Aerothermal analysis automation for preliminary axial compressor airfoil geometry generation

Tomas Diaz Jimenez

Department of Mechanical Engineering

McGill University, Montreal

November 2020

Thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Master of Engineering

Acknowledgements

I would like to thank my supervisor Professor Michael Kokkolaras for his continuous support and mentorship before and throughout this degree.

I am also thankful to my ETS, McGill, Siemens team, in particular Martin Staniszewski for providing a welcoming, supportive, and engaging workplace. We could not have asked for a better or more caring manager than Martin.

Finally, a special thanks to my research colleague Ahmed Bayoumy and our Siemens technical expert Nigel Gwilliam for their willingness and eagerness to share their valuable technical knowledge and expertise with me. Nigel's guidance was a strong enabler for the work that has been done in this thesis.

Abstract

Automated analysis and design are increasingly becoming a necessity in the domain of aeroderivative gas turbines. This work focuses on automation aimed at axial compressor aerodynamic design. An automation framework is developed for rapid generation of preliminary axial compressor airfoil geometry from throughflow aerodynamics results specific to Siemens Energy analysis tools and practices. The framework handles real and general designs with as many blade-rows as necessary and enables a simplified link to numerical optimization tools. The automation covers a common aero-thermal workflow which incorporates auto-blading, stacking, blade to blade analysis, and 3D CFD evaluation for a general multi-row component. It allows for an analyst to create new rotor or stator geometries from scratch by supplying a solved throughflow model or to retrofit and optimize existing geometries for new designs. The framework makes use of a compartmentalized object-oriented structure to maintain its general applicability to new models within the compatible suites. Three semi-automated stand-alone executable processes are created allowing for specific user input at certain stages of the workflow through custom developed, fully by-passable, graphical user interfaces to ensure ease of use. Integration of the codes into two possible existing multidisciplinary automation platforms is discussed and assessed. In addition, the automation and its suitability for use in design exploration and optimization for multi-row airfoil geometry creation is tested through four case studies where blades are re-designed for different components. Case study results demonstrate significant time savings when compared to current manual execution of the workflow in addition to tangible design improvements enabled by the use of existing, but under-utilized, numerical optimization tools.

Table of Contents

List of Tables

List of Figures

Nomenclature

Acronyms

AGT	Aeroderivative gas turbine
API	Application programming interface
CDA	Controlled diffusion airfoil
CFD	Computational fluid dynamics
DCA	Double circular arc
GUI	Graphical user interface
JSON	JavaScript object notation
MDO	Multidisciplinary design optimization
PS	Pressure side
RANS	Reynolds averaged Navier-Stokes
SS	Suction side
XDSM	Extended design structure matrix
XML	Extensible markup language

Symbols

BIA	Blade inlet metal angle
BOA	Blade exit metal angle
$SKEW$	Blade section re-stagger angle
$DELT$	Blade section lean
$XCEN$	Blade section sweep
δ	deviation angle
η	isentropic efficiency
T_0	Total temperature
p_0	Total pressure
m	Mass flow rate
N	Rotational speed

1 Introduction

1.1 Context

Aeroderivative gas turbines (AGTs) are land gas turbines that share design practices and materials with aircraft engines. Their competitive advantage in the power generation market comes from their ability to rapidly start-up and run at various speeds to satisfy variable power demands in addition to smaller and sometimes portable form factors suitable for remote job sites such as oil rigs. AGTs have become a competitive option to standard large industrial gas turbines, because they can work more effectively in conjunction with green power generation methods such as solar or wind as a buffer to make up for energy deficits in times of high energy demand. At nighttime, or in times of low wind where solar and wind energy may not be sufficient to satisfy energy demands by the public, AGTs are able to rapidly provide an alternative and reliable source of energy. Their light construction style allows for rapid start-up and shutdown cycles necessary for this type of use, which large gas turbines are not able to effectively accommodate.

The gas turbine market is a rapidly evolving one and is pushing towards increasingly shorter development times with continuous requirements for rapid and cost-effective design improvements. To satisfy rapid turnover times in the generation of new, high quality designs, engineering workflow automation is increasingly becoming a necessity in every facet of the development process [1]. AGTs are highly complex engineering systems involving many interacting disciplines in multi-level iterative processes. As such, they are prime candidates for multidisciplinary design optimization (MDO). Proper use of MDO techniques, particularly at the preliminary design stages, has been shown to lead to not only better performing preliminary designs, but also improved products at the detailed design level by allowing for more exhaustive

exploration of the early design space and encouraging more rigorous computationally driven design variable choices [2].

This project forms a component of a wider scope project at the AGT division of Siemens Energy Canada aiming to integrate engineering workflows into a multidisciplinary platform with the intent of ensuring effective and streamlined interaction of design and computational analysis processes in a collaborative context. The end goal is to leverage this platform and the developed automated workflows for data driven product development and MDO integration. The focus of this component of the project is the automation of throughflow driven preliminary compressor airfoil geometry generation, analysis, and optimization.

Axial compressors in aeroderivative gas turbines are highly complex and critical components which have a strong effect on overall performance of the system. Early focus on quick creation of highly efficient multi-row and multi-stage compressor designs is required for maintaining competitiveness in the AGT market. Compressor development is a very incremental process where efficiency improvements of as little as half of a percent may be worth pursuing. A main driver for performance of compressor designs is airfoil geometry, making automation of airfoil geometry generation for preliminary design studies a candidate with a strong value proposition for implementation.

1.2 Objectives and motivation

The objective of this work is to develop and test an automation framework that enables an aerodynamics analyst to rapidly generate preliminary airfoil geometry starting from an existing design throughflow model. This main objective is achieved by making use of existing mature and powerful analysis and design tools currently in use by Siemens Energy engineers. Tailor made

object-oriented code should be developed to supplement the capabilities of existing tools to facilitate their use in the preliminary airfoil geometry generation workflow, and to allow for integration of this workflow into a chosen multidisciplinary automation platform. To ensure development of a useable and useful industrial strength tool, the following requirements are imposed on the programs:

- The automation code must be able to handle a general single or multistage axial compressor.
- The code should be capable of generating airfoils either from scratch or to inherit existing designs.
- The code should reduce the number of non value-added tasks that the analyst needs to complete (i.e., reduce repetitive tasks, book-keeping, data management, and transfer).
- The code must include the possibility for user input where necessary, but also be capable of running in a fully automated manner (batch) if desired.
- The code must be able to run as standalone or as an easily integrated tool on a multidisciplinary automation platform of the engineer's choice.

1.3 Outline of thesis

The thesis has been divided into six chapters. The second chapter is a brief background literature review, where principles of compressor aerodynamic design are discussed in the context of this work along with existing methodologies. The third chapter covers the specific analysis and design workflow that is being automated. This includes a detailed step by step breakdown of the activities required. The fourth chapter focuses on the automation of the workflow, where involved software, architecture and processing approach are discussed including integration into multidisciplinary platforms and general capabilities. In the fifth chapter, results of case studies, where the framework

is fully and also partially used to generate and optimize real preliminary designs, are discussed commenting on general metrics such as relative aerodynamic performance improvements and time savings. Finally, a conclusion with closing remarks and suggestions for future work is presented in chapter six.

2 Literature review

2.1 Existing developments

Aerodynamic design and design optimization for multi-row axial compressors is a very commonly investigated topic. Frameworks incorporating partially similar workflows to the one applied here, where the throughflow is included and drives the initial geometry generation have been presented in sources [3-5]. Analysis automation and integration in the context of aerospace engineering are covered in sources [1, 6-8]. The most recent survey of MDO architectures is presented by Martins and Lambe [9].

Automation strategies intended for compressor airfoil geometry optimization tend to either skip the throughflow in the design workflow or avoid the 3D CFD, focusing on 2D analysis and design techniques when the throughflow is present [4, 5, 10, 11]. For example, Schnoes et al. apply a throughflow driven automated optimization method for multi-stage compressors that encompasses a similar workflow than the one automated here; however, they use 3D CFD only for final design performance evaluation and not to drive the 3D geometry optimization [4]. In this case optimization is only driven by the blade-to-blade analysis which occurs as well in the work of Büche and that of Sieverding et al. [10, 11]. Additionally, Oyama uses only a streamline curvature throughflow method for the airfoil optimization restricting the design variables to blade rigidity distributions avoiding blade-to-blade and 3D CFD analyses altogether [5]. Conversely, Siller et al. only make use of the full 3D geometry and 3D CFD with no consideration of throughflow and blade to bade aerodynamics for their implemented automation and design optimization methods [12].

Aerodynamics process automation is common in the literature as well; however, little emphasis is devoted to industrial strength automation in particular for the workflow discussed in this thesis and its integration. Studies have a tendency to focus on methodology for defining the process at a high level [13], defining more powerful parametrizations [14], or defining novel multi-discipline or multi-objective optimization problem formulations [12, 15]. In these studies, automation coding strategies are either glossed over, given implicitly, or implemented on a case by case basis, such that their ability to handle generalized design problems is not discussed and/or unlikely. For example, Siller et al. provide a detailed account of their employed automated process and multidisciplinary integration however only applied to a single one and a half stage model, making no mention of framework generality to other components [12].

The need for engineering analysis automation, integration, and general approaches are also topics available in the literature and can be found in [1, 16, 17]. Smith and Bardell outline a non workflow specific object oriented approach to automating engineering development tasks in a responsive and cost effective fashion that leverages existing methods and data with the intent to develop highly customised and continuously maintained automated processes [1]. Ebert stresses the importance of effective integration of automated processes for use in a complex collaborative engineering setting, more specifically by making use of PLM/ALM systems [16].

Industrial strength frameworks specifically applied to preliminary turbomachinery automated design and optimization are covered in sources [2, 8, 18]. The work of Panchenko et al. covers preliminary design optimization for general turbomachinery components outlining an integration framework which defines key requirements for automating the analysis and design processes involved. They highlight the need for data transfer, automatic execution, an intuitive user interface, and execution that is independent of the optimizer, among others [2]. Lagloire et al. specify a single

platform design and analysis automation architecture, applied to turbine rotors, where the coding approach is given in detail, including programming language, system and API call management, and user interface. They focus on code reusability to accommodate multiple parametric models, analysis code execution control through scripting, and data management [8]. The link of this analysis and design automation architecture to numerical optimization tools is then covered by the work of Ouellet et al., highlighting it usefulness and industrial usability [18].

2.2 Compressor design working principles

The AGT is a real implementation of the Brayton power cycle developed in the late 1800s consisting of four main thermodynamic processes: isentropic compression, constant pressure heat addition, isentropic expansion, and constant pressure heat rejection. In real application the cycle is open, continuously ingesting new air and expelling it once it has passed through the turbomachine. In the type of gas turbines being studied here, the first step in the cycle is achieved by an axial flow compressor, the second by a combustor, and the third by an axial flow turbine. In real systems the flow never possesses an ideal behaviour and aerothermodynamic losses become present throughout the machine. The design efforts are driven by an attempt to reduce these losses as much as possible while maintaining the desired power generation capabilities, range of operation, cost, mechanical integrity, and complexity [19].

The axial flow compressor is composed of multiple rotating and stationary airfoil blade pairs commonly referred to as rotors and stators. The purpose of the compressor is to ingest air and efficiently increase its static pressure. This is achieved through a repeated two step process shown in Figure 1 in which the rotating blade imparts kinetic energy to the flow. The flow is then turned and decelerated by travelling around the stators, after which it approximately recovers its initial velocity, but at higher static and total pressures.

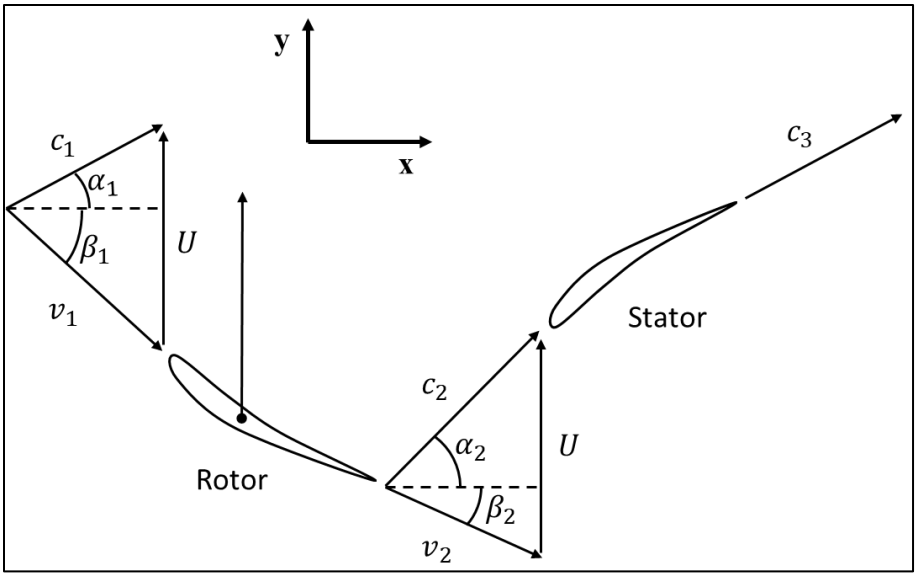

Figure 1: Rotor-stator pair velocity triangles

The aerodynamic design process incorporates multiple analysis methods at different levels of fidelity and dimensionality. The design process usually starts with a preliminary 1D mean line design intended to get a rough approximation of the annulus shape and velocity triangles at mid-height along with estimates of performance measures such as pressure, temperature ratio, etc. At the 2D level the design can be evaluated using a discretized axisymmetric model referred to as a throughflow, or a model to evaluate flow around airfoil sections called a blade to blade model [20, 21]. At the 3D level steady and unsteady RANS CFD solvers applying suitably chosen turbulence models are used to calculate the flow field in the simplified and real compressor geometry. The preliminary design phase usually includes analyses up to 3D steady CFD with simplified geometry in the duct which does not include seal clearances and leakage flow passages [22].

A quasi 3D approach to generating full rotor or stator geometries referred to as stacking is commonly used. Stacking consists of taking a set of 2D sections at different meridional heights for a blade design and lofting them together, and possibly smoothing the surfaces, to obtain a 3D geometry. Common parametrizations for 3D blading make use of deforming sections, re-stacking

and re-lofting for blade deformation [14]. Compressor blade geometries are tailor made to the desired component.

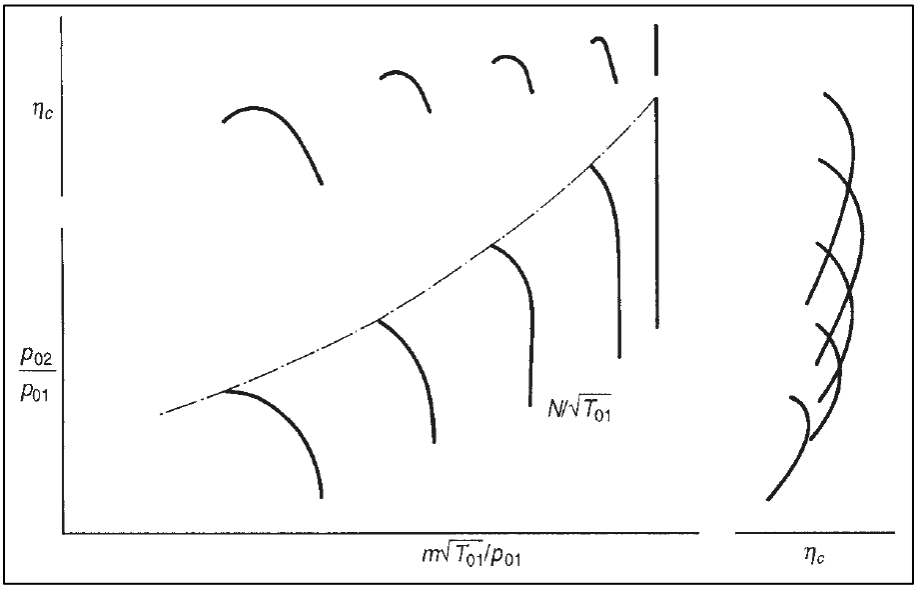

Figure 2: Axial compressor characteristics [20]

General thermodynamic performance measures for a compressor include total temperature ratio, total pressure ratio, efficiency, and mass-flow rate. These quantities are usually presented in grouped-parameter fashion where performance curves, referred to as characteristics, at any operating condition can be displayed (see Figure 2). The design point is the operating condition (flowrate, rotational speed, ambient conditions) for which the component is optimized; however, the compressor needs to be capable of operating at a very big range of speeds, ambient conditions, and power inputs in order to be useable. Off-design performance is an important attribute, and also influences the way the components are designed. In general, an airfoil geometry is designed so that it can maintain good performance over a large range operating conditions [21].

