope of SU2. Using second-order backward difference approximation to illustrate the procedure, the physical time derivative can be discretized to give:

$$\frac{\partial}{\partial t}\left(U_i|\Omega_i|\right) = \frac{3U^n - 4U^{n-1} + U^{n-2}}{2\Delta t}|\Omega_i|. \tag{2.19}$$

The system in Eq. (2.18) can be rewritten by adding the discrete physical time derivative in Eq. (2.19) to the discrete spatial residual $\mathcal{R}_i\left(V^n\right)$ to form an updated residual $\mathcal{R}_i^\star\left(V^n\right)$:

$$\frac{\partial}{\partial \tau}\left(\Gamma_i V_i|\Omega_i|\right) + \mathcal{R}_i^\star\left(V^n\right) = 0. \tag{2.20}$$

The conservative variables U required in the physical time derivative can be straightforwardly recovered from the primitive variables $U^n\left(V^n\right)$, such that $\mathcal{R}_i^\star$ remains dependent only on the primitive variables V^n. During the pseudo-time iterations, the known conservative variables of the past physical time steps, U^{n-1} and U^{n-2}, are naturally considered to be constants, and thus $\mathcal{R}_i^\star$ is not written down to be dependent on those states for the sake of brevity. As $\tau \to \infty$ the pseudo time derivative vanishes and the solution for U^n, respectively V^n, is found.

In the discretization of the pseudo-time derivative, the subscript p indicates the current pseudo timestep. The next pseudo-time level, $p+1$, is evaluated by a backwards Euler method (or implicit Euler), as:

$$\Gamma_i \frac{\Delta V_i^n}{\Delta \tau_i}|\Omega_i| = -\mathcal{R}_i^\star\left(V_{p+1}^n\right), \quad \Delta V_i^n = \left(V_i\right)_{p+1}^n - \left(V_i\right)_p^n. \tag{2.21}$$

A first-order linearization about pseudo time level p provides an approximation to the unknown $\mathcal{R}_i^\star\left(V_{p+1}^n\right)$:

$$\mathcal{R}_i^\star\left(V_{p+1}^n\right) = \mathcal{R}_i^\star\left(V_p^n\right) + \sum_{j\in\mathcal{N}(i)\cup i} \frac{\partial \mathcal{R}_i^\star\left(V_p^n\right)}{\partial V_j^n} \Delta V_i^n + \mathcal{O}\left(\left(\Delta V_i^n\right)^2\right), \tag{2.22}$$

where the sum over the Jacobian contributions already factors in the compact stencil over neighboring nodes (including the node i itself) that are used in classical FVM codes. Introducing Eq. (2.22) into Eq. (2.21), the following linear system should be solved to obtain the solution update ΔV_i^n:

$$\left(\Gamma_i \frac{|\Omega_i|}{\Delta\tau_i}\delta_{ij} + \frac{\partial \mathcal{R}_i^\star\left(V_p^n\right)}{\partial V_j^n}\right) \Delta V_j^n = -\mathcal{R}_i^\star\left(V_p^n\right), \tag{2.23}$$

where entries with a subscript j have to be summed over $j \in \mathcal{N}(i) \cup i$ and δ_{ij} represents the Kronecker delta. If a flux $\tilde{F}_{ij}$ has a stencil of points $\{i, j\}$, then contributions are made to the Jacobian at four points, or:

$$\frac{\partial \mathcal{R}^\star}{\partial V^n} := \frac{\partial \mathcal{R}^\star}{\partial V^n} + \begin{bmatrix} \ddots & & & & \\ & \frac{\partial \tilde{F}_{ij}}{\partial V_i^n} & \cdots & \frac{\partial \tilde{F}_{ij}}{\partial V_j^n} & \\ & \vdots & \ddots & \vdots & \\ & -\frac{\partial \tilde{F}_{ij}}{\partial V_i^n} & \cdots & -\frac{\partial \tilde{F}_{ij}}{\partial V_j^n} & \\ & & & & \ddots \end{bmatrix}. \tag{2.24}$$

For the density-based approach, the block contribution along the diagonal involving the volume and time step is multiplied by the matrix Γ_i and added to the Jacobian matrix for node i. In practice, the Jacobian terms are approximated to reduce the computational effort for constructing the linear system, or simply because a symbolic construction is infeasible. An exact Jacobian is naturally desired for optimal stability of the solution procedure, but if the specific application can be iterated until convergence, the inexact Jacobians are no longer relevant and theoretically do not influence the final solution. Eq.(2.23) can be rewritten as a fixed point iterator $V^n = G^n\left(V^n\right)$:

$$V_{p+1}^n = G^n\left(V_p^n\right) := V_p^n - \mathcal{Q}\left(V_p^n\right)\mathcal{R}^\star\left(V_p^n\right), \tag{2.25}$$

where $\mathcal{Q}$ is our approximate Jacobian from Eqn. (2.23), also given by

$$\mathcal{Q}\left(V_p^n\right) := \left(\Gamma\frac{|\Omega|}{\Delta\tau}\delta_{ij} + \frac{\partial\mathcal{R}^\star(V_p^n)}{\partial V^n}\right)^{-1}. \tag{2.26}$$

In Eqs. (2.25) and (2.26) the subscripts i, j indicating a specific node were omitted for the sake of brevity. The fixed-point iterator notation is useful in order to construct iterative discrete adjoint equations that use the same approximate Jacobian as the primal solver. The pseudo time-step $\Delta\tau$ that respects stability limits needs to be chosen, and the user can manipulate the pseudo-time step via the Courant-Friedrichs-Lewy(CFL) number.

Boundary Conditions

A numerical simulation can only model a part of the real physical domain, which requires the prescription of physical conditions on the resulting boundaries. These boundaries can be natural, like the no-slip wall which appears also in the real world, or motivated by computational cost to reduce the simulation effort like symmetry, periodicity or farfield conditions. Symmetry and periodic boundary conditions are crucial for reducing the necessary domain and therefore mesh size in the considered pin-fin-array array application in Sec. 5.1, and the farfield conditions delimit the domain for the unsteady pin in crossflow application in Sec. 5.2. An extensive discussion on boundary conditions and their implementation, for unstructured median-dual schemes in the case of SU2 but also cell-centered schemes, can found in the book of Blazek [Bla15].

The **no-slip wall** conditions is:

$$\bar{v} = 0, \tag{2.27}$$

with the condition being strongly enforced in SU2. This means, the respective degrees of freedom on the boundary are set to the respective value and the associated row in the linear system is neutralized by setting the diagonal entry to unity and others to zero. Note that here only the developed incompressible solver with the primitive variables as working variables is considered. In case of an active energy equation, either an isothermal or a heatflux wall for the temperature degrees of freedom are set in the scope of this thesis. In combination with the no-slip wall, one of the following conditions have to be met:

$$T = T_{Wall}, \quad \text{isothermal wall (Dirichlet)}, \tag{2.28}$$

$$\dot{q} = \dot{q}_{Wall}, \quad \text{heatflux wall (Neumann)}, \tag{2.29}$$

$$\dot{q} = h\left(T_\infty - T\right), \quad \text{heat transfer wall (Robin)}. \tag{2.30}$$

The temperature on the isothermal wall, in contrast to velocity for the momentum equations, is weakly enforced via a flux. Instead of setting the prescribed temperature value and deleting the degree of freedom from the linear system, a heatflux based on the temperature difference to a nearest normal neighbor is applied such that the boundary condition is fulfilled upon convergence. For a prescribed heatflux wall, the residual contribution to the cell is straight forward, as the remaining flux following Eq. (2.2) on a no-slip wall is a heatflux that is replaced by the value prescribed by the boundary condition. For heat transfer walls, the heatflux is not prescribed by the user, but computed locally based on the a given heat transfer coefficient h, a given reference temperature T_∞ and the local temperature T.

On **symmetry** boundaries, the following mathematical conditions have to be fulfilled:

$$\bar{n} \cdot \nabla s = 0, \tag{2.31}$$
$$\bar{n} \cdot \nabla \left(\bar{v} \cdot \bar{t} \right) = 0, \tag{2.32}$$
$$\bar{t} \cdot \nabla \left(\bar{v} \cdot \bar{n} \right) = 0, \tag{2.33}$$

where s stands for any scalar quantities, i.e. pressure and temperature. $\bar{n}$ denotes the normal vector and $\bar{t}$ a vector that is tangential to the symmetry boundary. In SU2 a reflected state is constructed that results in a flux which enforces these conditions[18].

For **periodic** boundary conditions, the degrees of freedom on either side of the periodic interface (note that SU2 requires a matching mesh on both sides of the interface) have to be identical. So for a periodic node pair i and j the requirement:

$$V_i = V_j \tag{2.34}$$

has to hold for the primitive variables. The occurring residuals, Jacobians, partial volumes, gradient contributions, etc. of either side of the interface are communicated between the periodic point-pairs such that upon convergence, each point-pair has an identical solution.

With a **farfield** boundary, an infinite domain is approximated such that the boundary condition that imposes free-stream conditions, optimally, has a negligible effect on the flow solution. For the incompressible flow solver, the flux of the boundary point to a constructed identical point that holds the free-stream values is computed.

2.2.2 Turbulence Modeling

The SST turbulence model as employed in SU2 is briefly introduced, as there are slight deviations towards the original SST-1994 version as listed on the website of the Turbulence model resource (TMR) of the Nasa Langley research center[19]. In order to stay

---

[18] This was implemented by the author in PR #657 `https://github.com/su2code/su2/pull/657` and PR #740 `https://github.com/su2code/su2/pull/740`.

[19] `https://turbmodels.larc.nasa.gov/sst.html` (Accessed 2022-08-09)

close to the TMR resources for better comparison, the Einstein summation convention is used in the subsequent equations. The motivation to implement a non-standard SST variant into SU2 remains unclear to the author. For the sake of reproducibility, the implemented SST variant is documented here, in case of a standard version, this would not have been done.

In a desired comparison with a commercial CFD-solver, e.g. Ansys Fluent, this circumstance should be kept in mind. A detailed comparison between the original SST-1994 [Men94], the updated SST-2003 [MKL03][20] and the SU2-variant, together with implications on an axisymmetric nozzle application, can be found in the master's thesis of Heimgartner [Hei21], which was supervised by the author.

The transport equations for turbulent kinetic energy and specific dissipation rate are:

$$\frac{\partial(\rho k)}{\partial t} + \frac{\partial \rho k u_j}{\partial x_j} = P_k - \beta^* \rho \omega k + \frac{\partial}{\partial x_j}\left[(\mu + \sigma_k \mu_t)\frac{\partial k}{\partial x_j}\right], \tag{2.35}$$

$$\frac{\partial(\rho \omega)}{\partial t} + \frac{\partial \rho \omega u_j}{\partial x_j} = \frac{\gamma}{\nu_t} P_\omega - \beta \rho \omega^2 + \frac{\partial}{\partial x_j}\left[(\mu + \sigma_\omega \mu_t)\frac{\partial \omega}{\partial x_j}\right] + 2\left(1 - F_1\right)\frac{\rho \sigma_{\omega 2}}{\omega}\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j}, \tag{2.36}$$

with $\sigma_k, \sigma_\omega, \beta$ and γ being blended between two constant values, e.g. for a generic variable ϕ one has:

$$\phi = F_1 \phi_1 + \left(1 - F_1\right)\phi_2, \tag{2.37}$$

with F_1 being defined in Eq. (2.45) and as all the constants are unchanged to the given TMR resource, they are not further listed here. ν_t is the turbulent kinematic viscosity. The production term for turbulent kinetic energy P_k is:

$$P_k = \mu_t S^2 - \frac{2}{3}\rho k \frac{\partial u_i}{\partial x_i}, \tag{2.38}$$

$$S = \sqrt{2 S_{ij} S_{ij}}, \quad S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right), \tag{2.39}$$

with the strain magnitude (or strain invariant) S and the strain rate tensor S_{ij}. For the turbulent kinetic energy production in Eq. 2.35 an additional production limiter is used:

$$P_k = \min\left(P_k, 20\beta^\star \rho \omega k\right). \tag{2.40}$$

---

[20] With PR #1560 https://github.com/su2code/SU2/pull/1560 (Accessed 2022-08-09) the SST-2003 model is available in SU2, which was after the thesis results had been finalized.

The production term for the specific dissipation rate $P\omega$:

$$P_\omega = \mu_t S^2 - \frac{2}{3}\max\left(\omega, \frac{\Omega F_2}{a_1}\right)\frac{\partial u_i}{\partial x_i}, \tag{2.41}$$

$$\Omega = \sqrt{2W_{ij}W_{ij}}, \quad W_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}\right), \tag{2.42}$$

$$F_2 = \tanh\left(arg_2^2\right), \quad arg_2 = \max\left(2\frac{\sqrt{k}}{\beta^\star \omega d}, \frac{500\nu}{d^2\omega}\right), \tag{2.43}$$

where Ω is the vorticity magnitude. For the production of specific dissipation rate, a production limiter is employed as well:

$$P_\omega = \max\left(P_\omega, 0.0\right). \tag{2.44}$$

The notable differences to the original SST-1994(m) version are, that the production terms P_k and P_ω are just computed as $P = \mu_t S^2$ without any further terms, and there is no production limiter for P_ω.

The blending function F_1 is computed by:

$$F_1 = \tanh\left(arg_1^4\right), \quad arg_1 = \min\left[\max\left(\frac{\sqrt{k}}{\beta^\star \omega d}, \frac{500\nu}{d^2\omega}\right), \frac{4\rho\sigma\omega 2k}{CD_{k\omega}d^2}\right], \tag{2.45}$$

$$CD_{k\omega} = \min\left(2\rho\sigma\omega 2k\frac{1}{\omega}\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j}, 10^{-20}\right), \tag{2.46}$$

where d is the distance from the field point to the nearest wall.

The turbulent eddy viscosity is finally computed as:

$$\mu_t = \max\left(\rho k \min\left(\frac{1}{\omega}, \frac{a_1}{\Omega F2}\right), 0.0\right), \tag{2.47}$$

with Ω is defined in Eq. (2.42) and F_2 in Eq. (2.43).

2.2.3 Solid Heat Conduction

In addition to the fluid flow, heat conduction in solid bodies is considered. This section introduces the PDE to be solved in the domain and the coupling of fluid and solid fields is discussed in the subsequent Sec. 2.2.4.

Following the notation for the fluid Eqs. (2.1), the heat equation in the domain $\Omega \subset \mathbb{R}^3$ is structured as follows:

$$\mathcal{R}\left(U\right) = \frac{\partial U}{\partial t} + \nabla \cdot \bar{F}^c\left(U\right) - \nabla \cdot \bar{F}^v\left(U, \nabla U\right) - S = 0, \tag{2.48}$$

where the working variables are simply $U = \{T\}$. The source term S could hold contribution of e.g. Joule heating, if additional electrical current were to be considered.

There are no convective fluxes in the solid body, and the remianing viscous fluxes can be written as:

$$\bar{F}^c(U) = \left\{ \, \cdot \, \right\}, \quad \bar{F}^v(U, \nabla U) = \left\{ \, \tfrac{\kappa}{\rho c_p} \nabla T \, \right\}. \tag{2.49}$$

As for the fluid equations, no temperature dependent material properties were considered in this thesis.

The spatial discretization of viscous fluxes via the finite volume method is identical to the fluid flow as well as the temporal discretization, both in pseudo and in physical time.

In terms of boundary conditions, isothermal, constant heatflux and symmetry boundaries are considered. The isothermal boundary condition is again weakly enforced as described for the fluid flow as well as the constant heatflux boundary, and for symmetry again the normal component of the temperature gradient has to vanish which leads to no additional flux contribution.

2.2.4 Conjugate Heat Transfer

In conjugate heat transfer problems, heat transfer between fluids and solids is coupled through a fluid-solid interface. The continuity of temperature and heat-flux has to be satisfied at the fluid-solid interface:

$$T_f = T_s, \tag{2.50}$$

$$\dot{q}_f = \dot{q}_s, \tag{2.51}$$

with the subscripts f, s indicating fluid or solid, respectively.

In SU2, the solution to the coupled problem is computed by exchanging suitable boundary values between zones, which are otherwise iterated independently. This approach is referred to as the *partitioned* approach, which allows for independent development of each solver and also of the coupling tools, e.g. using preCICE as coupling tool in a CHT simulation with OpenFOAM to solve the fluid domain and CalculiX to solve the solid domain as presented in cite. [Luc16]. The notable alternative of one unified system that contains the degrees of freedom of the fluid and solid zone that is solved for at once, is referred to as *monolithic* approach.

It remains to introduce the communicated values between zones and the applied boundary conditions in each zone for the partitioned approach in SU2. Note that the CHT applications shown in this work have a matching mesh interface which makes the use of a nearest-neighbor mapping of boundary nodes viable. In case of strongly varying mesh resolutions and a resulting non-matching interface mesh it is recommendable to explore options of interpolating communicated values across the interface.

On the fluid side, Dirichlet boundary conditions for Temperature are imposed, therefore the solid side only needs to transfer its Temperature values. The Dirichlet boundary condition for temperature on the fluid side of the CHT interface is strongly enforced, as opposed to the weak imposition that is done for the standalone isothermal boundary. In addition, no-slip conditions for the flow velocities are strongly enforced on the fluid interface, just as for the regular isothermal or heatflux wall. On the solid side, a Robin boundary condition (in this context often called convective or heat transfer boundary) is applied that computes a resulting heatflux based on a heat transfer coefficient h and a temperature difference between fluid and solid side:

$$\dot{q}_s = h \left(T_s - T_f \right), \tag{2.52}$$

where the choice of h is not dependent on the material properties, but ultimately an empirical quantity with implications on convergence speed, as shown in [ED16]. Alternatively, the resulting heatflux on the fluid interface could be transferred and applied directly to the solid domain. Differences in stability and convergence behavior on applying a Dirichlet condition to the fluid and a Robin/Neumann boundary to the solid or vice versa are discussed in [VS16].

The interface coupling process provides a natural possibility to introduce an additional iteration level, in which multiple iterations are performed in a zone with a constant interface boundary before updating the interface information by calling the respective transfer routine. In an unsteady multizone simulations using dual-time stepping as in this work, there are hence a total of three iteration levels: physical time iterations, outer iterations in which interface communication takes place and finally inner iterations in each zone with the interface boundary fixed. The specific applications considered in this thesis only use a single inner iteration per outer iteration, i.e. the interfaces values are always up-to-date with the zonal values, as no direct benefit of multiple inner iterations could be observed.

3 Development of an efficient adjoint-based design capability

In the first Sec. 3.1 of this chapter, modeling capabilities for streamwise periodic flow are introduced as an essential prerequisite for a computationally affordable evaluation of periodically repeating geometries as investigated in the application Sec. 5.1. The adjoint-based sensitivity computation leveraged by algorithmic differentiation follows in Sec. 3.2 and provides the foundation for the efficient design capabilities that are applied in this thesis.

3.1 Modeling streamwise periodic flow

Flows through periodically repeating geometries can be simulated (or approximated for non-academic cases) solely on a representative unit cell using the approach presented in this section. The following cyclic conditions of reoccurring conditions (or flow patterns) characterize streamwise periodic, or often called fully developed, flow:

$$p\left(\bar{x}\right) = p\left(\bar{x} + \bar{t}\right) + P_{\Delta}, \tag{3.1}$$

$$\bar{v}\left(\bar{x}\right) = \bar{v}\left(\bar{x} + \bar{t}\right), \tag{3.2}$$

$$T\left(\bar{x}\right) = T\left(\bar{x} + \bar{t}\right) - T_{\Delta}, \tag{3.3}$$

where $\bar{x}$ is any point in the flow domain, and $\bar{t}$ the periodic translation vector that connects two subsequent points in the domain, that exhibit the same flow pattern. The length of $\bar{t}$ represents the length in which flow patterns reoccur, therefore the simulation domain length for streamwise periodic simulation is chosen as the length of $\bar{t}$. Fig. 3.1 sketches a simulation domain Ω together with various quantities that will become relevant in the subsequent derivations. The domain can be placed arbitrarily in the coordinate system for the following derivation, but in case the periodic translation vector $\bar{t}$ coincides with a coordinate axes and the domain inlet is placed on the origin, much of the derivation simplifies.

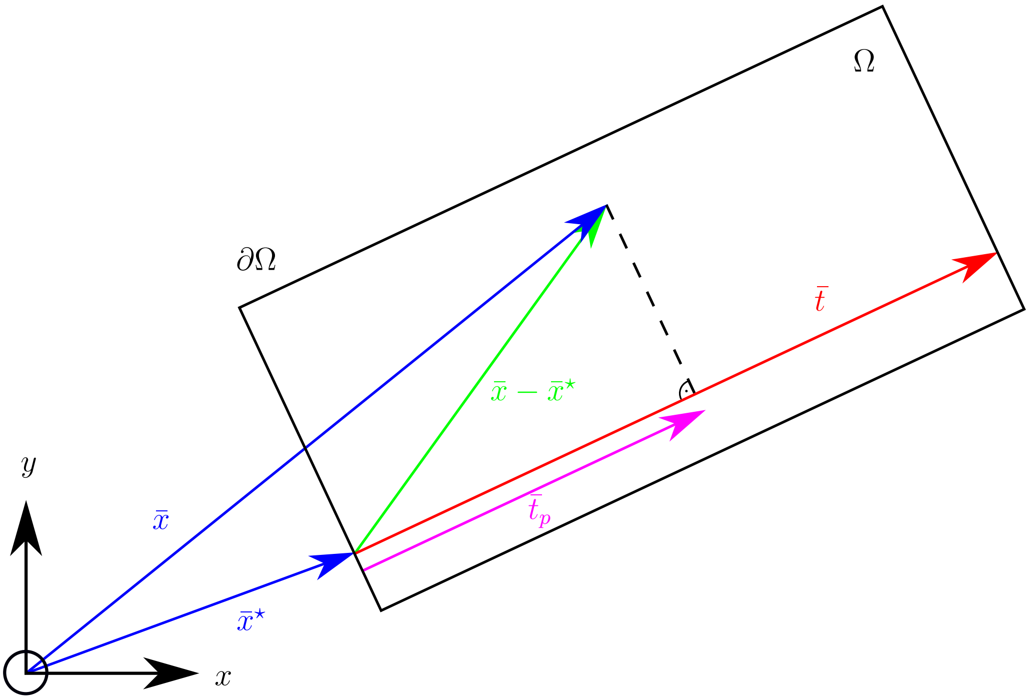


Figure 3.1: Illustration of the computation for the domain progress variable $\bar{t}_p$, in direction of the periodic translation vector $\bar{t}$.

From Eq. (3.2) follows, that velocity behaves truly periodic, in that the same velocities can be observed periodically in direction of the periodic translation vector $\bar{t}$. For pressure in Eq. (3.1) and temperature in Eq. (3.3) the shape of the pressure/temperature distribution at successive streamwise locations is identical but with a constant offset, P_Δ or T_Δ. The signs in front of the constant offsets is motivated by pressure decreasing in downstream direction and fluid temperature increasing in general cooling applications, such that P_Δ and T_Δ itself represent positive values. But a reversal in that logic is straight forward and does not require any change in the derivation.

Now in order to simulate streamwise periodic flow, the interfaces of the aforementioned representative unit cell have to be connected via periodic boundary conditions in flow direction. The prescription of periodic boundary conditions alone, at what is supposed to be the designated inlet and outlet, leads to the trivial solution of zero flow velocities and a constant pressure field. In many conducted internal flow simulations, a combination of in- and outlet boundary, e.g. velocity inlet and pressure outlet, acts as a momentum source term resulting in non-zero velocities. Therefore, additional source terms for momentum and energy, based on the streamwise periodic conditions in Eqs. (3.1)-(3.3) are derived in the following subsections based on the work of Patankar [PLS77] and Beale [Bea05], with an additional focus on possible boundary condition adaptions as well as implications on RANS equations in Sec. 3.1.4 or CHT simulations in Sec. 3.1.3.

3.1.1 Pressure Periodicity through the Momentum Equations

The previously introduced condition for pressure in streamwise periodic flows in Eq. (3.1) motivates the subdivision of the pressure field p into two components:

$$p\left(\bar{x}\right) = \tilde{p}\left(\bar{x}\right) - P_{\Delta}\frac{t_p}{\left\|\bar{t}\right\|_2}, \quad t_p = \frac{1}{\left\|\bar{t}\right\|_2}\left|\left(\bar{x} - \bar{x}^{\star}\right) \cdot \bar{t}\right|, \tag{3.4}$$

where the field $\tilde{p}$ is truly periodic with $\tilde{p}\left(\bar{x}\right) = \tilde{p}\left(\bar{x} + \bar{t}\right)$ and relates to the local fluid motions, whereas the second term relates to the global mass flow and increases linearly along the periodic translation vector. The factor to the pressure drop, along a streamwise unit P_{Δ}, is simply the progression along the periodic translation vector divided by its total length and is thus zero at the beginning of the periodic domain and scales linearly to one at the end of the domain. Fig. 3.1 motivates the computation of the domain progress variable t_p via orthogonal projection, with $x^{\star}$ being any point on the inlet surface and $\left\|\cdot\right\|_2$ the Euclidean norm.

Recall the previously introduced momentum equation in the incompressible Navier-Stokes Eqs. (2.1) that contain a divergence term dependent on pressure:

$$\frac{\partial \bar{v}}{\partial t} + \nabla \cdot \left(\rho \bar{v} \otimes \bar{v} + \bar{\bar{I}} p - \bar{\bar{\tau}}\right) = 0. \tag{3.5}$$

Inserting the above definition for pressure in streamwise periodic conditions (3.4) yields:

$$\frac{\partial \bar{v}}{\partial t} + \nabla \cdot \left(\rho \bar{v} \otimes \bar{v} + \bar{\bar{I}}\left(\tilde{p} - P_{\Delta}\frac{t_p}{\left\|\bar{t}\right\|_2}\right) - \bar{\bar{\tau}}\right) = 0. \tag{3.6}$$

Evaluating only the divergence of the pressure related term:

$$\nabla \cdot \left(\bar{\bar{I}}\left(\tilde{p} - P_{\Delta}\frac{t_p}{\left\|\bar{t}\right\|_2}\right)\right) = \nabla \cdot \left(\bar{\bar{I}}\tilde{p}\right) - \nabla \cdot \left(\bar{\bar{I}} P_{\Delta}\frac{t_p}{\left\|\bar{t}\right\|_2}\right) = \nabla \cdot \left(\bar{\bar{I}}\tilde{p}\right) - \frac{P_{\Delta}}{\left\|\bar{t}\right\|_2^2}\bar{t}, \tag{3.7}$$

leads to an additional volumetric source term that has to be added to the momentum equations while the original pressure term is now formulated in terms of the periodic pressure component instead of the physical pressure. Writing the momentum equations adjusted to the streamwise periodic conditions with respect to pressure gives:

$$\frac{\partial \bar{v}}{\partial t} + \nabla \cdot \left(\rho \bar{v} \otimes \bar{v} + \bar{\bar{I}}\tilde{p} - \bar{\bar{\tau}}\right) - \frac{P_{\Delta}}{\left\|\bar{t}\right\|_2^2}\bar{t} = 0. \tag{3.8}$$

As a practical sidenote for implementation, one has $\left\|\bar{t}\right\|_2^2 = \bar{t} \cdot \bar{t}$. Note that with the change to the periodic pressure $\tilde{p}$ as the solvers working variable, the physical pressure, if needed, has to be reconstructed as a post-processing step by evaluating the pressure decomposition in Eq. (3.4) for each grid node. This is especially necessary for objective functions based on physical pressure levels, e.g. the pressure drop between in- and outlet is of course zero evaluated for the periodic pressure component, while the evaluation with physical pressure yields the prescribed pressure drop P_Δ.

The pressure drop over the domain P_Δ has to be provided by the user based on the treated application, but the possibility to automatically determine the suitable P_Δ for a prescribed massflow through the periodic interface will be discussed later in this section. The periodic translation vector $\bar{t}$ is already uniquely defined by the periodic interface pair that is given by the user. The introduced source term generates no additional Jacobian contribution in case a P_Δ is directly prescribed.

As no adaption to the time derivative term had to made, the dual-time stepping approach employed in this thesis for time-accurate simulations remains valid as is. This is also the case for the derivation for the energy equation in Sec. 3.1.2.

The boundary condition (noslip wall, symmetry, etc.) do not need to be adapted based on the subdivision of the pressure field. In the upcoming Sec. 3.1.2 which introduces a subdivision of the temperature field, a notable adaption needs to be made for heatflux boundaries.

Prescribing a Fixed Massflow

In some real-world applications, the massflow over the periodic section is known (or strictly enforced) but not the pressure drop. Up to now, only a pressure drop P_Δ can be prescribed by the user.

In order to prescribe a specific massflow $\dot{m}_t$ (with subscript t for target), a continuous update mechanism for P_Δ is necessary, such that upon convergence a P_Δ value is found that results in the target $\dot{m}_t$. The initial guess for P_Δ has to be given by the user based on experience, but a good guess is not important in the author's experience and taking 1 as an initial value works well. After performing the first iteration with the initial P_Δ, the pressure drop can be updated based on the difference between current and target massflow, i.e. $\dot{m}_t - \dot{m}$. The current massflow over a surface $\partial\Omega$ is computed by:

$$\dot{m} = \int_{\partial\Omega} \rho\bar{v} \cdot \bar{n}\, dS. \tag{3.9}$$

Instead of simply updating the pressure drop based on the massflow difference and relaxation factor, an approximation of the required pressure drop based on Bernoulli's equation is done:

$$P_{\Delta}^{n+1} = P_{\Delta}^{n} + \phi \, \frac{\frac{1}{2}\rho\left(\dot{m}_t^2 - \dot{m}^2\right)}{\left(\rho A\right)^2}, \tag{3.10}$$

where the superscript n indicates the pseudo time iteration, A the surface area and $\phi \in (0, 1]$ a relaxation factor. A relaxation factor of $\phi = 0.1$ was found to be stable in all tested configurations. Note that in case of variable density flow, ρ is recommended to be area or massflow averaged over the periodic surface.

Note that Eq. (3.10) constitutes a fixed point equation for the streamwise periodic pressure drop and thus has to be handled accordingly in the derivation of the discrete adjoint equations that is based on fixed point operators. Sec. 3.2.4 covers the inclusion of additional fixed point equations beyond the flow solutions, and Sec. 5.1.4 discusses the implications on gradient accuracy.

3.1.2 Temperature Periodicity through the Energy Equation

The derivation for the inclusion of the temperature condition Eq. (3.3) in streamwise periodic flow is essentially identical to the previously performed derivation for the pressure field. With the restriction, that only heatflux boundaries can be considered. In flows with isothermal walls, Eq. (3.3) which states a reoccurring uniform temperature difference between streamwise cross-sections, does not hold as the flow temperature asymptotically converges against the wall temperature until in the limit actually $T\left(\bar{x}\right) = T\left(\bar{x} + \bar{t}\right)$ is fulfilled. Methods for including constant isothermal walls in streamwise periodic flows are presented in [PLS77; Bea06] but not further discussed in this work. In Sec. 3.1.3 an alternative approximate approach to handle streamwise periodic temperature is presented based on a boundary heat-sink, that is used to enable CHT simulations with cyclic repeating temperature profiles. Note that for this derivation, all material properties are considered to be constant and explicitly not temperature dependent. This is not a hard requirement per se, but the derivation would need to be redone taking this circumstance under consideration.

