POLITECNICO DI TORINO

Politecnico di Torino

von KARMAN INSTITUTE FOR FLUID DYNAMICS

MSc THESIS IN AEROSPACE ENGINEERING
MAJOR: PROPULSION SYSTEMS

Development of a data-driven design tool for turbomachinery blades: Exploration of secondary flows in 3D turbine cascades

Supervisors: Prof. Andrea FERRERO
Prof. Sergio LAVAGNOLI

Advisor: Ph.D. Gustavo LOPES

Candidate: Cosmo SCARIMBOLO

Submitted in collaboration with the
von Karman Institute for Fluid Dynamics

Academic Year 2024/2025
Graduation session April 2025

Abstract

Gas turbines, particularly those in aircraft propulsion, represent a cutting-edge engineering field. Although advances in cooling technologies and materials have led to mechanical efficiency above 90 percent, economic and physical constraints limit future increases. As a result, improving machine performance still relies on minimising aerodynamic losses, assuming engine performance parameters—such as turbine inlet temperature or bypass ratio—are already established, thereby defining the propulsive efficiency. Among these, secondary-flow-induced losses — arising from the complex interaction between the main passage flow and endwall boundary layers — still represent a challenge in the design phase. Despite several empirical correlations and simplified models have been proposed over the century of research, a comprehensive physics-based model that quantifies these losses across design stages, while incorporating secondary flows physical principles, has yet to be realised. This thesis develops a Python-based computational tool that automates the three-dimensional aerodynamic analysis of turbine cascades through NUMECA Cadence FINE^{TM}/Turbo software. The tool integrates key stages of the CFD (Computational Fluid Dynamics) workflow, including blades CAD (Computer Aided Desgin) generation through profile extrusion, mesh generation, boundary conditions computation, based on operational Reynolds and Mach numbers, and the setting of desired inlet boundary layer profiles. Following this, the tool automates simulation execution, its monitoring, and post-processing data, extracting key aero-thermodynamic quantities to evaluate aerodynamic losses. Through data collected from arbitrary blade configurations, this framework allows the preliminary investigation of secondary losses and their sensitivity to design parameters, drawing from methodologies established in prior research. The study is structured as followed: a literature review on secondary flows-induced and endwall losses; a methodology section outlining the tool development and key computational processes; a further validation and comparative

analysis of processed blade configurations; and a final discussion of results, literature comparisons, and future research directions. The long-term objective is to deliver quality data useful to establish loss correlations with design parameters and develop a breakdown of losses, aiming to evaluate the secondary-flow induced ones. Thus, advancing current analytical approaches for turbine cascade performance optimisation through machine-learning based models.

Keywords: secondary flows, endwall losses, turbine blade design, automated design Python tool, data-driven, optimization

Contents

Chapter 1

Introduction

Gas turbines, especially those in aircraft engines, represent the highest achievements in precision manufacturing. While these machines have reached mechanical efficiency levels above 90 percent through advances in materials and cooling systems, physical limits and cost barriers make further mechanical improvements impractical. As a matter of fact, minimising aerodynamic losses remains a pivotal priority in enhancing the performance of the machines.

Aerodynamic losses are driven by the flow boundary layer viscous dissipation when comes to interacting with walls and blades surfaces. In fact, they have historically been classified into profile losses - relatively straight-forward to quantify since driven by two dimensional flow -, endwall losses and secondary-flow induced losses, whose prediction in the machine design phase still determine a big challenge since they are generated from the interaction of the main passage flow and the endwall boundary layer, generating complex three dimensional vortex structures. With a particular focus on 3D turbine cascades, despite the availability of empirical correlations and simplified theoretical models, still no comprehensive analytical framework exists to quantify secondary losses accurately whichever design space is chosen, especially in the field of turbines which is particularly deepened in the study carried out.

The scope of this thesis is to develop a design Python-based tool that provides a robust computational framework to perform a three-dimensional CFD (Computational Fluid Dynamics) analysis - through the software NUMECA Cadence FINETurbo - starting from a two-dimensional turbine blade profile that has already been optimised to fulfil certain aerodynamic duties. Among all its features, it performs geometry extrusion - CAD generation -, the structured mesh generation from a template validated through grid convergence

studies; setting of the computational environment by computing boundary conditions with operational Reynolds and Mach numbers, and a desired inlet boundary layer profile as user inputs, then launching and monitoring the computation; and ultimately, the evaluation of the losses generated by the three-dimensional flow field in the cascade from the aero-thermodynamic quantities extracted. This workflow is subsequently automated to process arbitrary blade configurations, enabling systematic data collection and analysis, building upon the methodological framework established by J. Coull, [4]. The long-term research objective aims to establish correlations between design parameters and aerodynamic losses, while developing a loss breakdown of their typology and mechanisms of formation. This future development would extend the analytical approaches proposed until now.

This thesis is structured into four main chapters, beginning with a comprehensive literature review in Chapter 2, which explores the fundamental physics and mechanisms governing secondary flows and endwall losses in turbine passages. Chapter 3 portrays the methodology, describing tool development and its capabilities through the most significant steps of the aforementioned pipeline: creation of a blade, creation of a mesh, boundary condition management, computation and parallelization, extraction of data, and post-processing. Chapter 4 addresses testing and proving the automated algorithm, comparing successive sets of processed blades and changing sets of optimization parameters, and examination of outputted data. Chapter 5, in conclusion, illustrates acquired results, comparisons with present literature, and future work potential.

Chapter 2

Literature survey

Gas turbines play a critical role in aeronautics and power generation and operate in a Brayton cycle to efficiently convert fuel energy into work mechanically. This process first involves a compressor, pressurizing the working fluid before entering a combustion chamber. In the chamber, fuel is combusted, and temperature and level of energy in the fluid is increased. High-enthalpy flow then expands through a turbine, in which work is extracted mechanically in an attempt to drive both a compressor and, in case of aircraft "air-breathing" engine, to accelerate the fluid.

The overall efficiency of a gas turbine is determined by two key factors: mechanical efficiency and thermal efficiency. As advancements in materials technology drive mechanical efficiency towards its optimum level, thermal efficiency can then be maximized by increasing cycle temperatures, TIT (turbine inlet temperature), which influences the specific thrust -the overall output of the thermodynamic cycle. However, this involves sophisticated techniques for cooling and cutting edge temperature-resistant materials, not mentioning also that high combustion temperatures will result in high emissions of gases like NO_x. Alternatively, a propulsive efficiency gain can be reached by increasing the engine Bypass Ratio (BPR) - the fraction of cold, low speed mass-flow over the total mass flow ingested by the engine, but this is correlated to an increase of aerodynamic drag and weight, as the size of the engine tends to grow. Hence, a more sustainable approach can be minimising losses, in particular limiting the total pressure drops along the gas path within the engine components.

In an adiabatic turbine, where heat exchange with the surroundings is minimal, efficiency enhancement is a matter of finding and controlling entropy production sources. One of the prominent sources of entropy production in turbo-machinery components is **aerodynamic loss** due to viscosity dissipation in the flow field. Its control and minimisation is crucial to

achieve better performance, leading to a considerable rise in overall efficiency. Among the numerous sources of aerodynamic loss, secondary flows and endwall loss have a critical, specific role to play. Losses occur out of complex boundary layer, pressure gradient, and three-dimensional flow structures in blade passage. With a peak focus towards turbines, an accurate prediction and control of such loss remains a focal point in present-day turbine design.

Denton investigated the sources of loss in turbomachinery components and scrupulously classified them in both turbines and compressors, [9]. To what extents turbines, a drop in efficiency is strictly linked to any irreversible flow process that generates entropy. In fact, the most accurate means of measuring losses is entropy - as it does not depend on reference frame -, whereas it is very difficult to measure in experiments, at the same time. Hence, an acknowledged way of computing a machine efficiency for **linear cascades** design purposes, in this case turbines, is the **enthalpy loss coefficient**. It is defined as the ratio of the outlet real and isentropic enthalpy difference against the outlet isentropic enthalpy change from stagnation to static conditions - hence outlet kinetic energy. Through isentropic relations it can be expressed as function of static and stagnation pressures, as:

$$\xi = \frac{h_2 - h_{2_{is}}}{(h_2^0 - h_2)_{is}} \quad \Rightarrow \quad \xi = 1 - \frac{1 - \left(\frac{P_2}{P_2^0}\right)^{\frac{\gamma-1}{\gamma}}}{1 - \left(\frac{P_2}{P_1^0}\right)^{\frac{\gamma-1}{\gamma}}} \tag{2.1}$$

The expression is obtained with the adiabatic field assumption: $(h_2^0)_{is} = (h_1^0)_{is}$.