3 Analysis and design workflow

3.1 Workflow overview

The workflow incorporates three analysis processes, and three geometry generation steps. The analyses are throughflow, blade to blade (B2B) analysis, and 3D steady CFD. The geometry generation steps are 2D airfoil profiling, 2D airfoil stacking and 3D geometry lofting and deformation. Three iterative procedures are present in the workflow two of which are accounted for and one which is considered out of scope for this work. The first iterative procedure is the 2D airfoil profiling process which is computationally evaluated by the blade-to-blade analysis. If this loop is unsuccessful (i.e., no chosen/adjusted 2D profile provides the desired performance), the design throughflow analysis needs to be revisited; however, modification of the throughflow model is out of the scope of this project and thus not considered in the automation. The objective is to start from an existing throughflow model. Finally, the third iterative procedure appears at the 3D level where the 3D geometry is deformed and evaluated with steady CFD until the design meets the requirements (i.e., efficiency, capacity, pressure ratio, reaction, loading, etc.).

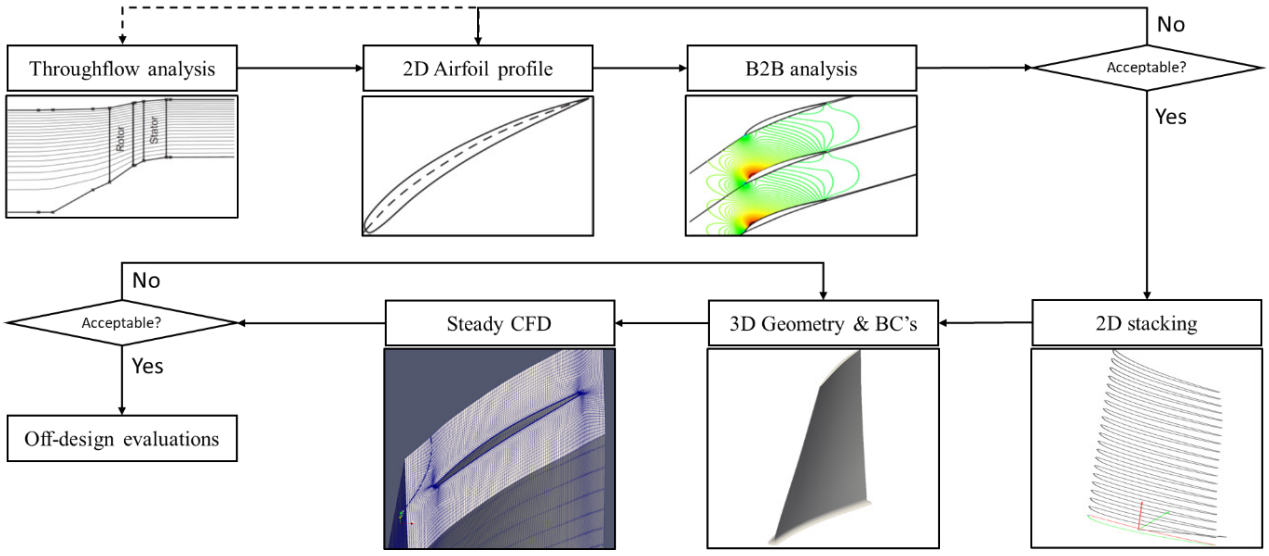

Figure 3: Design and analysis workflow overview

3.2 Design throughflow

The design throughflow, also referred to as the S2 analysis, is the driver for the entire design activity. The throughflow discretizes an axisymmetric section of the component and calculates the flow at each point along the mesh by solving the steady, adiabatic radial equilibrium equation iteratively [23]. If executed in design mode, the throughflow takes in an initial guess based on correlations or previous experience of the loss, and the target flow turning (input work) that each blade is intended to accomplish. The result provides an approximate distribution of camber-lines along the span of each blade but does not give an indication of the outer profile of the airfoil necessary to achieve the desired performance. Initial guesses of the losses, and enwall effects are indicated as pressure loss coefficients and blockage factors, respectively, at prescribed meridional and radial positions along each blade [21]. The losses can be later confirmed and re-matched if necessary, following the steady CFD calculation.

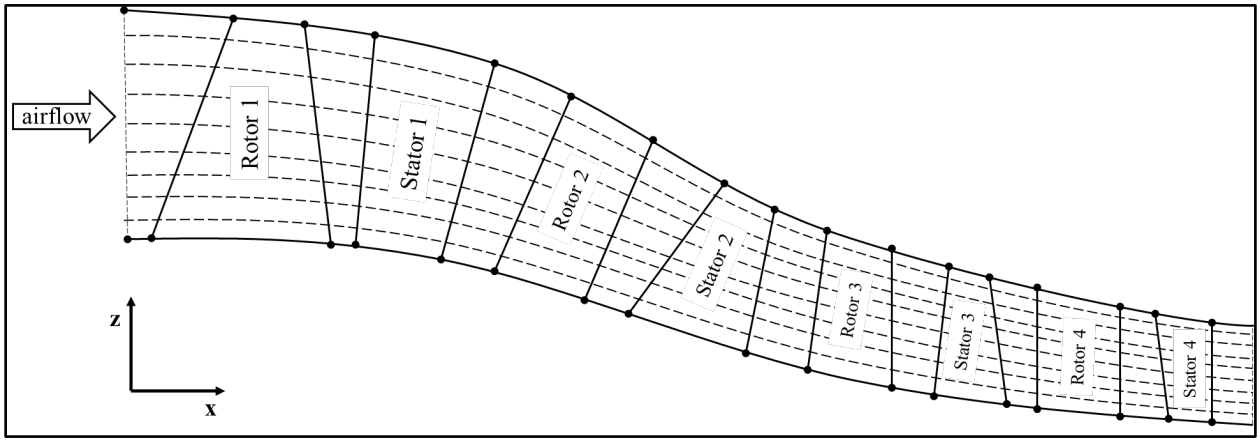

Figure 4: Compressor axisymmetric view

If matched properly, the throughflow analysis can be a high fidelity and computationally inexpensive analysis method for the full multi-stage compressor [23].

Generation of the throughflow model involves extensive interaction with the thermodynamic performance department to agree on parameters such as variable stator vane scheduling, target mass flow rate, target capacity, speed, etc. [19] This interaction is not considered; however, as will be discussed in a later chapter, performance modelling is also integrated into the multidisciplinary platform considered here which may allow to capture this interaction.

3.3 2D airfoil profiling

Having established the radial camber-line distributions of the airfoils along the meridional direction of the flow, a standard airfoil profile is fitted to the camber-line with standard radial thickness distributions and leading-trailing edge radii to generate profiles at multiple radial heights for each airfoil that can then be stacked and lofted together to generate a full 3D geometry. A method similar to the one proposed by Schnoes and Nicke is used to match the throughflow results to previously evaluated and optimized profiles available in a proprietary database [4].

Alternatively, the analyst has the option to select an existing airfoil geometry of their choice from a previous design that they can retrofit into the newly calculated throughflow aerodynamics. This type of 2D profiling is referred to as cloning.

3.4 Blade to blade analysis

Throughflow analysis provides an approximation of the flow velocity field along axial and radial positions in the turbomachine but assumes that the flow is axisymmetric. It provides no indication of how the flow behaves across the blade passages. Blade to blade analysis, also referred to as the S1 analysis, acts as the compliment to throughflow by eliminating the axisymmetric assumption but neglecting flow along the radial direction on the blades. Blade to blade analysis provides an approximation of the discretized flow-field across 2D spanwise cross sections of the airfoils in

order to estimate blade section properties such as lift, Mach number distribution, pressure loss, separation, incidence, diffusion, etc. [21]. The analysis is usually solved by the multiple blade interacting stream-tube Euler solver (MISES) which is a widely used turbomachinery analysis tool [24].

The blade to blade analysis is conducted for multiple 2D sections of each airfoil to determine whether the profile is capable of delivering the desired flow turning and ensure that the losses are acceptable. This can be done with as little as three sections along each blade span or as many as 21 sections. The analysis results can be presented in a condensed report which provides performance measures that can be rapidly used by the analyst to determine the suitability of the profile to the desired application without the need to go to higher fidelity, more computationally intensive methods.

3.5 2D airfoil exit air angle matching

Having cloned or fitted a profile to the new throughflow aerodynamics, there is no guarantee that the exit air angles along the span of each airfoil will yield the same quantities that were originally prescribed by the throughflow solution. Individual 2D sections are modified by changing their metal inlet angles (BIA) and exit angles (BOA) and reshaping the profile accordingly until the air angles are matched to a desired tolerance while maintaining a low amount of losses. A physical depiction of the metal inlet and exit angles and their possible effect on the geometry is shown in Figure 5.

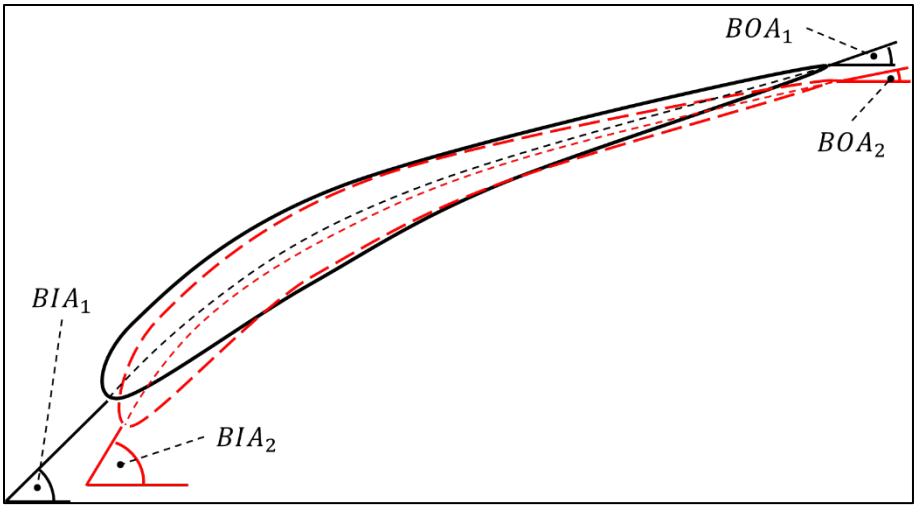

Figure 5: Re-profiling for incidence and deviation matching

The step can be considered a sub-optimization. The constrained, two variable design optimization problem is solved using a simple line search and usually involves less than 50 blade to blade evaluations per section. To reduce the losses and bring the airfoil geometry closer to the minimum of the airfoil loss polar, the problem is formulated such that flow deviation falls within a specified tolerance while also requiring that the flow Mach number spikes on the pressure and suction sides of the airfoil at the profile leading edge are of equal height. This incidence and deviation adjustment procedure is a heuristic method, and is mathematically formulated as follows:

$$\begin{aligned} &Find\ \boldsymbol{x}_* = \begin{bmatrix} BIA_* \\ BOA_* \end{bmatrix}\ such\ that: \\ &\left| M^{SS}_{peak}(\boldsymbol{x}_*, \boldsymbol{P}) - M^{PS}_{peak}(\boldsymbol{x}_*, \boldsymbol{P}) \right| \leq Tol_{peak} \\ &\left| \delta(\boldsymbol{x}_*, \boldsymbol{P}) \right| \leq Tol_{deviation} \end{aligned} \tag{1}$$

This procedure does not aim to find an optimal point (i.e., a minimum loss point), but rather, a feasible point. Once the two constraints are satisfied, the process is complete. A rough schematic of a Mach number distribution around an airfoil section highlighting how the spike difference between pressure and suction sides is measured is shown in Figure 6.

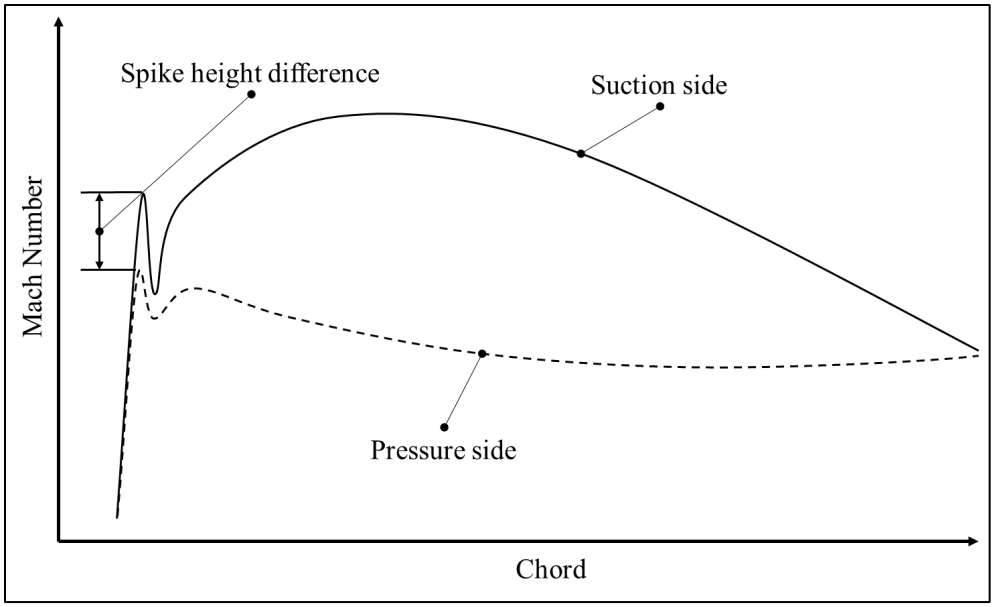

Figure 6: Mach number spike schematic

As previously mentioned, the 2D airfoil exit air angle matching needs to be repeated for multiple 2D sections, after which the 2D sections can be stacked, re-lofted, and smoothed to form the 3D airfoil. This iterative procedure involves multiple design loops involving two steps: 2D profiling to modify the metal angles and blade to blade analysis to extract the air angle deviations and Mach number spike heights.

3.6 3D steady CFD analysis

Three-dimensional features, such as fillets, blade tip clearances, and pennies are added to the geometry at this stage in the workflow before the computational flow analysis. The 3D CFD gives insight into 3D aspects of the flow that cannot be captured by the previous analyses. Secondary losses such as end wall effects, tip leakages which could only be guessed or estimated before are finally calculated at this stage [21].

The throughflow solution is used to prescribe a 1D radial distribution of exit pressures as the exit boundary condition for the 3D CFD analysis. The analysis uses a discretized representation, or

mesh, of the 3D geometry of the duct and airfoils. To reduce computational cost of the calculation, for each blade-row, the geometry is presented as a division of the annulus with a single airfoil and periodicity surfaces are specified to approximate the flow on the full duct. The interface surfaces between the rotating components and stationary components in the mesh are so called mixing planes and their axial location needs to be specified by the user. The flow is calculated by solving the steady Reynolds-averaged Navier–Stokes (RANS) equations iteratively on the discretized space [22]. A possible example of a CFD mesh for a single compressor stage indicating the location of a mixing plane is shown in Figure 7. The steady CFD analysis that is automated and conducted involves air modelled as an ideal gas in adiabatic, and viscous subsonic flow. The chosen turbulence model is the Spalart-Allmaras model employing wall functions.