Again, the temperature field is subdivided into two parts:

$$T\left(\bar{x}\right) = \tilde{T}\left(\bar{x}\right) + T_{\Delta}\frac{t_p}{\left\|\bar{t}\right\|_2}, \quad t_p = \frac{1}{\left\|\bar{t}\right\|_2}\left|\left(\bar{x} - \bar{x}^{\star}\right)\cdot\bar{t}\right|, \tag{3.11}$$

where $\tilde{T}$ is truly periodic with $\tilde{T}\left(\bar{x}\right) = \tilde{T}\left(\bar{x} + \bar{t}\right)$ and accounts for local effects. The second term, including T_{Δ}, accounts for global heat addition over the domain. Before inserting Eq. (3.11) into the low-Mach Navier-Stokes Eqs. (2.1), recall the energy equation that exclusively contains terms based on temperature:

$$\frac{\partial\left(\rho c_p T\right)}{\partial t}+\nabla\cdot\left(\rho c_p T\bar{v}-\kappa\nabla T\right)=0. \tag{3.12}$$

For the sake of simplicity, the time-derivative, advection and diffusion term are handled separately. Starting with the time-derivative term:

$$\frac{\partial\left(\rho c_p T\right)}{\partial t}=\frac{\partial\left(\rho c_p \tilde{T}\right)}{\partial t}+\frac{\partial\left(\rho c_p T_\Delta \frac{t_p}{\|\bar{t}\|_2}\right)}{\partial t}=\frac{\partial\left(\rho c_p \tilde{T}\right)}{\partial t}, \tag{3.13}$$

where no additional contribution has to be made as the second summand is time-independent for the here considered incompressible flows. An additional source term arises from the advection term:

$$\begin{aligned}
\nabla\cdot\left(\rho c_p T\bar{v}\right)&=\nabla\cdot\left(\rho c_p\tilde{T}\bar{v}\right)+\nabla\cdot\left(\rho c_p T_\Delta\frac{t_p}{\|\bar{t}\|_2}\bar{v}\right), &(3.14)\\
&=\nabla\cdot\left(\rho c_p\tilde{T}\bar{v}\right)+\frac{\rho c_p T_\Delta}{\|\bar{t}\|_2}\nabla\cdot\left(t_p\bar{v}\right), &(3.15)\\
&=\nabla\cdot\left(\rho c_p\tilde{T}\bar{v}\right)+\frac{\rho c_p T_\Delta}{\|\bar{t}\|_2}\left[\nabla t_p\cdot\bar{v}+t_p\nabla\cdot\bar{v}\right], &(3.16)\\
&=\nabla\cdot\left(\rho c_p\tilde{T}\bar{v}\right)+\frac{\rho c_p T_\Delta}{\|\bar{t}\|_2^2}\left[\bar{t}\cdot\bar{v}\right], &(3.17)
\end{aligned}$$

using that $\nabla\cdot\bar{v}=0$ in incompressible flow with constant density. Note that in contrast to the source term in the momentum equations which is constant in space, the source term in the energy equation is spatially varying based on the local velocity vector. The diffusion term does not add another source under the assumption that the gradient of thermal conductivity is zero. That assumption holds for laminar flows but not for turbulent flows with a non-zero eddy-viscosity gradient, which will be covered in Sec. 3.1.4. The diffusion term expands to:

$$\begin{aligned}
-\nabla\cdot\left(\kappa\nabla T\right)&=-\nabla\cdot\left(\kappa\nabla\tilde{T}\right)-\nabla\cdot\left(\kappa\nabla\frac{T_\Delta}{\|\bar{t}\|_2}t_p\right), &(3.18)\\
&=-\nabla\cdot\left(\kappa\nabla\tilde{T}\right)-\nabla\cdot\left(\kappa\frac{T_\Delta}{\|\bar{t}\|_2^2}\bar{t}\right), &(3.19)\\
&=-\nabla\cdot\left(\kappa\nabla\tilde{T}\right)-\frac{T_\Delta}{\|\bar{t}\|_2^2}\nabla\cdot\left(\kappa\bar{t}\right), &(3.20)\\
&=-\nabla\cdot\left(\kappa\nabla\tilde{T}\right)-\frac{T_\Delta}{\|\bar{t}\|_2^2}\left[\nabla\kappa\cdot\bar{t}+\kappa\nabla\cdot\bar{t}\right], &(3.21)\\
&=-\nabla\cdot\left(\kappa\nabla\tilde{T}\right)-\frac{T_\Delta}{\|\bar{t}\|_2^2}\nabla\kappa\cdot\bar{t}, &(3.22)\\
&=-\nabla\cdot\left(\kappa\nabla\tilde{T}\right),\quad \text{if } \kappa=\text{const.} &(3.23)
\end{aligned}$$

Writing the energy equation in terms of periodic temperature with the additional source term arising from the advection term yields:

$$\frac{\partial\left(\rho c_p\tilde{T}\right)}{\partial t} + \nabla\cdot\left(\rho c_p\tilde{T}\bar{v} - \kappa\nabla\tilde{T}\right) + \frac{\rho c_p T_\Delta}{\|\bar{t}\|_2^2}\left[\bar{t}\cdot\bar{v}\right] = 0. \tag{3.24}$$

Up to now, the choice of T_Δ was not discussed. In contrast to P_Δ, it is not a directly user chosen quantity, but implicitly prescribed via the heatflux boundary condition that dictate a T_Δ in a steady state system via:

$$\dot{Q} = \dot{m}c_p T_\Delta, \tag{3.25}$$

$$\Rightarrow T_\Delta = \frac{\dot{Q}}{\dot{m}c_p}, \tag{3.26}$$

where $\dot{Q}$ is the rate of heat flow in watts, i.e. the integrated boundary heatflux. $\dot{Q}$ is computed based on the user specified boundary heatflux from the boundary condition instead of the effective heatflux computed using the local temperature gradient times thermal conductivity.

The final energy equation formulated in terms of periodic temperature then is:

$$\frac{\partial\left(\rho c_p\tilde{T}\right)}{\partial t} + \nabla\cdot\left(\rho c_p\tilde{T}\bar{v} - \kappa\nabla\tilde{T}\right) + \frac{\dot{Q}\rho}{\dot{m}\,\|\bar{t}\|_2^2}\left[\bar{t}\cdot\bar{v}\right] = 0, \tag{3.27}$$

with the straight-forward Jacobian contribution:

$$\frac{\partial}{\partial V}\left(\frac{\dot{Q}\rho}{\dot{m}\,\|\bar{t}\|_2^2}\left[\bar{t}\cdot\bar{v}\right]\right) = \left(0, \frac{\dot{Q}\rho}{\dot{m}\,\|\bar{t}\|_2^2}\bar{t}, 0\right)^{\mathsf{T}}. \tag{3.28}$$

Boundary conditions

The now formulated energy equation uses the periodic temperature component as the working variable, but the heatflux boundary condition is still given regularly based on the physical temperature gradient $\dot{q} = -\kappa\nabla T\cdot\bar{n}$ in watts per square meter.

For the derivation of the adaption for the heatflux boundary condition, the finite volume method is applied to the energy equation in periodic temperature formulation and the known conditions on the boundary are inserted in the equation. This is a standard approach to translate the prescribed conditions on the boundary into flux contributions. For a heatflux wall, it is known that $\bar{v} = 0$ and $\dot{q} = -\kappa\nabla T\cdot\bar{n}$. First the energy equation is integrated over a finite volume followed by the applications of the divergence theorem. The advection term contribution vanishes by inserting the no-slip condition, and considering that the temporal derivative contribution and the previously derived additional source term are applied to each grid node in any case leaves the diffusion term:

$$-\int_{\partial\Omega}\left(\kappa\nabla\tilde{T}\right)\cdot\bar{n}=-\int_{\partial\Omega}\left(\kappa\nabla T\right)\cdot\bar{n}dS+\int_{\partial\Omega}\left(\kappa\frac{T_{\Delta}}{\|\bar{t}\|_2^2}\bar{t}\right)\cdot\bar{n}dS, \tag{3.29}$$

where the subdivision of the temperature field from Eq. (3.11) was inserted ‘backwards’ again for the periodic temperature component. As in the source term derivation, T_{Δ} is replaced by the formulation based on the rate of heat flow into the domain:

$$-\int_{\partial\Omega}\left(\kappa\nabla T\right)\cdot\bar{n}dS+\int_{\partial\Omega}\kappa\frac{\dot{Q}}{\dot{m}c_p\,\|\bar{t}\|_2^2}\left[\bar{t}\cdot\bar{n}\right]dS=0, \tag{3.30}$$

which is now a formulation based on the physical temperature gradient such that the normal heatflux boundary condition can be applied with the addition of one extra term that accounts for the streamwise periodic effects.

The derivation on symmetry walls identical up to Eq. (3.30), with the exception, that the advection term on the boundary is zero based on the flow tangency condition $\bar{v}\cdot\bar{n}=0$. The second, additional, term is zero, as a symmetry wall has to fulfill $\bar{t}\cdot\bar{n}=0$. The normal vector of the symmetry plane and the translation vector between the periodic surfaces have to be perpendicular to one another. This leaves the diffusion term that is zero based on the requirement $\nabla T\cdot\bar{n}=0$ on symmetry walls, such that ultimately again no flux contribution has to be made for the energy equation, just as in the non-periodic case.

In the context of boundary conditions it shall be noted that there is no ‘anchor’ for the absolute level of temperature in the absence of isothermal walls, inlets with a prescribed temperature or other boundaries that include a fixed temperature of some kind. The overall temperature level therefore depends on the initial conditions and the convergence history. This circumstance is especially noteworthy for temperature-based objective function, e.g. area averaged surface temperature, in that they should not be used for optimization as a higher/lower temperature might exploit numerical properties rather than a physical effects. Performing the same simulation with varying MPI-ranks and/or OpenMP threads can greatly influence convergence properties and thus the temperature level, which is certainly an unwanted property for a simulation tool. A possible remedy for this issue is an artificial temperature anchor, e.g. dialing in a massflow averaged temperature at the periodic inlet, as presented in the subsequent Sec. 3.1.3 on the treatment for CHT applications. The gradient of temperature however, i.e. local differences of temperature, is uniquely defined by the simulation setup and can thus be used in a straight-forward fashion for objective functions and other comparisons.

For using temperature dependent material properties, the above presented derivation needs to be revised as gradients of the properties in question are no longer zero. Additionally, it needs to be taken care of, that the material property is evaluated based on the recovered/physical temperature and the periodic component which represents the solvers working variable.

3.1.3 Alternative Treatment in coupled CHT Applications

In CHT simulations, isothermal boundaries are applied to the interface on the fluid side as established in Sec. 2.2.4 which disqualifies the use of true temperature periodicity as introduced in the previous section. The alternative presented in this section consists of keeping the unmodified energy equation formulated in physical temperature, but extracting the integrated heatflux induced over all boundaries via an artificial source term at the outlet. In essence, the total rate of heat flow $\dot{Q}$ in the previous approach is extracted via a volume heat sink, while here this is achieved via a heat sink at the outlet boundary. The quantity of $\dot{Q}$ given in watts is computed via integrating the heatflux over the outer boundary of the CHT simulation domain, which excludes fluid-solid interfaces. The outlet source term for the total rate of heat flow $\dot{Q}$ can be either area or massflow weighted, where massflow weighting was found to me more suitable to maintain the temperature profile over the periodic interface. The temperature level is bounded as the incoming and outgoing rate of heatflow over domain boundaries is balanced. In CHT applications as in Sec. 5.1, it is not unreasonable to assume that the fluid boundaries consists purely of interfaces, symmetry and periodic faces and the heat-load is applied to surfaces in the solid.

The resulting residual contribution to each vertex on the periodic outlet surface thus becomes:

$$Res \mathrel{-}= \frac{\dot{m}_{local}}{\dot{m}}\dot{Q}, \tag{3.31}$$

where the total rate of heatflow is massflow weighted for each element, with $\dot{m}_{local}$ being the massflow through the element and $\dot{m}$ the massflow through the periodic surface.

In addition to the boundary heat sink, a residual term based on the difference of the massflow averaged inlet surface temperature to a user-given value is employed at the very same boundary. Upon convergence, this fixes the inlet temperature T_{inlet} to the user-value T_{user} and remedies the issue of the arbitrary absolute temperature level:

$$Res \mathrel{+}= \frac{1}{2}\dot{m}_{local}\, c_p \left(T_{inlet} - T_{user}\right). \tag{3.32}$$

This temperature fixture might also be used in conjunction with the true streamwise periodic temperature approach in Sec. 3.1.2.

Note that with using the temperature fixture in Eq. (3.32), the initially constructed outlet heat sink based on integrating all boundary heat fluxes in Eq. (3.31) becomes obsolete, as Eq. (3.32) extracts as much rate of heat flow as required to dial in the fixed temperature. It is reasonable though to keep both for faster converging temperature levels.

3.1.4 Implications on RANS Turbulence Modeling

In this section, the inclusion of transport equations for turbulence models in streamwise periodic setups is discussed. In SU2 the widely used Spallart-Allmarras (SA, 1 equation) and Menter Shear-Stress-Transport (SST, 2 equations) models are implemented that are based on Boussinesq's approximation, see Eq. (2.5). With that, the viscosity field is no longer constant in space in case a constant laminar viscosity is prescribed, due to the spatially varying eddy viscosity field.

No adaptions have to be done for the transport equations of the scalar turbulent quantities, nor for their boundary conditions. The turbulence models are solely dependent on the velocity field and not on pressure or temperature, which would require an adaption based on the subdivision in a periodic and linear component.

In the derivation of the additional momentum source term for periodic pressure in Sec. 3.1.1, the circumstance of a zero gradient for the viscosity field is never used, thus no adaptions have to made for the momentum equations.

For the temperature periodicity through the energy equation, the gradient of thermal conductivity is considered to be constant in the derivation of the diffusion term, see Eq. (3.23). The thermal conductivity κ is a function of viscosity, see Eq. (2.6), and as such, also a function of eddy viscosity such that $\nabla\kappa \neq 0$ in turbulent flows. Inserting Eq. (2.6) at Eq. (3.23) alters the derivation of the diffusion term to:

$$\begin{aligned} -\nabla \cdot (\kappa \nabla T) &= -\nabla \cdot \left(\kappa \nabla \tilde{T}\right) - \frac{T_\Delta}{\|\bar{t}\|_2} \nabla\kappa \cdot \bar{t}, && (3.33) \\ &= -\nabla \cdot \left(\kappa \nabla \tilde{T}\right) - \frac{c_p T_\Delta}{\|\bar{t}\|_2} \nabla \left(\frac{\mu_L}{Pr_L} + \frac{\mu_T}{Pr_T}\right) \cdot \bar{t}, && (3.34) \\ &= -\nabla \cdot \left(\kappa \nabla \tilde{T}\right) - \frac{c_p T_\Delta}{\|\bar{t}\|_2 \, Pr_T} \left[\nabla \mu_T \cdot \bar{t}\right]. && (3.35) \end{aligned}$$

Formulating the novel source term contribution in terms of rate of heat flow as introduced in Eq. (3.26) yields:

$$-\frac{c_p T_\Delta}{\|\bar{t}\|_2 \, Pr_T} \left[\nabla \mu_T \cdot \bar{t}\right] = -\frac{\dot{Q}}{\dot{m} \, \|\bar{t}\|_2 \, Pr_T} \left[\nabla \mu_T \cdot \bar{t}\right]. \qquad (3.36)$$

The transport equation for energy in periodic temperature now writes:

$$\frac{\partial\left(\rho c_p \tilde{T}\right)}{\partial t} + \nabla \cdot \left(\rho c_p \tilde{T} \bar{v} - \kappa \nabla \tilde{T}\right) + \frac{\dot{Q}\rho}{\dot{m}\left\|\bar{t}\right\|_2^2}\left[\bar{t} \cdot \bar{v}\right] - \frac{\dot{Q}}{\dot{m}\left\|\bar{t}\right\|_2 Pr_T}\left[\nabla \mu_T \cdot \bar{t}\right] = 0, \qquad (3.37)$$

In conclusion, an additional source term in the energy equation of the Navier-Stokes equations is introduced, but otherwise the transport equations of the turbulence models can be included without further modifications. Note that the source term requires the gradient of the eddy viscosity, which is usually not required in standard FVM codes and thus has to be constructed and stored specifically for this purpose.

3.2 Discrete Adjoint Method

This section starts with selected fundamentals of AD in subsection 3.2.1, as it provides the necessary methods to solve the terms occurring in the adjoint equations that will be derived in subsection 3.2.2. A more in-depth and complete introduction to AD techniques can be found in [GW08]. In order to approach the unsteady multizone discrete adjoint problem, the derivation is made for a steady singlezone problem first, followed by the unsteady singlezone and the steady multizone problem. The additional differentiation of the volume and surface mesh workflow is handled in subsection 3.2.3, followed by remarks on the inclusion of additional fixed point equations that can occur in the streamwise periodic flow implementation in subsection 3.2.4. The section is closed with selected remarks on a memory-efficient implementation of the discrete adjoint solver in subsection 3.2.5.

3.2.1 Fundamentals of Algorithmic Differentiation

Algorithmic Differentiation builds on the fact, that every computer program, how complex it might be, is a mere sequence of elementary operations, e.g. addition, multiplication, sinus-function, etc. Each of those elementary operations is easy to differentiate with respect to their inputs. The chain rule can be applied to all elementary operations in the sequence to compute the required derivatives with machine accuracy. Only first order derivatives are required in this work, but higher order derivatives are possible to be computed with AD as well.

A distinction has to be made to both, symbolic differentiation and finite differences (or often called numerical differentiation). While symbolic differentiation provides exact derivatives, it is usually infeasible to provide the derivative in a mathematical expression for a complex computer program like a CFD-code, with the numerous occurring branches (e.g. every *if*-statement) and sheer number of operations involved. Note, that AD computes the derivative at a specific evaluation point, and the process has to be repeated if another point is chosen. Opposed to the mathematical expression in symbolic differentiation that, once constructed, provides a straight-forward evaluation over a range of inputs. Finite Differences, on the other hand, are straight-forward to apply without any required knowledge of the program's internals (i.e. in a black-box fashion), but due to the discretization error introduced by the user-chosen finite stepsize, the method is not as accurate as the two other. First-order forward differences are used later in this thesis, in Sec. 5.1.3 and 5.2.1, for the gradient validations against the sensitivities computed using the discrete adjoint method. In all methods, when executed with a computer, round-off errors are present due to the inexactness of how real numbers are represented in a computer (e.g. most prominently for this work the *double* datatype).

AD can have two general modes: forward and reverse, which are introduced in the following, where the reverse mode is the important technique for the discrete adjoint method in this work. Within the scope of this thesis, it is not crucial to understand how AD works in all detail, but it is important to understand what AD is capable of, and how to generally operate it.

In the SU2 software package the AD-tool Codipack [SAG19], written in and for C++ code, is used for its performance and its extension packages MeDiPack and OpDiLib [BSG21], which allow differentiating through MPI and OpenMP routines as well. Numerous other AD-tools for most programming languages exists [1].

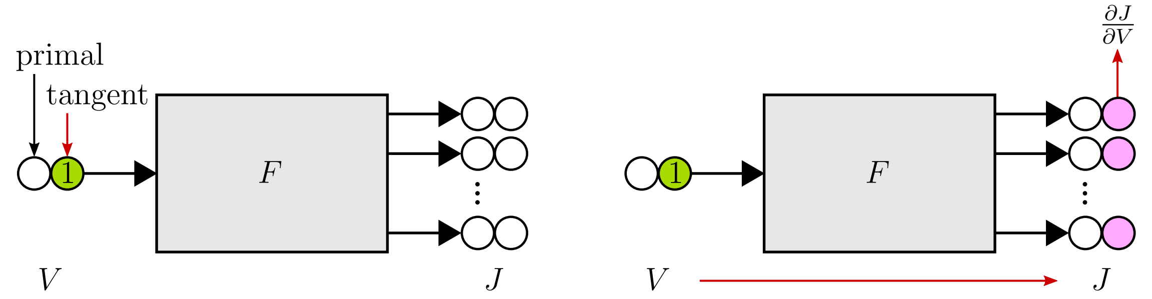
Codipack uses operator overloading AD, with the notable alternative being source code transformation AD. Very briefly, source code transformation AD tools create new, self-sufficient code containing the differentiated primal program, while operator overloading AD tools replace the computation datatype (e.g. double or float) which allows for primal evaluation and derivative computation within the same computer program. Next up, more details including some supporting illustrations on the functioning of forward and reverse mode for a generic operator overloading AD tool will be given.

Forward mode AD

In forward mode (or tangent-linear) AD, the derivative of arbitrary many outputs with respect to a single input (i.e. a scalar value) can be computed within a single program evaluation. In that characteristic, forward mode AD is equal to the complex step

---

[1] See `https://www.autodiff.org/` (Accessed 2022-07-10) for a curated list of AD-software packages.



(a) The tangent of the scalar input is seeded with 1, all other tangents are initialized with 0.

(b) After evaluating F, the desired derivative can be extracted from the tangent of the output.

Figure 3.2: Forward-mode AD.

method, or finite differences with the difference that finite differences requires 2 function evaluations. The sole input quantity has to be chosen a priori, and the sensitivities with respect to the chosen input of all intermediate values are computed alongside the primal value.

The forward mode of AD with an operator overloading tool, is illustrated in figure 3.2. The function F, representing an arbitrary computer program, computes a vector of outputs J based on the scalar input V and the task is to evaluate $\frac{\partial J}{\partial V}$. The usual datatype used in the computation (e.g. float or double) is replaced by a pair of the datatype (e.g. double-double), indicated by the two circles that represent one instance of the overloaded datatype. The value pair stores the regular primal value and derivative value with respect to the chosen input variable, also referred to as *tangent* value and often indicated by a dot over the variable, e.g. $\dot{J}$ is the tangent value to J. Now the tangent value of the input V, $\dot{V}$, is given the value of 1, while all other tangent values in the program are initialized to 0. This process of initialization of the derivative value is also referred to as seeding. During the subsequent evaluation of the program F, the derivative of every intermediate binary or unary operation f with input x and output y is evaluated as:

$$y = f\left(x\right), \quad \dot{y} \mathrel{+}= \frac{\partial f}{\partial x}\dot{x}. \tag{3.38}$$

Analytical derivatives for all occurring binary operations (including 2 operands for 1 output, like multiplication, addition, etc.) and unary operations (including 1 operand for 1 output like negation, trigonometric functions, etc.) have to be implemented in the AD-tool. The sensitivity information is propagated through the program alongside the primal computation, and once the outputs J are computed, the desired derivative $\frac{\partial J}{\partial V}$ can be extracted from the tangent value $\dot{J}$.

In case of multiple inputs, the described process has to be repeated for each scalar input. The increased cost with respect to compute resources can be mitigated by the so-called vector mode, or multi-directional forward mode, which is not further elaborated here. Even in the worst case of many inputs and one output, forward mode AD can be of valuable use: for gradient validation and in the discrete adjoint solver while preaccumulating during the recording phase, see subsection 3.2.5.

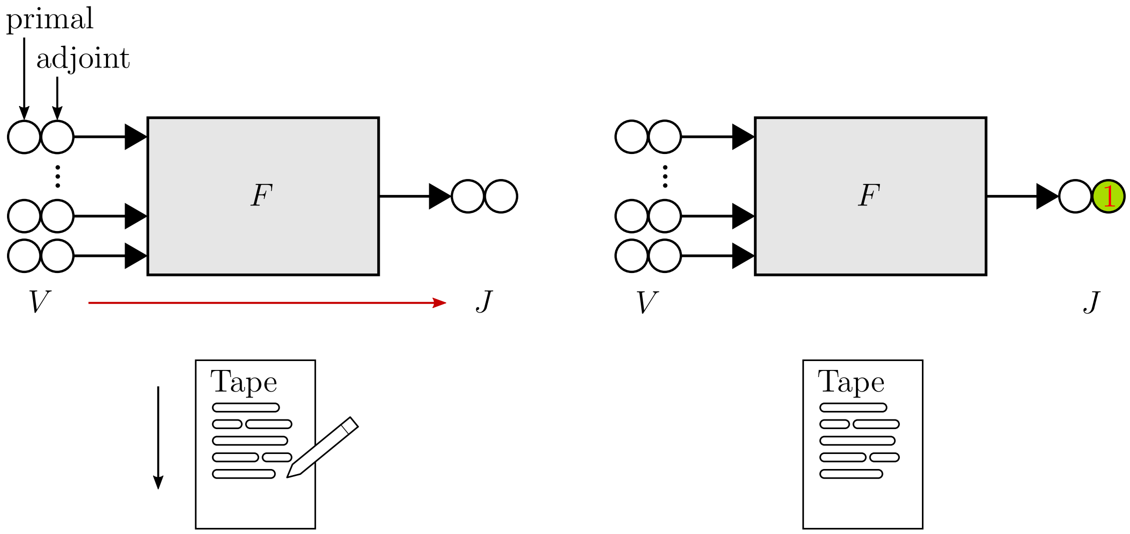
Reverse mode AD

With reverse mode AD, it is possible to compute the derivative of a chosen output with respect to arbitrary many inputs in a single execution. Thereby serving the opposite cause compared to forward mode AD. In contrast to forward mode AD, the derivatives are not computed alongside the execution of the primal program, but a reverse sweep is required in addition to the primal evaluation. Note, that for the purpose of this introduction, some presented concepts are introduced differently to their actual implementation in CoDiPack as used in SU2. These generalizations will be refined at the end of this subsection.

The working principle is explained with the aid of Fig. 3.3. Consider a computer program with the input vector V and the scalar output J. The computational datatype is again overloaded to hold its primal value and its derivative value, or called adjoint value.

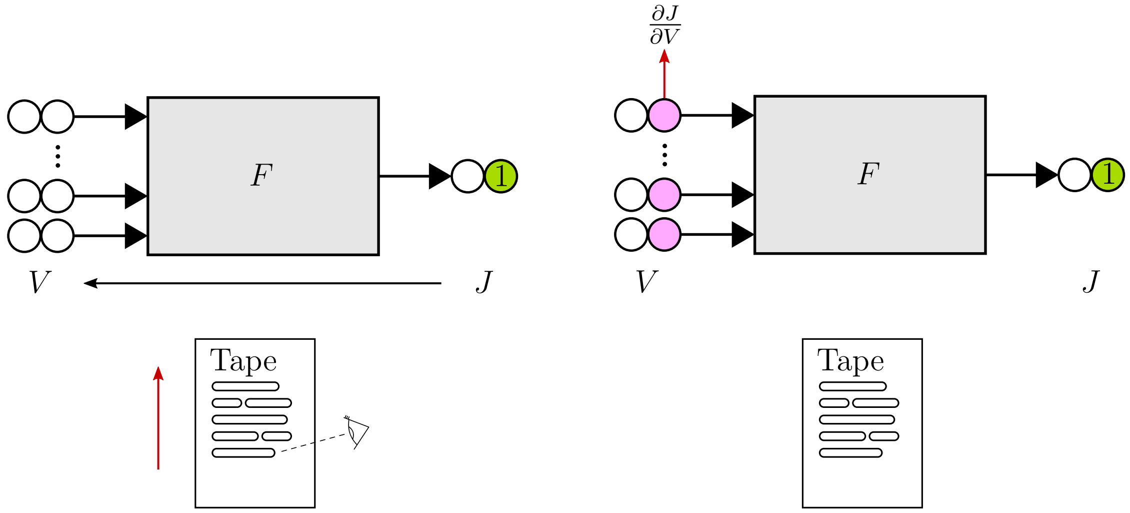
Unfortunately, the term *adjoint variable/value* is used for two different concepts within the scope of this thesis, which requires a clear distinction. Therefore, the just introduced adjoint value used by the AD-tool will be referred to as *AD-adjoint*, and the second adjoint variables introduced in the next subsection for the derivation of the discrete adjoint equations, will retain just that very name. In mathematical formulas, this thesis adopts the common notation for AD-adjoint values of an overbar, e.g. $\bar{x}$ is the AD-adjoint of x.

First, the primal program has to be evaluated. During that primal sweep, the so-called *tape* is recorded. In subfigure 3.3a, the red arrow points out, that the program evaluation is the active part and the black arrow at the tape indicates, that the additions to the tape are a mere consequence. The tape stores information about each computational operation (operation type and operands involved), such that the Jacobian with respect to the inputs of each operation can be provided at a later stage. The tape is a separate data structure that naturally grows with program length and causes an increased memory consumption compared to just a primal evaluation. Once the program evaluation reached the designated scalar output, the AD-adjoint of just that scalar output is set to unity, see subfigure 3.3b. This process is again referred to as seeding. All other AD-adjoint values



(a) Run the primal program and record the tape.

(b) Seed 1 into the AD-adjoint of the output.



(c) Interpret the tape, thus reversing the program flow.

(d) Extract the derivative from the AD-adjoints from the inputs.

Figure 3.3: Reverse mode AD with a scalar output variable and arbitrary many inputs.

are initialized to zero. Subsequently, the statements stored on the tape are interpreted in antichronological order, thus sweeping through the operations of the primal program in reverse order. This hierarchy is again indicated in subfigure 3.3c by a red and black arrow at the tape and program F, respectively. For each primal elementary operation f with inputs x and output y, the contribution to the input's AD-adjoints, $\bar{x}$ are:

$$y = f(x), \quad \bar{x} \mathrel{+}= \frac{\partial f}{\partial x}^T \bar{y}. \tag{3.39}$$

The symbolic derivatives of the elementary operations involved are again provided by the AD-tool itself. Finally, the derivative of the program's scalar output J with respect to the input vector U, can be extracted from the AD-adjoint of the input-vector, see subfigure 3.3d. In case, the sensitivities with respect to multiple outputs are to be computed, the procedure has to be repeated once for each output.

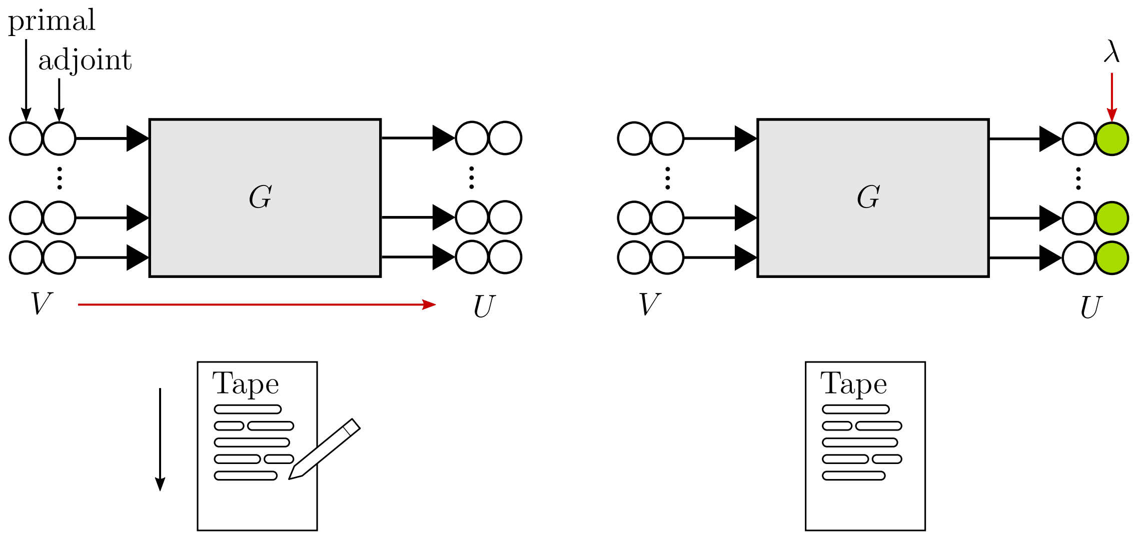
For the discrete adjoint equations that are derived in the next subsection, it is required to solve terms with the following structure

$$\lambda^T \frac{\partial G}{\partial V}, \tag{3.40}$$

with λ, G required to be vectors of identical size, and V a vector with not necessarily the same dimension. The function G represents a fixed point operator for a PDE-solver in this thesis' application, but the presented algorithm works for generalized cases. Ultimately, the generic input vector V will be replaced in the following subsection 3.2.2 by the PDE's solution state and the vector of all mesh node coordinates.

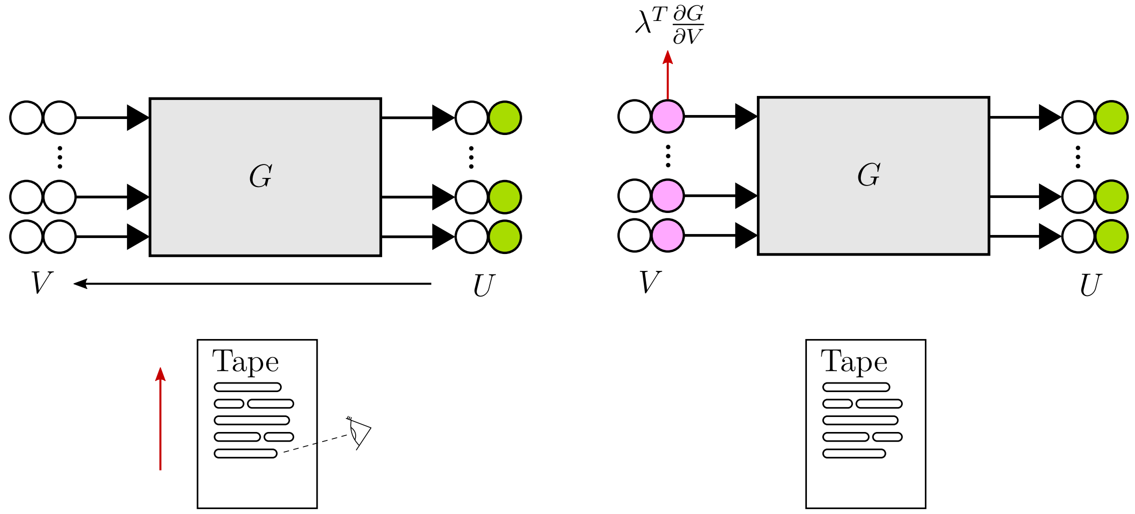
Applying either forward or reverse mode AD as introduced up to now for computing the matrix $\partial G / \partial V$ alone is computationally expensive in case G and V are both of high dimension. Forward mode AD scales with the number of inputs and reverse mode AD with the number of outputs, which renders both impractical for the problem at hand. Instead of evaluating the derivative and then computing the matrix-vector product, the whole expression in Eq. (3.40) can be evaluated efficiently using reverse mode AD in one procedure instead.

The evaluation of terms as in Eq. (3.40) using reverse mode AD, is illustrated in Fig. 3.4. The process is similar to the previous application of reverse mode AD featuring a scalar output, with the exception of the seeding process. After the tape is created alongside the primal evaluation, the vector λ is seeded into the output vector of G instead of just unity. After the interpretation of the tape, the solution of the vector matrix product $\lambda^T \frac{\partial G}{\partial V}$ can be extracted from the input vector. The described algorithm to solve for Eq. (3.40) is the core building block to evaluate the resulting discrete adjoint equations. Complications arise due to the involved multiple time steps and computational zones, that require additional care in their implementation.



(a) Run the primal program and record the tape.

(b) Seed the vector λ into the AD-adjoints of the outputs.



(c) Interpret the tape, thus reversing the program flow.

(d) Extract the derivative from the AD-adjoints from the inputs.

Figure 3.4: Reverse mode AD for a product of a vector and a Jacobian matrix. This scheme represent a core building block of the discrete adjoint in SU2.

With the general mode of operation of reverse mode of AD described above, some clarifications on the implementation in CoDiPack are made in the following, as mentioned at the start of the subsection. For further, in-depth information on CoDiPack, please consult [SAG19]. The AD-adjoint values introduced with the aid of Fig. 3.3a, are not stored alongside in the overloaded datatype itself, but in a separate list owned by the AD-tool. The overloaded datatype only carries the pointer to the location in that list, where the associated adjoint value resides.

There are two notable approaches to implement a tape, that stores the necessary information to be able to provide the required Jacobians during the tape interpretation phase: Jacobi-tape and primal-value-tape. The former computes the Jacobian entries during the primal sweep and only stores those directly on the tape. The latter stores the operation type and the involved operand values on the tape, and the Jacobian is computed during the tape interpretation based on those information. CoDiPack provides both approaches, but the Jacobi tape (in combination with expression templates) is used in SU2 due to its increased runtime efficiency [SAG19].

Expression templates enable the storage of complete assignments onto the tape as opposed to mere statements, i.e. a complex assignment such as $y = sin\left((a+b)*c\right)$ writes one tape entry as opposed to the three statements (addition, multiplication and sine function) for each elementary operation. The associated reduced memory usage and tape interpretation time are an important feature for an efficient operator overloading AD-tool.

So far, the provided illustrations were presented as isolated programs, where in reality, it is required to perform the AD evaluations only over specific parts of a program. In order to save memory, AD-tools provide functionalities to switch tape recordings on and off. In addition, specifically registering input allows to only write program parts on the tape which are dependent on these very inputs. All other computations are not relevant for the derivative computation and thus can only be executed with their primal values, excluding an entry on the tape.