In addition, Denton classifies the major loss categories in turbines by differentiating them on entropy-generation sources: **boundary layers** evolution on walls, due to viscosity shear stresses dissipating flow energy in near-wall regions; and **mixing processes**, unsteady phenomena that occur whenever two fluids with different momentum and vorticity encounter and exchange shear strain in order to balance their static pressure - cinematic constraint. An example of high shearing rates can be seen in wakes, separation vortices, leakage jets, and so on. As a consequence, viscous losses have historically been distinguished into **three** main families:

- **Profile loss** $\xi_{\textbf{profile}}$: generated by the interaction between boundary layers and blade surfaces, where the flow is bi-dimensional - hence far from endwalls - and typically account for **a third** of the total loss.

- **Endwall loss** $\xi_{\textbf{endwall}}$: often named as **secondary loss**, is related to the interaction between boundary layers and endwalls when passing through the blade passage. Since

this interaction also gives birth to the secondary flows, which then evolve from the passage to downstream - as further better explained -, it is still a **challenge** isolating the contribution of secondary flows three-dimensional complex evolution from the solely viscous effects of the boundary layers on the endwalls.

- **Tip leakage loss $\xi_{\mathbf{tip}}$**: a mixing loss that occurs in unshrouded blades, like rotors, where flow is driven in opposite direction of main flow due to pressure gradients: this produces vortices on blade tips, and consequent losses. Whereas, for shrouded blades, this loss can be accounted when leakages over seals occur. Even this loss source contributes to **a third** of the total loss.

A particular emphasis has to be placed on endwall losses, given their importance for this work objectives. As a consequence, it is essential to first expose the phenomenon of secondary flows in turbine cascades and the mechanisms that govern their generation, according to important findings in literature.

2.1 Mechanisms of Secondary Flows Formation

The research aimed to understand the physics of secondary flows started around the 1950s, at a time when three-dimensional viscous numerical analysis was not possible to perform. Therefore, the study of these complex flow phenomena relied exclusively on analytical approaches. In particular, inviscid and incompressible flow equations were employed to investigate the physics. Among the earliest models proposed, Hawthorne's study in 1955, [14], laid the basis for subsequent investigations in the field. He derived the vortex structures at the inlet and outlet of a cascade by applying vortex filament analysis to an inlet boundary layer profile, under the assumption that the flow laid within the bi-dimensional plane. Fig.2.1 shows that the orthogonal inlet vorticity distribution changes its orientation to stream-wise, after the flow entrains the curved cascade passage. At this stage, it is classified as secondary vorticity, and three main structures can be clearly distinguished at the cascade exit: **passage vorticity**, generated when the vortex filaments gets distorted when the flow transits within a curved passage; **trailing filament vorticity**, generated by the difference in Suction Side (SS) and Pressure Side (PS) velocities that stretches inlet vortexes; **trailing shed vorticity**, generated by the change of blade circulation in radial direction. As the sketch portrays, the most influencing structure is the Passage Vortex (PV).

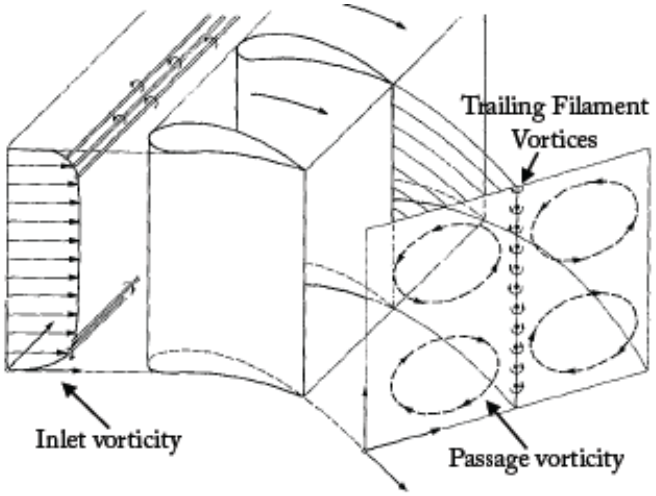
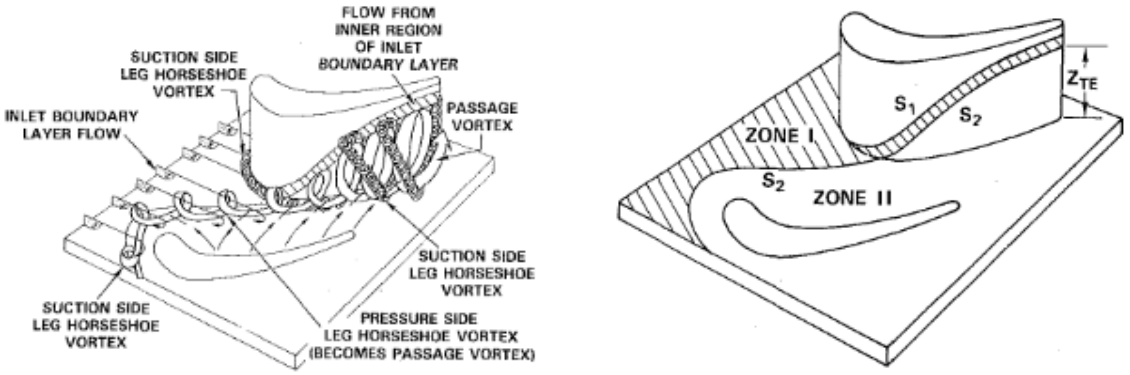


Figure 2.1: Hawthorne's cascade vorticity scheme, [14].

As previously stated, this model was recognised as the starting point for any following research proposed, along with Squire & Winter model. An important breakthrough that changed the perspective of the research was the observations made by Sieverding, [24]. In fact, he highlighted that prior approaches relied on balancing flow equation in a control volume, where flow quantities were evaluated only at the inlet and outlet. Whereas, also thanks to flow visualizations advances made in those years, it was outlined that a comprehensive investigation of secondary flows should have required the analysis of the interaction between the endwalls and the blades surfaces within the cascade passage. Building upon this reasoning, subsequent works -particularly from Langston et al. in the 1980s [16], which is among the most referenced- placed significant emphasis on the role of the Horse Shoe Vortex (HSV) into the understanding of secondary flows physics. In general, according to bluff bodies viscous fluid mechanics, the HSV is generated when an unskewed - bi-dimensional - boundary layer develops on a planar surface and encounters a bluff body, such as a cylinder, impinging on its front surface. The boundary layer separates just upstream the body due to an adverse pressure gradient imposed by the potential flow of the body. Here, the streamlines are suddenly decelerated, and subsequently deflected towards the endwall because of the strong downward pressure gradient that develops. These streamlines driven by viscosity then generate a vortical structure that rolls up the body surface, wrapping it around, resembling the shape of a horseshoe. This phenomenon occurs also in blade turbine cascades, but acquires a remarkable complexity. A comprehensive explanation of secondary flows was offered by Sharma & Butler in 1987, [23]. This study integrated flow visualization results available at that time with various models previously

published in literature, with some additional insights derived from authors' experiments and research activities. Therefore, given its comprehensiveness, it is reported in this work.