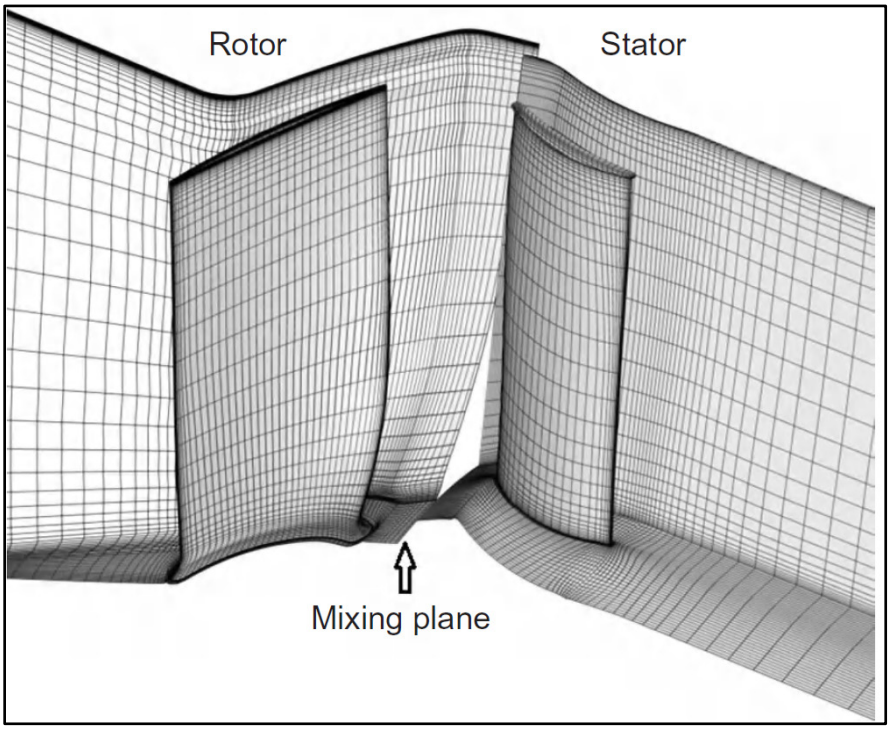

Figure 7: 3D mesh, and mixing planes [21]

3.7 Off design analysis

Having solved for the steady flow at the design point. The throughflow model can be modified iteratively to reflect performance at off design operating points. This involves an iterative process where the thermodynamic performance department provides operating conditions, variable stator vane (VSV) scheduling, and targets for pressure ratio and total exit temperature from carefully developed correlation models [19]. The new operating conditions are applied to the throughflow model and target values are matched to the throughflow solution by scaling the loss coefficients and shifting the amount of flow turning in the throughflow model until the pressure ratio and efficiency in the component throughflow reach the performance targets [23]. The off-design analysis outputs both a new throughflow model for the off-design point, and the new throughflow model's results which can be used to define new boundary conditions for new off-design blade to blade or 3D CFD calculations on pre-existing or newly created geometries.

4 Automation

4.1 Analysis software and scripting language

At the software level, the workflow can be split into three groups:

- Throughflow suite
- Stacking suite
- CFD suite

Each of the three groups is composed of sets of independent proprietary codes, with specific purposes and capabilities.

4.1.1 Throughflow suite

The throughflow suite encompasses one multi-tool computational platform which manages the throughflow solver itself, and the off design throughflow model generation tool. The throughflow suite is capable of incorporating and managing externally developed executable programs as a way to extend its built-in capabilities. It manages the creation and execution of the throughflow model, as well as extraction and post-processing of throughflow results.

The throughflow suite contains a wide range of built in and externally added capabilities for managing turbine and compressor throughflow models in a multidisciplinary context for example, for mediation to secondary air systems, or thermodynamic performance; however, its use in this work is reserved to existing compressor design throughflow evaluation and off design throughflow generation. Capabilities of the throughflow suite in a multidisciplinary context will be discussed more in detail in chapter 4.3 where code integration is touched upon.

The throughflow solver is a single batch-executable program that interacts with the main throughflow suite platform but is also capable of running externally as a standalone tool. The throughflow model is defined by two inputs: the station and annulus geometry, and the station aerodynamics. Both these inputs exist in independent XML tagged files and are the only necessary inputs for complete execution of the solver.

Similar to the throughflow solver, the off design throughflow generation tool also consists of a single standalone executable program. This tool was previously integrated into the throughflow suite as a plug-in and partially interacts with the platform itself.

4.1.2 Stacking suite

The stacking suite is a single software with built in solvers for pre/post processing and running blade to blade analysis, creating and parametrizing airfoil profiles and stacking sections. It possesses extensive capabilities for airfoil geometry generation, section stacking, deformation, profiling, etc. for geometry modelling processes from preliminary to detailed design phases including a built-in interface to CAD modelling software and optimization sub-routines. For the purposes of this work, the capabilities that are utilized are: auto-blading, cloning, blade to blade analysis, 2D airfoil exit air angle matching, and boundary condition and 3D geometry generation.

The auto-blading facilities of the stacking suite contain 3 standard databases for creating 2D cross sections to fit the throughflow aerodynamics: double circular arc (DCA) profiles, controlled diffusion airfoil (CDA) profiles, and a custom database of historically suitable profiles for specific components.

The cloning capabilities of the stacking suite allow for existing airfoil geometries to be re-fitted into new throughflow aerodynamics to more or less preserve an existing airfoil profile.

The blade to blade analysis tool makes use of a custom implementation of the solver MISES. Interaction to the solver for geometry definition, grid generation and refinement, data pre and post processing, solver execution, and results plotting is all handled by the stacking suite and directly built into its graphical and batch user interface. For the purposes of this work the blade to blade analysis facilities in the stacking suite are used only to determine how well the flow respects the air inlet and exit angles prescribed by the throughflow, and to generate human readable analysis reports for analysts to rapidly and easily evaluate the quality of newly generated designs at an intermediate stage in the workflow.

The stacking software already possesses a built-in implementation of the 2D airfoil exit air angle matching process discussed in chapter 3.5, which will be used directly as part of the automation process.

Boundary conditions and 3D geometry generation facilities in the stacking suite take an existing set of 2D stacked sections and output the 3D geometry of the airfoil and annulus in conjunction with post processed throughflow results as boundary conditions necessary to run 3D CFD on the blade row. The output format for the 3D geometry and boundary conditions is also natively compatible with the CFD suite.

The stacking suite also possesses powerful built in automation capabilities which facilitate its integration into the desired multidisciplinary platform. A majority of the stacking suite's possible commands via user interaction with its graphical user interface (GUI) have equivalent batch script commands that can be written into an input text file and run in a fully automatic manner. The input text file is written in a standard XML tree format further facilitating automation.

4.1.3 CFD suite

The CFD suite is composed of multiple independent codes for pre-processing, solving, and post-processing CFD models. The suite is fully compartmentalized, so that each code is a single, batch executable program. Interaction between the codes is accomplished entirely through natively compatible input and output files with the codes acting as intermediate black boxes. This work focuses on four specific components: mesh generator, single row-preprocessor, multi-row preprocessor, and steady solver.

A significant caveat of the CFD suite is that, although input and output files are natively compatible between its components, they come in the form of custom-formatted text files which require special treatment when processing them in an automated manner. The approach to addressing this problem is discussed in chapter 4.2.6.

4.1.4 Programming language

The prescribed scripting language is Python to ensure and preserve compatibility with other existing multi-disciplinary automation platforms that use it as their main programming language. Python is an interpreter based, fully object oriented, and high-level programming language that is ideal for automation work of this nature since it allows for rapid code prototyping and testing, and high coding productivity even with limited software development experience [25].

4.2 Framework

4.2.1 User interaction and user input

The matter of user interaction with the design and analysis process automation is a very important factor in the way the code is structured and its overall functionality. An ideal automation scenario would involve no need for user interaction at all. A drawback to full design automation is that it

leaves little room for engineering creativity, and can tend to limit modelling generality [17]. An important target for this work is that the developed tools should be able to accommodate a general compressor, which requires a very large amount of flexibility. For example, the throughflow may contain a number of stations corresponding to airfoils, say 10, however the analyst may only want to create new geometry for one or two of these airfoils, and clone the rest, or only generate data for a single blade row, or only the first couple stages. As the complexity of the model increases the number of combinations and permutations of the input parameters for the process become very large, and it becomes nearly impossible to programmatically establish the appropriate settings in advance.

The approach to maintaining the necessary flexibility in a way that involves minimal added effort and reduced non-value-added time from the user is through custom GUIs for certain components of the design workflow. The intent is to provide the user with a simple interface that allows them to rapidly choose options only related to the workflow and translate this choice into a complete input file that the chosen software can read without the need to directly interact with it, which would require familiarity and proficiency with the available commands and user interfaces. This approach avoids the need to populate input files by hand or having to interact directly with batch code.

Keeping in mind that this automation should still allow for full autonomous execution, as is required for design optimization studies, the GUIs are offered as a simple initial mediator. In this way partial automation can be used to initially setup a specific workflow and then the setup can be reused in a fully automated manner if it needs to be used as part of an iterative procedure.

Partial and full automation in the custom tools is achieved by fully compartmentalizing the execution, and data handling from the user interface and user interaction. The user interface will

only serve to generate the required data and inputs for the main execution to function in the form of an input file or programming object depending on how the execution is started. Decoupling the user interface from the execution allows for an easy transition to full automation once the desired settings have been defined by the user.

This approach to user input and interaction was chosen in favour of full automation in order to reduce the number of assumptions and pre-requisites necessary for the code to be able to handle general cases. It ensures that the newly created extensions of the tools do not become too restrictive which can act as a deterrent to their early adoption.

4.2.2 Compartmentalizing the workflow

In addition to user input, another necessary choice is the way the workflow is compartmentalized, or the way the activities are linked to one-another by the automation code. Compartmentalization or decomposition of the workflow is favoured to ensure modularity in an object-oriented programming context. A well-chosen decomposition can make the automation simpler while also enhancing integration into a multidisciplinary process [17].

At the software level the obvious choice would be to split the workflow into three, in this case each component corresponds to a specific analysis and design suite, such that the first component handles the throughflow suite, the second handles the stacking suite, and the third handles the CFD suite. At the "design process" level the workflow would be split into six components corresponding to throughflow, 2D profiling, blade to blade analysis, stacking, 3D geometry and boundary condition definition, and steady 3D CFD. This choice was decided against because it gives unnecessary granularity to the process. Components that go hand-in-hand do not need to be separated. For example, the 2D profiling and blade to blade analysis require no intermediate user action and one is always required after the other, so keeping their execution separate is

unnecessary. The final and preferred choice for "splitting up" the problem was to separate it into three components. The chosen split separates the process into three activities, auto-blading, 3D geometry and boundary condition generation, and 3D CFD evaluation. Depending on the desired use case, the process can be run through from start to finish, or only individual components may be used. After careful consideration and consultation with experienced users of the tools in question, this choice was selected because it mimics the analysts' process more closely than other alternatives. This method provides the natural pauses, if desired, where the workflow can benefit the most from external and expert user input. For example, auto-bladed profiles can be further adjusted manually by an expert and then re-incorporated into the remaining two parts of the automated workflow. This decomposition approach follows the one suggested by Smith, where each module is composed of a highly similar and repetitive task and can be used as a black box without restricting the modularity of the process [1].

4.2.3 Throughflow solving

A throughflow model and throughflow solution are necessary to initialize the design process. As previously stated, throughflow modelling is out of the scope of this work so the only required action is to run an existing model through a solver. The analysis is evaluated fully in batch and its purpose is to generate the inputs to the automated workflow shown in Figure 8. This is achieved through a single system call to the throughflow solver with the specified geometry and aerodynamics input files.

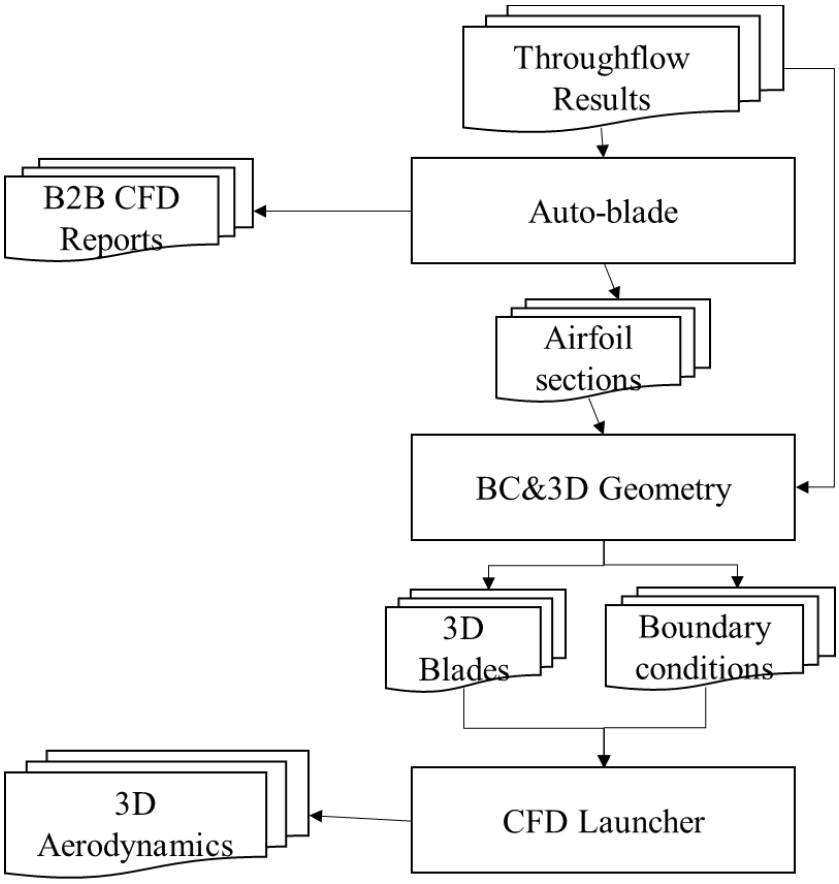

Figure 8: Automation, high level process

4.2.4 Auto-blading

4.2.4.1 Overview

The auto-blading program encompasses four activities performed by the stacking suite: airfoil profiling, airfoil cloning, blade to blade analysis, and exit ait angle matching which were discussed in chapter 3. It makes use of blade row aerodynamics specified by the throughflow analysis, and reference geometries, either from previous designs or rule-based databases to clone or generate new geometries, match their air angles and reduce their losses at the 2D level, and generate blade to blade analysis reports on all newly created geometry. The process generates all the data necessary for stacking new sets of airfoils. The basis of this process is effective bookkeeping and cross referencing of data combined with some reasonable assumptions on how the data is located.

4.2.4.2 Process inputs

The auto-blading automation requires three inputs: blade row aerodynamics, user settings, and reference geometries. The blade row aerodynamics are an output of the throughflow analysis. These results are in xml file format and contain flow properties such as temperatures, pressures, and air angles at blade inlet and exit. There exists one blade row aerodynamics file per blade-row present in the model. For example, for a 7-stage compressor with an inlet guide vane (IGV), there would be 15 blade-rows and, therefore, 15 files.

The user settings file is a custom created file containing all the necessary inputs and metadata for correct execution of the automation. The chosen file format for this user settings file is the JavaScript Object Notation (JSON). The JSON file type is a prescribed requirement to facilitate integration and follows the choice of Ramamurthy et al. in their collaborative design framework [7]. Regardless, it is an excellent choice as it can be loaded directly into a native python object completely eliminating the need for parsing.