Due to the cyclic properties of fixed point iterations of the primal program examined in this work, which override the solution variables for each iteration, it is not possible to refer to the AD-adjoints of the inputs by a variable in memory anymore. This requires the storage of indices of the inputs on the tape in a separate data structure.

In some literature instances [ASG15; ASG16] the primal program, e.g. G in Fig. 3.4, is considered to be treatable like a black-box using AD, making the development and implementation of the discrete adjoint solver independent of the particular implementation of the primal program. It is certainly true that the discrete adjoint solver has been implemented in a general way within SU2, such that a consistent discrete adjoint for evolutions and extensions of the primal capabilities will be available upon recompilation without the need to manually differentiate the new components. Nonetheless, there are relevant performance features in the discrete adjoint implemented in SU2, that require additional attention and programming effort such that the primal program is, in fact, not treated like a true black-box. The end of this section covers some of these features like external function handling for the linear solver, local preaccumulation in Sec. 3.2.5 or inclusion of additional fixed points in Sec. 3.2.4.

3.2.2 Derivation of Discrete Adjoint Equations

The adjoint method is a highly efficient sensitivity analysis technique that allows the calculation of the derivative of a single objective function (e.g. average temperature, pressure-drop, drag etc.) with respect to an arbitrary number of control parameters (design variables) at a fixed cost that is roughly equivalent to an additional solution of the primal problem. These sensitivities are used for gradient-based design optimization in the scope of this work, but application in the field of topology optimization, uncertainty quantification, etc. are possible, or even for simply gaining intuition into a complex system. With the primal problem formally defined in Sec. 2.2 and 3.1, we can now detail the process of constructing the discrete adjoint via algorithmic differentiation (AD) as implemented within SU2 [ASG15; ASG16]. There are multiple ways to derive the discrete adjoint equations, where here the method of Lagrange multipliers is used in conjunction with the solution residual based on the convergence of a fixed point iteration. The dissertation of Gomes [Ped21] provides a broad discussion of various aspects like derivation approaches, interpretation of adjoint variables, detailed implementation aspects, and more, which is a valuable resource, especially for SU2 developers.

Steady Singlezone Problems

The typical PDE-constrained optimization problem writes:

$$\begin{aligned} \min_{X} \quad & J\left(U, X\right), \\ \text{subject to} \quad & U = G\left(U, X\right), \end{aligned} \tag{3.41}$$

where J, the scalar objective function, depends on the design variables X (potentially a large vector) and the PDE state variables U of the discretized primal problem, e.g. pressure, velocities and temperature at each node for the present incompressible CFD-solver. The converged fixed point with fixed point operator G in the constraint implies a valid solution to the PDE problem. Through the constraint, the PDE state U is an implicit function of the design variables X [MN21]. To compute the objective J for a given set of DV's X, in practice, the PDE state U has to be found first.

For gradient-based optimization, knowledge of the derivative $\frac{dJ}{dX}$ is required. In order to compute said derivative, first the Lagrangian to the above optimization problem is set up, to transform the constrained optimization problem into an unconstrained one:

$$L\left(U, X, \lambda\right) = J + \lambda^{\mathsf{T}} \left[G\left(U, X\right) - U\right], \tag{3.42}$$

where the Lagrange multipliers λ are referred to as adjoint variables in this context. Since $G - U = 0$, it follows that $L = J$ and therefore $\frac{dL}{dX} = \frac{dJ}{dX}$. Consequently, the total derivative of the Lagrangian is taken:

$$\frac{dL}{dX} = \frac{\partial L}{\partial X} + \frac{\partial L}{\partial U}\frac{dU}{dX}. \tag{3.43}$$

Note that $\frac{\partial L}{\partial \lambda} = G - U = 0$. This works uses the convention that vectors, e.g. G, U and X, are considered to be column vectors only, and Jacobians are of dimension $\mathbb{R}^{dim(numerator)\times dim(denominator)}$, e.g. $\frac{\partial a}{\partial b} \in \mathbb{R}^{n\times m}$ with $a \in \mathbb{R}^{n}$ and $b \in \mathbb{R}^{m}$. In the total derivative, it is beneficial to avoid the evaluation of the computationally prohibitively expensive last term $\frac{dU}{dX}$. Sensitivities with respect to U involve the complete solution procedure of the PDE, e.g. the Navier-Stokes equations. G and J are mere operators which are much more straight-forward to differentiate. The DV's are fixed, the PDE solution is implied by the constraint, but the adjoint variables can be chosen freely. In order to avoid computation of $\frac{dU}{dX}$ the adjoint variables are chosen such that:

$$\frac{\partial L}{\partial U} \overset{!}{=} 0, \tag{3.44}$$

which results in what is referred to as the adjoint equation:

$$\frac{\partial J}{\partial U} + \lambda^{\mathsf{T}}\left[\frac{\partial G}{\partial U} - I\right] = 0, \tag{3.45}$$

or equivalently, written in a fixed point formulation for the adjoint variables λ:

$$\lambda^{\mathsf{T}} = \frac{\partial J}{\partial U} + \lambda^{\mathsf{T}}\frac{\partial G}{\partial U}. \tag{3.46}$$

With the iteration scheme being:

$$\lambda^{\mathsf{T}}_{p+1} = \frac{\partial J}{\partial U} + \lambda^{\mathsf{T}}_{p}\frac{\partial G}{\partial U}. \tag{3.47}$$

In accordance with the pseudo time steps of the primal solve, the internal iterations of the adjoint fixed-point iterator are denoted with the subscript p.

By introducing the fixed-point formulation, an adjoint can be constructed that resembles the iterative flow solver and enables the use of the same approximate Jacobian from the primal solve. Through the fixed-point formulation, the adjoint problem inherits the convergence properties of the primal problem [ASG15]. In particular, if the primal fixed point iterator G is contractive, the discrete adjoint fixed point scheme in Eq. (3.47) is contractive.

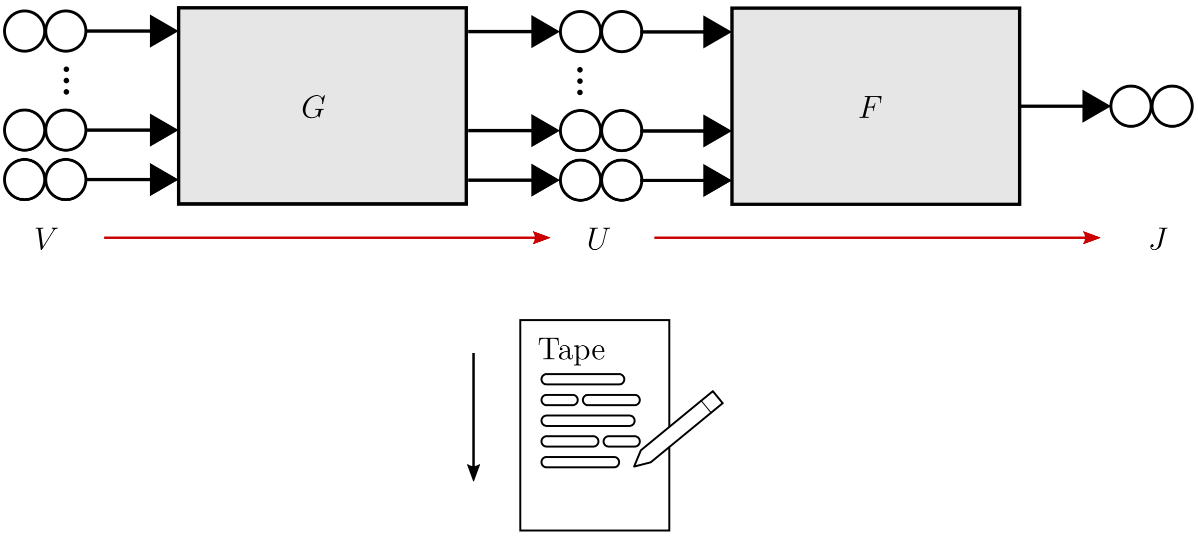
After solving the adjoint equation, the final derivative computes to the remaining term in Eq. (3.43):

$$\frac{dJ}{dX} = \frac{dL}{dX} = \frac{\partial L}{\partial X} = \frac{\partial J}{\partial X} + \lambda^{\mathsf{T}} \frac{\partial G}{\partial X}, \tag{3.48}$$

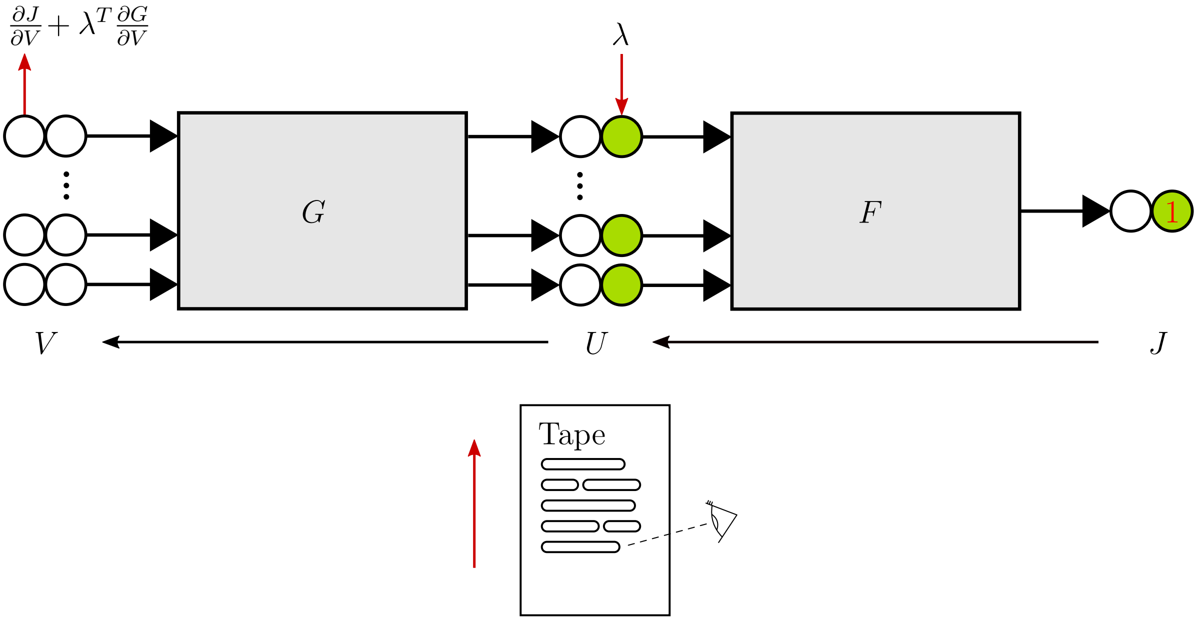
which is just a single evaluation. In conclusion, the adjoint equations (3.46) have to be solved before evaluating the desired derivative via Eq. (3.48). The occurring terms can be evaluated using AD with the methods presented in the previous Sec. 3.2.1.

The partial derivative terms in Eqs. (3.47), (3.48) for J can be evaluated following Fig. 3.3 and the matrix-vector products, including the partial derivative with respect to G, following Fig. 3.4. Instead of evaluating both terms in each equation individually and to add them up (alternatively, it is possible to add the OF gradient to the seed vector of the fixed point's output), it is possible to evaluate the linear combination, i.e. the complete equation, in one sweep. The process is illustrated in Fig. 3.5. Instead of taping the operators J and G individually, the operator G is written to the same tape, prior to J, see Fig 3.5a. Both seeds are set just as in the individual evaluations before, and the full tape is evaluated subsequently, see Fig. 3.5b. The linear combination as e.g. required in the adjoint equations can be extracted from the AD-adjoints of the input to G. At this point, the choice to make a clear distinction between the AD-adjoints and the adjoint variables from the Lagrange approach prevents possible confusion. Consider the evaluation of the above introduced adjoint equation (3.47) using the presented linear-combination approach, where the updated adjoint solution is set to the AD-adjoint variables of the input of G. If the distinction had not been made, a valid statement as *'The adjoint variables are set to the adjoint values'* might be unnecessarily confusing.

It would also be possible, to tape the consecutive evaluation of G and J with both, U and X, as input. This eliminates the need to re-record the tape with respect to X for the final derivative evaluation, but increases the number of entries on the tape that require evaluation during the tape reversal. The adjoint fixed point iteration (3.47), which only requires a tape with the input U, is done potentially thousands of times. It is therefore computationally more efficient to retape with respect to X afterwards which comes at a



(a) Record the fixed point and the objective Function in one sweep on the tape.


$\frac{\partial J}{\partial V} + \lambda^T \frac{\partial G}{\partial V}$
λ
G
F
1
V
U
J
Tape

(b) Seed both outputs, interpret the tape and extract the derivative at the inputs.

Figure 3.5: Evaluation of the discrete adjoint equations for the single zone implementation in SU2. Computation of the linear combination using AD.

constant time cost, compared to the additional tape interpretation cost that grows with the number of iterations to solve the fixed point.

In the above formulated optimization problem (3.41), the vector of design variables X was introduced without further specification. In this work, FFD-boxes are used to deform the surface mesh, which is subsequently used in a linear-elasticity based volume mesh-deformation routine that yields the deformed volume mesh. To be precise, not the updated surface mesh itself is used by the volume mesh-deformation routine, but only the vector containing the resulting deformations of said surface mesh. This vector of surface mesh deformations is denoted with S_Δ. The updated surface mesh is never explicitly used in the design cycle, and therefore the following optimization problem uses the deformation S_Δ as an intermediate result.

Extending the initial optimization problem by this two-stage mesh-deformation yields:

$$\begin{aligned} \min_{X} \quad & J\left(U, X\right), \\ \text{subject to} \quad & U = G\left(U, X\right), \\ & X = M\left(S_\Delta\right), \\ & S_\Delta = O\left(\alpha\right), \end{aligned} \tag{3.49}$$

where the (now redefined) volume mesh X is computed by a routine $M\left(S_\Delta\right)$ with the surface mesh deformations S_Δ as input, which, for its part, is computed by a routine $O\left(\alpha\right)$ that depends on the actual design variables α.

This does not invalidate the more general derivation above, but in the context of SU2, X is considered to be the vector of all mesh node coordinates and the adjoint solution process presented with the help of Fig. 3.5, only computes the sensitivities of the objective function with respect to this very volume mesh. Sec. 3.2.3 discusses the projection of the volume sensitivities onto the design variables in detail, i.e. how to compute the sensitivities with respect to the design variables on the basis of the volume mesh sensitivities. Ultimately, the resulting approach in SU2 allows the computation of sensitivities that are consistent with the discretized primal problem and allow for an exact gradient validation using finite differences in this work, see Sec. 5.1.3 and 5.2.1.

Unsteady Singlezone Problems

The steady single zone discrete adjoint derivation is extended by the concept of multiple timesteps for the unsteady case. Instead of just a single solution vector U to the PDE, the overall solution consists of one solution vector per timestep. The respective timestep will be indicated by a superscript in the following, e.g. U^n is the PDE solution in the n-th timestep, and the state vector containing the solutions from each timestep is denoted with $\mathbf{U} := \left(U^1, \dots, U^N\right)^\mathsf{T}$ where N is the total number of time steps. The physical time integration scheme chosen for this work is dual time stepping 2nd order, as presented in the previous Sec. 2.2.1. This approach leads to a fixed point in each timestep $n = 1..N$ of the form $U^n = G^n\left(U^n, U^{n-1}, U^{n-2}, X\right)$. The following derivation is based on that specific scheme, but the adjustments for dual time stepping schemes of varying order is straight-forward.

Expressing the unsteady PDE-constrained optimization problem changes the initial, steady-state formulation (3.41) to:

$$\begin{aligned} \min_{X} \qquad & J\left(\mathbf{U}, X\right) = \sum_{n=1}^{N} f^n\left(U^n, X\right), \\ \text{subject to} \quad & U^n = G^n\left(U^n, U^{n-1}, U^{n-2}, X\right), \quad n = 1..N, \end{aligned} \tag{3.50}$$

where the objective function J can be expressed as a weighted sum of contributions in each physical time step f^n, that exclusively depend on the timestep's solution U^n and the mesh state X. Any weighting of each timestep's contribution, be it zero to construct an objective function on a restricted time interval, is considered to be included in the function f^n. Note that the description here has been given for fixed-grid problems, for unsteady discrete adjoints with grid motion, the mesh also needs to be treated as time-dependent. The initial solutions U^0 and U^{-1} to start/restart the simulation, are given as constants.

We transform the constrained optimization problem in Eq. (3.50) into an unconstrained problem by forming the Lagrangian L:

$$L\left(\mathbf{U}, X, \boldsymbol{\lambda}\right) = \sum_{n=1}^{N} f^n\left(U^n, X\right) + \sum_{n=1}^{N} \left(\lambda^n\right)^{\mathsf{T}} \left[G^n(U^n, U^{n-1}, U^{n-2}, X) - U^n\right], \tag{3.51}$$

where the vector containing all adjoint solution states, $\boldsymbol{\lambda} := \left(\lambda^1, \dots, \lambda^N\right)^{\mathsf{T}}$, is introduced in the same fashion as $\mathbf{U}$. Note, that the time indexing for the adjoint states is linked to the primal solution, i.e. λ^n is the adjoint state corresponding to the primal solution U^n. Taking the total derivative writes:

$$\frac{dL}{dX} = \frac{\partial L}{\partial X} + \sum_{n=1}^{N} \frac{\partial L}{\partial U^n} \frac{dU^n}{dX}, \tag{3.52}$$

and to avoid the computation of the computationally expensive derivative of each PDE solution state with respect to the mesh, the adjoint solution states are chosen such that:

$$\frac{\partial L}{\partial U^n} \overset{!}{=} 0, \quad n = 1..N. \tag{3.53}$$

Which results in a sequence of adjoint equations to be solved:

$$\frac{\partial f^n}{\partial U^n} + \left(\lambda^n\right)^{\mathsf{T}} \left[\frac{\partial G^n}{\partial U^n} - I\right] + \left(\lambda^{n+1}\right)^{\mathsf{T}} \frac{\partial G^{n+1}}{\partial U^n} + \left(\lambda^{n+2}\right)^{\mathsf{T}} \frac{\partial G^{n+2}}{\partial U^n} = 0, \quad n = 1..N. \tag{3.54}$$

Transformed into a fixed point formulation with iteration subscript p:

$$\left(\lambda^{n}\right)^{\mathsf{T}}_{p+1} = \frac{\partial f^{n}}{\partial U^{n}} + \left(\lambda^{n}\right)^{\mathsf{T}}_{p} \frac{\partial G^{n}}{\partial U^{n}} + \left(\lambda^{n+1}\right)^{\mathsf{T}} \frac{\partial G^{n+1}}{\partial U^{n}} + \left(\lambda^{n+2}\right)^{\mathsf{T}} \frac{\partial G^{n+2}}{\partial U^{n}}, \quad n = N..1. \tag{3.55}$$

Note, that to compute the n-th adjoint state λ^{n}, knowledge of the adjoint solution states λ^{n+1} and λ^{n+2}, corresponding to the primal solutions at timesteps $n+1$ and $n+2$, is required. Therefore, the adjoint solution has to be solved backwards in time, compared to the primal evaluation. This circumstance is reflected in Eq. (3.55) by stating the equation for $n = N..1$. Eq. (3.55) is only strictly correct for $n = N-2..1$, as for $n = N, N-1$, the respective derivative terms of the fixed points G, that exceed the time iteration N, do not occur.

Once the adjoint flow states are computed, the final derivative can simply be evaluated:

$$\frac{dJ}{dX} = \frac{dL}{dX} = \frac{\partial L}{\partial X} = \sum_{n=1}^{N} \left[\frac{\partial f^{n}}{\partial X} + \left(\lambda^{n}\right)^{\mathsf{T}} \frac{\partial G^{n}}{\partial X} \right]. \tag{3.56}$$

The resulting steps for the unsteady adjoint are therefore: compute the primal and store the solution states that are required to accurately restart each fixed point operator for the taping process, solve the adjoint Eqs. (3.55) backwards in time and finally solve (3.56) for the wanted derivative.

In a practical implementation, some efficiency considerations can be made with respect to the computation of the required terms. In order to tape one instance of G^{n} for dual time stepping 2nd order, the solutions of U^{n}, U^{n-1}, U^{n-2} are required as stated in the optimization problem (3.50). The solution states of each time-step are stored on the hard drive and read into memory as necessary. Compared to the overall adjoint solution procedure, the required read/write times for the testcases regarded in this work are negligible and keeping all solution states in memory can quickly consume the available resources. A remedy in checkpointing strategies is discussed in Sec 3.2.5. Note that most current HPC systems provide multiple storage locations to their users, e.g. home, scratch and often specialized ones, with drastic performance implications. Writing numerous small files or individual large files[2] might require a special storage location for best performance and the author recommends reading the local cluster guidelines or getting in touch with the administrators. Both edge cases might occur for this type of unsteady application. The first, if thousands of timesteps are computed for a tiny mesh and each solution is written to disk, and the second, if the solutions for a large mesh are kept in memory and written to disk in a single file for potential further postprocessing.

---

[2] The TU Kaiserslautern cluster Elwetritsch used for this work mentions 10 000 files with sizes below 100 kB for the former case and files larger than 500 GB for the latter `https://elwe.rhrk.uni-kl.de/elwetritsch/Kurse/Largefiles.pdf` (Accessed 2022-08-01).

Now with the above-mentioned solution states for G^n, only the first two terms in the adjoint equation (3.55) can be computed. For the remaining two terms that are constant in the fixed point iteration, evaluation of the fixed points G^{n+1} and G^{n+2} is required. Therefore, these terms are computed and stored in the respective earlier iteration and just added to the fixed point iteration. Similarly, the sum in the final derivative (3.56) is not evaluated in succession to the adjoint equation as this would require again to load the solution states for all fixed points G^n, but the contribution of each time step to Eq. (3.56) is evaluated after the respective adjoint time iteration.

Steady Multizone Problems

For steady multizone problems, the steady singlezone derivation has to be extended by the concept of multiple zones. This work was done by Burghardt [Bur+22] and the provided reference contains extended details on the derivation, implementation, etc. compared to the explanation given here.

In the scope of this work, a zone is considered to be a separate mesh where altering physics, e.g. RANS and heat equation, are solved that are coupled via some boundary interfaces. A zone does not necessarily need to be a connected space, e.g. the 3 individual pin segments in the pin-fin cooling application in Sec. 5.1 are treated as just one zone. The provided derivation allows for a much more generalized view of a zone, in that the type of fixed point in each zone and the coupling between zones is arbitrary. The zone index is indicated by a subscript and similarly to the previous approach for the unsteady singlezone adjoint, the solution states are $\mathbf{U} := (U_1, \ldots, U_M)^\mathsf{T}$, and the adjoint states $\boldsymbol{\lambda} := (\lambda_1, \ldots, \lambda_M)^\mathsf{T}$, considering M distinct zones. In addition, the combined mesh state of the involved zones is $\mathbf{X} := (X_1, \ldots, X_M)^\mathsf{T}$.

Expressing the PDE-constrained optimization problem including multiple zones changes the initial, steady-state formulation (3.41) to:

$$\begin{aligned} \min_{\mathbf{X}} \qquad & J\left(\mathbf{U}, \mathbf{X}\right) = \sum_{m=1}^{M} f_m\left(U_m, X_m\right), \\ \text{subject to} \quad & U_m = G_m\left(\mathbf{U}, \mathbf{X}\right), \quad m = 1..M, \end{aligned} \tag{3.57}$$

where J is a sum of contributions f_m from each zone, that directly only depends on the solution and mesh state of the respective zone. The fixed-point in each zone G_m potentially depends on the solution and meshes state of all zones. In this work, this dependence between zones is due to the interface coupling, where parts of the solution and mesh state from one zone are used to prescribe a boundary condition in the other zone, see Sec. 2.2.4. Setting up the Lagrangian to the optimization problem (3.57):

$$L\left(\mathbf{U}, \mathbf{X}, \boldsymbol{\lambda}\right) = \sum_{m=1}^{M} f_m\left(U_m, X_m\right) + \sum_{m=1}^{M} \lambda_m^{\mathsf{T}} \left[G_m\left(\mathbf{U}, \mathbf{X}\right) - U_m\right]. \tag{3.58}$$

Taking the total derivative with respect to the mesh in zone m yields:

$$\frac{dL}{dX_m} = \frac{\partial L}{\partial X_m} + \sum_{l=1}^{M} \frac{\partial L}{\partial U_m}\frac{dU_m}{dX_l}, \quad m = 1..M. \tag{3.59}$$

Similar to the unsteady singlezone derivation, the adjoint variables are chosen such that:

$$\frac{\partial L}{\partial U_m} \overset{!}{=} 0, \quad m = 1..M, \tag{3.60}$$

as this avoids the evaluation of the computationally expensive derivatives of the solution states with respect the mesh parts. This requirement results in:

$$\frac{\partial f_m}{\partial U_m} + \left(\lambda_m\right)^{\mathsf{T}} \left[\frac{\partial G_m}{\partial U_m} - I\right] + \sum_{\substack{l=0\\ l\neq m}}^{M} \left(\lambda_l\right)^{\mathsf{T}} \frac{\partial G_l}{\partial U_m} = 0, \quad m = 1..M, \tag{3.61}$$

which can be reformulated to a fixed point iteration for the adjoint variables:

$$\left(\lambda_m\right)_{p+1}^{\mathsf{T}} = \frac{\partial f_m}{\partial U_m} + \left(\lambda_m\right)_p^{\mathsf{T}} \frac{\partial G_m}{\partial U_m} + \sum_{\substack{l=0\\ l\neq m}}^{M} \left(\lambda_l\right)^{\mathsf{T}} \frac{\partial G_l}{\partial U_m}, \quad m = 1..M. \tag{3.62}$$

The diagonal entry is listed individually as it carries the iteration subscript p and the cross terms are collected in the sum. After solving the adjoint equation, the final derivative with respect to the mesh in each zone computes to:

$$\frac{dJ}{dX_m} = \frac{dL}{dX_m} = \frac{\partial L}{\partial X_m} = \frac{\partial f_m}{\partial X_m} + \sum_{l=0}^{M} \left(\lambda_l\right)^{\mathsf{T}} \frac{\partial G_l}{\partial X_m}, \quad m = 1..M. \tag{3.63}$$

The resulting sensitivities with respect to each zonal mesh need to be projected onto some set of design variables that unify the contributions from all zonal meshes. A straightforward deformation based on the mesh sensitivities in the context of a CHT simulation, will most likely result in a non-matching interface that invalidates the solution. The handling of the contributions from multiple zones towards design variables is discussed in Sec. 3.2.3.

The implementation of the steady-state multizone discrete adjoint is significantly more complex compared to its singlezone counterpart, that was explained with the help of

Fig. 3.5. In order to efficiently evaluate the objective function gradient, diagonal and cross terms occurring in Eqs. (3.62) and (3.63), it is necessary to either create multiple tapes for the individual fixed points or to allow for partial evaluation of a global tape. The latter approach was implemented in SU2 by Burghardt [Bur+22] and consists of separating the full recording into sections containing the registration of input variables, dependencies capturing all pre- and postprocessing routines, objective function evaluation, data transfer between zones interface and lastly a part for each primal iteration including output registration. This separation allows for individual evaluation of the occurring terms, which in turn requires additional storage capacities for the objective function and the sum of cross terms as they have to be computed and stored prior to the adjoint fixed point iteration of a zone.

In the adjoint Eqs. (3.62), the cross terms and objective function derivative have to be available, and as it is not necessary to have all terms available separately, it is sufficient to add all these external contributions up into one data structure for each zone which is consequently named *External* in SU2. In the following unsteady multizone derivation, the additional dual time contributions are added to *External* as well.

Due to the cyclic dependency between the adjoint solutions of the zones through the cross terms, it is necessary to iteratively update the adjoint solutions in all zones and to update the cross terms for each zone based on the updated adjoint solution. In a straight-forward implementation, each zone performs one (inner) iteration of Eq. 3.62 before communicating updated cross terms (referred to as outer iteration). Prescribing multiple inner iterations is possible within SU2, but for the CHT applications presented in this work, one inner iteration only was used, as no benefit of multiple inner iterations could be observed.

Unsteady Multizone Problems

For unsteady multizone problems, the required modifications for the discrete adjoint derivation of multiple timesteps and multiple zones are brought together. This combination of complexities was pioneered by Crome [Ven20; VP20] for time-domain fluid-structure interaction problems. The indexing is kept consistent with the previous derivations, in so far as the subscript is used for zones and superscripts for timesteps. A total of M zones and N timesteps are considered. The aggregated solution states are defined as:

$$\mathbf{U} \coloneqq \left(U_1^1, \dots, U_1^N, U_2^1, \dots, U_2^N, \dots, U_M^1, \dots, U_M^N\right)^{\mathsf{T}}, \tag{3.64}$$

and additionally, just the solution of a specific timestep over all zones as:

$$\mathbf{U}^n \coloneqq \left(U_1^n, \dots, U_M^n\right)^{\mathsf{T}}. \tag{3.65}$$

The adjoint states are defined consistent to the solution states:

$$\boldsymbol{\lambda} \coloneqq \left(\lambda_1^1, \dots, \lambda_1^N, \lambda_2^1, \dots, \lambda_2^N, \dots, \lambda_M^1, \dots, \lambda_M^N\right)^{\mathsf{T}}. \tag{3.66}$$

The mesh state of the zones, which are considered constant in time in this work, is $\mathbf{X} \coloneqq (X_1, \dots, X_M)^{\mathsf{T}}$.

Expressing the unsteady multizone PDE-constrained optimization problem changes the initial, steady-state formulation (3.41) to:

$$\begin{aligned} \min_{\mathbf{X}} \qquad & J\left(\mathbf{U}, \mathbf{X}\right) = \sum_{n=1}^{N} \sum_{m=1}^{M} f_m^n \left(U_m^n, X_m\right), \\ \text{subject to} \quad & U_m^n = G_m^n \left(\mathbf{U}^n, U_m^{n-1}, U_m^{n-2}, \mathbf{X}\right), \quad m = 1..M, \quad n = 1..N, \end{aligned} \tag{3.67}$$

where the objective function consists of granular contributions in each timestep and zone, with f_m^n containing a potential weighting factor/function as already described for the unsteady singlezone adjoint. The primal fixed point G_m^n for a specific timestep n and zone m potentially depends on the solution states and meshes from all other zones due to the interface coupling, with the restriction that solution states of previous timesteps are only required from the current zone m itself. Note that this circumstance:

$$\frac{\partial G_m^n}{\partial U_l^{n-x}} = 0, \qquad l \neq m \wedge x > 0, \tag{3.68}$$

i.e. that a fixed point from zone m does not depend on previous solution states from other zones, will allow for a rather straight-forward extension of the multizone approach by the unsteady regime. This a posteriori motivates the introduction of $\mathbf{U}^n$ that collects all zonal solution states of one timestep, as it facilitates the formulation of the optimization problem.

The derivation does not deviate from the previous concepts employed for singlezone unsteady and steady multizone cases, but the following discussion of the implications on implementation is the principal concern. Setting up the Lagrangian:

$$L\left(\mathbf{U}, \mathbf{X}, \boldsymbol{\lambda}\right) = \sum_{n=1}^{N} \sum_{m=1}^{M} f_m^n \left(U_m^n, X_m\right) + \sum_{n=1}^{N} \sum_{m=1}^{M} \left(\lambda_m^n\right)^{\mathsf{T}} \left[G_m^n \left(\mathbf{U}^n, U_m^{n-1}, U_m^{n-2}, \mathbf{X}\right) - U_m^n\right]. \tag{3.69}$$

Taking the total derivative with respect to one zonal mesh part:

$$\frac{dL}{dX_m} = \frac{\partial L}{\partial X_m} + \sum_{n=1}^{N} \sum_{l=1}^{M} \frac{\partial L}{\partial U_m^n} \frac{dU_m^n}{dX_l}, \qquad m = 1..M, \tag{3.70}$$

and choosing the adjoint variables such that the following equation holds, eliminates the need to evaluate the computationally expensive derivatives of the solution states with respect to the mesh parts:

$$\frac{\partial L}{\partial U_m^n} \overset{!}{=} 0, \quad m = 1..M, \quad n = 1..N. \tag{3.71}$$

This requirement expands to:

$$\frac{\partial f_m^n}{\partial U_m} + \left(\lambda_m^n\right)^\mathsf{T} \left[\frac{\partial G_m^n}{\partial U_m^n} - I\right] + \left(\lambda_m^{n+1}\right)^\mathsf{T} \frac{\partial G_m^{n+1}}{\partial U_m^n} + \left(\lambda_m^{n+2}\right)^\mathsf{T} \frac{\partial G_m^{n+2}}{\partial U_m^n} + \sum_{\substack{l=0 \\ l\neq m}}^{M} \left(\lambda_l^n\right)^\mathsf{T} \frac{\partial G_l^n}{\partial U_m^n} = 0, \tag{3.72}$$

for $m = 1..M$ and $n = 1..N$. The equation reformulates into a fixed point iteration for the adjoint variables in every timestep and every zone:

$$\left(\lambda_m^n\right)_{p+1}^\mathsf{T} = \frac{\partial f_m^n}{\partial U_m} + \left(\lambda_m^n\right)_p^\mathsf{T} \frac{\partial G_m^n}{\partial U_m^n} + \left(\lambda_m^{n+1}\right)^\mathsf{T} \frac{\partial G_m^{n+1}}{\partial U_m^n} + \left(\lambda_m^{n+2}\right)^\mathsf{T} \frac{\partial G_m^{n+2}}{\partial U_m^n} + \sum_{\substack{l=0 \\ l\neq m}}^{M} \left(\lambda_l^n\right)^\mathsf{T} \frac{\partial G_l^n}{\partial U_m^n}. \tag{3.73}$$

Upon inspection, the adjoint iteration of the steady multizone approach in Eq. (3.62) is merely extended by the two additional contributions from previous timesteps of the regarded zone. Obviously, the reversal of the primal timestepping for the discrete adjoint applies just as for the unsteady singlezone adjoint in Eq. (3.55). The above-mentioned absent multizone cross terms due to previous solution states in Eq. (3.68), simply put, allows to wrap the approach developed for the unsteady singlezone discrete adjoint around the framework for multizone discrete adjoint without having to deal with a new type of term. This becomes even more apparent in the evaluation of the final derivative of the objective with respect to a mesh part. Compared to the steady multizone Eq. (3.63), the equation is wrapped in a loop over all timesteps with the additional required superscripts to indicate said timesteps:

$$\frac{dJ}{dX_m} = \frac{dL}{dX_m} = \frac{\partial L}{\partial X_m} = \sum_{n=1}^{N} \left[\frac{\partial f_m^n}{\partial X_m} + \sum_{l=0}^{M} \left(\lambda_l^n\right)^\mathsf{T} \frac{\partial G_l^n}{\partial X_m}\right], \quad m = 1..M. \tag{3.74}$$

In the implementation, each zone therefore organizes the inclusion of the unsteady contributions individually. In fact, this decentralized organization allows each zone to use a different time integration method, if required. As for the unsteady singlezone adjoint, the contributions of each timestep in the final derivative (3.74) are evaluated after the converged adjoint variables for that very timestep are present, as all the required primal solution states to tape the program are present in memory. Similarly, all considerations concerning primal solution storage and loading remain unchanged to the unsteady singlezone section.