(a) Secondary flows topology sketch, [23] **(b)** Endwall separation zones sketch, [23]

Figure 2.2: Secondary flows representation, Sharma & Butler

Fig.2.2a clearly shows the inlet boundary layer that approaches the cascade blades. In the blades Leading Edge (LE) proximity, it separates forming the HSV. Here the authors state that the greatest part of the fluid particles are entrained into the vortex, which is divided into two legs: **pressure and suction side leg**. Whereas, the remaining particles very close to the endwall gets transported towards the suction side of the adjacent blade through convection. During the generation of the HSV, the normal vorticity to the inlet boundary layer gets transformed into the streamwise component. In addition, the two legs rotate in opposite directions. The pressure side leg when wrapping the blade surface from the LE, is decelerated, while the counterpart on SS gets accelerated. As a consequence, the low momentum fluid at PS results to be more susceptible to the pressure gradient inside the passage, from the blade PS to the adjacent blade SS. This transit creates a new boundary layer on the endwall within the passage. Therefore, the pressure side leg entrains low speed flow particles which take part of this boundary layer, and then it starts to grow becoming the **PV**. On the other side, the suction side vortex - counter-rotating - tends to remain "sticked" on the blade SS surface, accelerating, until it reaches a point of minimum static pressure. This point is also reached by the PV - which follows the blade-to-blade pressure gradient -, and eventually both PV and suction side leg vortex interact. Their interaction follow the path of the particles from the inner part of the inlet boundary layer that gets convected towards the SS, climbing up the blade surface, as aforementioned, which creates a separation line visible in Fig.2.2b between the zones S_1 and S_2. When

this complex system of vortexes reaches the blade Trailing Edge (TE), it comes to mixing with the TSV, becoming the so-called Counter Vortex (CV), because of the high velocity difference in span-wise direction.
Later in time, an additional insight was outlined by Denton & Pullan, [8]. In this numerical analysis aimed to investigate the losses generated in a turbine stage basing on entropy generation mechanisms, it was highlighted that secondary flows are an **inviscid** phenomenon itself. However, their interaction with endwall and blade surfaces contribute to generate entropy, hence losses. As a consequence, no single dominant source of endwall loss exists. The mechanisms that most influence loss are: dissipation of inlet boundary layer on the rear blade endwall surface and of the "new" boundary layer formed within the passage by PV; PV and HSV suction side leg interaction on the blade surface that dissipates energy; downstream mixing loss.
In addition, the secondary vorticity field previously exposed generates an induced secondary velocity field. The related kinetic energy is called Secondary Kinetic Energy (SKE), and it is essentially associated with the secondary flows:

$$SKE = \frac{V_{\mathbf{norm}}^2 + V_{\mathbf{z}}^2}{V_{2_{is}}^2} \tag{2.2}$$

where V_z is the span-wise velocity field, $V_{\mathbf{norm}}$ is the secondary velocity obtaining projecting the local velocity in parallel direction to the endwall

$$V_{\mathbf{norm}} = V_{tan} \cos\beta - V_{ax} \sin\beta$$

where β is the angle between the axial and primary flow direction $\beta = \tan^{-1} \frac{V_{tan}}{V_{ax}}$. As further explained in [8], SKE is used to evaluate secondary flows shear interaction. As secondary flows development in the cascade is an inviscid mechanism, its dissipation through viscosity can be linked to induced loss. Hence, if the SKE coefficient is evaluated by mass averaging SKE, it is noted that it starts to decrease from TE to downstream. In fact, this reduction directly corresponds to a raise in the stagnation pressure loss coefficient, hence accounting for loss. The main challenge remains evaluating how much of this SKE decrease is strictly due to secondary flows dissipation.

2.2 History of Research on Endwall Losses

What hitherto discussed underlines that a comprehensive understanding of secondary flows physics has been acquired after almost 70 years of thorough research. However, its crucial contribution when computing - and mainly predicting - losses has been clear since the

beginning.

Since the early stages of research, as previously mentioned, the first models based on secondary flow theory have been proposed. Among those, the one which would be then revisited is from Hawthorne and his vortex filament analysis, [14], from which computed the streamwise circulation of the streamwise vortex structures at the outlet of the control volume- PV, CV and TSV - and presented as vorticity **amplification factors** (AFs) - ratios of outlet streamwise against inlet boundary layer vorticities -

$$AF = \frac{\Gamma_{sec} / \left(\frac{V_2}{C_{ax}} \frac{h}{2} p \cos \alpha_2 \right)}{\bar{\omega}_1 / (V_1 / C_{ax})}$$

where Γ_{sec} represents the secondary - streamwise - circulation at cascade outlet. Each vortex structure contribution is expressed mainly as function of **non-dimensional transit time**:

$$\Delta T^* = T^*_{PS} - T^*_{SS} = \int_{PS} \left(\frac{V_2}{V_{fs}} \right) d \left(\frac{S}{C_{ax}} \right) - \int_{SS} \left(\frac{V_2}{V_{fs}} \right) d \left(\frac{S}{C_{ax}} \right) \tag{2.3}$$

where $_{fs}$ quantities are computed at the edge of the boundary layer. Hawthorne highlights that the orientation of downstream stream-wise vorticity depends on the difference in PS and SS velocities, hence considering how much time fluid particles take to "run" over the blade surfaces. In fact, when PS velocities are low, the analysis predicts high intensity secondary flows. This is logical since the lower the flow momentum on PS, the more it is susceptible to blade-to-blade pressure gradient, and therefore the PV would be more "feeded".

Some years later, in the latest 1970s, Marsh [18] revised Hawthorne amplification factors. He applied Kelvin's circulation theory - hence evaluating the effects of simple vortex singularities on the net velocity field -, but also added corrections by adding compressibility effects. The expressions of AFs obtained by Marsh, which will be further re-examined, are reported:

$$\mathrm{AF_{PV}} = M^* \left(\frac{V_1}{V_2} \right)^2 \left[\frac{\Delta T^* C_x}{p \cos \alpha_2} + \frac{|\frac{V_2}{V_1} \sin \alpha_1 - \sin \alpha_2|}{\cos \alpha_2} \right] \tag{2.4}$$

$$\mathrm{AF_{CV}} = - \left(\frac{V_1}{V_2} \right) \left(M^* \frac{V_1}{V_2} \frac{\Delta T^* C_x}{p \cos \alpha_2} + \frac{(M^* - 1)}{\cos \alpha_2} \left| \frac{V_2}{V_1} \sin \alpha_2 - \sin \alpha_2 \right| \right) \tag{2.5}$$

$$\mathrm{AF_{SHED}} = - \left(\frac{V_1}{V_2} \right) \left(\frac{M^*}{\cos \alpha_2} \left| \frac{V_1}{V_2} \sin \alpha_2 - \sin \alpha_1 \right| - \frac{(M^* - 1)}{\cos \alpha_2} \left| \frac{V_2}{V_1} \sin \alpha_2 - \sin \alpha_1 \right| \right) \tag{2.6}$$

where $M^* = \left(1 + \frac{\gamma - 1}{2} M_1 \right)^2$, and therefore

$$AF_{\mathbf{MARSH}} = \sum_i AF_i = AF_{\mathbf{CV}} + AF_{\mathbf{PV}} + AF_{\mathbf{TSV}} \tag{2.7}$$

Later in the years, although, theory-based models were gradually replaced by empirical and semi-empirical approaches based on experimental campaigns. These models were thought to achieve better agreement with experimental data, while often diverging from the physics of the problem itself. For example, Dunham & Came [11] or Craig & Cox [7] models developed in the 1970s, or even the one from Sharma & Butler [23], are among these, and still today some of them are still used. For example, Sharma & Butler delivered an empirical correlation to estimate the separation line penetration height Z_{TE}, as in Fig.2.2b, based on an extended turbine cascade data available at that time - for example from works of Sieverding, Lanston et al., *etc.*
It resulted in:

$$\frac{Z_{TE}}{C_{ax}} = 0.15\frac{\varepsilon}{\sqrt{CR}} + f\left(\frac{\delta_1}{h}\right) \tag{2.8}$$

where it is correlated with flow turning angle $\varepsilon = |\alpha_1 - \alpha_2|$, convergence ratio $CR = (\rho U)_{exit}/(\rho U)_{inlet}$ and a function of ratio between inlet boundary layer thickness δ_1 and the blade height h, expressed as

$$f\left(\frac{\delta_1}{h}\right) = 1.4\frac{\delta_1}{h} - 2.73\left(\frac{\delta_1}{h}\right)^2 + 1.77\left(\frac{\delta_1}{h}\right)^3$$

According to them, evaluating Z_{TE} was considered a good factor for the estimation of the magnitude of secondary flow.
Further in the years, shedloads of empirical models exhibited varying performance in design stages. In particular, the sensitivity of design parameters to predicted losses was yet not universally established. This inconsistency may be attributed to the sparsity of available data, which acted as bottleneck in developing a reliable model.