The final, optional, inputs are the reference geometries. These are specified depending on whether the user needs to create a preliminary airfoil geometry from scratch, or simply clone and possibly adjust an existing one. These inputs can be retrieved in two different ways, either by importing a local copy of it from a remote storage location, or by finding a pre-existing local copy. Depending on the execution, this optional input will either be in the form of standard named files in XML format or marked as a set of reference numbers for their retrieval in the user settings file. The reference geometry is split into a specific number of 2D cross sections to be stacked. For example, if the reference geometry is split into 11 sections, the input to the tool will be 11 files per blade-row. For the same 7-stage compressor with an IGV, if all geometries are to be cloned, the reference

geometry will be defined by 165 XML files, or 15 reference numbers if the geometry is to be imported during execution.

4.2.4.3 Process outputs

The process creates several output files related to the newly created geometry, the imported reference geometry, if the function is chosen, the blade to blade 2D CFD solver, and the scripts used to run the process in batch with the stacking suite.

The newly created geometry is generated in the same format as the reference geometry: a set of XML formatted files each containing a single 2D blade cross section. If required, the imported reference geometry is also output as a set of xml files. The blade to blade solver's output covers a wide range of file types including .pdf for a human readable output summary, text, and binary. The MISES solver generates one output "set" per evaluated 2D cross section. A default of five cross sections are evaluated with the MISES solver per-blade for quick assessment of their performance. Finally, the process also outputs the auto-generated batch XML files that transfer the execution commands from the script to the stacking suite.

4.2.4.4 Process execution

Execution of the tool can start in two ways: fully or partially automated. If the execution requires initial user input, it will be started in a partially automated mode where the user input is given through a GUI. This GUI will help locate the inputs and select the desired options to generate the user settings file to be used. The user can select root folders where the blade row aerodynamics, and reference geometries will be located. Their selection will trigger an event where the root folders are sorted through recursively up to three subfolders down in order to locate all possible inputs without the need to locate them manually to avoid user error and book-keeping. As previously mentioned, for 10 rows at 11 sections per row this would constitute 110 geometry files

and 10 additional blade row aerodynamics files for a total of 120 files to be located. The execution allows for any number of input files to be located simply by selection of two folders under the assumption that all inputs can be found in no particular order within these and up to three subfolders down.

The next stage of the execution becomes bookkeeping. Having located the input files in the specified directories the geometry files need to be grouped together. Each file contains a unique name identifier which includes its model number, version, section, and corresponding blade row. Files with different section numbers, but matching identifiers are grouped together and indexed for correct tracking by the program. These identifiers are extracted by parsing the input files and extracting the correct tagged element from their XML element trees. The blade row aerodynamics files contain similar unique identifiers (model number, version, and blade row) these identifiers are cross referenced to geometry file groups in order to determine whether they make suitable inputs, and to avoid miss-matching reference geometries to blade row aerodynamics. Once the inputs have been correctly identified, indexed and cross referenced, they are displayed in the user interface where different options can be selected for each. The options were discussed in detail in chapter 4.1.2, and include exit air angle matching, cloning, and auto-blading from three different types of profile databases. In addition, at this stage the user can also select the target location for all outputs.

Once user interaction is complete an input settings file for use by the script itself is generated and signals the initialization of the pre-processor. The pre-processor takes all the settings and input files translating them into XML scripts in the custom tagged tree format required by the stacking suite for batch command execution. In addition, it creates a standard folder structure for the output to follow for simple output tracking by the user and direct compatibility with the other developed

tools. The input scripts are created at the “global level” for the activity and at the “row level”. Global input scripts run through the entire auto-blading, or MISES computation process for all desired blade rows in a single instance of the stacking suite. While row level scripts contain only commands for one blade row per file. These scripts are auto generated for both, traceability and flexibility. If the process requires re-running in another context or another type of setup, the input scripts can be used directly in conjunction with the stacking suite to generate the desired outputs again.

After the pre-processing step, the code handles the sequential system calls to the stacking suite to create, adjust, and evaluate the new geometry. The code calls one instance of the stacking suite per blade row. This type of execution is chosen over all at once because it is more robust. If a single blade-row fails, it will not affect execution of the others. In addition, this type of configuration will lend itself to simpler implementation with multi-processing or multi-threading if/when implemented to allow for multiple rows to be processed simultaneously.

If execution is started on a fully automated manner, the GUI step is bypassed, and execution begins at the pre-processor with the added requirement of a pre-existing input settings file (input.json). The full auto-blading process is displayed in Figure 9. A single instance of the process is required regardless of the number of blades or source of the blade row aerodynamics for either of the execution modes. Moreover, execution is not limited to a single compressor component at a time. Since the program treats and cross references each blade row separately, multiple blade-row aerodynamics for several different compressor models can be specified.

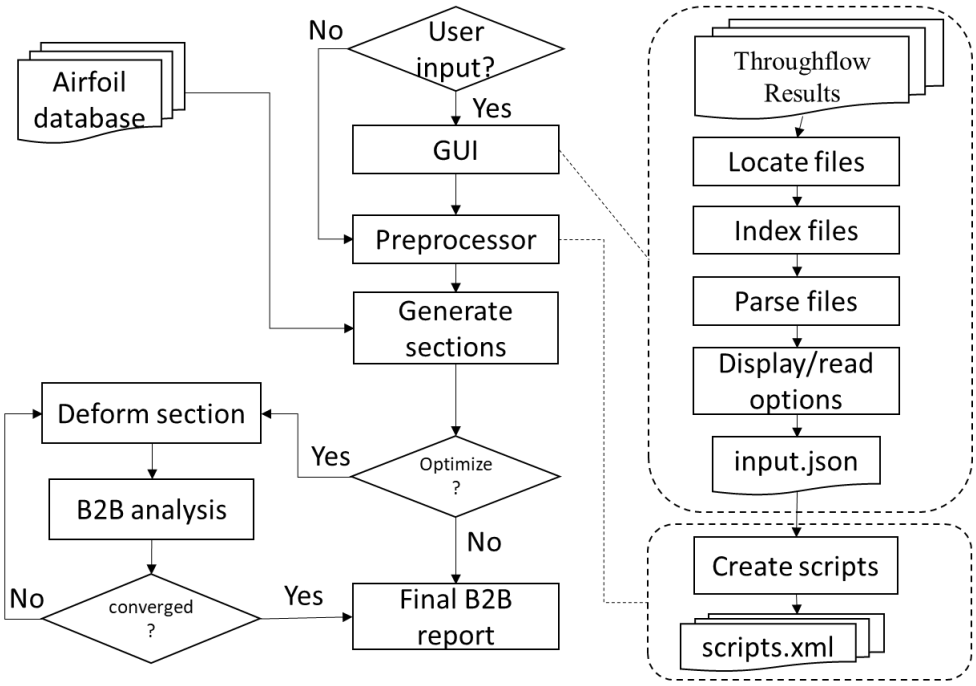

Figure 9: Auto-blading automation process

4.2.5 Blade definition and boundary conditions

4.2.5.1 Overview

The blade definition and boundary conditions program take the stacking data for the airfoils together with the throughflow aerodynamics results and post processes them in order to generate standard text files that define the 3D geometry of each airfoil.

As discussed in chapter 4.2.2, although the software that is used for this component of the workflow is the same as that of the auto-blading program the two capabilities are kept separate from each-other in order to ensure flexibility. In many cases, existing geometries can be directly re-used for new throughflow aerodynamics (i.e., newly created boundary conditions), as will be discussed in chapter 4.2.7 where off design evaluation is touched upon. If that is the case, having to go through the auto-blading component of the workflow every time would be of no use.

In addition, if the desired geometries are stored directly in the airfoil database, handling their importation becomes simpler if the software's capabilities are split into two. Similar to the auto-blading, the bulk of the effort is dedicated to book-keeping. The main challenge here is to cross reference the available airfoil geometry files to the throughflow aerodynamics results' data to make sure that station names, models, and components match. If they match, they become available to the analyst for use, otherwise they are ignored.

4.2.5.2 Process inputs

The blade definition and boundary conditions generator automation requires three inputs: throughflow station aerodynamics results, user settings, and blade-section geometries. The throughflow station aerodynamics results are an output of the throughflow analysis. Similar to the throughflow blade-row aerodynamics, these results are in XML file format but are contained in a single file per throughflow model rather than multiple ones. The single XML file contains results for the entire throughflow mesh.

The blade section geometries are in the same standard file format as the input and output geometry for the auto-blading tool. A natural and obvious source for the blade section geometries is the output from the auto-blading tool; however, any pre-existing blade section geometries can be chosen as long as the required data is compatible with the throughflow aerodynamics results (i.e., the specified blades fit in the duct defined in the throughflow). If the annulus geometry and axisymmetric section dimensions in the throughflow results match those in the section geometries, then they can be used. In addition, the blade section geometries can also be imported from a remote storage location if so desired. In that case the blade section geometries are not input files but, rather, input reference numbers indicating the geometry to be imported.

The user settings file is very similar to the file used for auto-blading. It is also a JSON file containing inputs and metadata for execution. A reference to all input files along with geometry reference numbers to be imported, and run-time options is contained in this user settings file.

4.2.5.3 Process outputs

Two custom formatted text files per blade are generated by the process, one containing the full 3D stacked geometry of the airfoil, and the other containing boundary condition data for use in 3D CFD. In addition, the process outputs any imported section geometry, if required, and auto-generated script files that transfer the execution commands from the tool to the stacking suite.

4.2.5.4 Process execution

Execution is very similar to the auto-blading process in that it uses a custom, by-passable GUI as well to generate the input settings to the code. As before, execution of the tool can start in two ways: fully or partially automated. If the execution requires initial user input, it is started in a partially automated mode where the user interface takes in the desired inputs. The user can select the input throughflow results file, and the directory where the airfoil section geometries will be located. The airfoil section geometries are found, indexed and grouped in the same way as with the auto-blading tool. In this process, the book-keeping step is simpler because there is only a single throughflow results file to locate/specify.

The throughflow results file is then parsed to cross reference the station names to the section geometries via their standard unique name identifiers to ensure compatibility, and to track their order of appearance in the annulus geometry. Once the inputs have been correctly identified, indexed and cross referenced, they are displayed in the user interface where available options are displayed and can be selected for each. The options in this case are reserved to importing geometry from an existing airfoil section database, toggling blade rows to enable/disable their processing in

the execution and setting the relative mixing plane location for each blade row. In addition, at this stage the user can also select a desired the target directory where all outputs will be written to.

Next, the pre-processor generates a single script file to the stacking suite to parametrize the 2D sections and generate all 3D blade geometry files and boundary conditions. After the pre-processing step, the code handles a single system call to the stacking suite to execute the auto generated script. The code calls one instance of the suite with all blade rows at the same time because the processing time is very short (of lower magnitude than initialization time for the stacking suite itself). Making multiple calls to the stacking suite would significantly lengthen execution time and parallel instances would increase computational overhead with non-significant relative time savings.

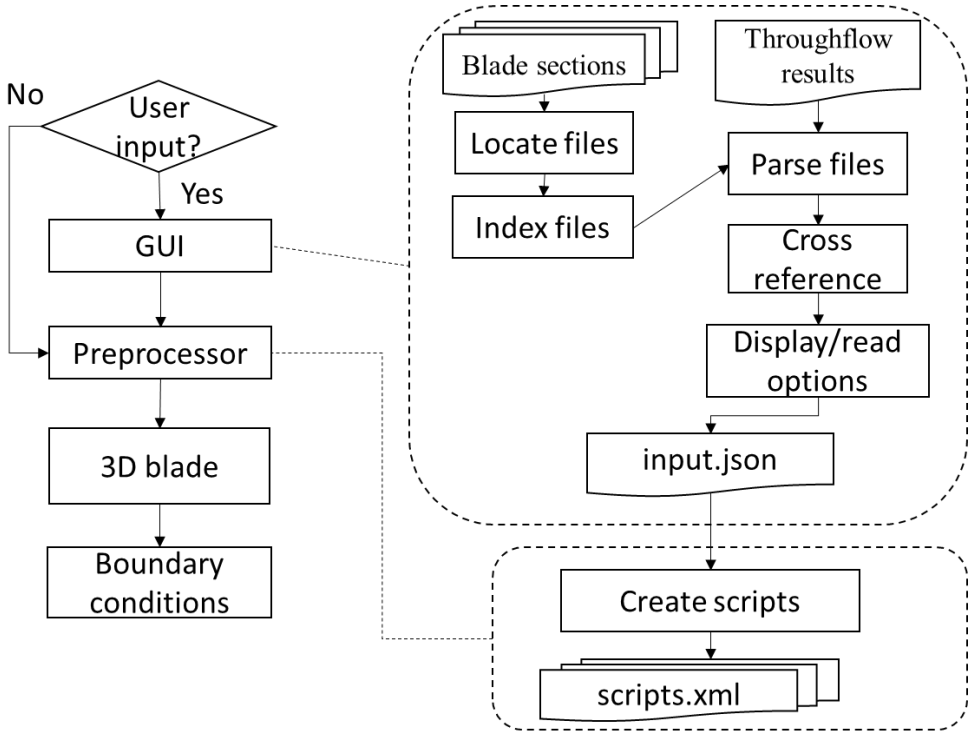

Figure 10: Blade definition and boundary condition generator process

If execution is started on a fully automated manner, the GUI step is bypassed, and execution begins at the pre-processor with a pre-existing input settings file (input.json). The full boundary condition and blade definition generation process is displayed in Figure 10. A single instance of the process is required regardless of the number of blades or source of the throughflow results for either of the execution modes.

4.2.6 CFD Launcher

4.2.6.1 Overview

Similar to the stacking suite, the CFD suite already possesses some batch processing capability via custom formatted input text files. In this particular case, the format is not standard such as XML, therefore parsing or creating general input files from scratch becomes more complicated, especially considering that they are quite case specific. For example, boundary conditions are defined in different ways for blade casing and hub depending on whether it is a cantilevered stator, shrouded rotor, etc., and the mesh parameters such as density, inflation, tip/hub clearance, special features (fillets, pennies, etc.) can vary even for different setups of the same component.

The objective for the automation of this component of the workflow is to create a standard format input file to output file relationship via a single executable that manages all other software specific inputs and system calls. This ensures that any existing, or newly created CFD setup can be readily and easily linked to an optimization algorithm.

The CFD launcher was only developed for fully automated execution since the codes from the CFD suite that it makes use of are already exclusively batch executable. Its automation is a bit more involved than that of the auto-blading and boundary conditions and blade definition generator, so the programming approach is covered in more detail.

4.2.6.2 Scripting approach

The CFD launcher is divided into four classes, each corresponding to a specific code in the CFD calculation suite: meshing, row preprocessor, multi-row and multigrid pre-processor, and steady solver. Calls to each corresponding code are handled by an ‘executor function’ which handles the instantiation of the objects and their correct interaction (see Figure 11).