3.2.3 Projecting Sensitivities onto Design Variables

At this stage, sensitivities with respect to the volume mesh coordinates were computed. Although it is theoretically possible to directly deform the volume mesh based on these sensitivities for every mesh node coordinate, the method is most likely to be highly unstable and will quickly result in self-intersecting meshes or otherwise nonphysical mesh states with respect to a real application. In that sense, the projection on some more coarse-grained design variables can be understood as a smoothing step for the raw volume mesh node sensitivities. It is very well within reason though to validate the volume mesh sensitivities directly using finite differences, or any other suitable method, as the whole design chain is of course more prone to additional error sources.

In SU2, first a surface mesh is deformed by a set of design variables, e.g. FFD-boxes in this work, and the resulting surface mesh deformations act as a Dirichlet boundary condition[3] for a linear-elasticity based volume mesh deformation algorithm. Therefore, the sensitivities are projected onto the set of design variables in two additional process steps, to yield the gradient that can be utilized for shape optimization. The linear elasticity based volume mesh deformation is differentiated analytically without the help of AD and for the final projection from the surface mesh towards the design variables, AD is applied in a straight-forward fashion.

For both, surface mesh and volume mesh, an update (X_Δ, S_Δ) to the initial mesh (X_0, S_0) is computed that is added onto said initial mesh to obtain the updated mesh (X, S):

$$X = X_0 + X_\Delta, \tag{3.75}$$

$$S = S_0 + S_\Delta. \tag{3.76}$$

The initial meshes are given naturally as a constant and the volume mesh obviously contains the surface mesh in both, initial and updated state.

From Volume Mesh Sensitivities to Surface Mesh Sensitivities

The necessary steps from volume mesh sensitivities towards final design variable sensitivities are independent on whether the simulation was steady or unsteady. This might not be generalizable to cases with a non-constant mesh. For multizone cases, the outlined process is done for each mesh part X_m independently, and the unification of sensitivity contributions is done in the subsequent projection to the design variables.

---

[3] Note that this is not strictly true for symmetry conditions that are allowed to move in their respective symmetry plane.

At this stage, the derivative of the objective function with respect to the volume mesh is known, so the chain is applied to the derivative of the objective function with respect to the surface mesh update:

$$\frac{dJ}{dS_\Delta} = \frac{dJ}{dX}\frac{dX}{dS_\Delta}. \tag{3.77}$$

To be precise, SU2 only deforms a subset of the complete surface mesh that can be specified by the user, not necessarily the complete surface mesh. The remaining surface boundaries remain unchanged, but this circumstance does not change the derivation.

The volume mesh nodes X are governed by an algorithm, that is denoted by M with $M(S_\Delta) = X$. In the case of SU2, this represents a linear-elasticity based mesh deformation algorithm that solves one linear system of the form:

$$AX_\Delta = \widetilde{S_\Delta} \rightarrow X_\Delta = A^{-1}\widetilde{S_\Delta}, \tag{3.78}$$

followed by the straight-forward mesh update given in Eq. (3.75). S_Δ is the update to the surface mesh computed by the FFD-algorithm and $\widetilde{S_\Delta}$ extends that vector by zeros for all volume mesh points, in order to achieve correct vector dimensions. The system matrix A is a function of the initial surface mesh S_0 as it uses inverse cell volume or wall distance[4] which includes the node positions of the original mesh, which includes the surface. The system matrix A specifically does not depend on the update to the surface mesh S_Δ, as everything is based on the initial mesh.

The chain rule in Eq. (3.77) can be transformed by expanding the volume mesh by its specification:

$$\frac{dJ}{dS_\Delta} = \frac{dJ}{dX}\frac{d}{dS_\Delta}\left(X_0 + X_\Delta\right) = \frac{dJ}{dX}\frac{d}{dS_\Delta}\left(X_\Delta\right) = \frac{dJ}{dX}\frac{d}{dS_\Delta}\left(A^{-1}\widetilde{S_\Delta}\right), \tag{3.79}$$

where the derivative with respect to the constant initial mesh is zero and the update of the volume mesh was replaced by Eq. (3.78). Applying the product rule yields:

$$\frac{dJ}{dX}\frac{d}{dS_\Delta}\left(A^{-1}\widetilde{S_\Delta}\right) = \frac{dJ}{dX}\left(\frac{dA^{-1}}{dS_\Delta}\widetilde{S_\Delta} + A^{-1}\frac{d\widetilde{S_\Delta}}{dS_\Delta}\right) = \frac{dJ}{dX}\left(A^{-1}\frac{d\widetilde{S_\Delta}}{dS_\Delta}\right), \tag{3.80}$$

where A, as introduced above, does not depend on S_Δ and thus the derivative vanishes. Lastly, the remaining terms are rearranged:

$$\frac{dJ}{dX}\left(A^{-1}\frac{d\widetilde{S_\Delta}}{dS_\Delta}\right) = \left(\left(A^T\right)^{-1}\frac{\partial J}{\partial X}^T\right)^T\frac{d\widetilde{S_\Delta}}{dS_\Delta} = \frac{dJ}{dS_\Delta}, \tag{3.81}$$

---

[4] Young's modulus, a measure of the stiffness of the material, is computed based on either inverse element volume or wall distance, see [Eco14] for further information.

using associativity of matrix multiplications and for an invertible matrix one has $\left(A^{-1}\right)^T = \left(A^T\right)^{-1}$. The terms in the transposed brackets in Eq. (3.81) can be interpreted as a solution to a linear system that needs to be solved prior to the multiplication with the outside matrix. The solution procedure consequently is to first assemble the system matrix A to then solve the derived linear system with the known vector $\frac{dJ}{dX}^T$ on the right-hand side. The solution vector of that linear system is then multiplied with the matrix $\frac{\partial \widetilde{S_\Delta}}{\partial S_\Delta}$ which can be interpreted as an extraction matrix. Extraction matrix, because it reshapes the solution vector from the dimension of the volume mesh vector to the surface mesh vector and extracts all the sensitivities associated with the update to the surface mesh.

The sensitivity of the update to the surface mesh can be computed without the use of AD. This is possible as the algorithm to compute the volume mesh from the surface mesh is a single linear system solve. Thus, the self-adjoint nature of linear systems can be used.

From Surface Mesh Sensitivities to Design Variable Sensitivities

Now that the surface mesh sensitivities $\frac{dJ}{dS_\Delta}$ are computed, the final step towards the design variable sensitivities can be done. The chain rule of the derivative of the objective function with respect to the design variables separated into the already known and unknown parts writes:

$$\frac{dJ}{d\alpha} = \frac{dJ}{dS_\Delta}\frac{dS_\Delta}{d\alpha}. \tag{3.82}$$

In order to compute the present matrix vector product, AD is applied as introduced in Fig. 3.4. The program to compute the surface mesh update $S_\Delta = O\left(\alpha\right)$ is recorded after registering α as input. Then the AD-adjoints of S_Δ (i.e. $\overline{S_\Delta}$) are seeded with $\frac{dJ}{dS_\Delta}$. The tape is interpreted, and the final derivative is extracted at $\bar{\alpha}$.

For multizone cases, this procedure is repeated for each zone independently and the zonal sensitivity vectors are simply added up as just one global set of design variables is used that acts on all zones:

$$\frac{dJ}{d\alpha} = \sum_{m=1}^{M} \left(\frac{dJ}{d\alpha}\right)_m. \tag{3.83}$$

This concludes the general procedure for sensitivity evaluation of a scalar objective function with respect to a set of design variables via the discrete adjoint method, as it is available in SU2. In the following, a number of more specific aspects concerning the discrete adjoint method via AD are discussed.

3.2.4 Handling Additional Fixed-Point Solutions

In the present work, an additional fixed point iteration occurs for streamwise periodic flow with a prescribed massflow, where a pressure drop is iterated until a user-defined global massflow is achieved, see Eq. (3.10). The effects on the gradient validation, of not taking this additional fixed point equation into account, are discussed with the help of a steady state testcase presented in Sec. 5.1.4.

In the previous derivation of the discrete adjoint method, the fixed point of the PDE state variables was introduced as a constraint to the optimization problem, e.g. in Eq. (3.41). Thus far, this constraint only includes the field solution, e.g. pressure, velocity and temperature at each mesh node for the incompressible Navier-Stokes solver. In case an additional fixed point iteration is solved in the overall solution procedure, even for an intermediate quantity, it has to be treated like the field solution using AD. That means for the implementation: input and output have to be registered on the tape, an additional adjoint variable is associated to the primal value and the seeding at the output with said adjoint variable plus extraction at the input has to be performed. Formally, this can be included by extending the solution vector U, and thus the adjoint vector λ, by the additional variable and the remaining derivation is unchanged. An equivalent approach is to add the additional fixed point equation as a further constraint to the optimization problem, which closely resembles the steady multizone discrete adjoint derivation.

For the sake of brevity, the extension to a steady singlezone optimization problem will be discussed here as the here considered additional fixed point only directly depends on and affects a single zone such that an extension to multizone or unsteady cases is possible without any further modification. The optimization problem with the additional fixed point equation writes:

$$\begin{aligned} \min_{X} \quad & J\left(U, P_{\Delta}, X\right), \\ \text{subject to} \quad & U = G\left(U, P_{\Delta}, X\right), \\ & P_{\Delta} = A\left(U, P_{\Delta}, X\right), \end{aligned}$$

where P_{Δ} represents the pressure drop for the streamwise periodic flow which its associated fixed point operator A. See Eq. (3.10) for the exact form of the fixed point iteration for streamwise periodic pressure drop. Following the same approach as for the optimization problem of the steady multizone Sec. 3.2.2, the following adjoint equations have to be solved:

$$(\lambda_1)^\mathsf{T}_{p+1} = \frac{\partial J}{\partial U} + (\lambda_1)^\mathsf{T}_p \frac{\partial G}{\partial U} + (\lambda_2)^\mathsf{T} \frac{\partial A}{\partial U}, \tag{3.84}$$

$$(\lambda_2)^\mathsf{T}_{p+1} = \frac{\partial J}{\partial P_\Delta} + (\lambda_1)^\mathsf{T} \frac{\partial G}{\partial P_\Delta} + (\lambda_2)^\mathsf{T}_p \frac{\partial A}{\partial P_\Delta}. \tag{3.85}$$

With the final derivative analogously being:

$$\frac{dJ}{dX} = \frac{dL}{dX} = \frac{\partial J}{\partial X} + (\lambda_1)^\mathsf{T} \frac{\partial G}{\partial X} + (\lambda_2)^\mathsf{T} \frac{\partial A}{\partial P_\Delta}. \tag{3.86}$$

In contrast to the previously presented multizone approach, no additional transfer routines, dependencies etc. have to be individually managed, which allows us to record all operations onto the same tape without explicit tape sectioning and the linear combinations in Eqs. (3.84), (3.85) and (3.86) are evaluated simultaneously after seeding both adjoint states.

In general, the expectation of being able to treat the primal fixed point as a black-box remains true, but additional fixed point equations might arise in different contexts that are not directly obvious to developers/users. Therefore, this possibility should be taken into consideration when debugging a failing gradient validation for an added feature. A helpful intermediate step to check is whether the residuals of the primal iteration after the taping process are in-line with the converged primal ones. If an additional variable needs to be stored into a solution file to achieve perfect restarts, then this variable most likely has to be treated as presented in this section.

A well-known example in CFD adjoint applications for a deliberately ignored additional fixed point is the *frozen turbulence* approach [SSP18], where the eddy viscosity field is considered to be given as a constant in the derivation of the adjoint equations. Especially in continuous adjoint applications, the frozen turbulence approach is often employed, as the differentiation of turbulence models is cumbersome and not always possible. Following that naming convention for the application in this work, it would be possible to deliberately use a *frozen pressure drop* approach by not treating the additional fixed point, but with discrete adjoint methods using AD there is no apparent reason to do so.

3.2.5 Memory-efficient implementation

Compared to a regular primal simulation, the memory requirements for the associated discrete adjoint computation is increased, due to the tape storing the computational graph, as well as the additional fields that need to be stored for the adjoint solutions and intermediate containers that are required for multizone or unsteady adjoints. This subsection highlights three techniques that can be used to reduce memory and disk storage requirements in certain situations.

Handling of the Linear Solver

When solving the Navier-Stokes equations or other PDE's, often an implicit Euler step is used to discretize the pseudo-time derivative, which requires setting up and solving a linear system of equations. This solution procedure for this linear system of equations of the general form:

$$Ax = b \tag{3.87}$$

is not taped during the one restarted primal iteration for the discrete adjoint, as it is implemented in SU2. The linear solve involves a great number of operations that would increase tape size considerably. The linear system of equations is self-adjoint, meaning we can solve the following equations [Gil08] to compute the AD-adjoints of the inputs, i.e. $\bar{A}$ and $\bar{b}$, instead:

$$A^T\bar{b} = \bar{x}, \tag{3.88}$$

$$\bar{A} = -\bar{b}x^T. \tag{3.89}$$

The AD-adjoints of the output, i.e. $\bar{x}$, are computed by the AD-tool in the tape reversal process and are known at this stage. The AD-adjoints of the right-hand side, i.e. $\bar{b}$, are the solution of a linear system with the already constructed system matrix A which is solved similarly to the primal solve. The AD-adjoints for the system matrix, i.e. $\bar{A}$, do not need to be computed as x, the solution to the primal linear system, represents the update to the solution variables which becomes zero if the fixed point is converged.

In conclusion, the linear solve is not taped and during reverse accumulation, the linear system in Eq. (3.88) is solved to retrieve the required AD-adjoints. The AD-tool needs to support external function handling, i.e. allowing the programmer to provide the Jacobians for specific areas of the code.

Local Preaccumulation

For the local preaccumulation method, the local Jacobi matrices of specific code sections are evaluated during the taping process, instead of storing the Jacobians for each individual statement (or expression) and processing those during the reverse accumulation. Preaccumulation reduces the memory requirements if the respective code section has only a few in- and outputs, but is comprised of many computational statements that would require tape entries[5]. The computational cost to evaluate these local Jacobi matrices becomes negligible compared to the benefits from memory savings with increasing iterations necessary to solve the adjoint fixed point. The Jacobian is evaluated with AD in reverse mode itself, if a preaccumulation section has more (or equal) inputs than outputs, and AD in forward mode in the remaining cases.

Note that the programmer has to identify suitable code sections and define inputs and outputs manually, which constitutes added effort and an additional error source. In doubt, it is therefore reasonable to deactivate local preaccumulation altogether when debugging a failing gradient validation.

Checkpointing

The last discussed memory/disk space reduction technique in checkpointing makes adjoints feasible for long simulations with memory-intensive tapes, by storing only a fixed number of checkpoints during the primal run. Intermediate sections of the full simulation can be restarted from these checkpoints, which are only then fully taped and the adjoints are evaluated on these intermediate sections. This process of subsequently adjoining parts of the program between the respective checkpoints can greatly reduce peak memory consumption.

Checkpointing can be applied generally to AD programs, but time-stepping procedures are particularly well suited if the computational cost of all time-steps is comparable as the individual time-steps provide natural locations for checkpoints. The most straightforward technique is uniform checkpointing where checkpoints are set in fixed time intervals, but there exists a nonuniform schedule for an a priori number of time-steps, that is provably optimal with respect to total number of forwards steps and checkpoint writes for a fixed number of available checkpoints, developed by Griewank and Walther [GW00]. Chapter 12 of the book [GW08] *Evaluating Derivatives* of the aforementioned authors is recommended at this point for further reading. In case of time-stepping schemes with not a priori known number of time-steps, dynamic checkpointing schemes exist, e.g. [WMI09].

For time-stepping schemes of higher order, e.g. more than just the previous time-step solution is required for restart, a checkpoint has to be comprised of multiple time-step solutions. In the present work that uses dual time-stepping 2nd order, a restart requires the solutions of two time-step.

---

[5] See `https://www.scicomp.uni-kl.de/codi/d6/de7/Example_15_Preaccumulation_of_code_parts.html` (Accessed 2022-08-04) for the associated CoDiPack Tutorial showcasing the possible memory savings.

4 Implementation, Sustainability and Maintainability Aspect

The first section 4.1 of this chapter points out the code contributions made by the author to the SU2-project. In the second section 4.2 the efforts to ensure maintainable and thus sustainable code are highlighted. The third and last section 4.3 is dedicated to result reproducibility with respect to the simulation results presented in the subsequent verification and application chapter 5. Note that a clear separation of the treated subjects in this chapter is not fully possible, and rather are interconnected to a certain degree.

4.1 Contributed code

Arguably the most direct way to support a coding project like SU2 and to ensure its longevity, is through reviewed code contributions. The reviewing process deployed in SU2 was introduced in Sec. 2.1.2. In the following, an overview over the total numbers[1] with respect to the authors code contributions to the overall project is collected, before then highlighting selected parts.

The author has 35 merged Pull Requests[2] in the main code repository of SU2. Within these Pull Requests over a 1000 discussion entries were made from various community members, including review comments, code suggestions, explanations by the author etc. Note that the content in these Pull Requests does not represent all contributions to the code, as commits were made to other Pull Requests, that were not opened by the author himself[3]. A total of 712 commits with 44 692 added and 56 263 deleted lines[4] were made to the main SU2 code repository. A change to an existing line counts as one deleted and one added line.

---

[1] All numeric data in this section is collected at the 5th of July 2022.

[2] `https://github.com/su2code/SU2/pulls?q=is%3Apr+is%3Amerged+author%3ATobiKattmann+`

[3] See `https://github.com/su2code/SU2/pull/1226` for an example of a PR abandoned by its original author that was finished by the author.

[4] `https://github.com/su2code/SU2/graphs/contributors`

Neither of the presented metrics are conclusive with respect to the involvement in the code development. Pull requests can contain only a one-line bugfix that was found, fixed and merged within one day, or the PR contains combined unrestricted work of half a year. The number of total commits also is subject to the individual developer's style, whether progress is committed in smaller or larger chunks. And number of lines are quickly misleading in large refactoring efforts, where big code blocks (e.g. whole classes) are moved to different files but the code itself is not altered. Another example is falsely committed mesh files that quickly contribute to added and removed lines up to multiple millions. However, taking all metrics together and with a short overview over the actual contributions and their consistency over time, it is reasonable to draw a conclusion on the activity (or passivity) of an individual developer.

In addition to the contributions to the main code repository, the SU2 project features multiple other repositories that complement the raw simulation code, as introduced in Sec. 2.1.1. The author contributed to the Testcase repository with 11 merged Pull requests[5], most of which representing an added regression test. 8 Pull Requests were made to the Tutorials repository[6]. The website, containing the documentation, tutorial descriptions, etc., was supported with 11 merged Pull Requests[7].

All above-mentioned contributions were made to the SU2 project itself, but as introduced in section 2.1.1, various libraries are included in the code that SU2 depends on. Hence, SU2 does not work on its own in a vacuum, but is embedded in, and surrounded by other pieces of software, and ensuring smooth interoperability between software components is a key factor. The author contributed one Pull Request each to MeDiPack[8], which handles AD for nearly all Message Passing Interface (MPI) functions, and to FADO[9], which is used as the optimization framework for the optimization runs performed in section 5.1.5 and 5.2.3. Making relevant changes to a third-party repository that are merged by the maintainer requires an increased effort compared to an accustomed repository and therefore need to be assessed differently. Code layout, employed data-structures, programming paradigms, etc. can differ quite drastically to the known concepts from the familiar projects. Even a small fix or addition to a foreign repository can therefore take a multiple of time compared to a similar contribution to the own main

---

[5] https://github.com/su2code/TestCases/pulls?q=is%3Apr+is%3Amerged+author%3ATobiKattmann+

[6] https://github.com/su2code/Tutorials/pulls?q=is%3Apr+is%3Amerged+author%3ATobiKattmann+

[7] https://github.com/su2code/su2code.github.io/pulls?q=is%3Apr+is%3Amerged+author%3ATobiKattmann+

[8] https://github.com/SciCompKL/MeDiPack

[9] https://github.com/pcarruscag/FADO/

project.

The symmetry boundary condition was completely redesigned and fixed to fulfill the required conditions from Eqs. (2.31)-(2.33). The previous implementation was incorrect, and the updated version is now utilized in the compressible and incompressible solver. The elementary unit-cell approach used for the pin-array evaluation in Sec. 5.1 depends on a correct symmetry boundary.

Another integral contribution for a well performing unit-cell approach are the additional source terms, adapted boundary conditions, etc. for streamwise periodic flow. The accompanying explanations and derivations are given in Sec. 3.1, but are also available on the project's website in the theory guide alongside a detailed tutorial.

In the context of streamwise periodic flow the special handling of additional fixed point iterations within the existing solver scheme was implemented, at the example of iterated pressure drop to dial in a prescribed massflow. The theory with respect to the adjoint equation is discussed in Sec. 3.2.4, and the consequences of neglecting said fixed point iteration are showcased in Sec. 5.1.4.

As enabling and performing shape optimization is the final objective of this thesis, it is consequently also necessary to be able to export the final designs in a widely utilized file-format to further process the geometry for e.g. 3D printing. The simple, but widespread supported STL format was added as an additional output option for that purpose.

A base class for transported scalar equations as e.g. for turbulence, species transport, etc. was created that unified a significant amount of duplicated code for those solver instances. The implications on maintainability of such a refactoring effort are discussed in the upcoming Sec. 4.2.

The remaining feature additions, bug-fixes and refactoring efforts undertaken by the author can be viewed on the development platform GitHub for further inspection, under the link given at the beginning of this section.

4.2 Efforts ensuring Maintainability

This section explores software maintainability in the context of the SU2 project and attempts to provide examples where possible. The author of this thesis strived to realize these ideas in the previously displayed code contributions, but claiming by oneself to have succeeded in that attempt is almost impossible. No formal, agreed-upon definition of maintainability exists, and suitable metrics are hard to come by, so the following reflects the author's perspective of five consecutive years as a developer in the project. First general coding aspects are introduced, followed by a larger scale with feature documentation, and the section closes with a few observations on maintainability in the SU2 project.

Maintainable code is easy to modify or to extend, without the risk that change will break existing code elsewhere. Before modifying existing code, it is crucial to first read and understand the present state. A major aspect in enabling a good understanding are naturally code comments, that allow to quickly get an overview of developer's intentions without having to decrypt the code itself. Comments are made on various scales: from the general intention of a complete class, over the purpose of a certain code block, down to the idea behind a specific line of code. More comments are definitely not better in the general sense, as bloating a file can reduce readability and also a minimum level of understanding needs to be expected from a developer. Another problem is that the background knowledge of a potential reader it not a priori known, so that the comments from one developer's perspective might not be helpful for another developer. In doubt, it is probably recommendable to stay consistent in amount and depth of commenting to the existing code.

In addition to helpful comments, descriptive variable/function/etc. names should be used to achieve readability without any comments at best. In that context, long and complicated names are entirely possible, but arguably the most important aspect is consistency. A common construct in SU2 are `Get<VarName>()` functions that simply return the private class variable `VarName`. The implication of coming across such a `Get`-function is of course the expectation of the just mentioned implementation, but if the `Get`-function e.g. alters different class members or includes calls to other functions, the developer might run into unexpected behavior. Another example of breaking consistency is to implement e.g. `Return<VarName>()` that also simply returns `VarName`. In order to increase the usage of consistent coding concepts, it is clearly beneficial to first get familiar with the codebase before introducing the own ideas. Being a multiphysics solver, SU2 developers might know a lot about one specific solver but not about the others. In such a scenario, employing the same structure to a reasonable amount via a parent class that all solvers inherit from, enables every developer to orient themselves on a high level even in completely unknown sections of the code.

It is valuable to avoid unnecessary code duplication wherever possible and to encapsulate functionality on a high level. A single source of origin is easier to identify, and all dependent code parts benefit from eventual changes. Unifying similar code from various code regions usually comes at the cost of increasing code abstractions that can reduce readability. Following that multiphysics solver example, the implementation of certain features as e.g. boundary conditions might be overwhelmingly identical between different solvers such that keeping both in sync leads to increased maintenance work in case of changes, or worse potential bugs might only be fixed for one implementation. The symmetry boundary condition redesigned by the author was implemented in the compressible and incompressible solver featuring identical code and was consequently unified for both solvers to avoid that code duplication.

Making interfaces between different code modules as explicit as possible, can reduce unpredictable and unwanted coupling effects. Modification of data-structures should be prohibited if not required by a certain method, which can be achieved in C++ by proper utilization of the *const* keyword. Being able to rule out certain data alteration of a specific function by code design, can greatly reduce debugging efforts. In the worst case, the developer is not aware of which methods alter what data, as that information is obscured in a deep call stack.

It is not possible to avoid bugs altogether, but spotting them early on through error handling and preparing the code to facilitate debugging can greatly reduce the accompanied effort to fix emerging bugs. Reasonable error handling checks, e.g. whether a function output represent a physical meaningful value or whether a combination of configuration options was tested to work, can signal a possible bug or a prohibited application of certain configuration options to the user without having to interpret simulation results first. With the implementation of a new feature, it is therefore sensible to only allow the feature to be used within the tested bounds, and throw an error otherwise. Once a further combination with that feature is tested, the restriction can be lifted for that very combination. Error checking has its limits, as an excessive use can degrade performance, e.g. if frequently arrays of the mesh size are checked for nonphysical values. In case a bug is not caught by error handling, which probably represents the majority, debugging efforts based on the nature of the error have to be made. Some debugging tools require special instrumentation of the code, such as e.g. the *fenv.h* library that was frequently employed by the author to find the occurrence of *NaN*'s which are often signaled by the screen output *SU2 has diverged*. The library is included in the source code together with some simple commands to trigger specific options. When not used, these options are disregarded in the compiled code, but upon uncommenting the specific lines and recompilation, the developer can quickly determine whether an unwanted *NaN* is the root cause. In conclusion, writing and instrumenting the code with the ability to fix bugs later on, leads to more bugs being found early and fixed quickly, and thus improves maintainability.

To some degree, each developer should be aware of the overall level of coding expertise in the community. Many developers in the SU2 project are driven by an engineering application or numerical aspects rather than pure software development. The introduction of excessively complex software constructs by one developer might therefore be an obstacle for the remaining majority of developers. Weighting the average community member's ability to contribute with e.g. increased performance through highly specialized code is certainly difficult. At best these highly specialized code sections can be encapsulated with a clear to understand interface, such that even an entry-level developer can understand the purpose and usage of e.g. a function, but does not necessarily has to fully grasp its implementation.

All of the above points are subject of discussion in each Pull Request, and it is the reviewer's task to judge the PR also with maintainability in mind. Especially aspects like consistency, helpful comments, error handling or avoiding code duplication can be enhanced by a proper review. The role of the PR review as a flawless quality gate can naturally not be maintained at all times in a software project as SU2. If for example a small feature addition in theory could motivate a larger refactoring effort to smoothly incorporate the new feature along the existing ones, the original developer is potentially not capable to do so due to time restrictions. Numerous code reviews were given and received by the author, and it can be stated that reasonable amounts of this thesis advances are enhanced by proper code reviews given by the community, that helped with overall improved functionality, conciseness and maintainability of the contributed code.

On a larger scale compared to coding aspects, feature documentation via tutorials, testcases or simply a written guide, tremendously improves maintainability. A written guide can provide a quick overview over the key components and operations of a feature without the obfuscation through possible fragmentation of the operations over multiple files, classes, etc. and over the program's runtime. This is especially helpful for inexperienced developers who get to know the codebase and the employed methods, as it is much easier to recognize patterns in the code if the exact theory is written down. Knowing what to look for avoids having to reverse engineer a whole feature purely based on the code, which can be vastly time-consuming. Especially the streamwise periodicity feature of the author has a theory guide on the project's website, but also e.g. the symmetry boundary condition theory is explained in an extensive comment at the respective functions start. With respect to a complete feature often including multiple configuration options, a working testcase, or at best tutorial, has to be available to ensure maintainability of said feature. Otherwise, a developer who is unfamiliar with the specific feature is unable to ensure its correctness, in particular when refactoring code parts. A working testcase, which additionally is ideally part of the regression tests, acts as a safety net for unwanted side effects due to code changes to the features utilized in that very testcase. Although not strictly enforced, larger features that are not checked through the regression tests should be consequently removed (or such regression test should be added) as its correct working is questionable and the potential maintenance effort to repair the feature is immense. In the various Pull Requests of the author, thorough regression testing is performed, which also acts as a safeguard for continuous work against other developer's code breaking the own. One example is the unsteady regression test[10] that safeguards the developments on the unsteady CHT adjoint with the results presented in Sec. 5.2.

In that context, a simplified approach to test-driven development, used and recommended by the author, is sketched. In a strict sense, test-driven development means to first design and implement a test and then developing the feature that ultimately has to pass that very test. That kind of rigorous approach is not feasible for the integration tests in SU2, that check for matching residuals, which is impossible to know in advance. Yet, validation data can be matched in approximation or plausibility checks e.g. in the limits of a feature's validity, can be judged by a developer based on a testcase. The author frequently designed these kinds of plausibility checks at the start of the feature development which can be easily converted into an integration test, once the desired result is reached. As an additional benefit of putting a test to use early on during the development phase, is knowing when the test was altered by own changes, which might reveal unwanted coupling effects that can then be resolved.

In a project like SU2, it cannot be assumed that the original author of any given section will be around once another community member is required to modify that exact code part. The necessary effort to thoroughly understand existing code such that the required modifications can be made without breaking existing features cannot be underestimated, and often times are on the same timescale as developing code from scratch. The lack of continuity in personnel is a major weakness of a code that, by and large, is developed by PhD students that have contracts with a temporary length and then often times leave the project entirely after their thesis. In the position of a PhD-candidate it often seems difficult to justify the additional effort for code generalization, appropriate commenting, regression test setup etc. but in the author's experience these efforts will pay-off in the long run, even for a singe PhD-student. As a PhD project usually spans 3 to 6 years, significant parts of the early written code will be used in much later stages of the PhD-time. Having written well-tested, maintainable code in

---

[10] https://github.com/su2code/SU2/tree/master/TestCases/coupled_cht/disc_adj_unsteadyCHT_cylinder

the early stages allows modifications in the later stages at a greatly reduced effort, and thus might save time overall. Another, less self-serving thought is if every developer were to thoroughly write and merge their code, everyone will benefit in return from others people contribution at some point and thus increase the research output of the whole community, including oneself. On the contrary, if all developers were to be secretive about their work, some developments will be done independently by different developers and neither of their contribution will be suitable to be used by others. Another aspect to consider is the funding source of a developer, which for PhD-students is often public money from research funds. In return, the author derives from that circumstance an increased obligation to openly publish their work in a usable format. According to the above reasoning, the author took care of merging the developed features, carefully documenting and safeguarding the contributed code and following the community advice given through code reviews in order to provide a sustainable contribution to the SU2 project.

4.3 Result Reproducibility

The reproducibility of results is a key property, and to some degree even requirement, for high-quality scientific work. The interpretation of what suffices to actually achieve reproducibility is not rigorously defined nor enforced throughout the scientific computing community. Is a research publication that details the implemented method without providing the source code sufficient, or does every detail down to compiler versions have to be provided together with the source code? The author is convinced that the latter should be attempted wherever possible, and the motivation for this section is to achieve result reproducibility to the best of the author's knowledge and belief.

All methods utilized for the applications shown in this thesis are implemented in the open source multiphysics and design solver SU2, and the results shown are created using the git tag v7.3.1 with a clean working directory and unchanged submodules. This git tag represents a specific commit[11] in the master version of the code, therefore all required code had to pass a quality-ensuring pull request into the develop version and subsequently, into the master version. The only instance where another commit[12] was used, is section 5.1.4 where deliberately a code without a certain feature, which was added by the author, was executed, to showcase the influence on solution's behavior.

The code was compiled using gcc in the version 11.2.0 and the MPI implementation

---

[11] 328a1b747a4785d13b749e7fb6cc4589fd1b9529

[12] 04178124ffc9594da1ba97941f5a0bdca3a59305

OpenMPI in the version 4.0.5. . Despite the availability of a shared-memory parallel approach with OpenMP, this feature was not used, as core counts did not exceed a single compute node.

All simulations were performed on the high-performance Computer Elwetritsch at the TU Kaiserslautern Kaiserlautern which is run by the HPC-Team of the RHRK[13]. The following data represents the state in July 2022. The cluster runs on the operating system Scientific Linux 7.9 (Nitrogen) and features various hardware options from Intel. Most notable are 299 Intel Skylake nodes with two XEON SP-6126 CPU's and thus 24 cores, a total of 205 Intel Sandybridge nodes with the two XEON E5-2670 CPU's (16 cores per node) in 32GB (105 nodes) and 64GB (100 nodes) options and 96 Intel Haswell nodes using two 8-core Intel E5-2640v3 with 64G RAM per node[14].

The optimization framework FADO was used with a specific commit[15] that represent a master version of the code.

Python is used in version 3.8.8 and the exact version numbers of the various used packages (e.g. NumPy, SciPy, pandas, etc.) are determined by the Anaconda 2021.05 distribution, that ships a specific set of mostly scientific python packages. FADO and various visualization scripts make use of Python.