These issues have been highlighted by Coull, 2017 [4] work. In addition, the absence of coherent data led to lack of consistent trends in the available models, along with the fact that they were a result of empirical approaches, which often overlooked the underlying physics of the phenomena.
As a matter of fact, Coull conducts a parametric study across a set of design parameters -13-, through RANS CFD numerical analysis - see further section 3.1. He argues that this approach represents the best way to collect data and develop a low-order correlation model. Moreover, he proposes a breakdown of the measured losses - each evaluated with the main parameters sensitivity -, and incorporates a theory-based model - specifically that of Hawthorne and Marsh - to try to predict secondary flow-induced losses. In particular, when discussing about endwall losses, he remarks that it comes very difficult to isolate

their partial contribution to the whole loss. Therefore, he adheres to Denton point of view: evaluating entropy generation rate on the surfaces. Hence, he distinguishes the sources of endwall loss in: **background dissipation loss** - which accounts for the boundary layer interaction with walls and represents losses measured regardless of the presence of the blade in the passage - and **secondary flow-induced** loss.
The first component is evaluated with [9] model: from the entropy generation rate per unit area

$$\frac{d\dot{S}_{\text{surf}}}{dA} = C_D \frac{\rho_{fs} V_{fs}^3}{T_{fs}}$$

he derives the loss coefficient as a function of blade passage parameters:

$$\xi_{\mathbf{CD}} \approx 2C_D \left(\frac{A_{\text{endwall}}}{h \cdot p \cdot \cos \alpha_2} \right) \int \left(\frac{T_2}{T_{fs}} \right) \left(\frac{\rho_{fs}}{\rho_2} \right) \left(\frac{V_{fs}}{V_2} \right)^3 d \left(\frac{A}{A_{\text{endwall}}} \right) \tag{2.9}$$

where the dissipation coefficient is kept constant as $C_D \simeq 0.002$, and the quantities with $_{fs}$ refer to free-stream, at the edge of the boundary layer.
Whereas, the secondary flow-induced losses are then calculated by subtracting the background dissipation loss from the total endwall loss:

$$\xi_{\mathbf{sec-flow}} = \xi_{\mathbf{endwall}} - \xi_{\mathbf{CD}}$$

Afterwards, Coull investigates how secondary flows could be correlated through a relation justified by the physical mechanism of their formation, rather then only an empirical-data based one. As a matter of fact, he attains to Hawthorne vortex-filament mathematical analysis, relying on the vorticity amplification factors, strictly linked to flow angles, velocities, and mainly on non-dimensional transit time, according to Eqs. 2.5 to 2.6, hence 2.7. He finds out that secondary flow-induced loss correlates almost **linearly** with the sum of vorticity amplification factors with compressibility correction by Marsh, with the relation:

$$\xi_{\mathbf{MARSH-fit}} \simeq 0.0021 \cdot AF_{\mathbf{MARSH}} \tag{2.10}$$

Eventually, the predicted global endwall loss results to be:

$$\xi_{\mathbf{pred.-endwall}} = \xi_{\mathbf{CD}} + \xi_{\mathbf{MARSH-fit}}$$

performing quite well when compared to the measured endwall loss from numerical simulations - as kinetic energy loss-, as in Fig.2.3.

In last instance, another crucial aspect to analyse when talking about secondary flows is the sensitivity to inlet conditions, a concept already introduced by Sharma & Butler. In a further work, Coull & Clark, 2021 [5] assessed the turbine cascade endwall loss sensitivity

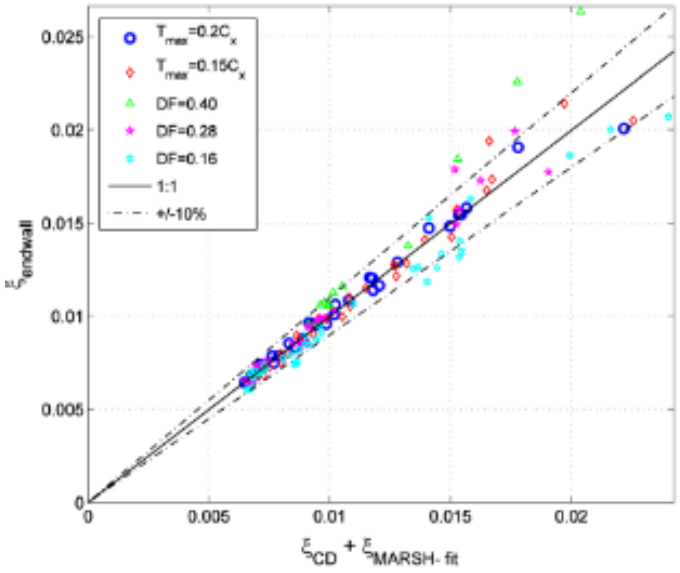


Figure 2.3: Predicted *vs.* measured endwall loss, model from Coull, 2017 [4].

to inlet conditions. In particular, the properties of the inlet boundary layer ingested by the cascade influence the topology of secondary flows. A combined influence of boundary layer thickness δ and the shape factor $H_{12} = \delta^*/\theta$, obtained with the integral boundary layer parameters:

- Displacement thickness δ^*:

$$\delta^* = \int_0^\delta \left(1 - \frac{u(x)}{U}\right) dx = \delta \int_0^1 \left(1 - \xi^{\frac{1}{n}}\right) d\xi$$

- Momentum thickness θ:

$$\theta = \int_0^\delta \frac{u(x)}{U}\left(1 - \frac{u(x)}{U}\right) dx = \delta \int_0^1 \xi^{\frac{1}{n}}\left(1 - \xi^{\frac{1}{n}}\right) d\xi$$

where $\xi = x/\delta$ is the normalised wall-normal coordinate. Particularly, H_{12} has most influence: higher shape factor boundary layers (laminar-like, $H_{12} \simeq 2.6$) produce very different secondary vorticity distributions if compared to turbulent profiles ($H_{12} \simeq 1.3$), due to the product of inlet vorticity and velocity $\omega_1 V_1$ peaks further from the endwall, hence the maximum vorticity generated at the outlet results to be further towards midspan - recalling Sharma & Butler Z_{TE}. Furthermore, it is shown that high shape factors lead to increase of SKE values by a factor of $2-3$ if compared to turbulent profile, maintaining the same thickness. This translates in up to **doubled mixed-out endwall loss** measured, when switching from a turbulent to a laminar-like profile, as in Fig.2.4. This result is

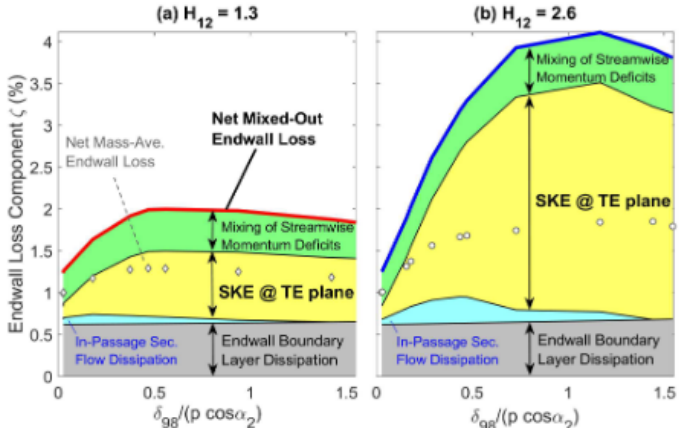


Figure 2.4: Endwall loss breakdown with different inlet boundary layer shape factors, varying normalised thickness, [5].

also confirmed by the losses breakdown proposed by de la Rosa Blanco et al., [12]. This conclusions are delivered through comparison of numerical data and the employment of Rankine vortex theory, evaluating the effects of a vortex of intensity Γ and diameter D on the obtained tangential velocity field when it founds to be at a distance d from the wall. Coull & Clark analysis, eventually, focus on the sensitivity of vorticity distribution due to secondary velocity, when comes to evaluating the behaviour of SKE with inlet conditions. Theory imposes that the secondary circulation Γ_{sec}, computed with Rankine theory from the secondary vorticity obtained from Hawthorne:

$$\omega_{sec} = \frac{\omega_1}{\cos\alpha_2}\left(\frac{V_1}{V_2}\frac{C_{ax}}{p}\Delta T^* + |\sin\alpha_1 - \frac{V_1}{V_2}\sin\alpha_2|\right)$$

where ω_1 is the inlet boundary layer vorticity, p the cascade pitch, C_{ax} the axial chord. Hence:

$$\frac{\Gamma_{sec}}{V_2 \cdot p\cos\alpha_z} = U^* \cdot \frac{\Delta T^* C_{ax}}{p\cos\alpha_2} + |\frac{V_1\sin\alpha_1}{v_2\cos\alpha_2} - U^*\tan\alpha_2|$$