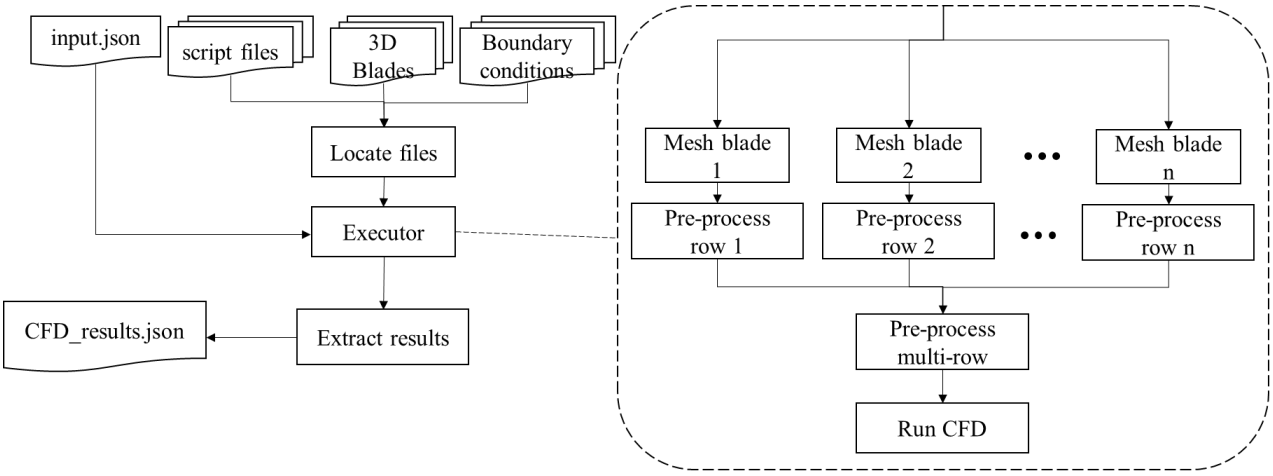

Figure 11: CFD Launcher script process diagram

4.2.6.3 Meshing

The meshing class has two objectives: to take individual airfoil geometries with their annulus and discretize them into a CFD grid or mesh, and to parametrize the airfoil geometry and apply a chosen set of 3-D deformations to the blade. The inputs to the meshing code are a single geometry file, or blade definition, and an input settings file or ‘script’ which includes the desired settings for generation of the mesh. The meshing tool outputs a “new” blade definition file with the deformed airfoil geometry and a CFD mesh to be used for the single row pre-processor. The ‘meshing’ step is executed one time per blade-row in the blade row’s own directory.

The meshing class is capable of parametrizing the blade with as many radial control points, specified in the launcher's input file, as desired for the study and apply the prescribed deformations of the three types shown in Figure 12.

The class contains four simple components that cover all the desired capability:

- Folder structure locator
- File locator
- Input file editor
- Meshing tool launcher

As the name suggests, the folder structure locator walks through the working directory attempting to match the subdirectories to the standard naming and folder structure used for setting up the CFD runs. This first component aims to locate the desired standard folder structure which will be used to find all necessary input files for the CFD suite to execute correctly. In addition, it assists with identifying all the airfoil names for the problem. These names can then be used to cross reference and assign the desired deformations in the main script input file.

Following successful execution of the folder structure locator, the file locator searches for the two required input files for each blade-row: the meshing input file and the blade geometry file. This function serves as a check to ensure that the problem has been setup correctly.

Having located the input file to the meshing software, the input file editor can partially parse the input file searching only for a specific command to activate geometry parametrization and deformation. In addition, the input file editor creates auxiliary files required by the meshing software which define the magnitude and location of the applied span-wise deformations on the airfoil being processed.

Finally, the meshing tool launcher can be called to run the meshing software on the blade and generate the mesh. The meshing tool launcher handles the batch system call to the meshing software while specifying the correct blade folder and input file. A single instance of the class is required per airfoil. Therefore a 5-stage compressor would require 10 meshing class objects to generate all 10 deformed blade geometry meshes.

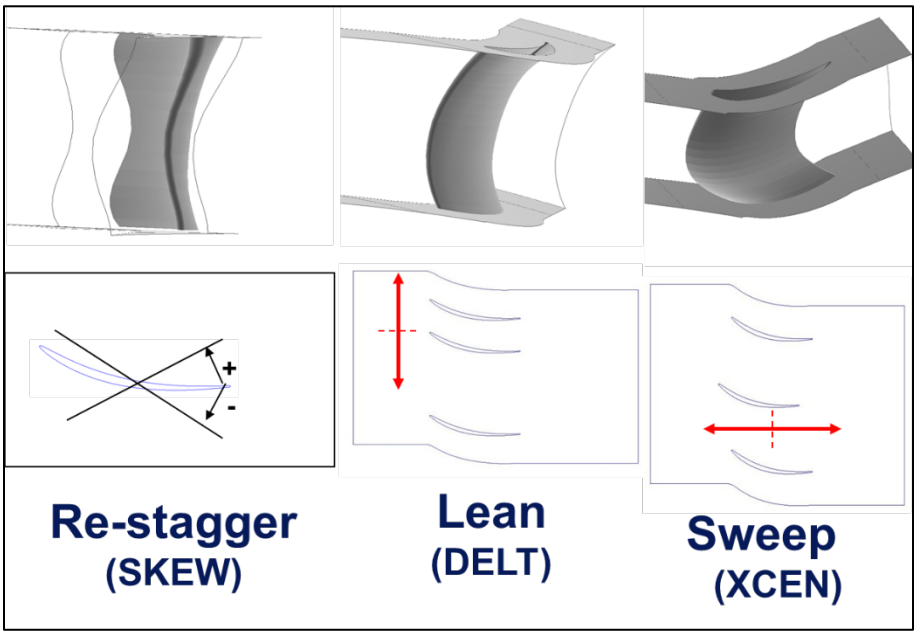

Figure 12: Parametric geometry deformations

4.2.6.4 Row pre-processor

The row pre-processor's purpose is to take the individual blade row's generated grid and convert it into a compatible format for the CFD solver. It also initializes the flow field in this new grid and takes the previously generated boundary conditions file and creates the boundary conditions for the CFD run.

The row pre-processor class is the simplest of the four classes only containing two components:

- File locator

- Row pre-processing tool launcher

Similar to the meshing class the row pre-processor file locator's purpose is to locate the desired input files for the pre-processing software. In this case only one input file needs to be located in the blade folder. This file requires no modifications, therefore once the file is located by the file locator function the row pre-processing tool launcher can be called. Again, similar to the meshing class, the row pre-processing tool launcher handles the batch system call to the meshing software while specifying the correct blade folder and input file. Again, one instance of the row pre-processor class is required per airfoil/blade row.

4.2.6.5 multi-grid preprocessor

Up to this point there exist one CFD model per blade row. The multi-grid preprocessor takes each individual row's grid and combines them into a global mesh which will be used for the steady calculation. The mixing planes (see chapter 3.6) act as reference planes to determine how the grids fit together. Moreover, this pre-processor also collapses the grids into a chosen number of lower density ones.

The multi-grid pre-processor is computationally expensive when run locally and can take a half hour or more of execution time if the setup is large enough (i.e., has more than 10 fine mesh blade rows). Fortunately, the multi-grid pre-processor, along with the solver itself, has the option to be executed remotely in a compute grid with a large number of processing cores. This option, although more difficult to implement allows for very significant pre-processing time savings particularly as the number for blade-rows grows, and thus is the preferred method for implementation.

When submitting the process to the compute grid, execution is no longer handled locally. The system call finishes once the pre-processing job is submitted to the compute grid successfully; however, this only signals the start of the multi-grid pre-processing step. The automation program probes the compute grid's queue at specified time intervals in order to verify the state of the pre-processing job. This ensures, both that the job has been submitted successfully, and eventually completed. Once there is confirmation that the job is complete, the output is verified in order to check for errors before moving on to the next step. If an error is encountered the program issues a critical warning and a non-zero exit code. A faulty pre-processing job does not allow to continue the workflow any further, thus premature termination of the program is suitable approach here.

4.2.6.6 Steady solver

The steady solver takes the output grids and initialized flow and boundary conditions from the multigrid pre-processor and solves for the steady state flow in the multi-row turbomachine model. The steady solver is executed in a remote compute grid with a specified number of computer cores to speed up the calculation.

The CFD solver class manages the submission of the job to the compute grid and monitoring of the job's status. Results for optimization studies are extracted from, so called, monitors which are text files where general performance measures at each iteration for the CFD run and for the model in question can be found. These include mass or area averaged stage efficiencies, capacities, temperature ratios, pressure ratios, turning, etc. Results can be extracted from monitors to close the input file to output file relation that is required to run an automatic optimization study.

Similar to the multi-grid pre-processor, the steady solver needs to be executed by a compute grid with a large number of processing cores in order to maintain a reasonable execution time. The same approach is followed for the solver, but with additional considerations. The CFD run may

diverge, therefore, convergence is also checked while probing the state of the job execution. If the run begins to diverge, or is simply unable to converge, there is no point on continuing the execution of the steady solver until it exhausts its desired iteration budget, this can be a waste of computational resources, and, more importantly, time; however, terminating the automation code itself with a non-zero exit code alone would not abort the run, as it is being handled by a completely different and compartmentalized compute resource. The code must be able to identify early non-convergence or divergence and, in addition mark the computation in the compute grid for deletion to ensure that the failed and "useless" calculation does not run its course.

4.2.6.7 Executor function

The executor function is in charge of managing the input file to output file 'chain' that is necessary to automate the entire process and allow for general use of the CFD suite with an optimization package. The function reads in the input file and distributes these inputs to each of the four objects that were just covered. In addition, it retrieves input scripts/files for the CFD suite and creates local copies of them so that the original model is never overwritten and can serve as a static template for multiple designs to be evaluated simultaneously. Finally, the executor function acts as the main execution logging, and error handler. It issues run-time logging messages and error codes, if issues are encountered while initializing objects or while running the process.

4.2.6.8 Calling the CFD launcher

The CFD launcher can be called as a script once the input file for the launcher itself and the necessary input files for the CFD suite have been created and placed in their standard folder structure. A key feature of the launcher is that its execution is fully compartmentalized and only linked to its working directory. This allows for easy use of parallel, or multi-threaded execution. The user can call the launcher from the same 'source model' into different working directories at

the same time. This feature is used extensively in the test case optimization studies shown in section 5 where the code is used in conjunction optimization programs. The ability to execute several designs simultaneously is very important since a single full fidelity function evaluation can take anywhere between one and five hours depending on the number of blade rows in the component and the maximum number of computer cores allowed for the solver per submission.

4.2.7 Off design evaluation

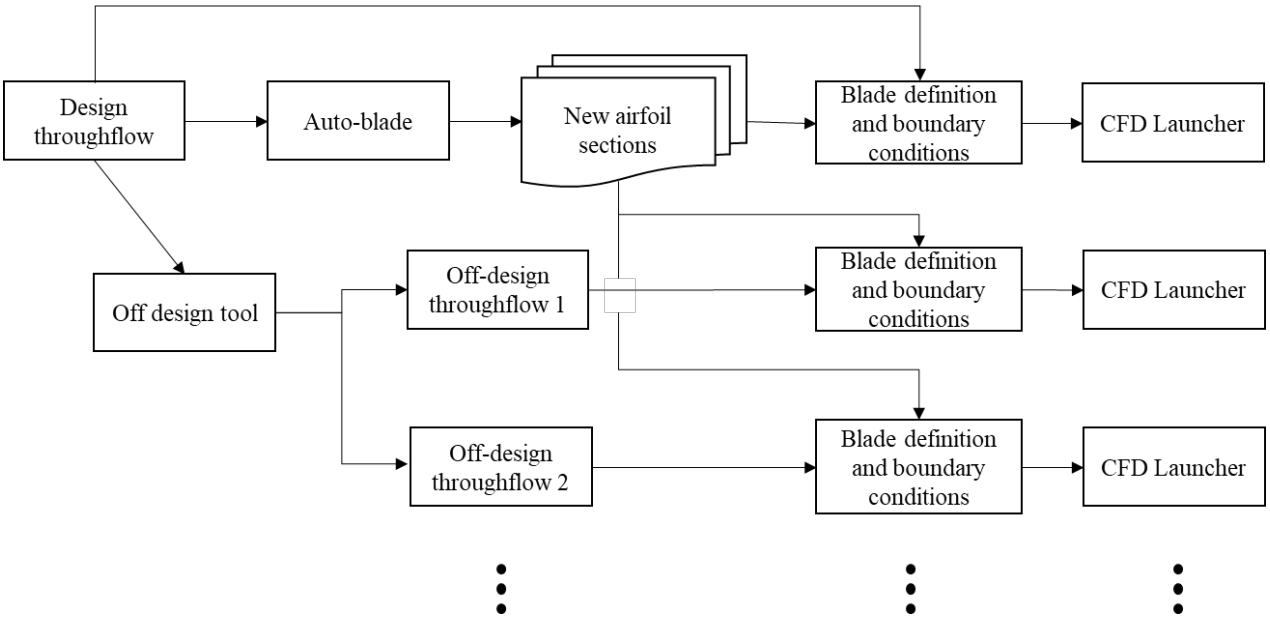

Figure 13: Automation for off design cases

Off design evaluation of new geometry generated by the auto-blading module can be performed by obtaining an off design throughflow analysis. The off design throughflow tool is covered in chapter 4.1.1. The workflow is repeated for off design throughflow results while skipping the auto-blading step and using the "design point" geometry to generate the blade definition and boundary conditions for the off design CFD models which are then handled by individual instances of the CFD launcher. A breakdown of the process for handling off design points is shown in Figure 13. Since all instances of the CFD launcher share the same geometry, a multi-point 3D optimization

may be conducted by defining the deformations for each CFD model as shared variables. This multi-point approach is applied in Case Study 4.

4.2.8 Automation remarks and challenges addressed

The automation makes use of existing functionality available in the software suites themselves and uses their ability to run in batch with customized input files. Although, arguably, the analyst may be capable of writing the input files from scratch, thus avoiding the use of the developed tools altogether; creating these files by hand is time consuming, prone to error, and case specific. The input files will be different every time the component changes, or a new component is considered, and may require hundreds of entries to be changed or added. This alternative may offer some limited time savings to the workflow when compared to the fully manual process; however, the analyst would still be stuck with a tedious non-added value task that the automation already takes care of.

Tomiyama notes that some main reasons for failed design automation in “intelligent platforms” include a lack of room for engineering creativity, too restricted design space due to limited modelling generality, and limited code maintainability [26]. Although reduced, room for engineering creativity is ensured by allowing for user interaction throughout the workflow and by allowing for external modelling input in intermediate steps when necessary. The design space is limited by the chosen and implemented parametrization; however, the automation focuses on generality, addressing this challenge by default, by ensuring that any compressor model can be handled regardless of size. As will be discussed in chapter 6 other parametrizations exist which the automation is readily able to accommodate, thus addressing the challenge of limiting the design space as little as possible as well.

Simpson and Martins identify an additional challenge to successful implementation of design automation for successful implementation into an MDO framework: a component based approach in which individual modules are reusable for different applications [27]. The compartmentalized, and object-oriented nature of the codes addresses this challenge by ensuring code modularity and reusability of the processes without requiring a full code overhaul to implement additional capability. These attributes also address the challenge of code maintainability. Further, existing capability has not been restricted to the same context as that for which the automation was intended. As is demonstrated in Case study 4, components of the automation are usable for different types of problems and models. For example, for Case study 4, the CFD launcher was used for a turbine model design exploration study rather than a compressor without any modifications to the existing codes.

In addition, the challenge of "packaging" the entire CFD process into a single executable for a general model with a single text input file and text output file is fully taken care of so that little to no extra scripting is required for setting up automated optimization studies. This capability was not available before in the CFD suite, and negatively influenced users' desire to perform automated design exploration and optimization studies at the 3D level.

4.3 Code usage in a multi-disciplinary automation platform

4.3.1 Automation platforms

Two in-house developed multidisciplinary automation platforms are considered for integration. The first automation platform is intended for throughflow driven turbine and compressor design. Its main advantage is that it offers direct compatibility with the throughflow suite and offers out of the box parametric throughflow design, execution, and post-processing options. Other capabilities can be built on top of it via custom programmed objects or plug-ins which interact

directly with the platform's execution, or fully isolated executable programs which are called by the platform but do not depend on it to function. The automation codes would fall under the latter category. Their integration only requires knowledge of the standard output data hierarchy and structure from the computational platform which is always consistent. Although very convenient and suitable for this particular workflow, the automation platform becomes restrictive for more complex workflows in a multidisciplinary context. This platform is ideal if the design process is throughflow driven and centered around aerodynamic analysis; even more so if it only involves aerodynamics as its single discipline. However, if more general integration of several different workflows and disciplines is required, usage of the platform becomes less advantageous due to challenges related to data management, version control, operating system dependent processes, and case specific analysis interactions. In addition, the platform does not offer strong collaborative design capability, limiting its use to a single user at a time per design.