The visualization software Paraview in version 4.4 is used for the contour plots in this thesis and was used extensively throughout the development process for data visualization. Paraview is available under a permissive BSD license[16] and describes itself as *"an open-source, multi-platform data analysis and visualization application"*[17].

All meshes were created using the open-source meshing tool gmsh [GR09] in the version 4.8.4. Gmsh is available on all common operating systems under the GPL 2.0 license[18].

Altogether, the major components of the design chain utilized in this work consist of open-source tools: gmsh for meshing, SU2 and FADO for simulation and optimization, Paraview and Python for postprocessing, with all of these tools running on a Linux distribution including non-proprietary compilers and interpreters. This tool-chain allows nearly everyone with access to the internet and a CPU with x86-architecture to fully reproduce, and even to freely modify or extend the presented results.

The simulation files (configuration files, gmsh mesh scripts and the meshes itself, postprocessing and visualization scripts for the exact plots from the result section) for

---

[13] `https://elwe.rhrk.uni-kl.de/` Accessed 2022-07-05

[14] `https://elwe.rhrk.uni-kl.de/elwetritsch/hardware.shtml` Accessed 2022-07-05

[15] d4dfa56b1f6f52e6e362c7954d2f2474b55b263d

[16] `https://www.paraview.org/paraview-license/` Accessed 2022-07-08

[17] `https://www.paraview.org/overview/` Accessed 2022-07-08

[18] `https://gmsh.info/` Accessed 2022-07-08

full result reproduction shown in this thesis are available in a GitHub repository of the author[19]. This is not only a mere attempt to ensure straightforward and exact reproducibility, but also an invitation to modify or extend the capabilities to the reader's own needs, as this is ultimately a core characteristic of open-source.

---

[19] `https://github.com/TobiKattmann/Thesis-Simulations`

5 Verification and Application

After the introduction of the required primal and adjoint simulation methods in connection with the operating principles of the SU2 solver in the previous chapters, this chapter highlights two distinct applications with selected accompanying validations. First, Sec. 5.1 deals with a steady state, streamwise periodic, pin-fin heat exchanger array on an elementary unit cell. In the second application in Sec. 5.2, an unsteady CHT case is considered with a cylinder in crossflow.

5.1 Steady coupled CHT Staggered Pin Array

This application case is for steady state, turbulent, incompressible flow through a 2D array of heated cylinders with coupled solid domains representing the pins in the array. A typical pin array, for e.g. power electronics cooling, can consist of tens up to a couple of thousand individual pins. Instead of an (often prohibitively) expensive simulation on the entire pin array, a common simplification is to only use a characteristic unit cell, see Fig. 5.1. Through mirroring at symmetry boundaries and translating at periodic boundaries, one recovers an approximation to the initial/full array. See Fig. 5.3 how the elementary unit cell embeds into a larger part of a pin array. Note that this image already features the constrained optimized pin design instead of the initial, round pin. It is however within reason to presume that general design rules can be derived from such a reduced, fast to evaluate, unit cell approach.

This section starts with a thorough case description including geometry/mesh description, material properties, boundary condition and finally, employed numerical methods. The validity of the streamwise periodic approach is shown with a numerical experiment in Sec. 5.1.1, followed by the complex, periodicity-preserving, mesh deformation routine in Sec. 5.1.2. An in-depth gradient validation is carried out in Sec. 5.1.3, with a subsequent discussion in Sec. 5.1.4 of the consequences on gradient validation, that come with incorporating additional fixed-point equations. Finally, Sec. 5.1.5 explains the constrained optimization setup and explores the optimization process and results.

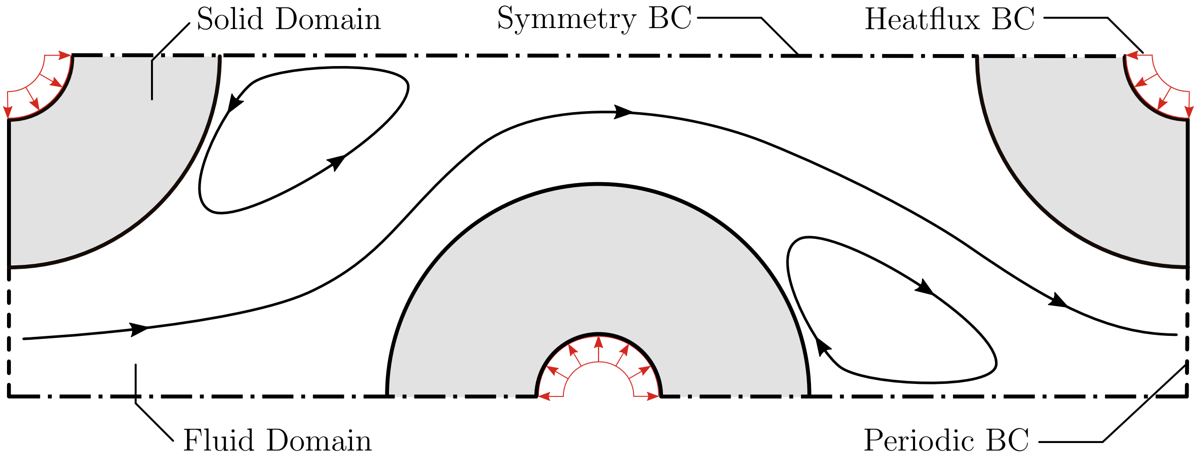


Figure 5.1: Simulation domain schematic. The solid zones are contrasted in gray to the white fluid body, where arrows indicate the overall flow topology. Heatflux boundary conditions are indicated with red arrows and symmetry planes with a dash-dotted line.

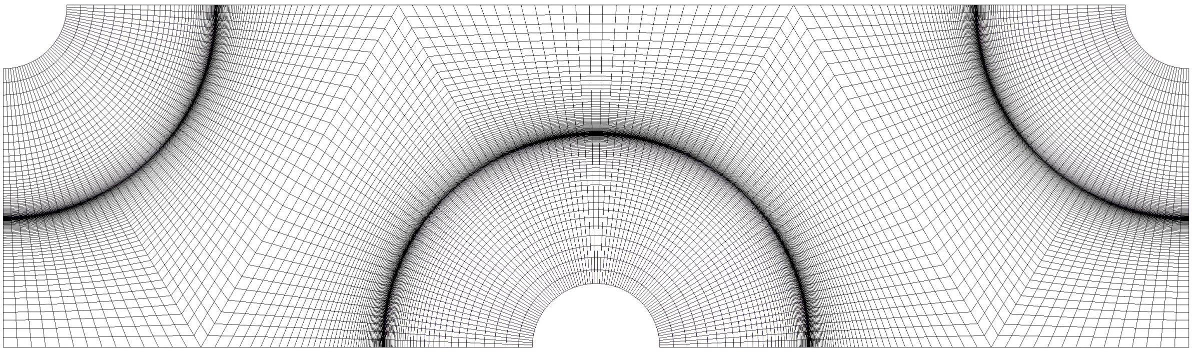


Figure 5.2: Fully structured mesh of the periodic simulation setup. The no-slip walls on the fluid-solid interfaces are resolved such that $y^+ < 1$ is achieved everywhere.

Geometry and Mesh description

The geometry of the simulation domain is fully characterized by three quantities: distance between pin midpoints 6.44 mm (three neighboring pins form an equilateral triangle which is oriented such that the downstream pin rows are staggered), the inner pin radius 0.6 mm and outer pin radius 2 mm. The pins are true circles, i.e. no ellipsoidal shapes. Additional symmetry and periodic boundaries are connected via straight lines through the pin midpoints, as shown in Fig. 5.1.

The computational mesh is fully structured, e.g. only quadrilaterals are used, and consists of 8477 elements in the fluid zone and 6747 elements in the solid zone, see Fig. 5.2. The height of the first cell layer is chosen such that a $y^+ < 1$ is obtained

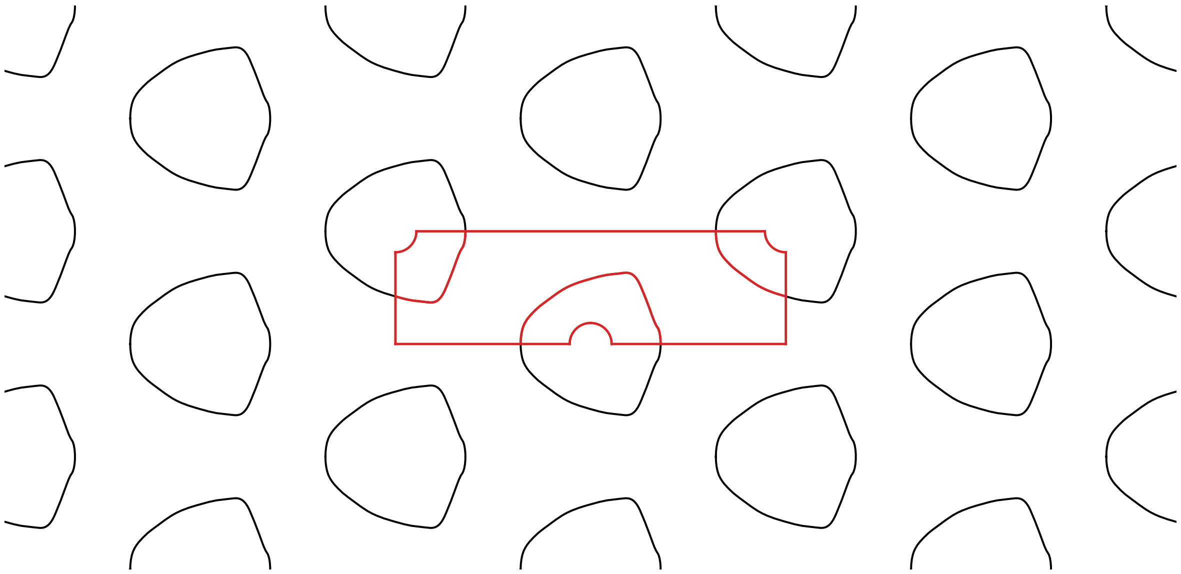


Figure 5.3: Extended pin array by mirroring and translating the original simulation domain that is indicated in red. The array features the constrained optimized design, presented in Sec. 5.1.5, instead of the initially round pins. This setup showcases the periodicity-preserving geometry and mesh deformation, as the whole pin array only consists of a single pin-shape.

everywhere on the no-slip walls. The primary purpose of the unit cell application is to verify the gradient accuracy of the discrete adjoint method in these cases and to showcase the optimization abilities.

Material properties

The fluid's material properties represent a generic automotive cooling liquid with, with material properties generally similar to water. All parameters are considered to be constant, with temperature dependency in particular not taken into account. The density of the incompressible fluid is set to $\rho = 1045\, kg/m^3$, the specific heat capacity at constant pressure to $c_p = 3540\, J/(kg\, K)$ and the laminar dynamic viscosity $\mu_{lam} = 1.385\text{e -3}\, Pa\, s$. The laminar and turbulent Prandtl numbers are $Pr_{lam} = 11.7$ and $Pr_{turb} = 0.9$, respectively, where the laminar Prandtl number implies a thermal conductivity of $\kappa = 0.42\, W/(m\, K)$ after Eq. (2.3).

The constant material properties in the solid represent an aluminum alloy with density $\rho = 2719\, kg/m^3$, specific heat capacity $c_p = 871\, J/(kg\, K)$ and thermal conductivity $\kappa = 200\, W/(m\, K)$. As an alternative to aluminum alloys, heat sinks manufactured from copper alloys are often installed due to its increased thermal conductivity of $\kappa \approx 400\,W/(m\,K)$.

Boundary conditions

Inlet and outlet in the fluid domain are prescribed as periodic boundaries, and streamwise periodicity (or fully developed flow) is used in conjunction, as introduced in Sec. 3.1. The massflow over the domain is set to $0.85\,kg/s$ and the inlet bulk temperature to $338\,K$. By iterating over a pressure drop that is ultimately used in the associated source term to achieve the prescribed massflow, the resulting pressure drop in the initial configurations computes to $208.02\,Pa$. The inlet bulk temperature is enforced by an outlet heat sink, and theoretically allows for quick evaluation over various temperature differences between fluid and solid that can occur over the downstream coordinate of a pin-fin array due to the overall heating of the cooling liquid.

Symmetry boundaries on top and bottom of the domain are indicated in Fig. 5.1 by dash-dotted lines. Alternatively, mirroring the domain along the lower symmetry axes to obtain one full pin and applying periodic boundary conditions in spanwise direction is possible in the SU2 framework as well. Due to the further complication of the mesh deformation routine, and limited accuracy enhancements, the former approach using symmetry boundaries is used.

Handling of the CHT-interfaces between fluid and solid is described in Sec. 2.2.4.

On the three inner pin arcs of the solids a constant heatflux of $5e5\,W/m^2$ is applied, which represents the heat loss of an attached device, e.g. power electronics. All remaining walls are considered to be adiabatic. Note, that during optimization, the inner pin surface remains clamped such that the integrated heatflux over the solid's domain boundary, i.e. the rate of heat flow in watts, does not change.

Numerical methods

Convective fluxes in the incompressible solver are discretized using a flux-difference-splitting (FDS) with second order accuracy achieved by a MUSCL approach, see [Eco20] for a detailed explanation. No limiters are used for variable reconstruction, and gradients are computed via the Green-Gauss theorem, for the variable reconstruction in the MUSCL scheme and the viscous fluxes. The pseudo-time derivative is discretized using an implicit Euler scheme. The Menter SST turbulence model as described in Sec. 2.2.2 without wall function is used.

Similar to the flow solver, the gradients in the heat conduction solver are computed via the Green-Gauss theorem, and the pseudo-time derivative is evaluated with the implicit

Euler scheme.

In both solvers, the resulting linear system is solved using FGMRES with an ILU preconditioner. The CFL number and number of linear solver iterations are modified between primal and adjoint setup for optimal performance in either setup and for robustness throughout the complete optimization history. In the author's experience, lowering the CFL number and/or increasing the amount of linear solver iterations can stabilize the convergence of the adjoint problem. In the present case for example, the CFL number is decreased by an order of magnitude in both zones, and the increased amount of linear solver iterations is used in primal and adjoint solver for an eased setup.

See Sec. 4.3 on result reproducibility, in case of any unclear or missing case description, on how to obtain the used configuration scripts, meshes or visualization scripts.

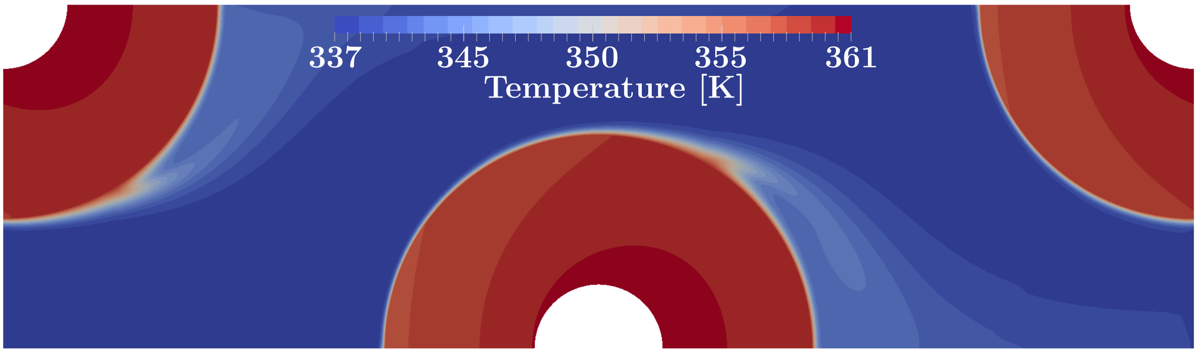


Figure 5.4: Temperature contour lines for the streamwise periodic elementary unit cell of a pin-fin heat exchanger.

To finalize the case introduction, the primal temperature contour lines are given in Fig. 5.4 that show a thin temperature boundary layer on the middle pin and a noticeable heat convection into the fluid body at the separation point at approximately 1 o'clock. The upstream part of the pin exhibits an increased cooling compared to the downstream portion, which is largely covered by a recirculation area beyond the separation point. Yet, due to the high thermal conductivity in the solid compared to the fluid, the heat is spread rather uniformly over the entire pin in contrast to the temperature distribution in the fluid.

5.1.1 Validation of the Streamwise Periodic Approach

It has to be validated, that the streamwise periodic approach is capable of accurately predicting the flow state after arbitrary many pin rows in the array. It naturally depends on the specific simulation setup (geometry, material properties, boundary conditions, etc.)

after how many downstream repeated geometries the flow is considered fully developed. The question, after how many pin rows an approximate periodic state is reached, needs to be answered for each application and simulation setup individually and whether it therefore it is even reasonable to employ a streamwise periodic simulation. A simulative validation is carried out, which additionally answers the previous question for this exact unit-cell setup. The streamwise periodic result is compared to a simulation on a domain, that features multiple copies of the periodic domain in the downstream direction. The massflow is held constant in all simulations. For the simulation with downstream repeated unit-cells, the inlet boundary condition is changed to a velocity inlet and the outlet to a pressure outlet with gauge pressure zero. For the velocity inlet, two cases could be considered:

1. The inlet profile is a uniform block profile, thus the flow will develop to a downstream repeating nature only after a few pins.
2. The outlet/inlet profile from the streamwise periodic simulation is extracted and prescribed at the non-periodic inlet. In this case, the flow should not be altered in downstream repeating section.

In this thesis the former method, inlet-velocity block profile, is shown as the latter simply replicates the exact profiles for all downstream cross-sections as intended and therefore is not as suitable to showcase.

The results of the comparison between periodic and non-periodic results are plotted in Fig. 5.5. The overall agreement after 16 pin-rows for velocity, pressure, temperature and eddy viscosity is excellent and proves the validity of the streamwise periodic assumption. For velocity, pressure and temperature, the profile match after 8 or 12 slices already is quite significant and would justify the usage of the streamwise periodic approach, in case no quantities that directly depend on eddy viscosity are of interest.

Comparing velocity profiles is straight-forward as it requires no scaling of the results, and each cross-section should provide the exact same results. For pressure and temperature, the simulation result needs to be adapted according to the streamwise periodic assumption in order to be comparable. With a constant heatflux boundary condition on the pin surface, the absolute temperature naturally rises from with increasing downstream length in the non-periodic setup, but the profile shape should stay constant. The same concept applies to the pressure but with the reverse effect, as the pressure continuously drops in downstream direction. In order to just compare the profile, free of the absolute value, the profiles are linearly shifted such that the temperature or pressure at the lowest y-coordinate position is zero.

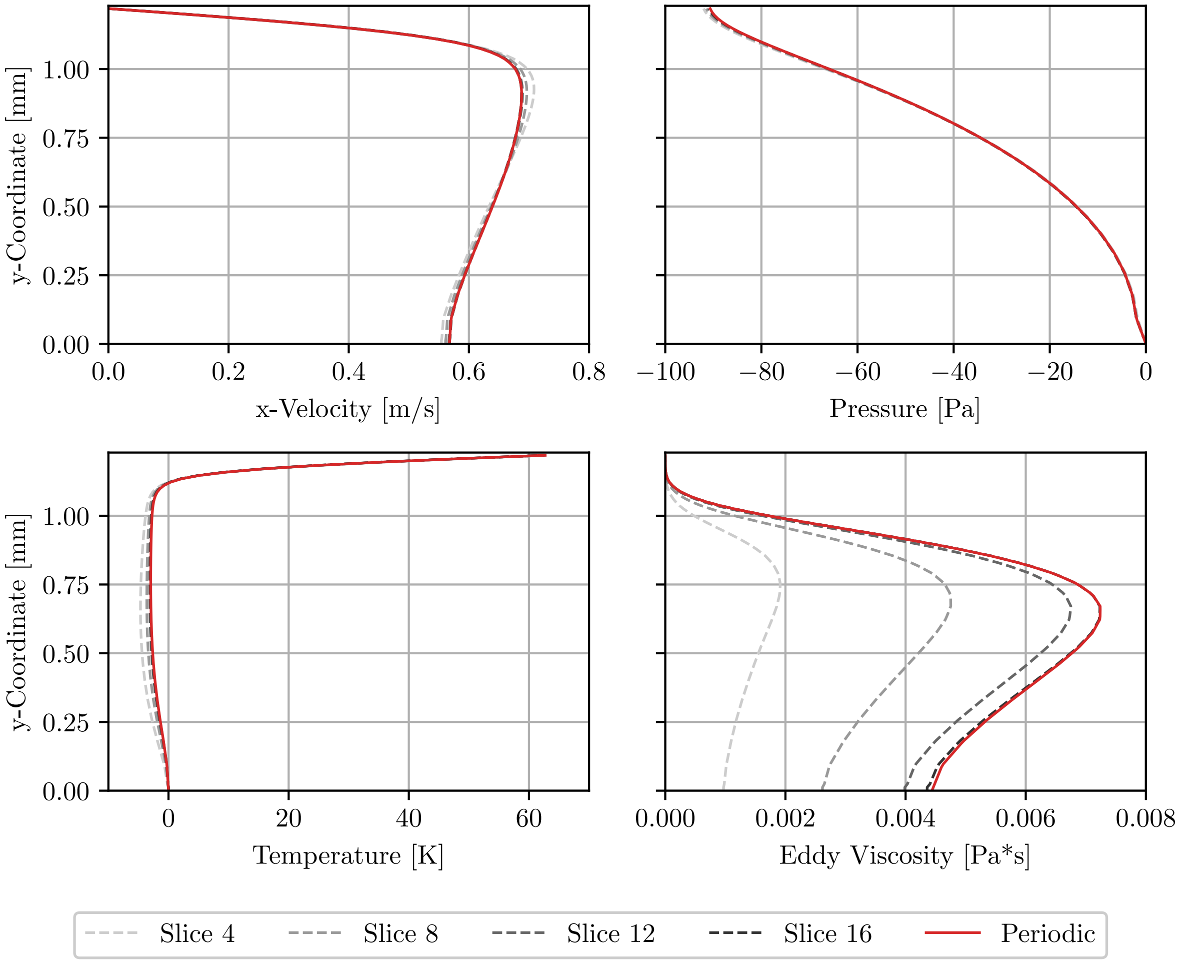
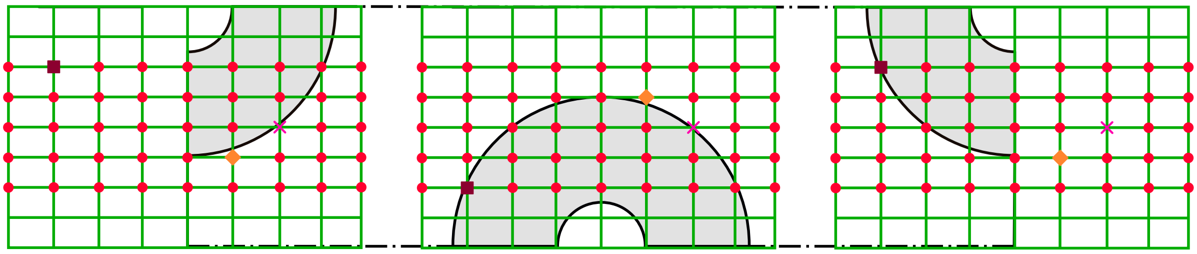
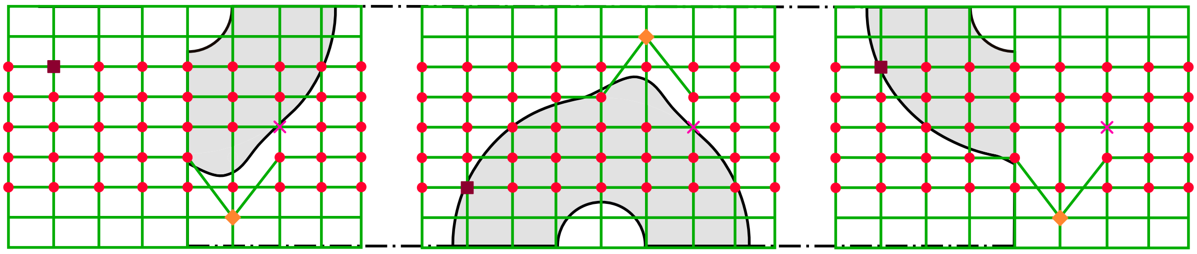


Figure 5.5: Comparison of x-velocity, pressure, temperature and eddy viscosity profiles for a streamwise periodic simulation (red line) and an extended simulation domain with many pin rows (profiles extracted at 4 locations indicated by dashed lines with a darkening gray scale for further downstream locations).

Note, that the above test was performed using a fluid-only case, but the results using a CHT case show no apparent difference, so they are excluded here for the sake of brevity. In any case, the difference between a CHT and fluid-only case exclusively influences the temperature profiles as the energy equation in the present scenario is only weakly coupled to the continuity and momentum equations, meaning that the energy equation depends on the velocity field, but the continuity and momentum equations do not depend on the temperature field.



(a) FFD-setup for the optimization using three identical, but mirrored, FFD boxes.



(b) Deformation of the three FFD-boxes that yield equal pin shapes.

Figure 5.6: Mirrored FFD-box setup where equal deformation is applied to all three FFD-boxes and thus the surface mesh. The complete FFD-lattice is in green color while the control points that are active in the vertical axis are indicated by a red dot or other symbol. These 'special' symbols represent an equal point in all FFD-boxes and thus facilitate orientation.

5.1.2 Mesh Deformation Routine

Ensuring a suitable and robust geometry and mesh deformation setup for the optimization was one of the major challenges for the present testcase. This is mostly due to the two quarter pins on the periodic interface that have to undergo the same deformation as the half pin in the middle of the domain. That approach of course enforces, that the simulated pin-array consists of a single pin-shape only, as shown in Fig. 5.3 for the constrained optimized pin. This also guarantees the required matching surface boundary and mesh on the periodic interface[1].

The main focus here will be on the surface deformation procedure using FFD, as for the linear-elasticity based volume mesh deformation procedure, all used surfaces in the setup are simply considered clamped.

As already introduced previously, the surface mesh and volume mesh deformation procedures are separated in SU2. For the surface mesh deformation, the imposed FFD-

---

[1] The periodic boundary conditions, as implemented in SU2, require a matching mesh at the periodic interfaces.

boxes setup is presented and showcased for a sample deformation in Fig. 5.6. The three green lattices represent separate FFD-boxes and the symbols (e.g. red dots) indicate actively used points, the remaining points stay clamped. Two rows on the top and bottom are clamped specifically as B-Splines of 3rd order in y-direction are used as the interpolation function for the FFD-boxes, which ensures no point movement of the top and bottom symmetry boundaries. The B-Spline order in x-direction is 4 and this parameter arguably influences the pin's shape the most, given an otherwise fixed FFD-box. Higher order B-Splines take more FFD-point deformations into account for the local deformation of the surface mesh, while at the lower order, the local FFD-deformation prevails, i.e. the support of the B-Spline depends on its order. The available alternative to B-Splines in SU2 are Bézier blending functions, which have a support over the full FFD-lattice and thus creates extremely smooth shapes at the expense of a decreased design space, due to the lack of possible drastic local deformations. Bézier blending functions are employed for the unsteady shape optimization in the subsequent Sec. 5.2.

A movement of FFD-points in x-direction is not considered in this work, as this creates further complications when it comes to retaining a valid shape for the connected quarter pins. If the pin midpoint section moves horizontally, points/elements would need to be removed at one quarter pin and reattached at the other quarter pin to assure a suitable mesh within this FFD framework.

The front and back FFD-box are mirrored in Fig 5.6 based on the middle box, and of course translated, along the x-axis compared to the middle FFD-box. Some symbols (brown square, orange diamond, pink cross) are used for easy identification of the same point in each FFD-box. A sample deformation is applied to the orange diamond in Fig. 5.6b to showcase the final working-wise of the FFD-box setup.

In order to apply an equal movement to each FFD-box in an optimization cycle, one can either compute sensitivities for one FFD-box (in this case the middle one) and map these appropriately onto the other FFD-boxes by mirroring, or compute sensitivities for all FFD-boxes individually, average the values and then apply these averaged values to all FFD-boxes. The former is used for the optimization presented Sec. 5.1.5. Note, that this approach of course constitutes a slight inconsistency between the computed gradient and applied shape optimization. Therefore, the mesh deformation for the gradient validation, see the subsequent Sec. 5.1.3, will be restricted to the middle pin only.

For the gradient validation, only design variables according to Fig. 5.7 are considered. The seven FFD-point displacements in vertical direction are only applied to the middle pin in order to retain a fully consistent discrete adjoint gradient.

Each surface patch is now either clamped, i.e. remains unaltered, or is affected by

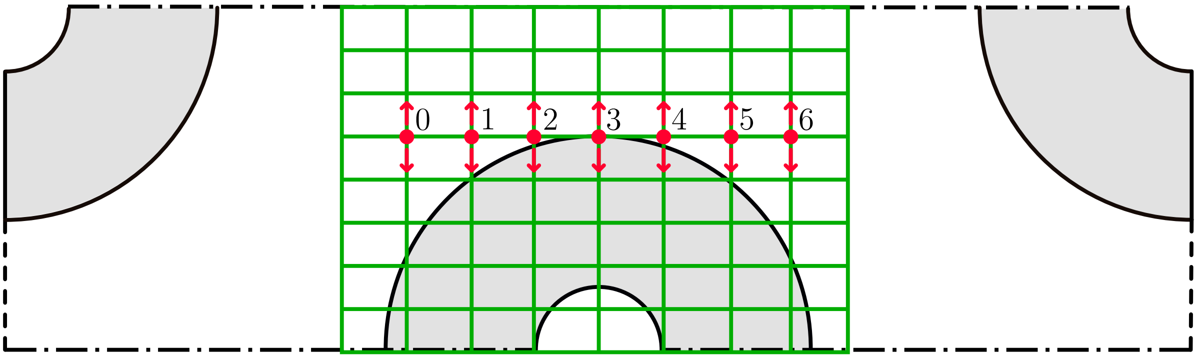


Figure 5.7: The FFD box in green encloses exclusively the middle half pin for the purpose of gradient validation. Active design variables are marked in red, and are only altered in the vertical y-coordinate, as indicated by the red arrows.

the FFD-Box movement. The inner-pin circle segments and the symmetry boundaries are clamped, while consequently all other surface segments (fluid-solid interfaces and periodic front- and backface) are moved by the FFD-box. An alternative, only deforming the fluid-solid interface and allowing in-plane movement of surface mesh points in the volume mesh deformation phase, was not viable due to a resulting non-matching mesh on the periodic interface. This also motivates the extension of the FFD box over the complete domain in vertical direction.

5.1.3 Gradient Validation

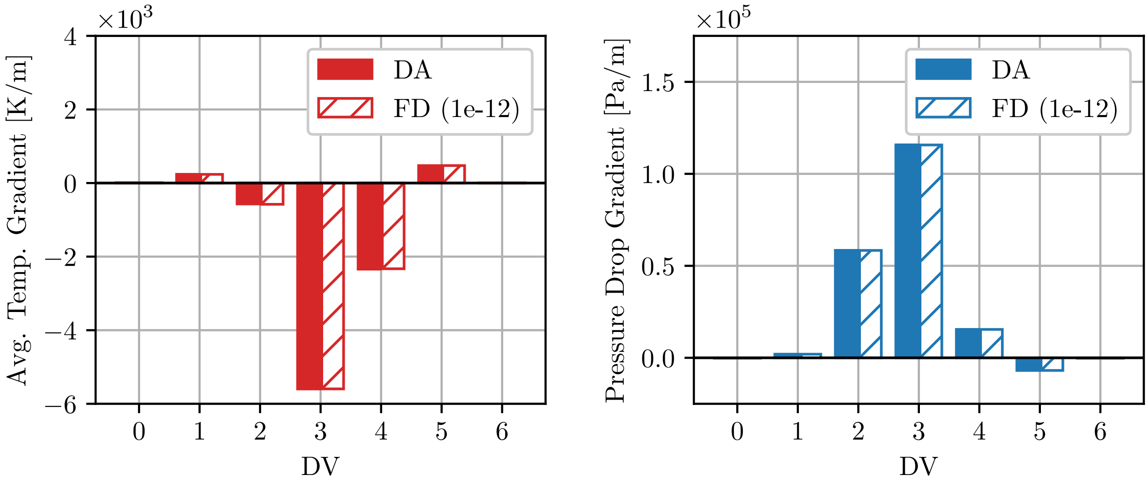


Figure 5.8: Gradient validation of surface average temperature and pressure drop objective functions with a fixed finite difference stepsize of 1e -12.

Before attempting a full optimization, sensitivity computed via the discrete adjoint method is validated against finite differences. Alternative approaches to finite differences are the complex step method which is not implemented in SU2 or using forward mode AD, which, at the time of writing is not operational for the here discussed testcases.

The particular type of finite difference used is the 1st order forward difference, which for a scalar design variable and objective function writes:

$$\frac{dJ}{d\alpha} \approx \frac{J\left(\alpha + \Delta\alpha\right) - J\left(\alpha\right)}{\Delta\alpha}, \tag{5.1}$$

with $\Delta\alpha$ being the stepsize. In this context note, that the design variables α in the FFD approach are the deformations to the FFD-points in the Cartesian coordinate system, and their value thus allows an intuitive understanding of resulting surface deformations. That means, based on a characteristic length of the domain, a suitable finite difference step (or a range thereof) can be defined. The author found a factor to the characteristic length of 1e -6 to be a reasonable starting point.

The two objective functions considered for the elementary unit cell application are surface average temperature, evaluated on the surface of the middle inner pin, and pressure drop over the domain, that is taken between the two periodic interfaces which represent the fluid 'inlet' and 'outlet', naturally using the physical (recovered) pressure instead of the periodic pressure component, following Eq. (3.4).

A straight-forward result of the gradient validation for both objectives is given in Fig. 5.8, that makes use of a fixed finite difference step size of 1e -12. The design variable numbering of this figure and other figures in this paragraph follows the layout in Fig. 5.7. Albeit not giving sophisticated insights in the exact gradient accuracy, Fig. 5.8 attest a sufficient sensitivity accuracy, to confidently use the discrete adjoint gradients in an optimization cycle.