, where $U^* = \left(1 - \sqrt{1-(V_1/V_2)^2}\right)$. When evaluating different blade geometries, the normalised secondary circulation varies since dependent on ΔT^*: higher turning angles increase cross-passage pressure gradient, since lower PS velocities, hence SKE increases. But, it remains constant with inlet conditions. Since real data obtained from numerical analysis behaves differently from theory, a parameter is introduced in order to evaluate **only** secondary vorticity distribution due to SKE:

$$\Pi_{SKE} = \frac{\zeta_{SKE}\left(\frac{h}{2}p\cos\alpha_2\right)}{\Gamma_{sec}^2/V_2^2}$$

ζ_{SKE} is SKE coefficient, $\frac{h}{p \cdot \cos \alpha_2}$ represents the ratio between the channel height and the effective passage width, $\frac{\Gamma_{sec}^2}{U_2^2}$ is the square of the normalized secondary circulation. It represents a non-dimensional parameter that characterises the effects of vorticity distribution on the SKE field generated. in other words, it quantifies how a certain secondary vorticity distribution generates SKE: high values of Π_{SKE} stands for vorticity distributions that maximise the SKE field, and low values vice-versa. Fig.2.5 shows the relation between the non-dimensional vorticity distribution and the boundary layer thickness normalised by the orthogonal cascade passage area $\delta_{98}/p \cos \alpha_2$, and parametrised with different boundary layer characteristics, H_{12} or Blasius power law exponents n. It identifies three flow regimes: **Thin Boundary Layer Regime** ($\delta_{98}/p \cdot \cos \alpha_2 <\sim 0.1$): Relatively low Π_{SKE} values. Vorticity concentrated near the endwall; strong vorticity cancellation effect due to "mirror effect" - Rankine theory; minor span-wise penetration of secondary flows. **Buffer Regime** ($\delta_{98}/p \cdot \cos \alpha_2 \approx$ 0.1-0.5): Transition zone with rapid increase in Π_{SKE}; optimal balance between vorticity strength and its distribution. **Thick Boundary Layer Regime** ($\delta_{98}/p \cdot \cos \alpha_2 >\sim 0.5$): Π_{SKE} reaches a maximum and then begins to decrease. Vorticity more dispersed throughout the passage with larger vortex structures but with reduced intensity, beneficial for secondary flows development in the passage, hence losses.

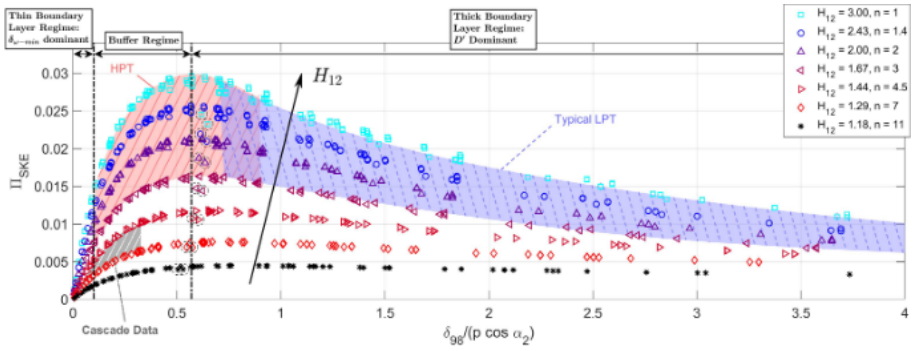


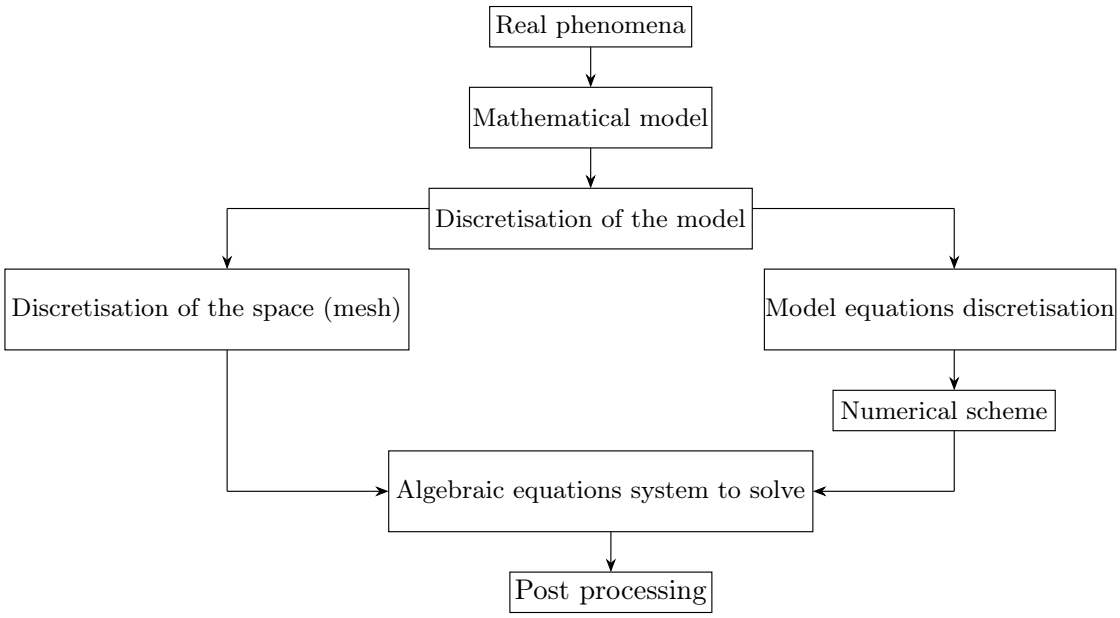
Figure 2.5: Non-dimensional vorticity distribution in relation to flow regimes, parametrised with inlet boundary layer properties, from secondary flow theory, [5].

Chapter 3

Tool development

As forementioned, the aim of this tool is to robustly automate the 3D numerical analysis of a selected number of optimised blade profiles and to provide an estimation of the losses. This could help the designer in estimating principal parameters responsible for secondary-flow loss of energy, from loading of a blade and proceeding to the impact of an inlet boundary layer. The tool development phases, which are consistent with the actual workflow sequence, will be the focus of this chapter.

3.1 The role of Computational Fluid Dynamics



3. TOOL DEVELOPMENT

Computational Fluid Dynamics (CFD) is a powerful means of analysis and prediction of a fluid flow behaviour in motion, driven by its governing equations. To accomplish this, their numerical solution is required. The way that each problem is modelled and how this model is treated numerically gives birth to a certain type of CFD "technique".
The foundation of CFD, whereas, is the Navier-Stokes equations, which analytical solution is still unknown: here comes the utility of solving them numerically. For a generic fluid, with no hypothesis involved regarding its thermodynamics or its behaviour in space and time, they state that every fluid particle, in a confined domain and in a defined time frame, must obey to **three conservation laws**:

- **Mass**:
$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{V}) = 0$$
- **Momentum**:
$$\frac{\partial (\rho \mathbf{V})}{\partial t} + \nabla \cdot (\rho \mathbf{V} \otimes \mathbf{V}) = -\nabla p + \nabla \cdot \tau + \rho \mathbf{g}$$
- **Energy**:
$$\frac{\partial E}{\partial t} + \nabla \cdot [(E + p)\mathbf{V}] = \nabla \cdot (k \nabla T) + \Phi$$

where ρ is the density, $\mathbf{V}$ is the velocity vector, p is the pressure, $\mathbf{g}$ is the gravitational acceleration, E is the total energy per unit volume, k is the thermal conductivity, T is the temperature, and ϕ represents a generic expression of the dissipation function and τ represents the shear stress tensor: its components are computed following the Boussinesq approach, adopting the Stokes hypothesis for the bulk viscosity - its components are found to be: $\tau_{ij} = 2\mu \left[S_{ij} - \frac{1}{3} \frac{\partial u_k}{\partial x_k} \delta_{ij} \right]$, following Einstein notation.