The second automation platform is more generalized with no native analysis to it. The principal advantage is version control, smart, and automated data transfer between general workflow activities, and full traceability of data. It consists of an in-house framework aimed at collaborative design and analysis for general computer aided engineering activities and tools. The platform is based on the framework developed by Ramamurthy et al. in which they describe an object-oriented software that allows advanced data management and workflow simulation [7]. They outline a framework incorporating a systems engineering module allowing the user to select, configure, and execute custom developed workflows and models from previously defined and linked inputs. The framework is python based and incorporates an Application Programming Interface (API) that allows on-site developers to easily program new tailor-made workflows upon engineers' request.

Workflows are displayed in standard user-friendly user interfaces which reduce the time required by the engineers to get familiar with new activities and design processes.

The platform also provides post-processing tools for data analysis, and data sharing. Multidisciplinary workflows have already been developed in order to run certain iterative analysis processes. Models, outputs, and other specified results are saved and shared in a PLM database for full design revision traceability [16]. The platform is under development and continuous deployment and offers the ability to link custom processes directly to one-another. In addition, built-in linking of general workflows to commercial optimization packages is under development. Workflows for other disciplines such as performance and secondary air systems at Siemens Energy have already been successfully integrated into this platform. Their integration is partially covered by the work of Peoc'h [6].

4.3.2 Integration

The developed workflow automation can be readily integrated to some extent into either computational platform. For integration into the first platform, the automation codes would be kept as independent executable programs which would not interact with the platform's native framework per-se but would rather high-jack the way in which it generates and structures input and output file data. In this way the automation programs would rely on the outputs generated by other tools built into the platform, such as the throughflow solver, for their execution.

For the second computational platform, full integration is possible. Since the platform is written in python as well, inputs may be passed directly as python objects to the programs, and the need to keep all codes as independent executables is eliminated. In addition, all data transfer and tracking are handled and specified directly and robustly by the platform itself which eliminates the need for added assumptions on where and how to find the necessary data.

5 Case studies

5.1 General considerations

Case studies are performed as tests of the automation capability. A total of four case studies were conducted with varying levels of complexity and specific applications. The case studies aim to mimic real design activities that the automation tool might encounter, and in some cases test the ability of individual components to work fully independently from one another. Benchmarks, in which the automation's efficacy vs that of the manual process are also conducted to highlight the possible benefits in terms of time and usability for non-expert users. Case studies 1 and 2 cover full airfoil redesigns and 3D geometry optimization with different parametrizations, while case study 3 covers 3D constrained geometry optimization. Finally, in case study 4 we demonstrate an extraneous use of the CFD launcher: its ability to handle 3D turbine designs. A simple 3D turbine design exploration study is shown in this final case study.

5.2 Case study 1: single stage redesign

We start the case studies by performing a redesign of a single stage for an intermediate pressure compressor beginning with a previously designed rotor-stator pair of airfoils taken as the reference for the study. The study will cover all capabilities of each sub-component of the automation in a full design walkthrough, from the auto-blading to the 3D CFD optimization at the "design-point". The walkthrough includes the following steps:

- Existing airfoil geometry is directly used to generate boundary conditions and blade definition files for CFD using the newly generated throughflow results.
- Existing airfoil geometry is cloned into the new throughflow blade-row aerodynamics.

- Existing airfoil geometries are cloned into the new throughflow blade-row aerodynamics and adjusted along 4 reference radial sections where their exit air angles are matched.
- New airfoil geometries are generated from scratch from the three different available types of archives and fitted directly to the throughflow results.
- New airfoil geometries are generated from scratch, and blade to blade calculations are performed to correctly fit the air exit angles prescribed by the throughflow.
- All new geometries are used to generate boundary conditions and blade definition files.
- All designs are evaluated via 3D steady CFD.
- The best design is parametrized and optimized at the 3D level with an off the shelf optimization tool.

General, overall performance measures are evaluated via 3D CFD where imposed boundary conditions at the stage exit are static pressures calculated by the throughflow and generated by the blade definition and boundary conditions generator module in the automation framework. From the CFD results the most efficient setup is obtained and optimized further with CFD assisted optimization using 2 methods. The problem is formulated as an unconstrained optimization problem with 9 design variables and a single objective: efficiency.

The 9 design variables include re-stagger of three sections along the rotor and stator spans, lean on the stator hub and tip, and sweep on the stator tip. This formulation is intended to demonstrate the full 3D deformation capability of the CFD Launcher without creating an inordinate amount of design variables for the Blackbox optimization algorithms to handle. The optimization problem is formulated with normalized design variables as follows:

$$\min_{\boldsymbol{x}} \quad f(\boldsymbol{x},\boldsymbol{p}) = -\frac{\eta(\boldsymbol{x},\boldsymbol{p}) - \eta_{baseline}}{\eta_{baseline}} * 100$$

$$\text{Subject to} \quad \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$$

$$\text{Where} \quad \boldsymbol{x} = \begin{bmatrix} SKEW_1^{R1} \\ SKEW_2^{R1} \\ SKEW_3^{R1} \\ SKEW_1^{S1} \\ SKEW_2^{S1} \\ SKEW_3^{S1} \\ DELT_1^{S1} \\ DELT_3^{S1} \\ XCEN_3^{S1} \end{bmatrix}, \quad \boldsymbol{l} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{u} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{2}$$

The isentropic efficiency measure, η, is extracted from the CFD results by the CFD automation code. The input to output file relationship that is established by the CFD launcher allows for simple input communication via parsing and re-generating the input file and parsing and extracting the result from the output file. Commercial optimization packages such as Heeds or iSight already offer extensive and powerful parsing capabilities which minimize the user's programming skill requirements for quickly setting up optimization studies. Alternatively, a function handle can be easily programmed if the user is comfortable with parsing the input and output files. This enables relatively simple integration into a wider variety of optimization methods, including SciPy, MATLAB's fmincon and other algorithms, or even custom written programs. To demonstrate this option the problem is solved using both the commercial package, Heeds, and a custom, python based, simplified parallel implementation of the Mesh Adaptive Direct Search (MADS) algorithm. Details of the MADS algorithm and its convergence properties can be found in the work of Audet and Hare [28]. Setup for Heeds is handled entirely by the package's parsing and processing capabilities, while the setup with MADS involves a python programmed function handle, thus applying the two aforementioned options for setting up the optimization studies.

Airfoil stacking is performed through seven different setups outlined in Table 1 along with their resulting efficiency improvement at the design point. All but one of the configurations offers an improvement in efficiency with the highest relative improvement of 0.46% for Setup 6. This setup is used as the base geometry for the 3D optimization hoping to further improve the stage's performance. The base geometry corresponds to all design variables half-way between their endpoints ($\boldsymbol{x_{baseline}} = [0.5 \quad 0.5 \quad 0.5 \quad 0.5 \quad 0.5 \quad 0.5 \quad 0.5 \quad 0.5 \quad 0.5]^T$).

Table 1: Case study 1, generated geometry efficiency

Setup	**Description**	**Relative efficiency change [%]**
1	CDA Auto-blade	+ 0.2299
2	Baseline clone	+ 0.0146
3	Baseline clone with matched exit air angles	+ 0.0146
4	DCA auto-bade	+ 0.1404
5	DCA with matched exit air angles	- 0.0835
6	Custom rule database auto-blade	+ 0.4642
7	Custom rule database auto-blade with matched exit air angles	+ 0.2966

Along with efficiency for each configuration, other relevant performance measures are also reported such as capacity, total pressure ratio, and total temperature ratio to demonstrate the automation's general results extraction capability. Overall performance measures for all tested re-designs together with the 3D optimization results are displayed in Figure 14.

The 3D Optimization results are summarized below:

$$\boldsymbol{x}_* = [0.18 \quad 0.01 \quad 0.90 \quad 0.14 \quad 0.11 \quad 0.03 \quad 0.00 \quad 1.00 \quad 0.01]$$
$$f_{min} = -0.92436 \tag{3}$$

Thus, the relative improvement in efficiency with respect to the baseline design following the full 3D CFD optimization is approximately 0.924% or double the improvement obtained from auto-blading alone. The stopping criteria for the MADS and Heeds optimizations are a mesh size tolerance of 1E-3 and a maximum of 1200 Blackbox evaluations respectively. The MADS optimization reached its mesh tolerance after 1127 evaluations, while the Heeds optimization exhausted its budget. It is worth noting that the baseline design is already of high quality to begin with further highlighting the usefulness of the framework being tested.

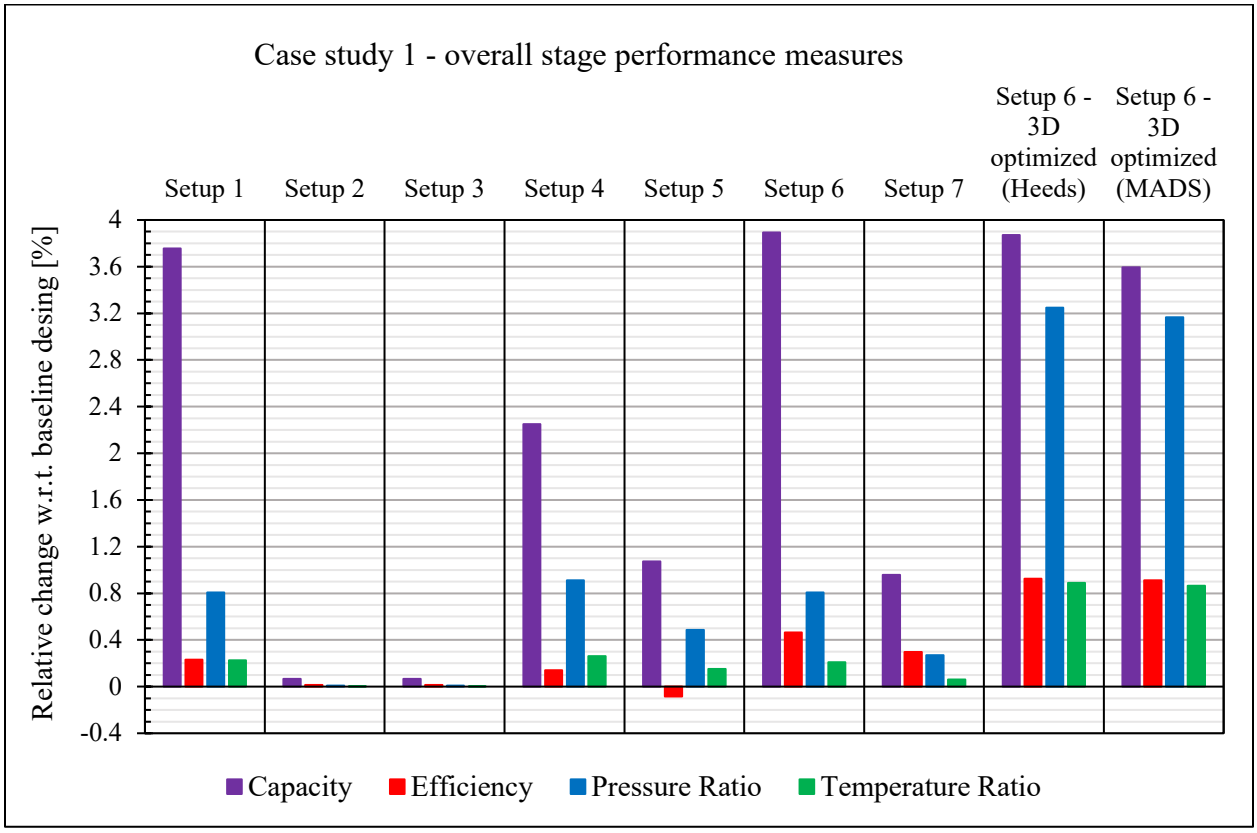

Figure 14: Case study 1, general model performance results

Three relevant geometries for the single stage are compared in Figure 15, where the baseline design is shown in gray, setup 6 in blue, and the 3D optimized geometry in green. Significant geometrical

differences can be observed between the 3 cases demonstrating that the automation and parametrization functioned as expected.

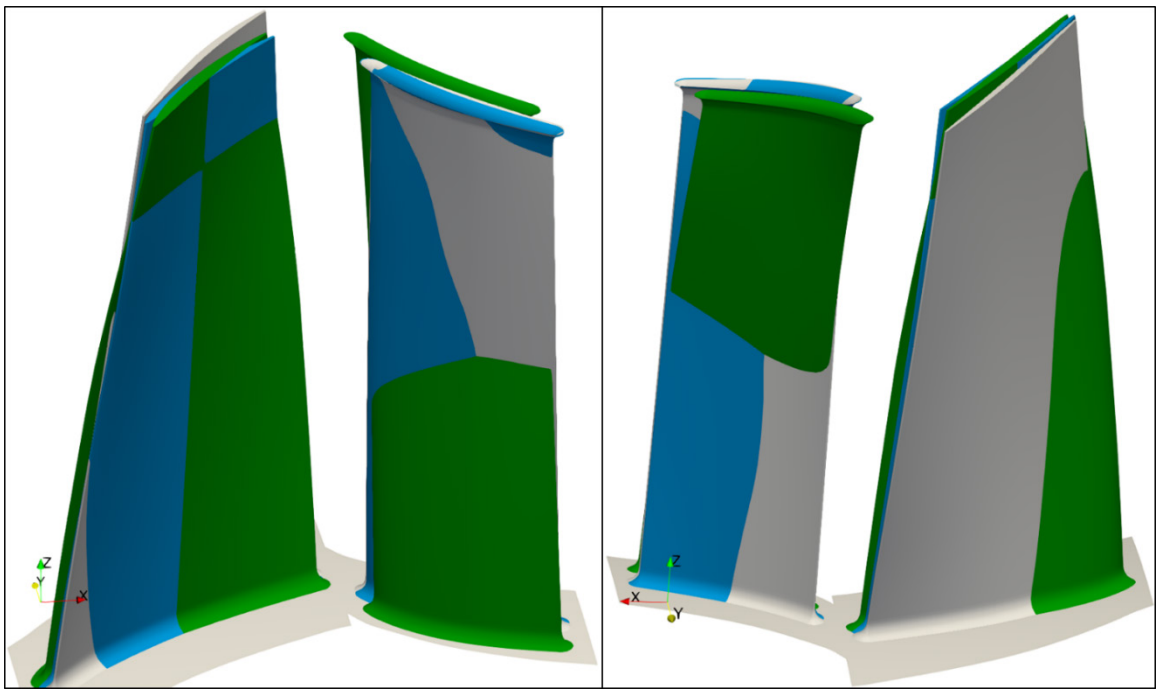

Figure 15: Case study 1 geometry comparison

In addition, some timing considerations are also reported to highlight the other advantage that automation brings to the table: time savings and reduction of non-value-added tasks. Time spent on each task is tabulated and subdivided into two categories: software run time, and user input time. Software run time accounts for the time spent waiting for calculations or other outputs to complete. This is the time that the analyst spends waiting for the software to "run" and does not require any direct user input. User input time, as the name suggests is taken as the time spent by the user while actively interacting with the software (i.e., making selections, clicking, modifying values, etc.). The preliminary airfoil generation process was repeated with an identical setup for all seven cases without use of any developed automation tools to make a meaningful comparison of the time savings.