A more in-depth discussion of the gradient validation can be done at the hand of Tables 5.1 and 5.2 as well as Fig. 5.9. The former two tables feature the comparison of discrete adjoint and finite difference sensitivity for two distinct design variables, with one table for each objective. The latter figure plots the relative sensitivity difference over the finite difference stepsize, and thus provides a more intuitive insight into the table data.

The first row of the tables lists the finite difference step-size via a magnitude p, where 1e -p is the finite difference stepsize. The second column lists sensitivity values reported by finite differences for one specific design variable, but the first row contains the discrete adjoint gradient. Matching digits to the discrete adjoint gradient are colored black, and non-matching in red for better visually indication. In the third, the respective relative

magnitude p	FD DV 3	rel diff (%)	FD DV 4	rel diff (%)
DA	-5.596844515e3	-	-2.340265548e3	-
-4	-5.689626723e3	1.6578	-3.111159434e3	32.940
-5	-5.619300461e3	0.4012	-2.433546105e3	3.9859
-6	-5.596967922e3	0.0022	-2.340294692e3	0.0012
-7	-5.595393258e3	0.0259	-2.330330336e3	0.4245
-8	-5.595213906e3	0.0291	-2.329331556e3	0.4672
-9	-5.595199468e3	0.0294	-2.329235201e3	0.4713
-10	-5.595233574e3	0.0288	-2.329259701e3	0.4703

Table 5.1: Gradient validation for surface average temperature for DV3 and 4. Magnitude p indicates the finite difference stepsize with 1e -p. Relative difference in % is computed after Eq. (5.2). The first row holds information for the discrete adjoint gradient and matching numbers are indicated in red.

magnitude p	FD DV 3	rel diff (%)	FD DV 4	rel diff (%)
DA	1.15758245956e5	-	1.5445492703e4	-
-4	1.35501101282e5	17.055	2.9906661869e4	93.627
-5	1.17692049932e5	1.6706	1.6744531940e4	8.4105
-6	1.15894398048e5	0.1176	1.5551885104e4	0.6888
-7	1.15716304806e5	0.0362	1.5433860479e4	0.0753
-8	1.15698482188e5	0.0516	1.5422077800e4	0.1516
-9	1.15696751578e5	0.0531	1.5420951399e4	0.1589
-10	1.15697096987e5	0.0528	1.5421356920e4	0.1563

Table 5.2: Gradient validation for pressure drop for DV3 and 4. The remaining table information are identical to table 5.1.

difference to the discrete adjoint gradient is shown, which is computed as follows:

$$\text{rel. diff.} = \frac{|\text{DA}_{Sens} - \text{FD}_{Sens}|}{|\text{DA}_{Sens}|}, \tag{5.2}$$

with an additional factor of one hundred to convert the fraction to percent. Fourth and fifth column shows the data for a different design variable. Design variables 3 and 4 were chosen due to their slightly enhanced gradient match.

With an outer pin radius of 1e -3m, the largest finite difference stepsize of 1e -4m is not expected to provide an accurate comparison, but serves as an indication on how aggressive the first geometry modification in the optimization run for this specific case

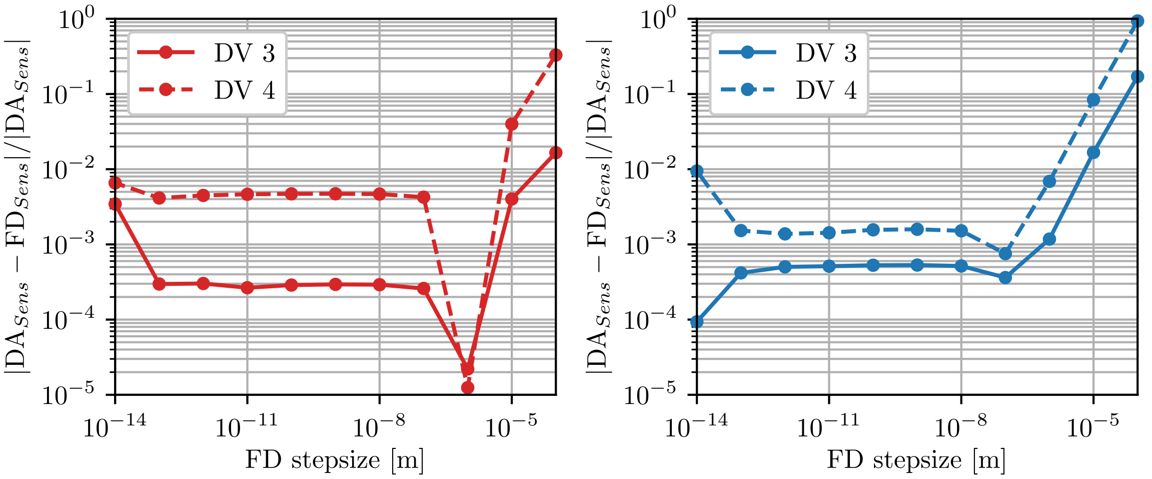


Figure 5.9: Sweep over finite difference step range with relative gradient difference on the y-axis. For average temperature objective on the left in red, and for pressure drop in blue on the right.

can be chosen. The optimal stepsize at DV4 for the average temperature objective is 1e -6m with a relative difference between the discrete adjoint and finite difference sensitivity of just 0.0012%. For the same DV and the pressure drop objective, this optimum is found at 1e -7m with 0.0754% relative difference. In Fig. 5.9 it is visible at a glance, which finite difference stepsizes provide optimal gradient validation. When continuously lowering the finite difference step, it is expected to initially see a lower relative error, but then an increasing relative error due to subtractive cancellation, as described in section 6 of [MN21]. The author cannot provide a conclusive explanation for the plateau in relative difference for both objectives in the range of 1e -8m and 1e -13m. For other design variables beyond 3 and 4, the relative differences have a similar magnitude.

Note the opposite sign of the gradient between the two objectives, that suggest an opposite shape change for both objectives. This circumstance indicates that the two objective functions are conflicting, meaning that optimization of one, will lead to a worsening of the other. Moving the FFD-box point, at e.g. DV3, up (enlarging the pin) leads to a decreased average temperature but increased pressure drop. This conflict will be handled in more detail in the following optimization Sec. 5.1.5.

Overall, the agreement between DA and FD sensitivities is very good, and the gradients are certainly suitable for accurately finding local optima. A clear justification for the remaining deviations in gradient accuracy, especially for pressure drop, cannot be given.

Prior to the gradient validation, it is a necessary/advisable step to ensure perfectly working restarts of the primal solution for the taping procedure in the discrete adjoint. This is not guaranteed and not trivial due to the combination of preprocessing and postprocessing routines that are used to compute derived values and that handle communication between solvers. With added complexity, like turbulence models or a multizone CHT approach, the processes have to be well understood to ensure a proper restart that exactly behaves like an additional iteration. Another frequent source of error is the inexact resetting of the initial primal solution in the implementation. Over the course of a discrete adjoint simulation the primal fixed-point is taped multiple times and to ensure a consistent gradient, the initial primal solution has to be restored each time.

5.1.4 Sensitivities with Additional Fixed-Point Equations

The sensitivities in a streamwise periodic setup with prescribed massflow require an additional adjoint equation, to be correctly validated. Recall that for prescribed massflow the pressure drop is subject to an iterative process after Eq. (3.10). The accompanying theory is described in a previous chapter. 3.2.4.

This section highlights the consequences of neglecting the additional fixed point equations at hand of the just presented gradient validation from Sec. 5.1.3. Solely the straightforward plot 5.8 with a fixed finite difference stepsize is re-considered, as an in-depth comparison does not provide any valuable insights, due to the large sensitivity deviations. The first obvious observation in Fig 5.10 is of course that the finite difference gradient remains unchanged to the previous gradient validation, as only the discrete adjoint evaluation is altered.

The second evident feature is, that the pressure drop objective gradients reports sensitivities close to zero without the additional adjoint equation across all design variables. Without the additional adjoint equation, the sensitivity acts as if a constant pressure drop were prescribed, in which case the sensitivity would of course be zero. The sensitivity procedure is missing the link, that the pressure drop is a derived quantity which is coupled to the flow solution. For the discrete adjoint program the pressure drop appears to be fixed, thus the sensitivity is zero. It is evident that no optimization or any other valuable contribution could be derived from such a defective sensitivity. An equivalent example for applications with frozen turbulence would be to set up an objective function that solely depends on the eddy viscosity field. The sensitivity will consequently report a zero value, as the eddy viscosity field is assumed to be constant in frozen turbulence.

For the surface average temperature objective function, the deviation is large, albeit

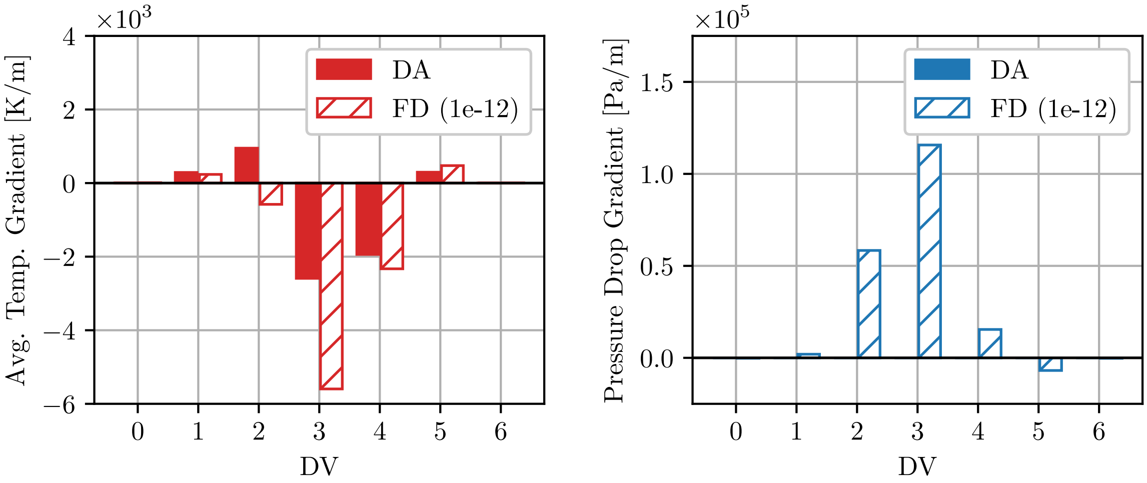


Figure 5.10: Unsuccessful Gradient Validation without special treatment of the pressure drop fixed-point. Finite difference sensitivities equal to Fig. 5.8.

it might still be useful enough to in conjunction with a robust optimization workflow that works well with imperfect gradients.

Note, that these results were not created using SU2 v7.3.1 but with a specific commit ID[2] which does not contain any handling of the mentioned additional fixed pint equation.

5.1.5 Constrained Shape-Optimization Results

At this stage, the primal solution method of streamwise periodic flow was validated in Sec. 5.1.1, a suitable geometry parametrization and mesh deformation routine was introduced in Sec. 5.1.2 and the validity of the gradients computed by the discrete adjoint method was confirmed in Sec. 5.1.3. This necessary groundwork enables the set-up of an optimization, with the confidence of getting a reasonable outcome. Naturally, the optimization control parameters are chosen a priory and potentially require a few adaptions to provide a well converging, and not prematurely failing, optimization setup. Two examples are appropriate bounds to the FFD-box deformations to guarantee a high-quality mesh, and scaling parameters to the objective and constraint to not overly weight one over the other.

The optimization cycle in Fig. 5.11[3] pictures a simplified representation with the fundamental steps. Based on an initial geometry that is governed by a set of design variables, the optimization loop is entered. Given the mesh for the current design,

---

[2] with the commit ID 04178124ffc9594da1ba97941f5a0bdca3a59305

[3] Image layout taken from [Eco14] with minor adaptions.

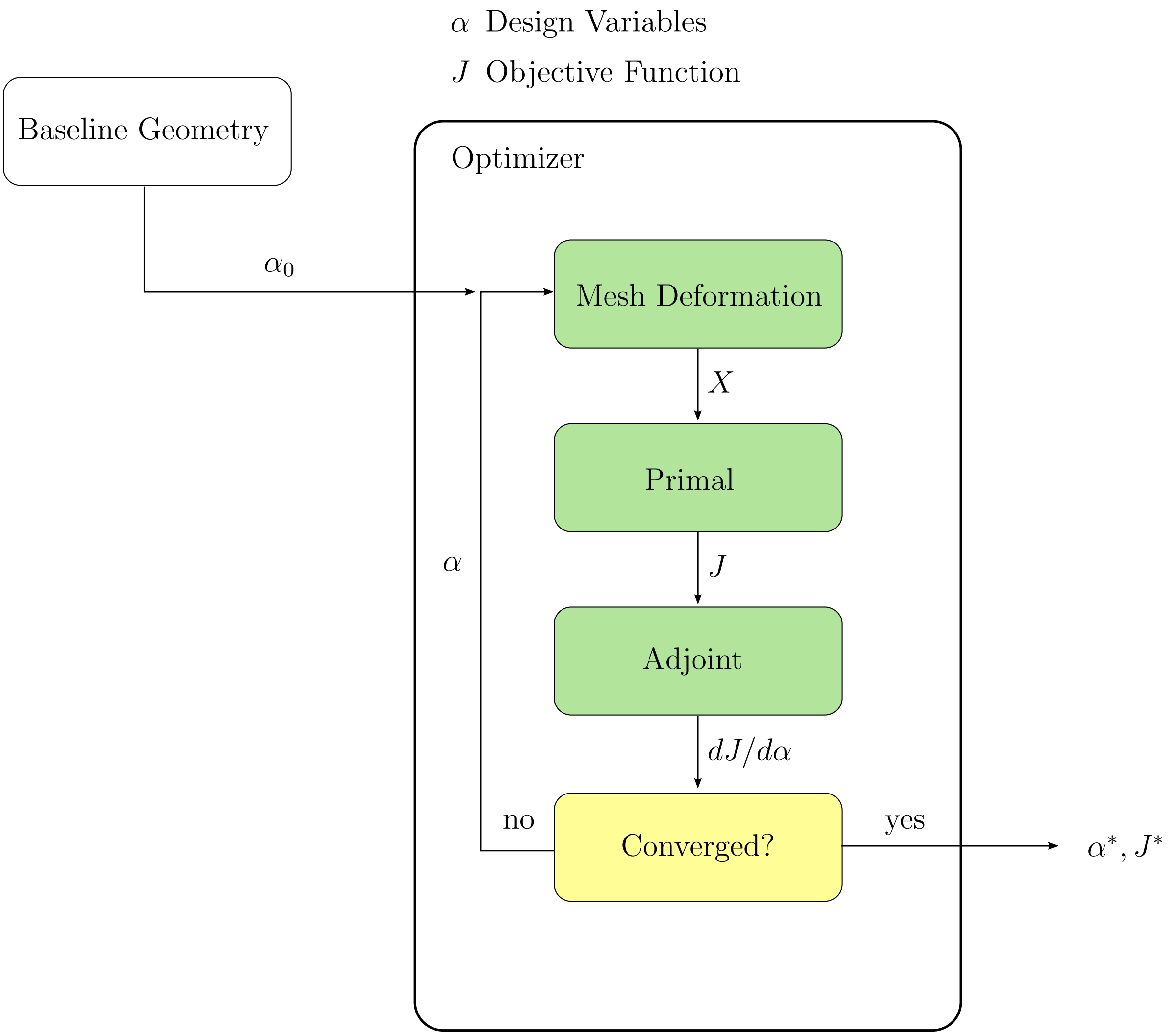


Figure 5.11: Optimization cycle: Based on an initial geometry the primal, adjoint and mesh deformation routines are used until convergence is reported or other exit criteria are met.

the primal solution is computed, which yields the objective and constraint functions values J. In the subsequent step, the sensitivities for the objectives and constraint with respect to the set of design variables are computed via the discrete adjoint method. The following exit criterion block for the optimization is not uniquely placed, as the exit criteria can be numerous: based on a tolerance of subsequent objective values, gradient norm, optimizer iterations or others. The optimization loop can therefore be aborted in all stages and not just a single block as in the simplified representation. If the gradient computation was successful and no exit criterion is met, the optimizer determines a new set of design variables to evaluate and the cycle repeats. It is also possible that the optimizer decides not to evaluate the gradient to a specific design, if it is considered to be worse than previous with respect to the objective and constraints. In this the thesis, the optimization loop terminates predominantly because no design improvement can be found in the available design space in spite of non-vanishing gradients norms.

The lightweight python tool FADO[4] is used to orchestrate the entire optimization process. It provides a general interface to compute objective functions, constraints, sensitivities and intermediate steps like mesh deformation. For all the various tasks in SU2, separate configuration files might be needed with only minor differences in between them, which can easily be handled via replacement functions in FADO. This enables the utilization of just a single configuration file, which eases later changes to common options. It is also straight-forward to introduce additional pre- or postprocessing steps, e.g. for the unsteady optimization in Sec. 5.2.3 the upper and lower side sensitivities are averaged to yield a symmetric deformation.

The optimizer of choice employed in this thesis is SLSQP[5] from the widely available SciPy python package. The simplicity of setting up an operational workflow make the SciPy implementation of SLSQP an appealing option.

The optimization target is set to minimize the surface average temperature on the inner middle pin, whilst constraining the pressure drop over the domain for the fixed massflow to not exceed the value of the initial design. For the equality constraint of $208.023\,Pa$ and the temperature-based objective function, separate gradients are computed and combined by the optimizer, instead of a formulating a single, weighted objective function containing both. From the previous gradient validation and additional optimizations of the objectives alone (that will be presented), it is known that the objectives are conflicting, i.e. the sensitivities of each objective generally suggest opposite deformations. For the present case the average temperature gradient suggest an enlargement of the pin whereas the pressure drop gradient suggests a reduction in pin size.

The optimization history with function values is shown in Fig. 5.12, including the respective gradient norm values in the same figure. As expected, the objective gets minimized while the equality constrained value circles around the fixed value where it closely fulfills the equality constraint at the optimization end. It is unreasonable to expect the OF to be strictly monotonically decreasing or that the constraint stays constantly fulfilled. The gradient, by its definition, is only valid in the current point of evaluation, i.e. with an aggressive large deformation the non-linearity of the problem

---

[4] `https://github.com/su2code/FADO`

[5] `https://docs.scipy.org/doc/scipy/reference/optimize.minimize-slsqp.html`

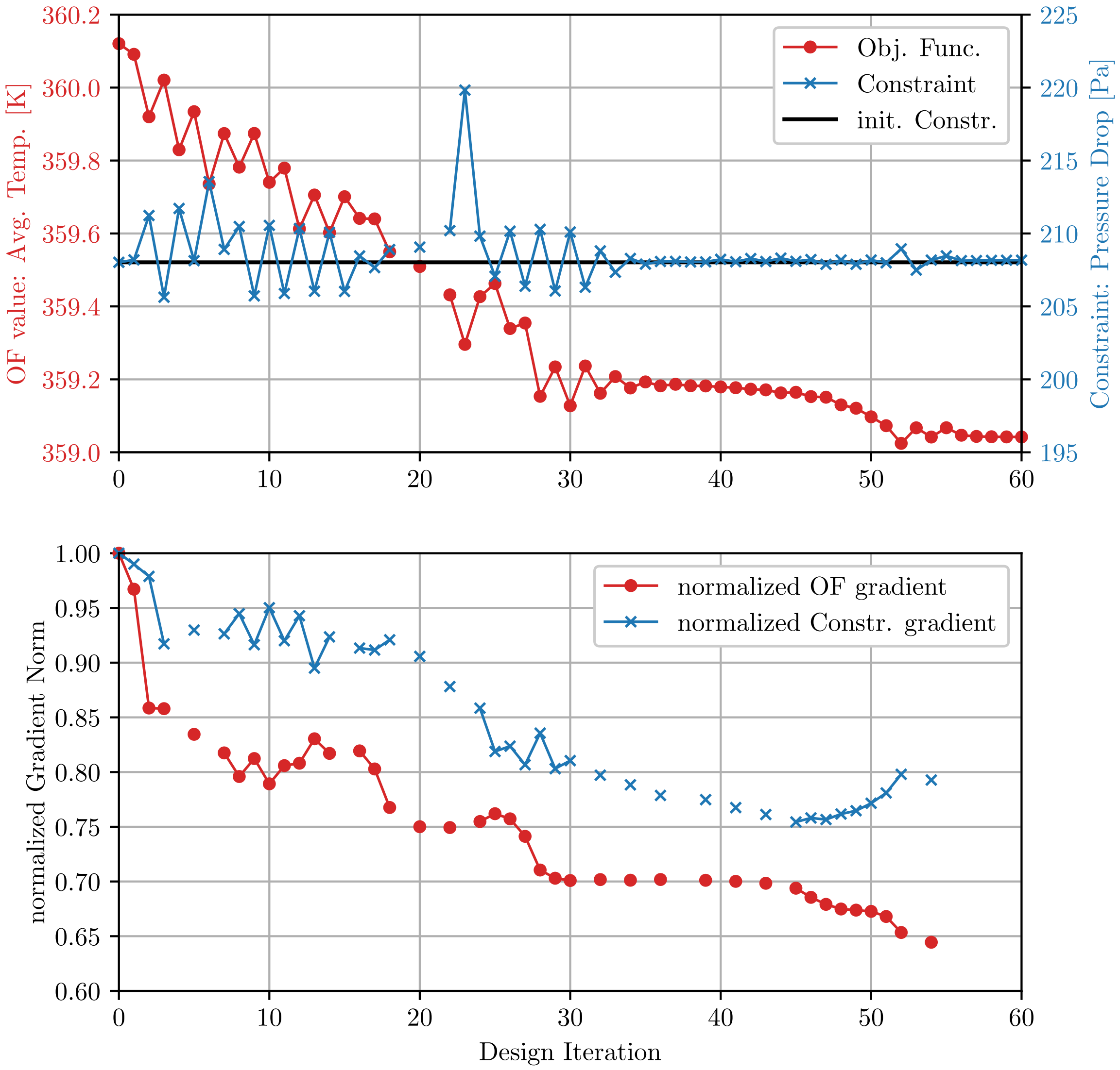


Figure 5.12: Optimization history for the elementary unit cell of a pin-fin heat exchanger. Objective function value and constraint value in the upper image and their respective gradient norm values in the lower. Line elements are only drawn if two subsequent evaluation points represent valid points, i.e. points miss due to invalid meshes, or gradients are not computed because the optimizer rejected the design for further evaluation.

might cause non-compliance with the constraint or worsening of the objective. The resulting ‘zig-zag’ course balances large deformation steps into the objective direction and subsequent restoration of compliance with the equality constraint. In the initial ~ 30 iterations, the major improvements are made to the design, while in the second half merely minor improvements are made until finally the optimizer is unable to find better designs. The gradient norm of the objective drops by approximately a third, but a vanishing gradient norm cannot be achieved, as the constraint and FFD-point bounds hinder further deformations in the objective gradient direction. The optimizer allows for a certain level of control via scaling parameters, but all things considered, the optimizer is used as a black box to a certain degree. A suitable set of scaling parameters has to be found by the user for a reasonably fast optimization.

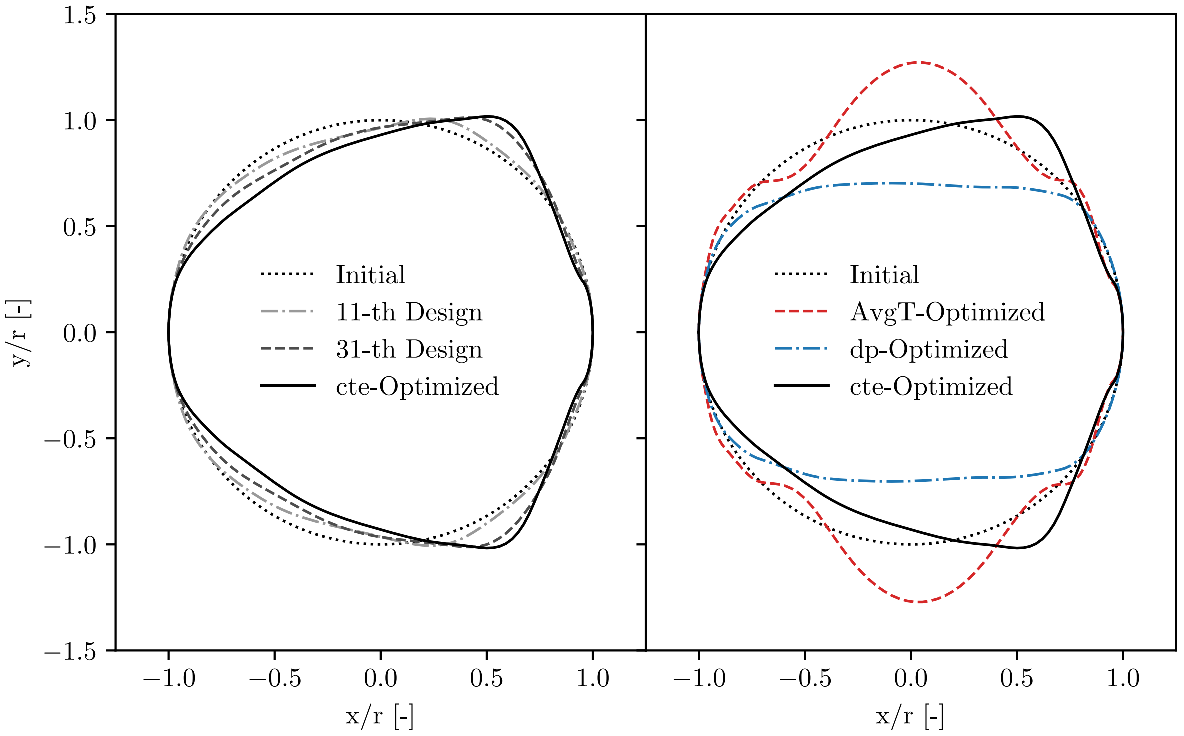


Figure 5.13: Comparison of full pin-shapes during the constrained (cte) optimization at various design's during the optimization, on the left. On the right, single objective optimization designs of average temperature and pressure drop are compared to the initial and cte-optimized shapes.

The optimized pin shape compared to the baseline geometry is shown in Fig. 5.13 together with two intermediate design steps that visualize the transition. The optimized geometry is shaped like a wedge, increasing the surface area prior to the flow separation at the newly formed sharp corner, thus allowing more heat to be effectively transported into the fluid body. The recirculation region in the cylinder's wake attributes only to a reduced amount to the overall cooling of the pin. The wedge shape is a compromise for the pressure drop constraint to be fulfilled whilst minimizing the temperature on the inner pin (which is not shown on these images). On the right side of Fig. 5.13, the single objective optimized designs are shown in comparison. The pressure drop optimized solution simply decreases the projected area against the flow direction and only does not decrease its size further due to a failing meshing algorithm. The outer pin surface gets too close to the clamped inner pin. The average temperature optimized pin on the other hand enlarges specifically in the pin midpoint to yield a greatly increased pin-surface. Due to the increased solid's thermal conductivity compared to the fluid, a larger surface area increases cooling capability of the pin. The pin-enlargement stops due to reached FFD-box point bounds.

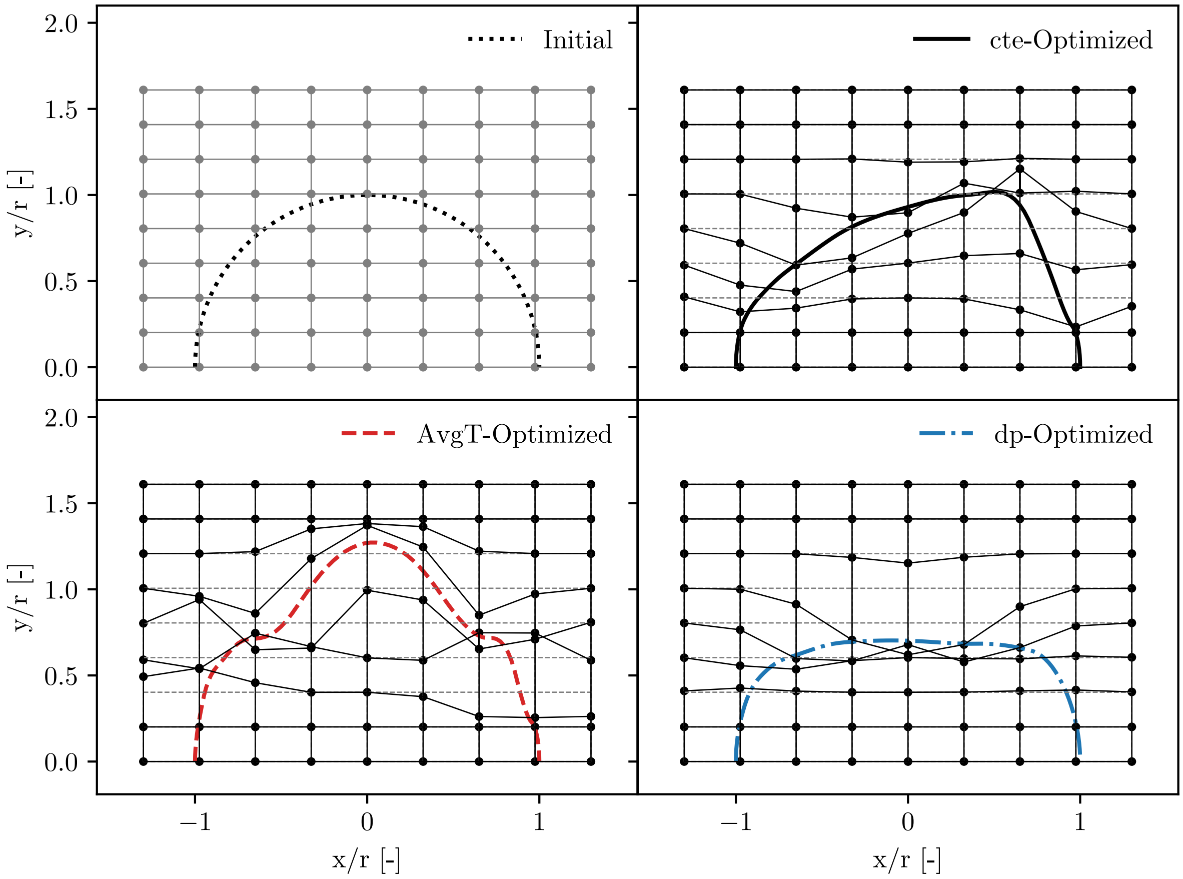


Figure 5.14: FFD-boxes that yield the constrained and single-objective optimized designs. Single-objective optimizations are severely limited by necessary FFD bounds and limits in the mesh deformation workflow.

The exact FFD-box deformations for the optimized designs are shown in Fig. 5.14. Note, that self-intersection of the FFD-lattice does not automatically yield an invalid geometry, such that a prevention of this mechanism is not strictly required. All optimized FFD-boxes exhibit minor and even more drastic forms of self-intersection of the FFD-box. The principal takeaway of the observed single-objective optimization is the unveiling of available design space with the complete employed setup. That certainly includes limitations of the FFD method itself, but large restrictions of the design space can be attributed to the author's choice of lattice resolution, B-Spline degree, active FFD-points as well as boundary positioning choice e.g. the inner pin diameter.

5.2 Transient Coupled CHT Cylinder in Crossflow

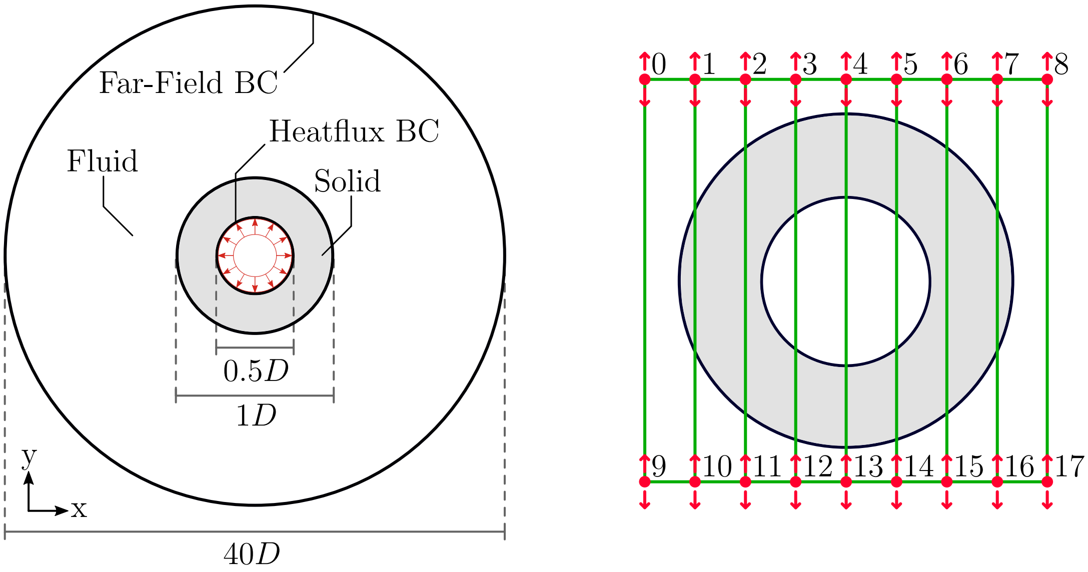
The well-established 2D cylinder in crossflow is used as an unsteady testcase, with the extension to a CHT case by introducing a hollowed solid pin. In previous joint work with Crome [Ven+19], the primal unsteady incompressible solver was validated against experimental and numerical data in literature, by monitoring lift, drag and the Strouhal number for the exact primal fluid setup utilized in this work.

After a brief introduction to the simulation setup, a gradient validation is performed in Sec. 5.2.1. The challenges in defining a suitable objective function for the present unsteady testcase are discussed in Sec. 5.2.2, before closing the section with a presentation of the optimization results in Sec. 5.2.3.

Geometry and Mesh description

The fluid domain around the cylinder is extended by a solid domain, which does not fill the cylinder entirely, but leaves an inner cylinder surface in order to properly set meaningful boundary conditions. Figure 5.15a sketches the geometry with all relevant length specifications relative to the cylinder diameter D. The cylinder diameter is $D = 1\,m$ for the geometry and mesh in this work. The inner cylinder, delimiting the solid domain, has a diameter of $0.5\,D$ and the farfield boundary is $20\,D$ away from the pin center.