Specifically, $S_{ij} = \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right)$ represents the strain-rate tensor. Even though diving deeper into their fascinating physical and mathematical background is of great interest, this lies beyond the scope of this thesis.
The challenge, therefore, lies in selecting an appropriate modelling approach: the simplifying assumptions, their mathematical expression and the numerical methods for the resolution of the highly non-linear system of Partial Difference Equations (PDE). As a matter of fact, an inherent error ε_{tot} lies between a CFD solution of a certain flow field and its actual behaviour in real life. Each step that drives towards the numerical solution, introduces a corresponding fraction of error: the modelling error ε_{model}; the discretisation (or truncation) error $\varepsilon_{\Delta x, \Delta t}$ that depends on how space and time are discretized from the real continuous "world": its order of magnitude is comparable to the order of Taylor

expansion truncation; the grid error ε_{grid}, eventually, represents the source of inaccuracy introduced by the physical domain discretisation: the mesh.
In addition, the higher complexity regards the turbulence modelling. Turbulence, indeed, portrays a fundamental characteristic inherent to fluid flows and it manifests through statistical and chaotic fluctuations of the flow quantities, hence almost impossible to predict through a mathematical relation. In fact, above a critical Reynolds number, flow state naturally transitions from laminar to turbulent. Historically, turbulent phenomena are associated to scales, namely the average dimension of the flow structures involved, such as vortices and eddies, since only an empirical analysis could be employed. Kolmogorov's energy scale, in particular, describes the hierarchical transfer of kinetic energy from larger to smaller eddies due to energy dissipation effects, ultimately leading to Kolmogorov micro-scales, where viscous effects are dominant.
Therefore, this multi-scale approach influences the treatment to numerical modelling of fluids. In particular, the littler the scales to catch, the higher the complexity of the simulation. According to Kolmogorov's theory, the ratio between the largest and smallest turbulent scales is comparable to $Re^{\frac{3}{4}}$. This clearly affects the spatial discretisation: in order to capture the smallest dissipative scales, the average mesh cell dimension must be comparable to it, in all the three dimension, hence $\approx Re^{\frac{9}{4}}$. In addition, when solving the equations, the temporal resolution must decrease accordingly to spatial one to mantain numerical stability ensured by the Courant-Friedrichs-Lewy (CFL) condition.
Ergo, different computational approaches are available based on the desired turbulent scales resolution. Direct Numerical Simulation (DNS) resolves every energetic scale from the large eddies to the Kolmogorov's micro-scales requiring no assumptions, hence no modelling, but being the most computationally prohibitive, especially when dealing with high Re flows. Large Eddy Simulation (LES) resolves every scale larger than a grid depth filter Δ - that can be a value rather than an adaptive function of the flow field - and modelling any smaller energy scale, offering a fair trade-off between accuracy and computational cost. For design-oriented problems, where minimizing computational demand is crucial, modelling every turbulent scale through closure equations, to add at the PDE system, can also deliver a good trade-off with results obtained.

3.2 RANS: Reynolds-Averaged Navier-Stokes

This is the case of the Reynolds-Averaged Navier-Stokes (RANS) approach: time-averaging the equations so that any flow quantity fluctuation in time is cut off. Closure equations are

thought to replace those phenomena. The RANS approach theory is based on the arbitrary decoupling of a quantity as a sum of its averaged value and the fluctuation value. Given A a flow quantity:

$$A(x,t) = \bar{A}(x,t) + A'(x,t) \tag{3.1}$$

where, dealing with statistically stationary problems:

$$\bar{A}(x,t) = \frac{1}{T_f}\int_{-T_f/2}^{T_f/2} A(x,t+\tau)d\tau \tag{3.2}$$

For compressible flows, where also density is decoupled, this way equations get more complicated to treat. Therefore, Favre averaging helps to deal with great variations in density:

$$\tilde{A}_i(x) = \frac{1}{\rho}\lim_{T\to\infty}\int_t^{t+T} \rho A(x,t)dt \tag{3.3}$$

When comes to averaging momentum and energy equations, a new term comes out and needs to be modelled: the so-called Reynolds stress tensor $\tau_{ij} = -\rho\overline{u_i' u_j'} = -\rho(\overline{u_i u_j} - \bar{u}_i\bar{u}_j)$, which takes into account small scale fluctuations behaviour. It is a symmetric matrix, and its trace $tr(\tau_{ij})$ is linked to the fluctuations kinetic energy. A strong approximation in this approach occurs when modelling it through the *linear Boussinesq model* [2]. Here, the approximation stands in the introduction of the *turbulent viscosity* μ_t. Since viscous effects are driven by molecular diffusion, which consequently depends on the average particle velocity, Boussinesq leveraged this concept in order to link the turbulent energetic scales with the mean flow field:

$$\tau_{ij} = 2\mu_t\tilde{S}_{ij} - \frac{2}{3}\mu_t\frac{\partial\tilde{u}_k}{\partial x_k}\delta_{ij} - \frac{2}{3}\bar{\rho}k\delta_{ij} \tag{3.4}$$

written in Einstein notation. The stress tensor is splitted into an isotropic and anisotropic contribution.

Therefore, the introduction of the closure equation 3.4 adds one more variable when comes to the solution of the system of PDEs: μ_t itself. As a consequence, an infinite literature is available for choosing it, identifying a turbulence model. It is possible to classify them into: algebraic models (*i.e.* Baldwin-Lomax), one-equation models (*i.e.* Spalart-Allmaras) or two-equation models (*i.e.* k-ω, k-ε, Menter SST, etc.). It has to be highlighted that any turbulence model is conceived starting by a strong empirical hypothesis, since their expressions are often filled with coefficients. That is to say, physical application has its own suitable model.

That being stated, the resolution of NS equations, most commonly for RANS models, nowadays is relegated to commercial solvers. In these cases, through optimisation of the

computing processes, the setting up of GUIs (Graphical User Interface), the elevated integration with CAD softwares, these softwares enhance the CFD workflow providing a streamlined workflow and eliminating codes manipulations.

3.3 NUMECA *FINE^{TM}/Turbo*, *IGG^{TM}* and *Python* environment

The tool has been developed in Python, a high-level programming language known for its intuitive syntax, but mostly for its extensive library ecosystem that allows to implement algorithms that perform whatever wanted task, in particular handling data very easily. Moreover, it is highly integrated in NUMECA Cadence commercial softwares suite: it allows the user to execute the same tasks that can be made through the GUI, but through tailor-made scripts. Hence, it is highly arranged to automated processes, where no user interaction is needed.

Solving a CFD problem requires three fundamental steps: *meshing*, meaning to discretise the space of the flow domain into cells; *solving* numerically the Navier Stokes equations through a solver, from which depends the nature of the numerical solution of the flow obtained; *post processing* the data, which means to extract all the aero-thermodynamic data of interest to be processed or just visualised.

In order to perform these tasks, NUMECA provides a specialised software for each of them. The first, *IGG^{TM}* - *Interactive Grid Generator* - is "an Interactive Geometry modeler and Grid generation system for multiblock structured grids" [30] that, among its features, provides an in-sight CAD modeler which can handle and edit external CAD geometries and, of course, a powerful meshing tool that creates multi-block structured meshes that most fit the geometry - with algebraic or elliptic smoothing algorithms. Furthermore, in the design point of view, it provides an automatic and fast mesh generator, *AutoGrid5^{TM}*, that allows the user to create the desired grid by touching the desired parameters, but ensuring quality control, because a bad-quality mesh can produce inexact results or fatalities during the computation. In the tool framework, the latter will be used. Then, the the following is *FINE^{TM}/Turbo* solver, a "state of the art 3D multiblock finite volume flow solver" [29] capable of solving both inviscid flows, governed by the Euler equations, and viscous flows, described by the Navier-Stokes equations. It enables the simulation of both internal and external flow fields, with all relevant settings and characteristics integrated within a single project. Furthermore, it supports computation in *batch* mode, allowing simulations to be executed without user interaction via the graphical interface, thereby enhancing its

compatibility with automated workflows. Lastly, with the purpose of analysing results, the third in-house software is *CFView*^{TM} which particularly embodies the automation adaptibility. In fact, it makes quantities analysis very manageable thanks to execution of user-made Python scripts.