Table 2: Case study 1, time metrics

Activity type	**Run time [s]**	**User input time**			**Overall time change [%]**
		Manual [s]	**Automated [s]**	**Time difference [s]**	
Auto-blading	119	203	54	- 149	- 46.3
Auto-blading + exit angle matching	367	203	58	- 145	- 25.4
Blade definition generation	91	107	57	- 50	- 25.1

Reporting of time performance measures are all under the assumption that software run time is identical in both cases. Since the commands to the stacking suite are identical for the manual and the automated case, this is a suitable assumption; however, the fact that the automation does not require initialization of the stacking suite's graphical user interface may lead to slightly higher time savings than those tabulated here. Overall, the automation reduces the user input time by over 70 percent for auto-blading and over 45 percent for blade definition and boundary condition generation. Furthermore, it is worth mentioning that the manual user input time will scale linearly with the number of blade rows present in the component being auto bladed since the user must repeat the same process for each new blade. In this case the time to auto-blade a single stage (two rows) is 203 seconds. For an eight-stage system the time will be multiplied by eight, or about 1600 seconds. As will be demonstrated in case study 2, this is not the case for the automated process. The user input time remains roughly constant regardless of the number of airfoils being auto bladed, so the benefit becomes more significant as the "size" of the compressor increases.

5.3 Case study 2: six-stage compressor stator redesign

For the second case study we take the last six stages of an intermediate pressure compressor and redesign its stators with the auto blading capabilities. Following the redesign, the stators are also re-staggered at the 3D level to further improve overall component efficiency. The purpose of this case study is to prove scalability of the framework while also demonstrating its flexibility. In this case, all new components will be a combination of pre-existing geometries with newly cloned or generated ones.

Similar to case study 1, this study covers all capabilities of each sub-component of the automation in a full design walkthrough, from the auto-blading to the 3D CFD optimization at the "design-point". The breakdown of the design activities covered is as follows:

- A new throughflow aerodynamic model is imported into the throughflow suite and solved.
- Existing airfoil geometries for all six rotors and stators are directly used to generate boundary conditions and blade definition files for CFD using newly generated throughflow results.
- Existing airfoil geometries for all six stators are cloned into the new throughflow aerodynamics.
- Existing airfoil geometries for all six stators are cloned into the new throughflow aerodynamics and adjusted along 4 reference radial sections to match their exit air angles.
- New airfoil geometries for the six stators are generated from scratch from three different types of archives and fitted directly to the throughflow results.
- New airfoil geometries for the six stators are generated from scratch, and blade to blade calculations are performed to correctly fit the air exit angles prescribed by the throughflow.

- Blade definition and boundary condition files are generated for all auto-blading/cloning configurations by combining the newly generated stators with the pre-existing rotors.
- Each preliminary airfoil geometry configuration along with the baseline design are evaluated using 3D CFD and a "best performing design" is chosen for 3D optimization.
- 3D optimization of the best performing design is conducted.

The 3D optimization is, again, formulated as an unconstrained optimization problem with 6 design variables (one angular shift per stator). For this case, each of the stators is re-staggered by a constant angle along its span. The deformation capability of the CFD automation has already been demonstrated in case study 1, so the deformations are made simpler in order to maintain a low number of design variables.

The optimization problem is formulated with normalized design variables as follows:

$$\min_{\boldsymbol{x}} \quad f(\boldsymbol{x},\boldsymbol{p}) = -\frac{\eta(\boldsymbol{x},\boldsymbol{p}) - \eta_{baseline}}{\eta_{baseline}} * 100$$

$$\text{Subject to} \quad \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$$

$$\text{Where} \quad \boldsymbol{x} = \begin{bmatrix} SKEW_{all}^{S1} \\ SKEW_{all}^{S2} \\ SKEW_{all}^{S3} \\ SKEW_{all}^{S4} \\ SKEW_{all}^{S5} \\ SKEW_{all}^{S6} \end{bmatrix}, \quad l = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad u = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{4}$$

Again, the objective is to maximize the relative change in efficiency with respect to the baseline design.

As can be observed in Table 3 only half of the newly generated geometries offer an improvement with respect to the baseline design. The best new design is setup 2 with a relative efficiency

improvement of 0.39%. Setup 2 is used as the base geometry for the 3D optimization. For this case study, the optimization is only done with Heeds, and the resulting design is shown below:

$$\boldsymbol{x}_* = [0.47 \quad 0.19 \quad 0.00 \quad 0.00 \quad 0.04 \quad 0.32]$$
$$f_{min} = -0.9147 \tag{5}$$

Table 3: Case study 2, tested setups

Setup	**Description**	**Relative efficiency change [%]**
1	CDA Auto-blade	- 0.5634
2	CDA Auto-blade with matched exit angles	+ 0.3928
3	Baseline clone	- 0.0012
4	Baseline clone with matched exit angles	- 0.0012
5	DCA auto-bade	+ 0.2682
6	DCA with matched exit angles	+ 0.2198
7	Custom rule database auto-blade	- 0.2022
8	Custom rule database auto-blade with matched exit angles	+ 0.3504

The 3D optimized geometry from setup 2 offers an additional relative efficiency improvement of slightly over 0.91% but with reduced capacity as reported in Figure 16. The significant efficiency improvement comes at the cost of reduced component capacity at the prescribed static exit pressure. This trade-off may be acceptable depending on product requirements. Nevertheless, the case study's optimization results demonstrate that automated analysis for use with numerical

optimization is now readily available regardless of the compressor's configuration, and design exploration is greatly facilitated.

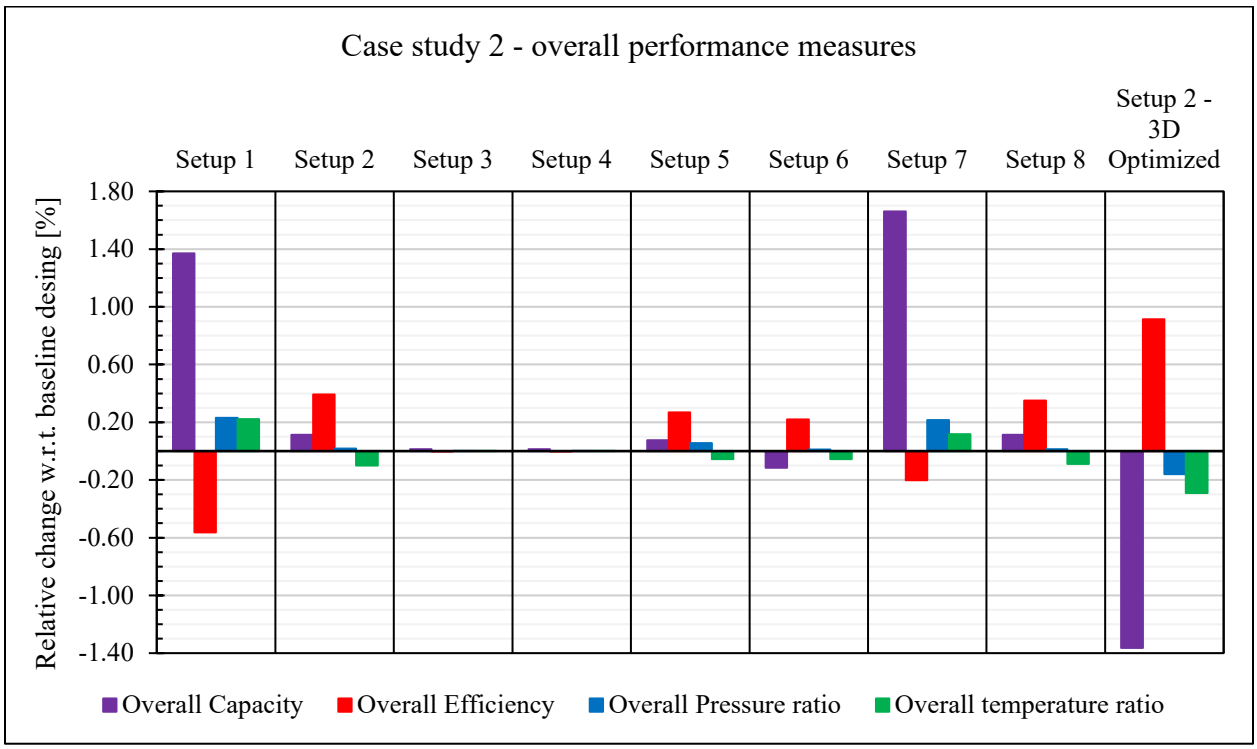

Figure 16: Case study 2, general model performance results

We also note that the type of rapid and preliminary design space exploration that is made possible by the automation framework can help mitigate design performance issues early on and with low added effort by an analyst or multidisciplinary engineer. This is demonstrated in Figure 17 where areas of flow separation and flow reversal around the six-stage compressor are shown. The separation bubble present and highlighted in the baseline design is eliminated during the preliminary redesign. For this case study, where the component has several stages, the time savings become significantly higher and are summarized in Table 4. This is due to the fact that manual user input has to be repeated for each blade row. For this case the user input time is more than 26

times lower in the semi-automated case than the manual case for the auto-blading process. Overall time spent is also greatly reduced.

Table 4: Case study 2, time metrics

Activity type	Run time [s]	User input time			Overall time change [%]
		Manual [s]	Automated [s]	Time difference [s]	
Auto-blading	326	1215	39	- 1176	- 76.3
Auto-blading + exit angle matching	944	1215	46	- 1169	- 54.1
Blade definition generation	53	643	76	- 567	- 81.4

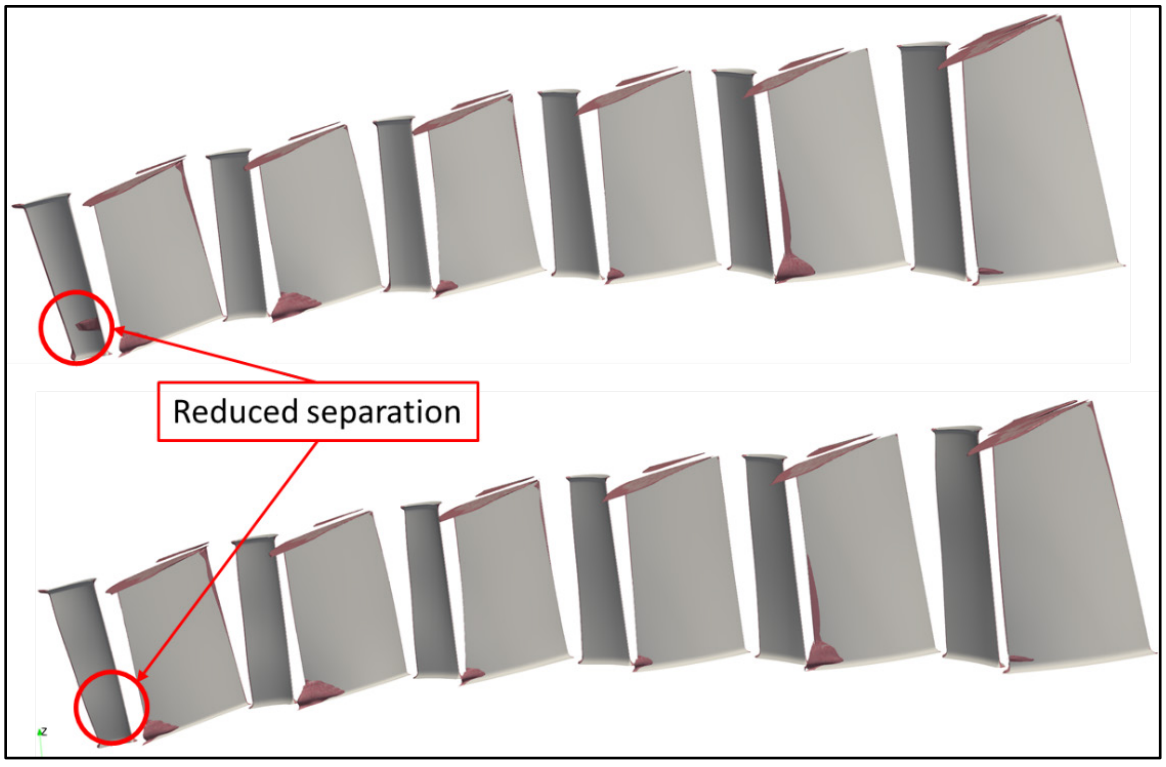

Figure 17: Case study 2, flow separation, baseline (top) vs. optimal design (bottom)

5.4 Case study 3: multirow compressor constrained 3D optimization

This third case study focuses solely on the capability of the CFD launcher to handle pre-existing models for constrained 3D geometry optimization. Here, a two-stage low pressure compressor is redesigned for efficiency improvement while targeting a desired capacity range at the prescribed exit static pressure. The inlet guide vane and stators are left unmodified, while the two rotos are re-staggered at three points each along their span. The constrained optimization problem is formulated as follows:

$$\min_{\boldsymbol{x}} \quad f(\boldsymbol{x},\boldsymbol{p}) = -\frac{\eta(\boldsymbol{x},\boldsymbol{p}) - \eta_{baseline}}{\eta_{baseline}} * 100$$

Subject to:

$$\begin{gathered} \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \\ g_1(\boldsymbol{x},\boldsymbol{p}) = \left(\frac{\dot{m}(\boldsymbol{x},\boldsymbol{p})\sqrt{T_{01}}}{p_{01}}\right) - 0.99\left(\frac{\dot{m}\sqrt{T_{01}}}{p_{01}}\right)_{baseline} \leq 0 \\ g_2(\boldsymbol{x},\boldsymbol{p}) = 0.985\left(\frac{\dot{m}\sqrt{T_{01}}}{p_{01}}\right)_{baseline} - \left(\frac{\dot{m}(\boldsymbol{x},\boldsymbol{p})\sqrt{T_{01}}}{p_{01}}\right) \leq 0 \end{gathered} \tag{6}$$

Where:

$$\boldsymbol{x} = \begin{bmatrix} SKEW_1^{R1} \\ SKEW_2^{R1} \\ SKEW_3^{R1} \\ SKEW_1^{R2} \\ SKEW_2^{R2} \\ SKEW_3^{R2} \end{bmatrix}, \quad \boldsymbol{l} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{u} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

The problem is again solved using Heeds with the algorithm SHERPA. Results of the optimization study are shown in Figure 18 where feasible points are shown in green and infeasible ones are shown in red. The best feasible design is defined as follows:

$$\begin{gathered} \boldsymbol{x}_* = [0.68 \quad 0.16 \quad 0.42 \quad 0.99 \quad 0.26 \quad 0.54] \\ f(\boldsymbol{x}_*, \boldsymbol{P}) = -0.2019 \end{gathered} \tag{7}$$

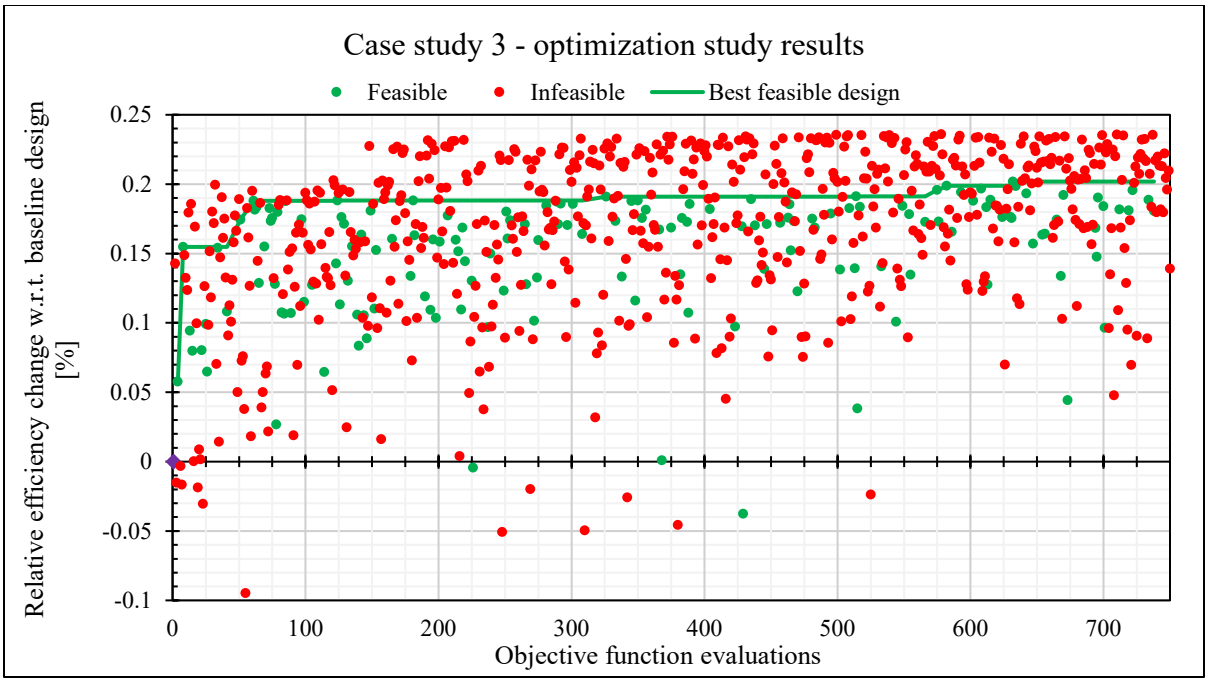

Figure 18: Case study 3 optimization results

The geometry for this design is compared to the baseline geometry in Figure 19. The relative efficiency improvement for the optimal design is of 0.20 percent, and the second capacity constraint $g_2(\boldsymbol{x}, \boldsymbol{p})$ is active for this solution. We also note that the baseline design did not satisfy the design constraints. By use of the CFD launcher in conjunction with an off the shelf optimization package an improved feasible design was easily obtained.