The mesh is fully structured, i.e. only quadrilateral elements are used, and consists out of 22200 elements in the fluid domain and 4800 elements in the solid domain. A progression towards the no-slip fluid-solid interface is done in both zones, in order to properly resolve the boundary interface. See figure 5.16 for a close-up view of the mesh close to the cylinder. The remaining mesh topology stays fully structured, with a coarsening of the grid resolution towards the farfield boundary.



(a) Simulation domain setup with boundary conditions. The pin diameter D being 1 meter.

(b) FFD-lattice with numbering for the gradient validation. Arrows indicate that only vertical displacement is considered.

Figure 5.15: Sketches of the simulation domain (left) and shape deformations setup (right). Note that only the outer pin surface is deformed by the FFD-box, the inner pin remains clamped.

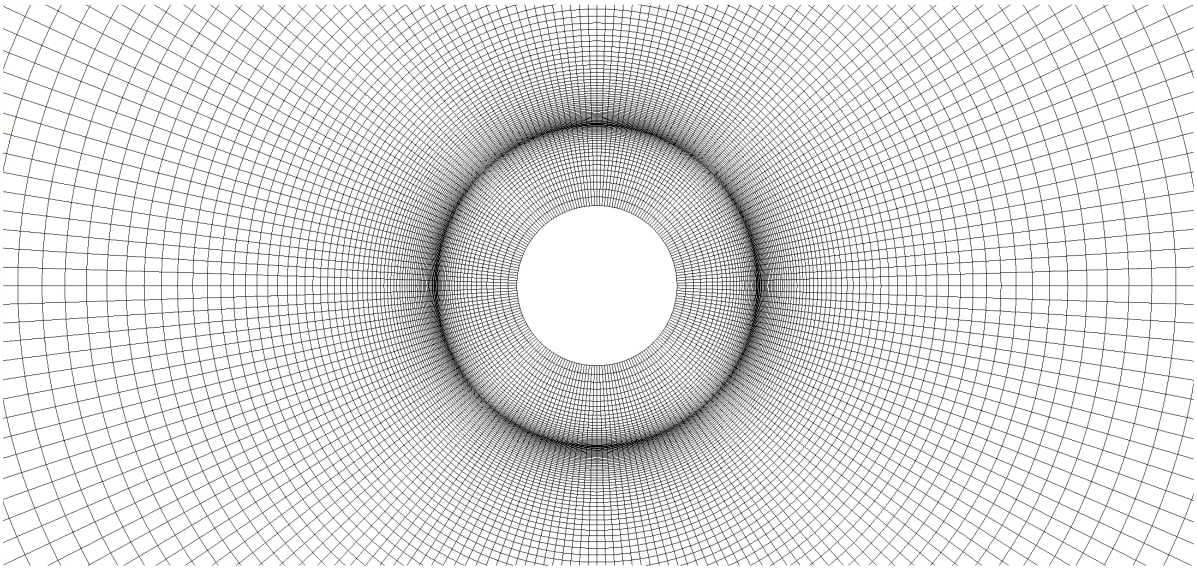


Figure 5.16: Mesh close-up centered around the pin. Although not specifically labeled, the mesh refinement at the fluid-solid interface reveals the different zones. The mesh contains 22200 elements in the fluid domain and 4800 elements in the solid domain.

Material properties

All material properties listed in the following are used as constants, e.g. no temperature dependent material properties are taken into account. The fluid represents water (note that an incompressible solver is used) with the material properties are listed below:

name	symbol	value
density	ρ	$1e3\,kg/m^3$
dynamic viscosity	μ_{lam}	1e -3 $Pa\,s$
specific heat capacity	c_p	$4180\,J/(kg\,K)$
laminar Prandtl number	Pr_{lam}	6.96

Table 5.3: List of fluid material properties.

The used material for the solid domain represents an aluminum alloy, frequently used in cooling application in various compositions. The material properties are given in the table below:

name	symbol	value
density	ρ	$2719\,kg/m^3$
thermal conductivity	κ	$200\,W/(m\,K)$
specific heat capacity	c_p	$0.8710\,J/(kg\,K)$

Table 5.4: List of solid material properties.

For the purpose of demonstration of optimization capability, the given value for the specific heat capacity is chosen, in order to achieve a larger amplitude in surface average temperature on the solid's inner pin, as a reaction to the occurring vortex shedding.

Boundary conditions

The boundary condition setup for the present 2D CHT cylinder in crossflow is straightforward, and also indicated in figure 5.15a.

On the outer circle of the fluid zone, a farfield boundary condition is applied. The chosen Reynolds number for the case is $Re = 100$ which represents a laminar regime where vortex shedding occurs behind the cylinder. In order to reach the specified Reynolds number:

$$Re = \frac{\rho v D}{\mu}, \tag{5.3}$$

the remaining variable in the freestream velocity is set to 1e -4 m/s in positive x-direction. The freestream temperature is set to $15\,°C$ or equivalently $288.15\,K$.

A heatflux boundary condition of $1e3\,W/m^2$ is applied on the inner pin of the domain, representing a heating element that requires cooling, e.g. power electronics from various applications. The heatflux boundary is indicated by red arrows in figure 5.15a.

Handling of the CHT-interfaces between fluid and solid is described in Sec. 2.2.4.

Numerical methods and time-stepping

The convective fluxes in the fluid are discretized using a flux difference splitting (FDS) scheme, and second order accuracy is achieved by the MUSCL scheme without an additional slope limiter. Gradients for reconstruction of viscous fluxes are computed using the Green-Gauss scheme. A constant CFL number is used but is reduced by one order of magnitude for the adjoint computation, as this helps to stabilize the convergence in the experience of the author.

For the solid heat conduction domain, the gradients are computed using the Green-Gauss scheme as well. Apart from that, no noteworthy additional methods were used. Note that the exact setups can be inspected in all detail as described in Sec. 4.3.

The following remarks on time-stepping are equally applied to the fluid and solid zone, so unless explicitly specified, the description is valid for both. A dual-time stepping 2nd order is used as described in section 2.2.1, thus the discretization of the pseudo-time derivative is done using the implicit Euler scheme. A constant physical time-step length is chosen to aim for 60 time-steps per full shedding cycle, and an a-prori estimation for the required time-step length is based on the Strouhal number:

$$St = \frac{fD}{v}, \tag{5.4}$$

with the frequency f in $1/s$ of the vortex shedding, the characteristic length D in m (here the cylinder diameter) and flow velocity v in m/s. The Strouhal number for this specific case of vortex shedding behind a circular cylinder at Reynolds number 100 is known from literature and previous investigations using the identical solver [Ven+19; Ven20] and is set to 0.166 for this estimation. The velocity is the freestream velocity at $1e$ -4 m/s. The remaining quantity in the shedding frequency can thus be computed to $1.66e$ -5 Hz. Via the period length T being $T = 1/f$ and an aimed at 60 timesteps per shedding cycle, the necessary time-step length is approximately 1004s.

The actual amount of time-steps per shedding cycle observed in the simulation is 61, i.e. the estimation was slightly off, which cannot be completely avoided in a discrete setting like the present. In the optimization section 5.2.3, 15 period lengths' for a total of 915 timesteps will be used with an optimization window of 5 period lengths' of 305 timesteps.

Geometry parametrization

The fluid-solid interface, i.e. the outer pin boundary, is the surface that we want to deform. The farfield and first of all the inner pin boundary remain fixed. The fixed inner pin surface guarantees a constant rate of heat flow in watts over the domain boundary during the optimization.

An FFD-box is used for its ease of use in this application and availability in the utilized code framework. The FFD-box fully encloses the pin and consists of 18 points. Only the vertical y-movement is considered, leaving a total of 18 design variables, see figure 5.15b. As the designing surface is fully submerged into the FFD-box, i.e. the FFD-box boundaries do not cut any designing surface, the application is much more straight-forward compared to the previous presented steady case, cf. section 5.1.2.

The used FFD blending is done via Bézier curves that results in smooth shapes despite low-resolved FFD-boxes, and is preferred over B-splines in this application as no limited support of the blending function is required. The downside is of course a limited design space compared to a low-order B-Spline blending. For the gradient validation and optimization the same FFD setup as before is used.

5.2.1 Gradient Validation

The sensitivities, computed by the discrete adjoint for the present unsteady CHT case, are validated by comparison against first order forward finite difference sensitivities.

The simulation duration is one complete shedding cycle for both, primal and adjoint. The objective functions, surface averaged inner pin temperature and drag of the cylinder surface, are time-averaged over the full shedding cycle with equal weighting of each timestep contribution.

The primal simulations are not run from freestream values, but from restart files that restart the simulation at a lift and drag peak that could be interpreted as the start of a shedding cycle. These files, from two timesteps for dual time-stepping 2nd order, are extracted from a simulation that already reached the quasi-steady state of the periodic shedding cycles. With 61 timesteps per shedding cycle, that results in a reasonably fast to evaluate setup for the numerous primal simulations required for the finite difference and the discrete adjoint.

In a straight-forward comparison of DA and FD sensitivities over all design variables

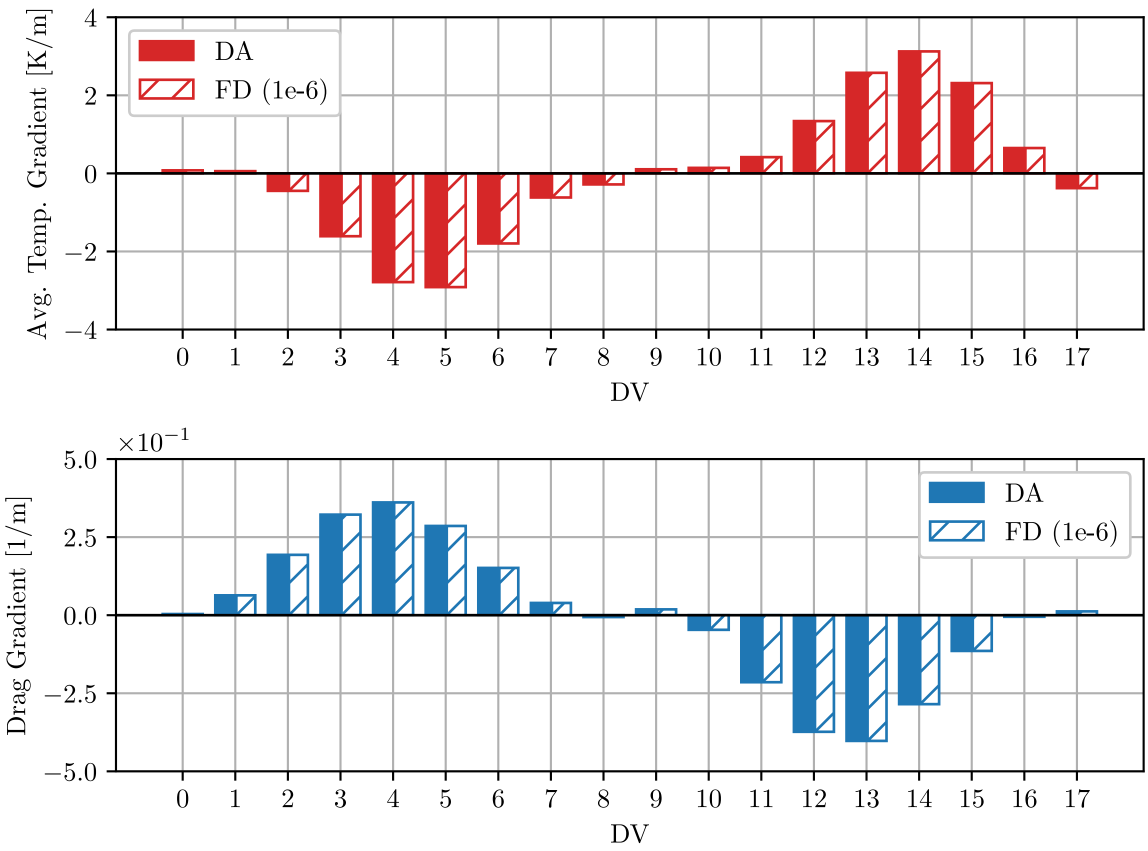


Figure 5.17: Gradient validation of surface averaged temperature and drag objective functions. Note that the imperfect symmetry between upper and lower side arises from inaccurate tracking of the exact shedding window with a limited number of timesteps.

with an FD stepsize of 1e -6 m, see figure 5.17, the overall agreement is excellent and certainly sufficient for all practical optimization purposes. A closer inspection of the gradient accuracy is performed below. The finite difference stepsize is simply chosen by taking 1e -6 m of a representative length of the to be designed object, which provides a good starting point for gradient validations. In the present case the representative length is naturally the pin diameter, but the choice becomes less clear for other, more complex, applications.

In order to fully determine the achievable gradient accuracy, a range of finite difference steps has to be evaluated for each design variable. This procedure also provides an optimal FD-step for an overall evaluation with a fixed stepsize, as performed in the previous paragraph.

For two selected design variables 4 and 13, being the middle points of the upper

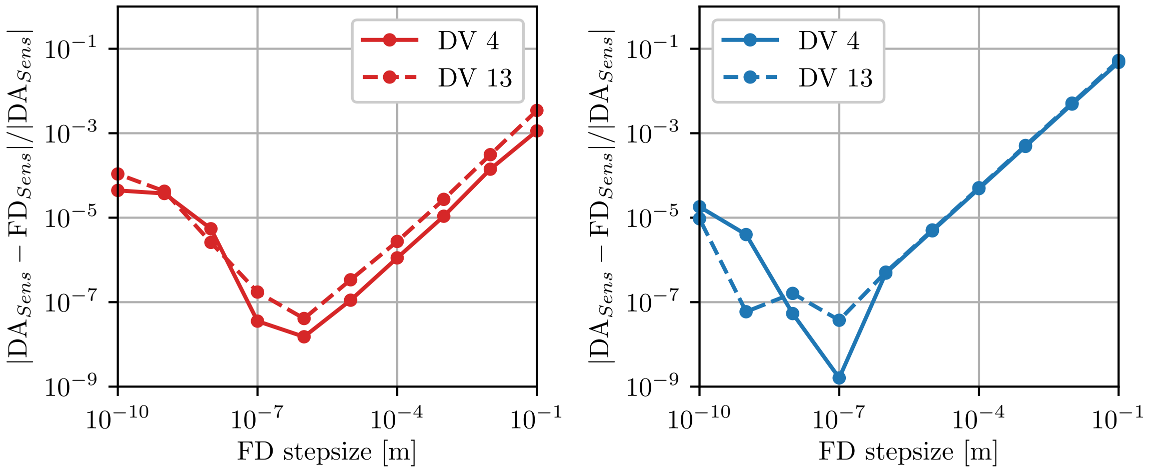


Figure 5.18: Sweep over finite difference step range with relative gradient difference on the y-axis. For average temperature objective on the left in red, and for drag in blue on the right.

and lower row of the FFD-box, see Fig. 5.18, a sweep over multiple FD-stepsizes is performed with a stride of one order of magnitude. The two DV's are chosen as their central position yields the largest deformations at the same displacement out of all design variables. Figure 5.18 shows the results for the surface average temperature objective function in red on the left, and for drag in blue on the right. The optimal accuracy for both objectives is not achieved at the same FD-stepsize, but overall the relative difference is below 0.1% across the observed values in a range of 10^{-3} to 10^{-10} . The optimum is at 10^{-6} for surface average temperature and at 10^{-7} for drag, respectively.

The curve form for the relative difference sweeps in Fig. 5.18 is expected: for large stepsizes a high truncation error occurs and after a valley with the optimal accuracy the round-off errors through the primal solver get dominant.

Additionally, to the line plot in figure 5.18, the gradient validation results are shown in table 5.5 and table 5.6 for both objective functions. The first row lists the DA derivative, and the subsequent rows the FD derivative with the red color indicating non-matching numbers. Additionally, the relative difference, computed after equation 5.2, to the DA value is given. The achieved agreement between DA and FD derivative for this unsteady CHT testcase is excellent, with relative differences below 0.00001% for all considered cases. Visualizing matching digits is another intuitive mean of showcasing accuracy in most cases but can be misleading at times, e.g. consider the edge case 1.000... versus 0.999... where no matching digits are achieved.

magnitude p	FD DV 4	rel diff (%)	FD DV 13	rel diff (%)
DA	-2.7852042730	-	2.5764232907	-
-1	-2.7820359710	1.1375474079e-1	2.5674963880	3.4648431850e-1
-2	-2.7855965580	1.4084605648e-2	2.5756228427	3.1068184045e-2
-3	-2.7852344518	1.0835432068e-3	2.5763533133	2.7160673582e-3
-4	-2.7852073878	1.1183694342e-4	2.5764162256	2.7421847623e-4
-5	-2.7852045832	1.1138609945e-5	2.5764224176	3.3887097505e-5
-6	-2.7852042308	1.5150111880e-6	2.5764231850	4.1021565999e-6
-7	-2.7852041739	3.5559178250e-6	2.5764228439	1.7339908113e-5
-8	-2.7852195216	5.4748887415e-4	2.5764165911	2.6003201919e-4
-9	-2.7851001505	3.7384150634e-3	2.5763142730	4.2313574733e-3
-10	-2.7853275241	4.4252114843e-3	2.5761437427	1.0850233230e-2

Table 5.5: Gradient validation for surface average temperature for DV4 and 13. Magnitude p indicates the finite difference stepsize with 1e $-p$. Relative difference in % is computed after Eq. (5.2). The first row holds information for the discrete adjoint gradient and matching numbers are indicated in red.

magnitude p	FD DV 4	rel diff (%)	FD DV 13	rel diff (%)
DA	0.3610934912	-	-0.4025129357	-
-1	0.3782510748	4.7515626808e-0	-0.3812776084	5.2756881816e-0
-2	0.3628630057	4.9004329990e-1	-0.4004358762	5.1602306375e-1
-3	0.3612694443	4.8727844207e-2	-0.4023065028	5.1286027164e-2
-4	0.3611111003	4.8766139794e-3	-0.4024923214	5.1214067918e-3
-5	0.3610952552	4.8853072459e-4	-0.4025108740	5.1222613281e-4
-6	0.3610936669	4.8664178557e-5	-0.4025127289	5.1392192897e-5
-7	0.3610934906	1.6068829409e-7	-0.4025129207	3.7299883815e-6
-8	0.3610935106	5.3736165127e-6	-0.4025130007	1.6129263499e-5
-9	0.3610949317	3.9892418055e-4	-0.4025129118	5.9365719239e-6
-10	0.3611000387	1.8132465200e-3	-0.4025091371	9.4373457743e-4

Table 5.6: Gradient validation for drag for DV4 and 13. The remaining table information are identical to table 5.5.

Similar to the gradient validation for the steady testcase in the previous chapter 5.1.3, the sign difference of the derivative values between the objective functions indicates the resulting optimization conflict. Consider the drag derivative value for DV 4, which has a positive value. DV 4 is on the top row of the FFD-box so displacing the FFD-point in positive y-direction, thus enlarging the cylinder, will intuitively increase the cylinder's drag coefficient. This intuitive expectation is confirmed by the positive derivative value and thus provides a good sanity check. Between DV 4 and DV 13, the derivative obviously switches its sign, as a displacement in positive y-direction results in a reduced cylinder size. The derivative sign switches for the surface averaged temperature objective function compared to drag, which results in an opposite deformation and explanation thereof. This circumstance showcases the conflicting nature of the two considered objectives.

Due to the integration of one full period of the present periodic shedding process, the theoretical expectation would be to obtain identical absolute derivative values with opposite sign of two FFD-Box points at the same downstream location, but in the top and bottom FFD-row. DV 4 and DV 13 present such a pair, and for the surface average temperature objective the DA derivatives are approximately -2.8 and 2.6. A similar effect can be observed for the other FFD-point pairs from upper side (DV's 0-8) and lower side (DV's 9-17) of the cylinder's FFD-box. The slight imperfection can be explained with the inherent inaccurate temporal discretization of the shedding cycle which fails to exactly track beginning, midpoint and end of the shedding cycle. With an increasing number of timesteps the true period could be captured more precisely at the expense of growing compute cost. The result is a marginally varying weighting of the upper and lower side contributions.

5.2.2 Challenges in Objective Function Definition

The initial design provides a periodic vortex shedding behind the cylinder, which is reflected in the periodically oscillating values of interest: surface average temperature and drag. These values of interest over a time-series are then converted into a scalar quantity via some form of representative time-average over a specific time window, that is not necessarily the full simulation length. This section works out the challenges in defining a physically reasonable objective function and adjoint integration length taking computational costs into account, and discusses the approach employed in this thesis.

For the gradient validation in the previous section, the objective function was the time average over the full simulation length, with the simulation running for exactly one period starting from a converged solution. The discrete adjoint is integrated backwards over the complete primal as well. The converged solution used is from the initial geometry and 'converged' means, that the initial nonphysical state set at the simulation start, has no more influence on the solution, and periodic shedding occurs. The issue in terms of physical correctness with that approach is, that the utilized converged solutions are only valid for the initial geometry, but for the finite difference, the deformed geometries are restarted for just one period from that very same solution. That one period is of course not enough for the solution to become independent of the initial solution. However, albeit not physically sound, that approach is very well suited and valid, to validate the gradients with respect to the discrete optimization problem. In an optimization, independent of the chosen restart values, it is therefore necessary to ‘wash out’ the nonphysical transient that comes with each design change before evaluating the objective function.

In an optimization, the goal is of course to optimize the true time average value, i.e. the simulation needs to reach its periodic (or otherwise cyclic) state before the time averaged objectives can be known. A ‘brute-force’ approach would be to let each design start from e.g. constant freestream values, and to update a running time average after each time step until that very running average converges to a suitable exit condition. Afterwards, by integration the adjoint over the complete time interval, the discrete adjoint will yield a gradient that is fully consistent with the primal problem. That approach is quite general and robust against significant period length shifts, including that a design in the optimization cycle does not exhibit any more unsteady behavior. The drawback is of course it being computationally quite expensive, and there is evidence of instability of the gradient evoked by chaotic behavior. This becomes an issue with increasing integration lengths and for scale-resolving simulations like (D)DES, LES or DNS. A discussion of the topic and proposed remedy in windowing is presented and applied in SU2 by Schotthöfer et al. [Sch+; Sch20]. Another issue with integrating the adjoint over the complete time series is, that the resulting sensitivity then contains an influence of the nonphysical initial transient phase. Which is desirable to avoid, as only the physical, stabilized portion of the simulation is of interest. It is therefore recommendable to only backwards integrate the adjoint over a fraction of the primal time-window that constitutes a physical state.

In this work, a computationally less demanding approach was taken compared to the ‘brute-force’, by restarting each new design from the converged solution of the initial design but letting the simulation run for 10 full periods before starting the optimization window of 5 full periods. In the optimization window, the objective functions are time-averaged with constant weighting factors, and the adjoint is integrated backwards only for the optimization window of 5 periods. This of course makes for a primal simulation duration of 15 periods, based on the initial design’s period length, which is fix. This would become problematic if the period length increases up to the point, where the time average in the optimization window does not represent the true time average. The occurring errors in the gradient due to missing tracking of period start and end are accepted in a cost-accuracy trade-off for the present case.

See the dashed lines in figure 5.22 for the time-history of the objective function values for the initial design, including indication of the optimization window. Note for that figure, that drag and surface average temperature cycle twice during one period as both exhibit an identical behavior, independent on whether a vortex sheds at the upper or lower side.

The adjoint is integrated backwards in time only for the length of the optimization window. Like so, only information of the design's true shedding cycle has an influence on the gradient, but as mentioned above, the optimization window most likely does not contain an integer amount of cycles which results in a non-perfect time-average. The cost-accuracy trade-off has to be made for each specific case by setting the period/time-lengths of the optimization window and the initial transient phase to reach a physical state. Unfortunately, the user has no real alternative in this approach other than making an educated guess based on the time-solution of the initial geometry, but an a posterior check has to be made whether the window length is still reasonable for the optimized geometry. In figure 5.22 the solid line represents the solution of an optimized solution and as the objectives are in a periodic state in the optimization window with only a slight shift in period length, the chosen window lengths can be concluded as suitable.

In conclusion, the applied approach consists of using restart files of the initial design throughout the complete optimization, and fixing an initial wash-out phase and a subsequent optimization window. The shortcomings in inaccurate period resolution and required a posterior checks are made up for and outweighed by the ease of set up and low computational cost.

5.2.3 Optimization Results

After the introduction of the unsteady CHT case setup in Sec. 5.2, a gradient validation in Sec. 5.2.1 and the discussion of a suitable objective function with the associated sensitivity computation in Sec. 5.2.2, this section presents constrained shape optimization results.

The optimization objective is to minimize time-averaged surface average temperature on the inner pin surface, with an inequality constraint on time averaged drag of the cylinder to not exceed the value of the initial geometry, 1.332.

The geometry parametrization is done via an FFD-box as presented in the previous Sec. 5.2, with an additional step explained in the following to preserve symmetry along the downstream axis. It can not be assumed, that the gradients will be identical for the top and the bottom row of the FFD-box. This leads to an asymmetrical pin, with respect to the x-axis, and is not desired behavior as the cause for this behavior are slight imperfections in the objective function definition and thus the gradient. Therefore, the arithmetic average of the gradients of lower and upper side is equally set for both sides and handed to the optimizer in a simple postprocessing step. This gradient postprocessing can be easily incorporated within the FADO optimization framework via a simple Python-script. The optimized designs with their associated, symmetric FFD-boxes are shown in figure 5.21. The discrepancies in gradients between lower and upper side were found to be small, but added up over multiple optimization steps the non-symmetric nature of the shapes were noticeable, albeit the overall design direction was quite similar. Lastly, the bounds of the FFD-box point displacements is set to 0.3 in negative and positive y-direction, especially in order to avoid a collapse of the outer pin-surface onto the inner pin surface.

The optimization history is shown in figure 5.19, which features the evolution of objective and constraint value as well as the respective normalized gradient norms. As expected, the optimizer takes more aggressive steps in the first 10 iterations, thus violating the constraint which is always recovered in at most 2 subsequent iterations. In the following 30 iterations the gains per iteration are less significant, but the objective continues to drop, and at iteration 42 the optimization can be considered as converged. The constraint value in the final design is very slightly violated by 0.0009 at a value of 1.3329. The objective function in time-averaged surface average temperature dropped by 3.7 Kelvin from $357.5K$ to $353.8K$.

Due to the drag constraint and the reached FFD-box bounds for selected points, the gradient norm of the objective function cannot vanish and only drops by about ∼12 percentage points. The gradient norm of the constraint is added for the sake of completeness, and does allow for any conclusive statement. The normalization in the plot is done via division by the initial design's gradient norm.

Selected shapes from the optimization history are shown on the left side of Fig. 5.20. The general design direction is towards a predominant wedge shape (or v-shape) with the thickest cross-section being shifted downstream. In the front to middle section, the pin shrinks, whereas it thickens in the rear section. A notable aspect is that design 17 is slightly slimmer in the front section compared to the final design, i.e. a more aggressive wedge shape in design 17 is slightly reversed to arrive at the final constrained optimized design. The optimized shape for this case resembles the steady optimized shape from the pin-array in Sec. 5.1.5 in its primitive design rules, such that the same

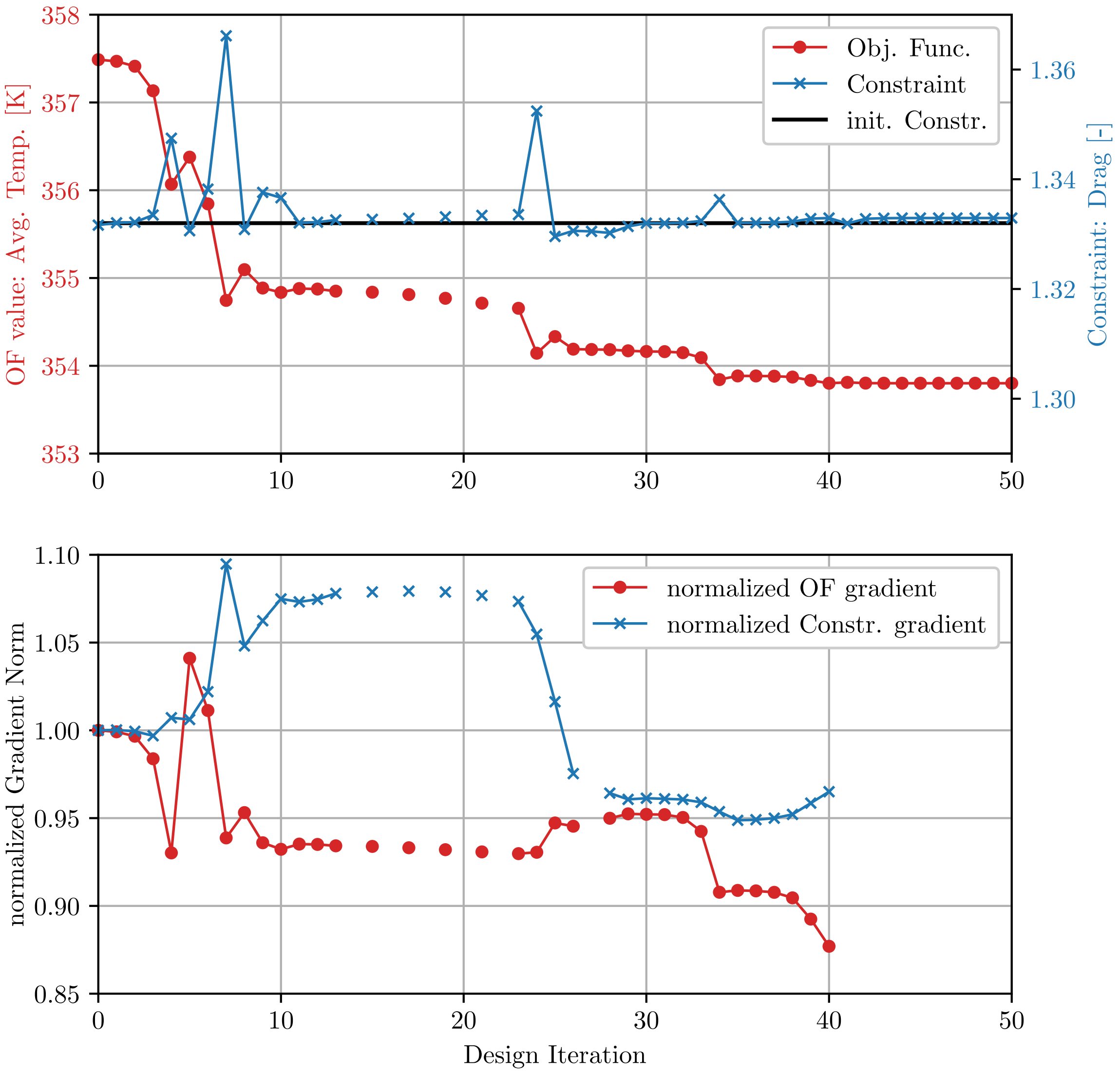


Figure 5.19: Optimization history for unsteady cylinder in crossflow. Objective function value and constraint value in the upper image and their respective gradient norm values in the lower. Line elements are only drawn if two subsequent evaluation points represent valid points, i.e. points miss due to invalid meshes, or gradients are not computed because the optimizer rejected the design for further evaluation.

physical explanation applies. Note as well that, despite not being explicitly enforced, an approximate pin-volume constraint is achieved by the constrained setup.

The right side of Fig. 5.20 features optimized design with respect to both objectives functions without an additional constraint. The resulting shapes are straight-forward:

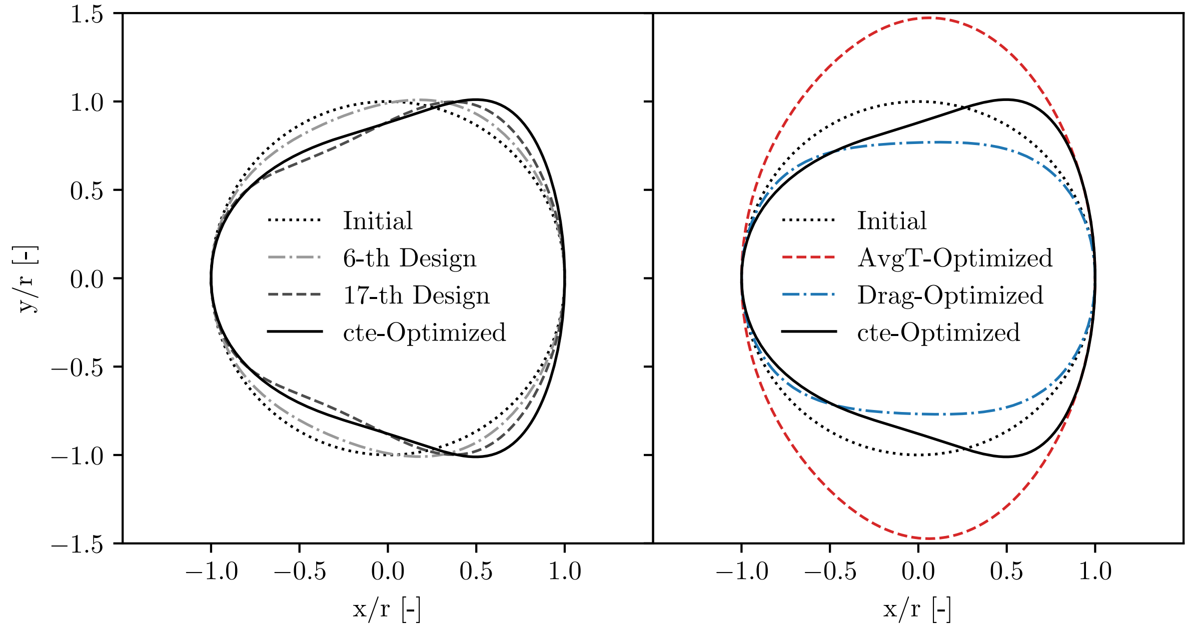


Figure 5.20: Comparison of full pin-shapes during the constrained (cte) optimization at various design's during the optimization, on the left. On the right, single objective optimization designs of average temperature and drag are compared to the initial and cte-optimized shapes.

for surface average temperature the pin thickens to gain an increased surface area to transfer heat into the fluid domain, and for drag the primitive design direction is naturally to shrink the projected area against the flow direction and to ultimately collapse entirely (which does not happen here due to FFD-limitations).