In order to gain a more comprehensive understanding of the tool framework development, it is beneficial to focus on the file formats that these softwares handle.

- *IGG*^{TM} : The CAD blade extension selected for this project is *.IGES*, although many other formats are also supported. Geometry files, which store the coordinates of curves and surfaces, typically have either a *.geom* or *.dat* extension. For geometry generation, the *.geomTurbo* file is particularly essential, as it is the only geometry format recognised by *AutoGrid5*.

  The final mesh consists of several key files:

  - *.igg* – the primary mesh file,
  - *.bcs* – defines boundary conditions for each patch,
  - *.trb* – a template file containing all parameters that define mesh characteristics,
  - *.qualityReport* – provides a detailed assessment of mesh quality indices.

- *FINE*^{TM}*/Turbo*: the main project file has *.iec* extension: contains any kind of useful information of the computation, comprising mesh file, boundary conditions, numerical models, etc. A project file can have many computations related to it, each of them with their subdirectory. where result files are present. Among them, those to be highlighted are: *.run* file, containing all computation parameters readable by the solver and *CFView*^{TM}, but also computation management information for the machine itself - *i.e.* sequential or parallelised process, number of processors to be involved *etc.* - ; *.cgns* file, that contains the whole solution - so it is the one that occupies most memory on the machine -, *.res* and *.std* files that keep track of the solution convergence history; and the *.batch* file, which is a batch script that launches the computation in the directory

- *CFView*^{TM}: for the interest of the project, in order to visualise the computation, only the *.run* file is needed. Apart from this, Python scripts will be essential for data extraction, so *.py* extension files, so called *MACROs*.

3.4 Overview

In order not to confuse the reader with the development of this tool codes and detailed approaches, and to maintain all of the steps involved in clarity, this chapter will first discuss a step-by-step development process - to get an 'horizontal' vision of the workflow - and then cover the actual functionalities of the tool: the automation of processes and data handling. The first part can be arranged in the following key points:

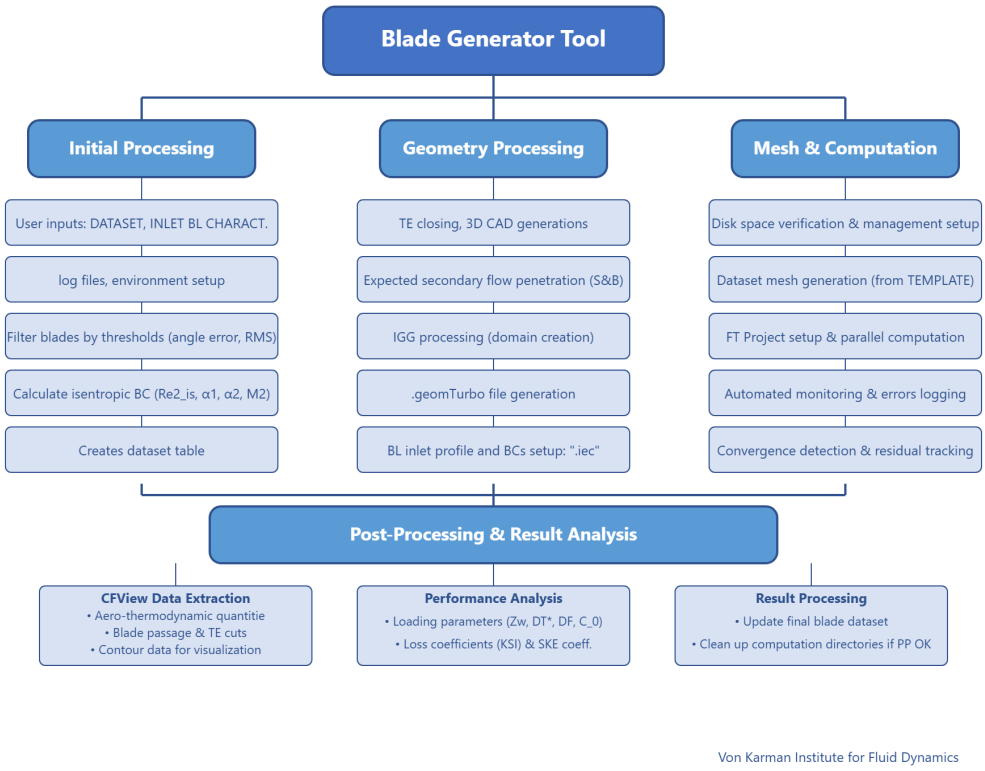


Figure 3.1: Tool structure

- Processing of the 2D blade geometry file, which involves the problem of closing its Trailing Edge and the creation of the CAD file, through a specific Python script.
- Definition of the computational domain and generation of *.geom* and *.geomTurbo* files in IGG^{TM} for seamless import into $AutoGrid5^{TM}$: mesh generation, specifying approaches and methodologies consulting literature.
- The CFD methodologies and workflow are then discussed in detail: from the boundary conditions set by user, to the numerical schemes and turbulence models, ensuring stability and robustness.

- Exploration of the potential extreme test cases for secondary flows formation, given their sensitivity to both blade turning and inlet boundary layer: two representatives are selected for a mesh independance study, and finally leading to the definition of a suitable mesh, subsequently used as template for the whole dataset of blades generated. A trade-off between computational time and solution convergence is then evaluated.

- Demonstrating the entire automated process: the pre-processing of the entire set of input geometries, the computing of boundary conditions based on user input parameters, the mesh generation of the entire dataset, the creation of project files, the launch of computations, real-time monitoring, error logging, post-processing, and data handling.

3.5 Geometry and CAD

The first feature of the tool is the 3D blade geometry generation, as forementioned. As input, an already optimised blade profile (x, y) set of coordinates is given to the code. This profile is the result of a distinct blade generation tool that is intended to generate geometries that satisfy user-defined criteria, specifically *aerodynamic duty* — such as inlet and outlet flow angles $\alpha_{1,2}$, Reynolds number $Re_{2_{is}}$, and outlet Mach number M_2 — under specified operating conditions. Additionally, the tool allows for customisation of the blade *aerodynamic style* by defining the loading through a non-dimensional Mach number distribution on the pressure and suction sides, controlled via four key parameters, as in Figure 3.2. To achieve this, a deep learning model has been implemented [21].

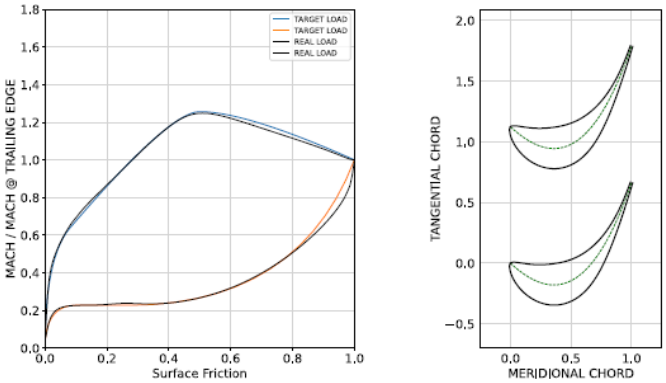


Figure 3.2: Blade loading - target vs real; blade profiles

The real loading is computed through a 2D CFD simulation through MISES, which outputs, in particular, the values of α_2, M_2 and profile loss coefficient ξ computed given $Re_{1_{ax}}$ and M_1. The value of the loss does not include trailing edge loss: in fact, in order to impose the Kutta condition at the TE, it is left arbitrarily **open**, as the sketch in Figure 3.3.