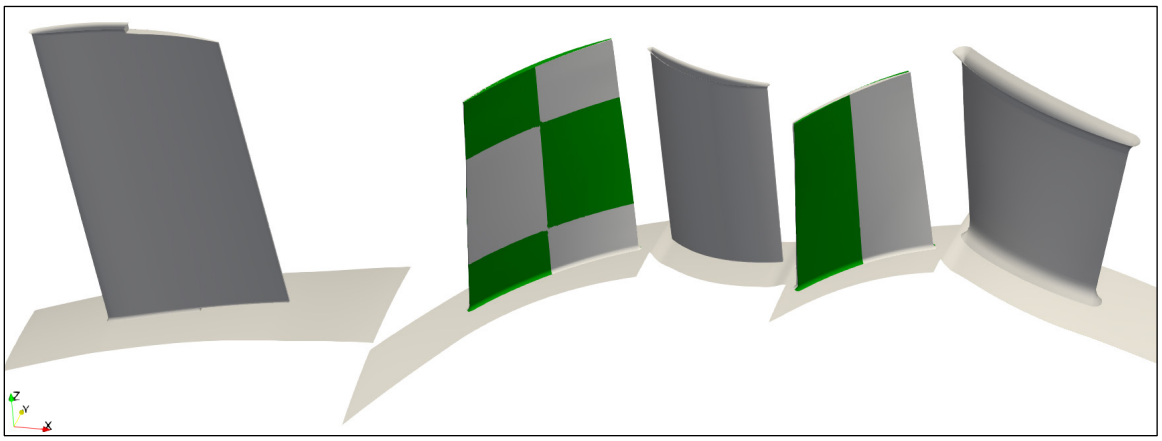

Figure 19: Case study 3 geometry comparison

5.5 Case study 4: multirow turbine 3D optimization

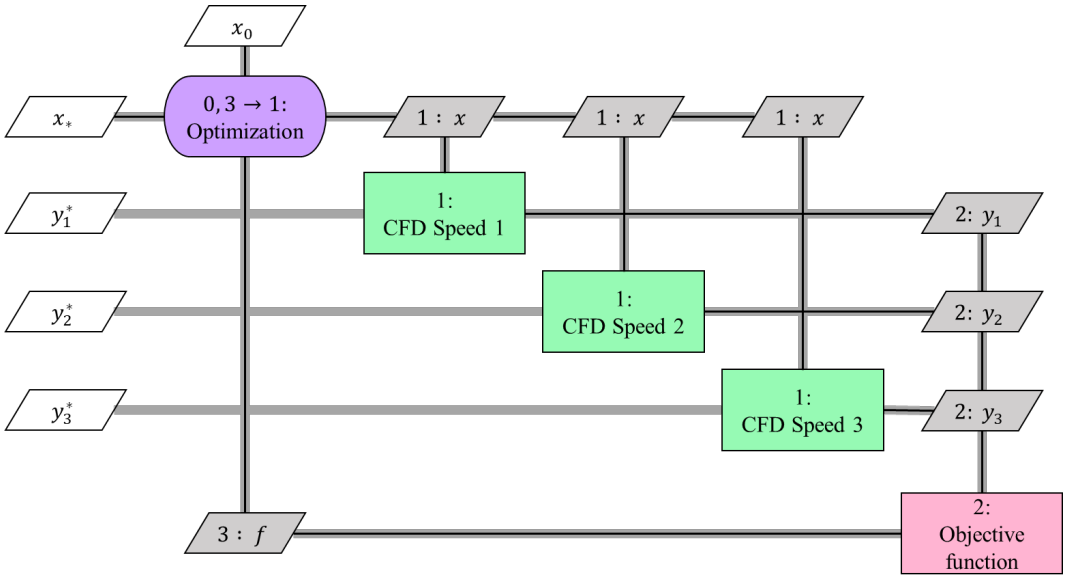

Figure 20: Case study 4, problem XDSM

For this special and final case study, extra capability of the CFD launcher is tested. The way in which turbine and compressor models are defined in terms of data structure, input files, and processing sequence in the CFD suite is very similar; to the point where an existing steady adiabatic turbine CFD model can be readily linked to the CFD launcher for automated optimization. In addition, in chapter 4.2.6 we mention the CFD launcher's ability to evaluate multiple models simultaneously for fast multi-point optimization. These two extra capabilities are used for this case study, where we conduct a design optimization problem on a three-stage turbine CFD model at three different rotational speeds.

The objective function for this study is defined as a weighted sum of the component's overall isentropic efficiency and specific work output at all three speeds. The three analyses share a single design variable which is the re-stagger angle of the third stage vane. The problem formulation is described mathematically with the normalized design variable as follows.

$$\min_{x} \quad f(x, \boldsymbol{P}) = -\sum_{i=1}^{3}\left(\frac{\eta_i(x, \boldsymbol{P})}{(\eta_i)_{baseline}} + \frac{w_i(x, \boldsymbol{P})}{(w_i)_{baseline}}\right)$$

$$\text{Subject to} \quad l \leq x \leq u \tag{8}$$

$$\text{Where} \quad x = SKEW_{all}^{V3}, \quad l = 0, \quad u = 1$$

The problem is visually presented in Figure 20 where we show its extended design structure matrix (XDSM) based on the convention developed by Lambe and Martins [29]. The output of each analysis is denoted by the letter "y" and includes the two performance measures used to define the objective function.

Code execution, and setup of the study were successful; however, run results are inconclusive, possibly indicating that the design variable choice, and/or problem formulation have been improperly selected. A detailed account of the results for this case study is pending approval.

6 Conclusion

6.1 Closing remarks

An automation framework for general preliminary axial compressor airfoil generation specific to Siemens Energy Canada's computational tools was developed and tested. The automation makes use of and extends existing capabilities in the tools to enable rapid generation of high-quality airfoil geometries starting from a throughflow model and progressing onto blade to blade analysis and 3D CFD. Emphasis was placed on its ability to handle general compressor designs through case studies which demonstrated its successful application, and out of the box usability. The code was shown to facilitate and enhance preliminary design space exploration by providing significant time savings to the existing workflow without restricting it and providing a useable and user-friendly link to currently available, but rarely used numerical optimization methods.

The case studies also highlight certain weaknesses, which include the inability to define a more general optimization problem on a section by section basis at the 2D level in conjunction with the MISES solver, and the lack of a more involved multi-fidelity approach to coordinating the use of 2D and 3D modelling such as the one presented by Bayoumy and Kokkolaras [30]. Ultimately the automation is restricted to a single and specific process which does not naturally lend itself to more sophisticated problem formulations without further software development, leaving ample room for future work.

6.2 Future work

6.2.1 Integration into a multidisciplinary platform

Although integration of the tool into a multidisciplinary platform is discussed here, the automation tools have not yet been fully linked and tested. Case studies were all performed with the tools as

stand-alone programs. In theory, integration should not be too difficult, however, this is yet to be confirmed in practice. An additional challenge that will need to be addressed following integration is interaction with other disciplines. The aerothermal discipline is strongly coupled to other disciplines when it comes to preliminary compressor design such as performance, mechanical integrity, lifing, and secondary air systems. Some of these interactions have been partially captured in other stages of the overarching research project from which this particular research is a part of. Implementation of these interactions would enable a more complete and powerful automated multidisciplinary design workflow where true MDO problems can be formulated and numerically solved.

6.2.2 Code refactoring and extra capabilities

In addition to integration, refactoring of the automation code to follow best practices in terms of robust software development is still required to ensure fault free use by early adopters and testers. As it stands, the automation works and has been somewhat tested, but limited time and attention has been dedicated to error handling, reporting, and standardized automated testing. This aspect still needs to be addressed more thoroughly. There is also the matter of maintaining and possibly expanding the developed tools. As desired use by analysts evolve, capability may need to be expanded, for example, to accommodate more intermediary steps that may have not been considered here, or to extend the workflow. An important addition being considered is a more powerful parametrization. In this work the geometry deformation at the 3D level is relatively simple; however, the software being utilized already possesses more sophisticated deformation techniques, such as free-form deformation, which may be worth exploring in order to further increase the efficacy of the framework in generating high quality preliminary geometries.

References

[1] A. Smith and N. Bardell, "A driving need for design automation within aerospace engineering," in *11th Australian International Aerospace Congress, Melbourne, Australia*, 2005: Citeseer.

[2] V. Panchenko, H. Moustapha, S. Mah, K. Patel, and M. Dowhan, "Preliminary multi-disciplinary optimization in turbomachinery design," PRATT AND WHITNEY CANADA CORP LONGUEUIL (QUEBEC), 2003.

[3] S. Markus, V. Christian, and N. Eberhard, "Design optimization of a multi-stage axial compressor using through flow and a database of optimal airflows," *Journal of the Global Power and Propulsion Society,* doi: 10.22261/JGPPS.W5N91I.

[4] M. Schnoes and E. Nicke, "A Database of Optimal Airfoils for Axial Compressor Throughflow Design," *J. Turbomach,* vol. 139, no. 5, 2017, doi: 10.1115/1.4035075.

[5] A. Oyama Akira, "Multiobjective Optimization of a Multi-Stage Compressor Using Evolutionary Algorithm," (in eng), *38th AIAA/ASME/SAE/ASEE Joint Propulsion Conferenceamp; Exhibit,* 2012.

[6] T. Peoc'h, "Automation & integration of secondary air system workflow for multidisciplinary design optimization of gas turbines," École de technologie supérieure, 2019.

[7] A. Ramamurthy, J. V. del Rio, F. Villeneuve, and J. Veer, "Development Of A Sysml Framework For Gas Turbine Design Under Uncertainty," 2016.

[8] F. Lagloire, Y. Ouellet, B. Blondin, F. Garnier, and H. Moustapha, "Single platform integration environment for turbine rotor design and analysis," *Journal of Energy and Power Engineering,* vol. 8, no. 9, 2014.

[9] J. R. R. A. Martins and A. B. Lambe, "Multidisciplinary design optimization: A survey of architectures," *AIAA Journal,* vol. 51, no. 9, pp. 2049-2075, 2013, doi: 10.2514/1.J051895.

[10] D. Bu¨ che, G. Guidati, and P. Stoll, "Automated design optimization of compressor blades for stationary, large-scale turbomachinery," in *Turbo Expo: Power for Land, Sea, and Air*, 2003, vol. 36894, pp. 1249-1257.

[11] F. Sieverding, B. Ribi, M. Casey, and M. Meyer, "Design of industrial axial compressor blade sections for optimal range and performance," *J. Turbomach.,* vol. 126, no. 2, pp. 323-331, 2004.

[12] U. Siller, C. Voß, and E. Nicke, "Automated multidisciplinary optimization of a transonic axial compressor," in *47th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition*, 2009, p. 863.

[13] I. Martin, L. Hartwig, and D. Bestle, "A multi-objective optimization framework for robust axial compressor airfoil design," *Structural and Multidisciplinary Optimization,* vol. 59, no. 6, pp. 1935-1947, 2019, doi: 10.1007/s00158-018-2164-3.

[14] J. Trepanier, A. Lupien, and C. Tribes, "A 3d parameterization for transonic fan blade multidisciplinary design," *Aeron Aero Open Access J,* vol. 1, no. 1, pp. 31-39, 2017.

[15] L. Sommer and D. Bestle, "Curvature driven two-dimensional multi-objective optimization of compressor blade sections," *Aerospace science and Technology,* vol. 15, no. 4, pp. 334-342, 2011.

[16] C. Ebert, "Improving engineering efficiency with PLM/ALM," *Software & Systems Modeling,* vol. 12, no. 3, pp. 443-449, 2013.

[17] M. Tarkian, "Design automation for multidisciplinary optimization: A high level cad template approach," Linköping University Electronic Press, 2012.

[18] Y. Ouellet, C. Savaria, F. Roy, H. Moustapha, and F. Garnier, "A Preliminary design system for turbine discs," *International Journal of Turbo & Jet-Engines,* vol. 36, no. 3, pp. 329-338, 2019.

[19] p. Rolls-Royce, *The jet engine*. Chichester, West Sussex: Wiley (in English), 2015.

[20] H. Cohen, G. F. C. Rogers, P. Straznicky, H. I. H. Saravanamuttoo, and A. R. Nix, *Gas turbine theory*, 7th edition / ed. Upper Saddle River: Pearson (in English), 2017.

[21] S. L. Dixon and C. Hall, *Fluid mechanics and thermodynamics of turbomachinery*. Butterworth-Heinemann, 2013.

[22] T. Belamri, P. Galpin, A. Braune, and C. Cornelius, "CFD analysis of a 15 stage axial compressor: Part I—Methods," in *ASME Turbo Expo 2005: Power for Land, Sea, and Air*, 2005: American Society of Mechanical Engineers Digital Collection, pp. 1001-1008.

[23] M. V. Petrovic, A. Wiedermann, and M. B. Banjac, "Development and validation of a new universal through flow method for axial compressors," in *ASME Turbo Expo 2009: Power for Land, Sea, and Air*, 2009: American Society of Mechanical Engineers Digital Collection, pp. 579-588.

[24] M. Drela and H. Youngren, "A User's Guide to MISES 2.53," *Massachusetts Institute of Technology, Cambridge, MA,* 1998.

[25] C. Miller, C. Hersberger, and M. Jones, "Automation of common building energy simulation workflows using Python," in *Proceedings of Building Simulation*, 2013, pp. 210-217.

[26] T. Tomiyama, "Intelligent computer-aided design systems: Past 20 years and future 20 years," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AI EDAM,* vol. 21, no. 1, p. 27, 2007.

[27] T. W. Simpson and J. R. R. A. Martins, "Multidisciplinary Design Optimization for Complex Engineered Systems: Report From a National Science Foundation Workshop," *Journal of Mechanical Design,* vol. 133, p. 101002, 2011.

[28] C. Audet and W. Hare, *Derivative-free and blackbox optimization*. Cham, Switzerland: Springer (in English), 2017.

[29] A. B. Lambe and J. R. Martins, "Extensions to the design structure matrix for the description of multidisciplinary design, analysis, and optimization processes," *Structural and Multidisciplinary Optimization,* vol. 46, no. 2, pp. 273-284, 2012.

[30] A. H. Bayoumy and M. Kokkolaras, "A Relative Adequacy Framework for Multi-Model Management in Design Optimization," *J. Mech. Des,* vol. 142, no. 2, 2020, doi: 10.1115/1.4044109.