The average temperature optimized shape is mostly limited by the FFD-box bounds, see Fig. 5.21. But the not-exceeded FFD-points in the pin's front section indicate, that a minimum of aerodynamic performance has to be retained. In the unconstrained drag optimization, the limiting factor is a degrading mesh. Note that a fixed inner pin surface sets a hard limit to the maximum shrinkage of the outer pin/cylinder. As the mesh deformation process results in an invalid mesh beyond that shown design, the front and back FFD-points remain nearly undeformed. The gradient value at the front and back is much lower compared to the middle section, and with more conservative FFD-bounds, these section would most likely shrink after the middle section stopping at the bounds. Both unconstrained optimized designs are a valuable indicator for the available design space, which the used geometry/mesh deformation and optimization setup is able to achieve. The trivial shape solution to the unconstrained drag optimization would be

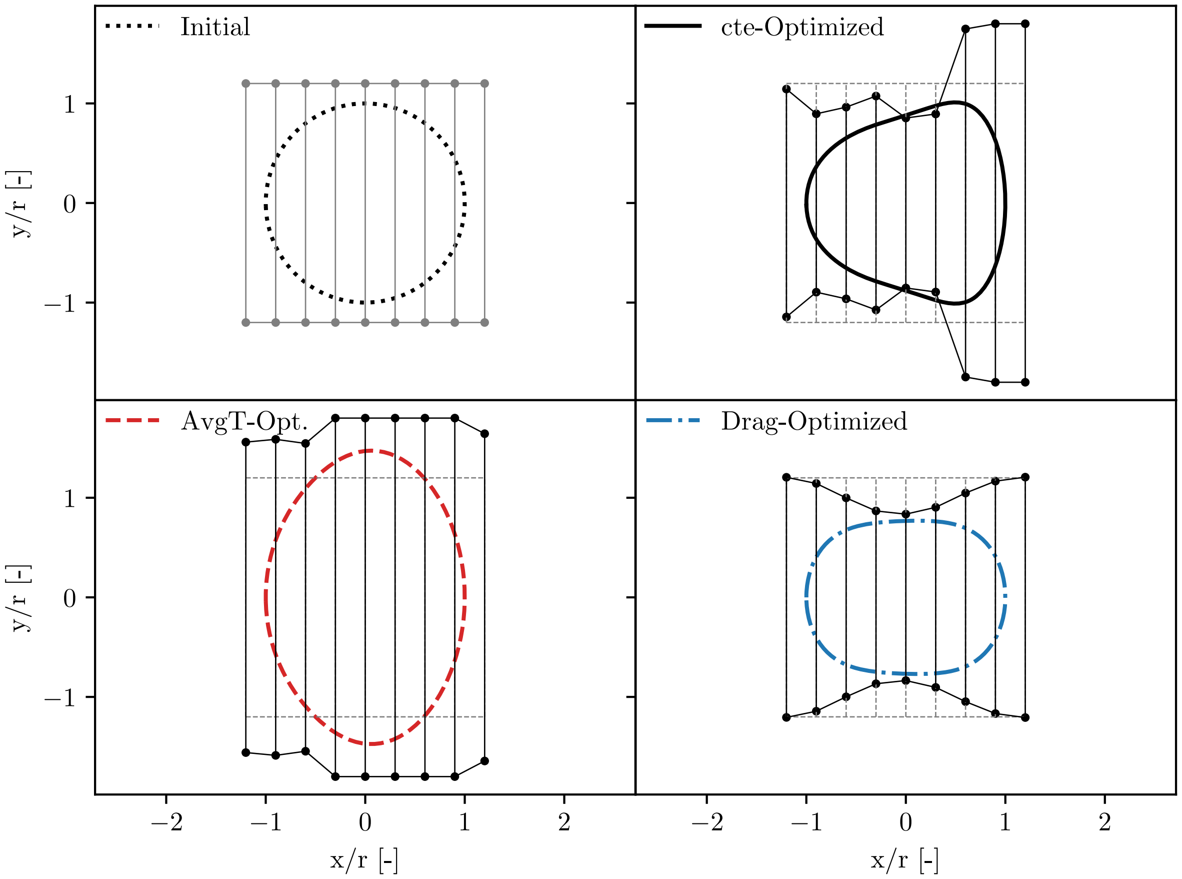


Figure 5.21: FFD-boxes that yield the constrained and single-objective optimized designs. Single-objective optimizations are severely limited by necessary FFD bounds and limits in the mesh deformation workflow.

e.g. a fully collapsed pin, which is not achievable with the used setup.

As a final step to prove the validity of the chosen optimization window and start-up length, the evolution of surface average temperature and drag of the constrained optimized design is shown in Fig. 5.22, alongside the same values for the initial design. Recall that all designs are restarted from the same converged solution based on the initial design, and thus a ‘wash-out’ phase of the nonphysical transient response is required. It can be noted, that the chosen 10 periods before the 5 periods of optimization window are sufficient to reach a new stable periodic regime. To the right of Fig 5.22, additionally the results of an FFT (Fast Fourier Transformation) analysis over the optimization window are shown. The intuitive observations is confirmed that there is only a slight decrease in period length, i.e. an increase of the dominant frequency. The less pronounced frequency peak for the optimized design can be explained by the optimization window

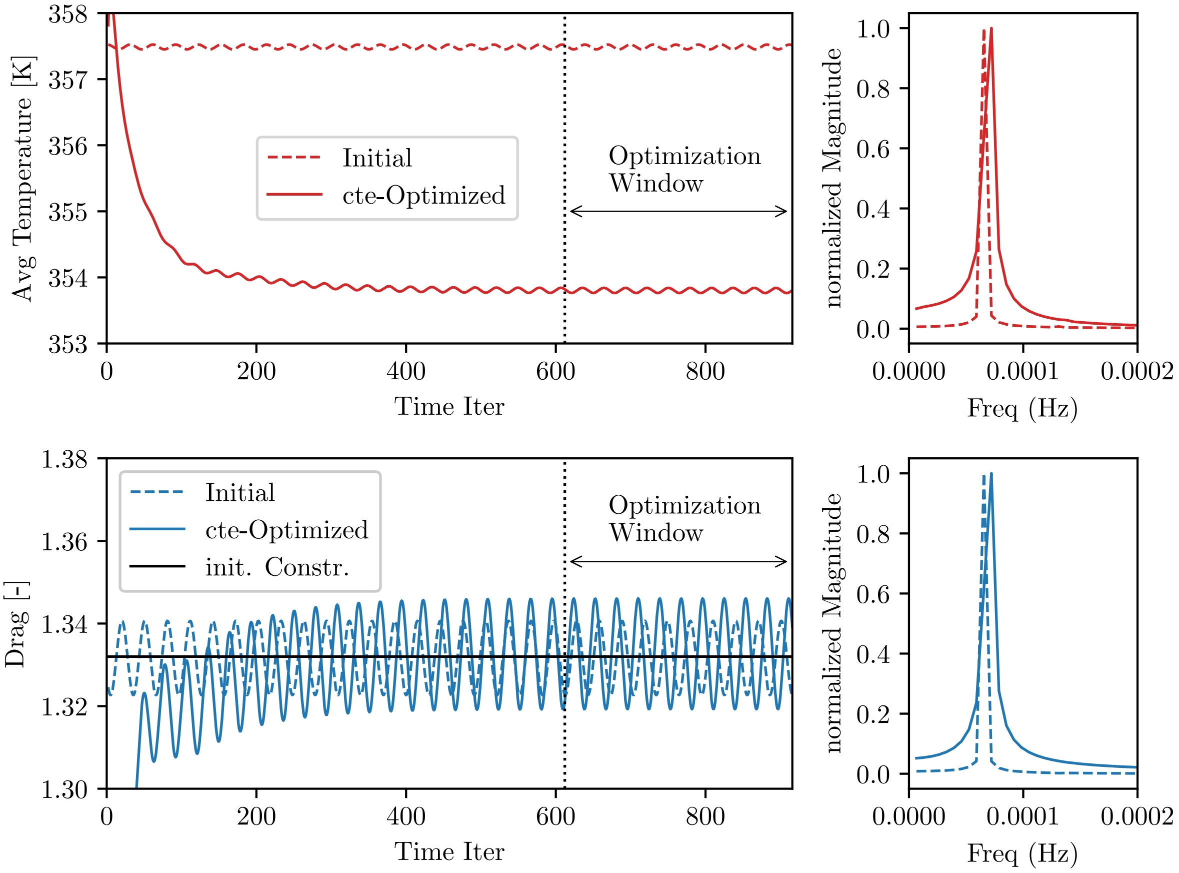


Figure 5.22: Time-history and FFT frequency analysis of the objective function and constraint value for the initial (dashed line) and constrained optimized (solid line) design. The primal simulation duration features an initial transient phase of 10 periods followed by 5 periods of the optimization window where the objectives are recorded, and the adjoint is integrated backwards.

not capturing an integer number of cycles, which is known as spectral leakage.

In summary, the here presented optimization setup builds on an accurate and validated gradient, but introduces slight inconsistencies through the specific optimization formulation that successfully balances computational cost and accuracy, and thus enables time-efficient unsteady CHT optimizations.

6 Conclusion and Outlook

6.1 Conclusion

This thesis has investigated discrete adjoint-based shape optimization for steady and unsteady conjugate heat transfer cases.

For this purpose, the available density-based incompressible was extended to handle unsteady flows with a dual-time stepping scheme. This unsteady incompressible solver was validated against literature data in preliminary work, and the identical primal fluid zone setup was employed for the unsteady application in this thesis.

As the primary application considered in the present work are heat exchangers based on pin-arrays, the concept of streamwise periodic flow was introduced to approximate an extended pin-array on a computationally efficient representative elementary unit cell. Pin-arrays potentially consist of thousands of pins, such that the implementation of streamwise periodic flow enables a computationally feasible performance evaluation of a pin design for steady state applications. In addition to the unsteady application, a steady state testcase for a pin array was set up and investigated. The streamwise periodic approach of momentum and energy source terms is extended to handle arbitrary boundary conditions related to the energy equation, as required for CHT, by applying an outlet heat sink to the energy equation as well as dealing with prescribed massflow. A simulative validation of the streamwise periodic approach was made against a simulation with numerous downstream repetitions of the unit cell domain.

The gradients of scalar objective functions with respect to an arbitrary set of design variables, that are used for gradient-based shape optimization, were computed using the discrete adjoint method. Based on a generic multi-physics discrete adjoint setup and a discrete adjoint setup for unsteady FSI, the discrete adjoint for unsteady CHT application was constructed. A detailed introduction into algorithmic differentiation was given, to pave the way for a step-by-step explanation of the discrete adjoint method. The discrete adjoint was first introduced on a plain singlezone steady formulation, and after independent introduction of required modifications for singlezone unsteady and multizone steady setups, the unsteady multizone discrete adjoint for CHT was constructed.

For streamwise periodic flow, the additional fixed point equation in pressure drop for a prescribed massflow required a treatment in the discrete adjoint approach, which is consistent to the regular PDE state variables. The essential impact of this specialized contribution was showcased at the example of the steady state application.

In the elementary unit cell application, a sophisticated geometry and mesh deformation workflow had to be set up. This workflow guarantees the preservation of a valid periodic domain and symmetry boundary conditions, as well as equal deformation of all involved pin-segments in order to retain just a single pin shape in the pin-array. The gradient validation with respect to surface average temperature and pressure drop objectives showed very good results. The subsequent optimization highlights the robustness of the workflow, including mesh deformation, primal and adjoint computation, over a complete shape optimization cycle.

Finally, the unsteady conjugate heat transfer application features a single cylinder in laminar crossflow, with a hollowed inner pin as the solid domain. Excellent results were achieved in the gradient validation of time-averaged surface averaged temperature and drag. The challenges in defining a suitable objective function and setting up a discrete adjoint workflow were discussed, and the approach used in this thesis of employing a fixed nonphysical transient wash-out phase followed by a concise optimization window was thus motivated. This constitutes an increased effort compared to steady-state cases, that has to be reevaluated and tuned for each case and no one-fits-all solution could be found. Successful optimization results for the unsteady setup underline the applicability and validity of the unsteady multizone gradient over a considerable range of design modifications.

The SU2 project in general was introduced, covering the codebase with its various aspects including the development platform and affiliated documentation, as well as the community and its engagement via various roles. The achievements of this thesis naturally build on the existing SU2 framework developed by many researchers, showcasing the flexible expandability of the project, which is substantially enabled by the concept of open source built around an active community. All the methods are implemented in the open-source solver SU2, and the author's code contributions were highlighted throughout the thesis. The results presented in this work are aimed at to be fully reproducible and therefore easy to modify and extend, which is one of the primary motivations for the extended effort that is accompanied by such aspiration.

6.2 Outlook

The interest in multi-physics simulation has grown over the recent years, in parts due to availability of coupled solvers in widespread commercial and open-source solvers, as well as computational affordability due to advances in hard- and software. A similar observation can be made for time-accurate simulations. Associated therewith is an increased requirement of gradient availability for optimization efforts. The present thesis contributes an efficient approach via the discrete adjoint method to compute shape gradients for CHT applications at the example of a simple laminar cylinder in crossflow.

A natural progression in the field would be the extension to applications featuring more complexities as e.g. turbulent regime, temperature dependent material properties, increasing the design space in the optimization, 3D domain, less predictable unsteady flow topology due to more challenging geometries, extending optimization windows, and many more possibilities. This helps to better understand the practical limits of the current implementation and can help to guide prioritization of future work.

An interesting comparison in terms of coupling approaches would be between the current segregated approach and a monolithic solver. A monolithic solver loses parts of the compelling generic nature of the segregated approach as implemented in SU2, but is considered to be more stable and robust in the primal execution. In terms of impacts on robustness of the unsteady adjoint in particular, windowing [Sch20] was mentioned as a noteworthy remedy but also the method of least squares shadowing [WHB14; Blo+18] was already successfully applied to fluid dynamics applications. It is known, that the sensitivity computation for unsteady applications can break down, and thus becomes unusable, due to chaotic behavior of the dynamical system [AZG19], and the aforementioned methods can supposedly alleviate this problem.

The principal application considered in this work were pin-fin heat exchanges. While that constitutes a widespread application in industrial devices and therefore also a rich research field, different CHT cases like turbine blade cooling or thermal management of batteries under consideration of Joule heating offer intriguing research questions. Introducing the developed methods and capabilities to a wider audience from other fields is potentially best done by showcasing shape meaningful optimization for various (unsteady) CHT applications.

If manufacturable designs are supposed to be generated by the optimization directly, one has to additionally formulate and implement geometry constraints, which adds yet another level of complexity to the existing design chain. This is often a necessity though for industrial devices that can have quite restricting manufacturing processes, up to the point where there is barely any available design space. In these cases, gradient-based optimization is potentially overkill, and methods like evolutionary algorithms might be more straight-forward. To achieve widespread use and acceptance in the industrial world, simulation tools like SU2 for gradient-based optimization have to provide a rich set of applicable geometry constraints.

Quite generally, a 'stacking of complexities' can be observed, and this thesis that builds on SU2 makes no exemption. Primal capabilities for unsteady incompressible flow and heat domain needs to be coupled, streamwise periodic flow has to be available, a discrete adjoint solver for the unsteady coupled problem needs to be validated and the geometry and mesh deformation workflow needs to work as intended. All of that needs to exhibit stable convergence over the optimization procedure for shape optimization, and optimally, show a performance like industry-leading commercial tools. For a small, rotating development team, mostly consisting of PhD-students, like in SU2, it therefore is even more substantial to focus on sustainable feature development and maintenance. Like so, even further advances can be made, without the incorporated chain of code modules and methods breaking for every case anew.

Bibliography

[ASG15] Tim A. Albring, Max Sagebaum, and Nicolas R. Gauger. “Development of a Consistent Discrete Adjoint Solver in an Evolving Aerodynamic Design Framework”. In: *16th AIAA / ISSMO Multidisciplinary Analysis and Optimization Conference*. 2015. DOI: 10.2514/6.2015-3240.

[ASG16] Tim A. Albring, Max Sagebaum, and Nicolas R. Gauger. “Efficient Aerodynamic Design using the Discrete Adjoint Method in SU2”. In: *17th AIAA / ISSMO Multidisciplinary Analysis and Optimization Conference*. 2016. DOI: 10.2514/6.2016-3518.

[AZG19] Tim A. Albring, Beckett Y. Zhou, and Nicolas R. Gauger. “Challenges in Sensitivity Computations for (D)DES and URANS”. In: *AIAA Scitech 2019 Forum*. 2019. DOI: 10.2514/6.2019-0169.

[Bea05] Steven B. Beale. “On the Implementation of Stream-Wise Periodic Boundary Conditions”. In: *Proceedings of the ASME Summer Heat Transfer Conference* 2 (Jan. 2005), pp. 771–777. DOI: 10.1115/HT2005-72271.

[Bea06] Steven B. Beale. “Use of Streamwise Periodic Boundary Conditions for Problems in Heat and Mass Transfer”. In: *Journal of Heat Transfer* 129.4 (Dec. 2006), pp. 601–605. ISSN: 0022-1481. DOI: 10.1115/1.2709976.

[BGE19] Ole Burghardt, Nicolas R. Gauger, and Thomas D. Economon. “Coupled Adjoints for Conjugate Heat Transfer in Variable Density Incompressible Flows”. In: *AIAA Aviation 2019 Forum*. 2019. DOI: 10.2514/6.2019-3668.

[Bla15] Jiri Blazek. *Computational Fluid Dynamics: Principles and Applications*. Ed. by Hayley Gray. Third Edition. Butterworth-Heinemann, 2015. ISBN: 9780080999951. DOI: 10.1016/C2013-0-19038-1.

[Blo+18] Patrick J. Blonigan et al. “Least-Squares Shadowing Sensitivity Analysis of Chaotic Flow Around a Two-Dimensional Airfoil”. In: *AIAA Journal* 56.2 (2018), pp. 658–672. DOI: 10.2514/1.J055389.

[BSG21] Johannes Blühdorn, Max Sagebaum, and Nicolas R. Gauger. *Event-Based Automatic Differentiation of OpenMP with OpDiLib*. Preprint: `https://arxiv.org/abs/2102.11572`. 2021.

[Bur+22] Ole Burghardt et al. "Discrete adjoint methodology for general multiphysics problems". In: *Structural and Multidisciplinary Optimization* 65.28 (2022). DOI: `10.1007/s00158-021-03117-5`.

[Car+13] Angelo Carnarius et al. "Optimal control of unsteady flows using a discrete and a continuous adjoint approach". In: *IFIP Conference on System Modeling and Optimization*. Ed. by Dietmar Hömberg and Fredi Tröltzsch. Springer Berlin Heidelberg, 2013, pp. 318–327. ISBN: 978-3-642-36062-6. DOI: `10.1007/978-3-642-36062-6_32`.

[CB16] Julien Cohen and David Bourell. "Development of Novel Tapered Pin Fin Geometries for Additive Manufacturing of Compact Heat Exchangers". In: *27th Annual International Solid Freeform Fabrication Symposium – An Additive Manufacturing Conference*. Nov. 2016.

[Chr94] Bruce Christianson. "Reverse accumulation and attractive fixed points". In: *Optimization Methods and Software* 3.4 (1994), pp. 311–326. DOI: `10.1080/10556789408805572`.

[Cop+] Sean Copeland et al. "Adjoint-Based Goal-Oriented Mesh Adaptation for Nonequilibrium Hypersonic Flows". In: *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. DOI: `10.2514/6.2013-552`.

[CS14] Scott Chacon and Ben Straub. *Pro Git*. 2nd ed. Apress, Berkeley, CA, 2014. ISBN: 978-1-4842-0076-6. DOI: `10.1007/978-1-4842-0076-6`.

[Eco14] Thomas D. Economon. "Optimal shape design using an unsteady continuous adjoint approach". Supervisor: Juan J. Alonso. PhD thesis. Stanford University, Department of Aeronautics and Astronautics, Aug. 2014.

[Eco20] Thomas D. Economon. "Simulation and Adjoint-Based Design for Variable Density Incompressible Flows with Heat Transfer". In: *AIAA Journal* 58.2 (2020), pp. 757–769. DOI: `10.2514/1.J058222`.

[ED16] M. P. Errera and F. Duchaine. "Comparative study of coupling coefficients in Dirichlet–Robin procedure for fluid–structure aerothermal simulations". In: *Journal of Computational Physics* 312 (2016), pp. 218–234. ISSN: 0021-9991. DOI: `10.1016/j.jcp.2016.02.022`.

[Gil08] Mike B. Giles. “Collected Matrix Derivative Results for Forward and Reverse Mode Algorithmic Differentiation”. In: *Advances in Automatic Differentiation*. Ed. by Christian H. Bischof et al. Springer Berlin Heidelberg, 2008, pp. 35–44. ISBN: 978-3-540-68942-3. DOI: `10.1007/978-3-540-68942-3_4`.

[GP00] Mike Giles and Niles Pierce. “An Introduction to the Adjoint Approach to Design”. In: *Flow, Turbulence and Combustion* 65 (Apr. 2000). DOI: `10.1023/A:1011430410075`.

[GP20] Pedro Gomes and Rafael Palacios. “Aerodynamic-driven topology optimization of compliant airfoils”. In: *Structural and Multidisciplinary Optimization* 62.4 (2020), pp. 2117–2130. DOI: `10.1007/s00158-020-02600-9`.

[GP97] M. Giles and N. Pierce. “Adjoint equations in CFD - Duality, boundary conditions and solution behaviour”. In: *13th Computational Fluid Dynamics Conference*. 1997. DOI: `10.2514/6.1997-1850`.

[GPG19] K.T. Gkaragkounis, E.M. Papoutsis-Kiachagias, and K.C. Giannakoglou. “Conjugate heat transfer shape optimization of internal cooling systems using continuous adjoint in OpenFOAM”. In: *Student Submission for the 7th ESI OpenFOAM Conference 2019, Berlin - Germany*. 2019.

[GR09] Christophe Geuzaine and Jean-François Remacle. “Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities”. In: *International Journal for Numerical Methods in Engineering* 79.11 (2009), pp. 1309–1331. DOI: `10.1002/nme.2579`.

[GW00] Andreas Griewank and Andrea Walther. “Algorithm 799: Revolve: An Implementation of Checkpointing for the Reverse or Adjoint Mode of Computational Differentiation”. In: *ACM Trans. Math. Softw.* 26.1 (Mar. 2000), pp. 19–45. ISSN: 0098-3500. DOI: `10.1145/347837.347846`.

[GW08] Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Society for Industrial and Applied Mathematics, 2008. ISBN: 978-0-89871-659-7. DOI: `10.1137/1.9780898717761`.

[Hei21] Mark Heimgartner. “Implementation and verification of turbulent species transport with composition dependent fluid properties in SU2”. MA thesis. University of Twente, 2021.

[Hue+15] Jan Hueckelheim et al. “Time-averaged steady vs. unsteady adjoint: a comparison for cases with mild unsteadiness”. In: *53rd AIAA Aerospace Sciences Meeting*. 2015. DOI: 10.2514/6.2015-1953.

[Jam03] Antony Jameson. *Aerodynamic Shape Optimization Using the Adjoint Method.* Lectures at the Von Karman Institute, Brussels. 2003.

[JST81] Antony Jameson, Wolfgang Schmdit, and Eli Turkel. “Numerical solution of the Euler equations by finite volume methods using Runge Kutta time stepping schemes”. In: *14th Fluid and Plasma Dynamics Conference*. 1981. DOI: 10.2514/6.1981-1259.

[KA17] H. L. Kline and J. J. Alonso. “Adjoint of Generalized Outflow-Based Functionals Applied to Hypersonic Inlet Design”. In: *AIAA Journal* 55.11 (2017), pp. 3903–3915. DOI: 10.2514/1.J055863.

[Kha04] Waqar Khan. “Modeling of Fluid Flow and Heat Transfer for Optimization of Pin-Fin Heat Sinks”. http://hdl.handle.net/10012/947. PhD thesis. University of Waterloo, Mechanical Engineering, Jan. 2004.

[KSK97] George Karypis, Kirk Schloegel, and Vipin Kumar. *PARMETIS: Parallel Graph Partitioning and Sparse Matrix Ordering Library*. Tech. rep. https://hdl.handle.net/11299/215345. University of Minnesota Digital Conservancy, 1997.

[Luc16] Cheung Yau Lucia. “Conjugate Heat Transfer with the Multiphysics Coupling Library preCICE”. MA thesis. Technische Universität Muenchen, 2016.

[Men94] Florian R. Menter. “Two-equation eddy-viscosity turbulence models for engineering applications”. In: *AIAA Journal* 32.8 (1994), pp. 1598–1605. DOI: 10.2514/3.12149.

[MKL03] Florian R. Menter, Martin Kuntz, and Robin Langtry. “Ten years of industrial experience with the SST turbulence model”. In: *Turbulence, heat and mass transfer* 4.1 (2003), pp. 625–632.

[MN21] Joaquim Martins and Andrew Ning. *Engineering Design Optimization*. Oct. 2021. ISBN: 9781108833417. DOI: 10.1017/9781108980647.

[Nim+18] Sravya Nimmagadda et al. “Low-cost unsteady discrete adjoints for aeroacoustic optimization using temporal and spatial coarsening techniques”. In: *2018 AIAA / ASCE / AHS / ASC Structures, Structural Dynamics, and Materials Conference*. 2018. DOI: 10.2514/6.2018-1911.

[OI01] Paulo J. Oliveira and Raad I. Issa. "An Improved PISO Algorithm for the computation of Buoyancy-Driven Flows". In: *Numerical Heat Transfer, Part B: Fundamentals* 40 (2001), pp. 473–493. DOI: 10.1080/104077901753306601.

[Pal+13] Francisco Palacios et al. "Stanford University Unstructured (SU2): An open-source integrated computational environment for multi-physics simulation and design". In: *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. 2013. DOI: 10.2514/6.2013-287.

[Pal+14] Francisco Palacios et al. "Stanford University Unstructured (SU2): Open-source analysis and design technology for turbulent flows". In: *52nd Aerospace Sciences Meeting*. Jan. 2014. DOI: 10.2514/6.2014-0243.

[Pat+18] Biswaranjan Pati et al. "Numerical investigation of pin-fin thermal performance for staggered and inline arrays at low Reynolds number". In: *International Journal of Heat and Technology* 36 (June 2018), pp. 697–703. DOI: 10.18280/ijht.360235.

[Ped21] Gomes Pedro Carrusca. "A High-Performance Open-Source Framework for Multiphysics Simulation and Adjoint-based Shape and Topology Optimization". PhD thesis. Imperial College London, Department of Aeronautics, June 2021. DOI: 10.25560/95887.

[PLS77] S. V. Patankar, C. H. Liu, and E. M. Sparrow. "Fully developed flow and heat transfer in ducts having streamwise-periodic variations of cross-sectional area". In: *Journal of Heat Transfer* 99.2 (May 1977), pp. 180–186. ISSN: 0022-1481. DOI: 10.1115/1.3450666.

[Rub+18] Antonio Rubino et al. "Adjoint-based fluid dynamic design optimization in quasi-periodic unsteady flow problems using a harmonic balance method". In: *Journal of Computational Physics* 372 (2018), pp. 220–235. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2018.06.023.

[SA92] P. Spalart and S. Allmaras. "A one-equation turbulence model for aerodynamic flows". In: *30th Aerospace Sciences Meeting and Exhibit*. 1992. DOI: 10.2514/6.1992-439.

[SAG19] Max Sagebaum, Tim Albring, and Nicolas R. Gauger. "High-Performance Derivative Computations Using CoDiPack". In: *ACM Trans. Math. Softw.* 45.4 (Dec. 2019). ISSN: 0098-3500. DOI: 10.1145/3356900.

[San17] Ruben Sanchez Fernandez. “A coupled adjoint method for optimal design in fluid-structure interaction problems with large displacements”. PhD thesis. Imperial College London, Aeronautics, Sept. 2017. DOI: `10.25560/58882`.

[Sch+] Steffen Schotthöfer et al. “Windowing Regularization Techniques for Unsteady Aerodynamic Shape Optimization”. In: *AIAA Aviation 2020 Forum*. DOI: `10.2514/6.2020-3130`.

[Sch20] Steffen Schotthöfer. “Regularization Techniques for Unsteady Aerodynamic Shape Optimization”. MA thesis. TU Kaiserslautern, 2020.

[SSP18] Matthias Schramm, Bernhard Stoevesandt, and Joachim Peinke. “Optimization of Airfoils Using the Adjoint Approach and the Influence of Adjoint Turbulent Viscosity”. In: *Computation* 6.1 (2018). ISSN: 2079-3197. DOI: `10.3390/computation6010005`.

[STN14] A. Sen, Markus Towara, and Uwe Naumann. “A discrete adjoint version of an unsteady incompressible solver for OpenFOAM using algorithmic differentiation”. In: *11th World Congress on Computational Mechanics (WCCM XI)*. July 2014, pp. 5014–5023.

[Tow18] Markus Towara. “Discrete Adjoint Optimization with OpenFOAM”. PhD thesis. RWTH Aachen University, Dec. 2018. DOI: `10.18154/RWTH-2019-00475`.

[van79] Bram van Leer. “Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov’s method”. In: *Journal of Computational Physics* 32.1 (1979), pp. 101–136. ISSN: 0021-9991. DOI: `10.1016/0021-9991(79)90145-1`.

[Ven+19] Charanya Venkatesan-Crome et al. “Discrete Adjoint for Unsteady Incompressible Flows Using a Density-based Formulation”. In: *AIAA Aviation 2019 Forum*. 2019. DOI: `10.2514/6.2019-3669`.

[Ven20] Charanya Venkatesan-Crome. “A coupled discrete adjoint method for optimal design with dynamic non-linear fluid structure interactions”. PhD thesis. Imperial College London, Aeronautics, Dec. 2020. DOI: `10.25560/91983`.

[Vid20] Mahening Citra Vidya. “Modeling and Adjoint Optimization of Heat Exchanger Geometries”. Supervisor: T.H. van der Meer. PhD thesis. University of Twente, Department of Thermal and Fluid Engineering, Jan. 2020. DOI: `10.3990/1.9789036549004`.

[VP20] Charanya Venkatesan-Crome and Rafael Palacios. "A Discrete Adjoint Solver for Time-Domain Fluid-Structure Interaction Problems with Large Deformations". In: *AIAA Scitech 2020 Forum*. 2020. DOI: 10.2514/6.2020-0406.

[VS16] Tom Verstraete and Sebastian Scholl. "Stability analysis of partitioned methods for predicting conjugate heat transfer". In: *International Journal of Heat and Mass Transfer* 101 (2016), pp. 852–869. ISSN: 0017-9310. DOI: 10.1016/j.ijheatmasstransfer.2016.05.041.

[Wan09] Qiqi Wang. "Uncertainty quantification for unsteady fluid flow using adjoint-based approaches". Supervisor: Parviz Moin, Gianluca Iaccarino, Anthony Jameson, Amin Saberi. PhD thesis. Stanford University, Institute for Computational and Mathematical Engineering, Jan. 2009.

[Wel+98] H. G. Weller et al. "A tensorial approach to computational continuum mechanics using object-oriented techniques". In: *Computers in Physics* 12.6 (1998), pp. 620–631. DOI: 10.1063/1.168744.

[WHB14] Qiqi Wang, Rui Hu, and Patrick J. Blonigan. "Least Squares Shadowing sensitivity analysis of chaotic limit cycle oscillations". In: *Journal of Computational Physics* 267 (2014), pp. 210–224. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2014.03.002.

[Whi17] Frank M. White. *Fluid mechanics*. 8th Edition. McGraw-Hill Education, 2017. ISBN: 9789385965494.

[WMI09] Qiqi Wang, Parviz Moin, and Gianluca Iaccarino. "Minimal Repetition Dynamic Checkpointing Algorithm for Unsteady Adjoint Calculation". In: *SIAM J. Scientific Computing* 31 (2009), pp. 2549–2567. DOI: 10.1137/080727890.

[WMS97] Jonathan Weiss, Joe Maruszewski, and Wayne Smith. "Implicit solution of the Navier-Stokes equations on unstructured meshes". In: *13th Computational Fluid Dynamics Conference*. 1997. DOI: 10.2514/6.1997-2103.

[WS95] Jonathan M. Weiss and Wayne A. Smith. "Preconditioning applied to variable and constant density flows". In: *AIAA Journal* 33.11 (1995), pp. 2050–2057. DOI: 10.2514/3.12946.

[Zho+16] Beckett Y. Zhou et al. "An Efficient Unsteady Aerodynamic and Aeroacoustic Design Framework Using Discrete Adjoint". In: *17th AIAA / ISSMO Multidisciplinary Analysis and Optimization Conference*. 2016. DOI: 10.2514/6.2016-3369.

[Zho+17] Beckett Y. Zhou et al. “Reduction of Airframe Noise Components Using a Discrete Adjoint Approach”. In: *18th AIAA / ISSMO Multidisciplinary Analysis and Optimization Conference*. 2017. DOI: `10.2514/6.2017-3658`.

Curriculum Vitae

Personal Data:

Name: Tobias Kattmann

Education:

1998 - 2002: **Elementary School**
Lindenschule Buer, Melle, Germany

2002 - 2004: **Orientation Level**
Lindenschule Buer, Melle, Germany

2004 - Mai 2011: **Secondary School**
Gymnasium Melle, Melle, Germany

Sep 2011 - Mai 2015: **B.Sc. Simulation Technology**
University of Stuttgart, Stuttgart, Germany

Jun 2015 - Jun 2015: **M.Sc. Simulation Technology**
University of Stuttgart, Stuttgart, Germany

Aug 2017 - Sep 2024: **PhD studies Scientific Computing**
Technical University of Kaiserslautern, Germany

Work Experience:

Aug 2017 - Jun 2021: **PhD Candidate**
Robert Bosch GmbH, Renningen, Germany

Jul 2021 - Dec 2021: **Simulation Engineer**
Bosch Thermotechniek B.V., Deventer, The Netherlands

Nov 2022 - Feb 2024: **Computational Scientist**
Luminary Cloud, Inc, Redwood City, United States

Aug 2024 - today: **System Developer Algorithm**
Carl Zeiss SMT GmbH, Jena, Germany