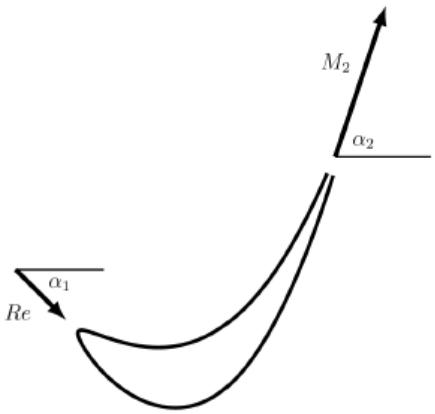


Figure 3.3: Optimised blade airfoil sketch

Hence, the first task of tool's *blade_processor.py* is to appropriately close the trailing edge.

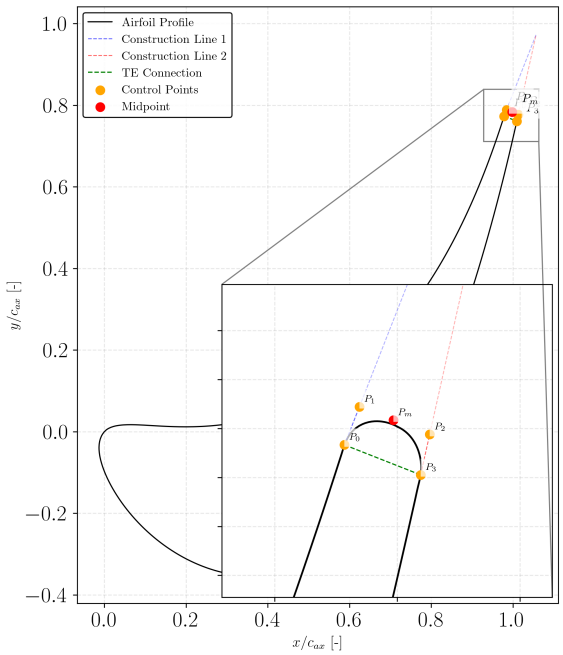


Figure 3.4: Trailing edge: NURBS curve with control points

This means to shape it in a way that profile curvature must be kept constant, otherwise spurious oscillations in the solution flow field would occur in the edges proximity, hence where the curvature function is discontinuous. Specifically, curvature is defined as $\kappa(s) = \frac{d\theta}{ds}$ - where s is the curvilinear coordinate along the airfoil and $\theta(s)$ is the angle between the tangent line and the abscissa x. In order to avoid edges, a C^2 continuity condition has to be established [1]: when building the trailing edge closing curve, both $\kappa(s)|_{s=s_{P_0,P_3}}$ and $\frac{d\kappa(s)}{ds} = \frac{d^2\theta(s)}{d^2s}\Big|_{s=s_{P_0,P_3}}$ have to be constant.

This is done through a geometric enforcement. Considered P_0 and P_3 as the opened airfoil endpoints: the two lines tangent to the profile, intersecting them, are tracked and kept as construction lines - blue and red dashed lines in Figure 3.4, shaping the so called wedge angle. Given the conjunction line between P_0 and P_3, $\overline{P_0P_3}$, it is than translated parallelwise of a quantity $d = k \cdot \overline{P_0P_3}$, the displacement parameter, expressed proportionally to the axial chord C_{ax}. The intersection with it and the construction lines yields the points P_1 and P_2.
These will become the NURBS - *Non-Uniform Rational B-Splines* - control points with the airfoil endpoints. This is implemented through a NURBS curve construction that ensures geometric continuity. The key function `compute_intersection_point` calculates the control points necessary for a smooth TE closure:

$$\begin{aligned}
&\mathbf{P}_0 : \text{Starting point of the open airfoil} \\
&\mathbf{P}_3 : \text{Ending point of the open airfoil} \\
&\mathbf{v}_r : \text{Tangent vector at } \mathbf{P}_0 \\
&\mathbf{v}_s : \text{Tangent vector at } \mathbf{P}_3 \\
&d : \text{Distance parameter (typically } 0.5 \cdot \|\mathbf{P}_3 - \mathbf{P}_0\| = f(C_{ax}))
\end{aligned} \tag{3.5}$$

The algorithm proceeds as follows:

1. First, P_{mid} is the midpoint of the segment $\overline{P_0P_3}$:

$$P_{mid} = \frac{P_0 + P_3}{2} \tag{3.6}$$

2. Computing the normalised vector of $\overline{P_0P_3}$:

$$\mathbf{v}_{norm} = \frac{P_3 - P_0}{\|P_3 - P_0\|} \tag{3.7}$$

3. The normal direction $\mathbf{n}$ to $\overline{P_0P_3}$:

$$\mathbf{n} = \frac{\mathbf{v}_{norm} \times (\mathbf{v}_{norm} \times \mathbf{v}_r)}{\|\mathbf{v}_{norm} \times (\mathbf{v}_{norm} \times \mathbf{v}_r)\|} \tag{3.8}$$

4. The control point P_m is located at distance d from P_{mid} along $\mathbf{n}$:

$$P_m = P_{mid} + d\mathbf{n} \tag{3.9}$$

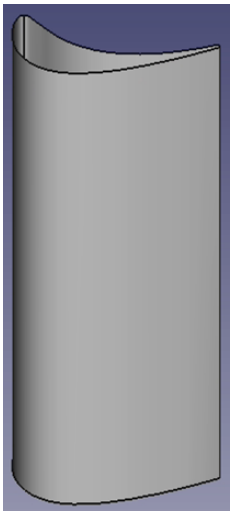
Eventually obtaining P_1 and P_2 through intersection of them:

$$P_1 = P_0 + \left(\frac{(P_m - P_0) \cdot \mathbf{v}_r}{\|\mathbf{v}_r\|^2} \right) \mathbf{v}_r \tag{3.10}$$

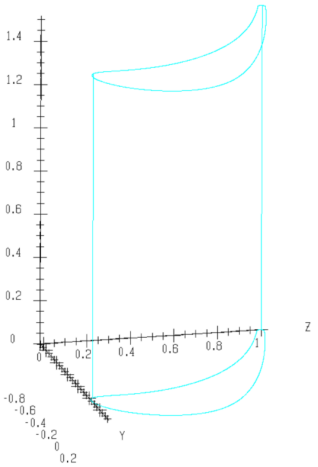
$$P_2 = P_3 + \left(\frac{(P_m - P_3) \cdot \mathbf{v}_s}{\|\mathbf{v}_s\|^2} \right) \mathbf{v}_s \tag{3.11}$$

Finally a NURBS curve is built with `compute_nurbs_curve`, which can be given a vector of $n+1$ weights that work as "attractors" for the curve. For this simple circular arc case, a `weights=None` condition is given. An optional function for curvature computation is implemented, adhering to the mathematical formulation proposed by Agromayor et al. in their appendix [1]: $\kappa(s) = \frac{|\mathbf{T}'(s) \times \mathbf{T}''(s)|}{|\mathbf{T}'(s)|^3}$. The curvatureis computed using the cross product of the first and second derivatives of the tangent vector $\mathbf{T}$, normalised by the cube of the tangent vector magnitude.

Once the airfoil is closed, it is first scaled on both (x, y) ensuring unitary C_{ax}. subsequently, after having splitted the blade coordinates into (PS) and (SS) an interpolation of the points has been done with BSplines - through `create_SS_PS`. Once the airfoil is checked and correctly closed, it gets extruded linearly of a quantity h as user input through `create_extruded_spline`, which creates the LE and TE conjunction lines and the SS and PS surfaces through *BRepPrimAPI_MakePrism* operation. Eventually, an *.IGES* CAD file is created with `export_blade_to_iges`, and saved in the output directory of *.IGES* files. In order to accomplish of these tasks, especially the last ones, without interacting with a CAD software, a Python library has been employed: *Open CASCADE*, [31].



(a) Output *.IGES* visualised in *FreeCAD*.



(b) Screenshot of the *.IGES* file imported into IGG^{TM}.

Figure 3.5: Blade *.IGES* file output.

The features explained thus far are executed specifically by `blade_convert_IGES_v2.1.py`. The script is loaded inside `blade_processor.py`, which is a class that implements this file handling and blade conversion robustly providing detailed error reporting and logging. Actually, it loads dynamically the aforementioned script after checking directories and using the blade number that was obtained from an input file. The variables in output are handled through dynamic parameter injection - *kwargs*.

Once generated the CAD file with the correct extension handled by IGG^{TM}, the following step consists of creating the domain extension lines, **hub** and **shroud**, and it is operated by `igg_processor.py`. Their definition is necessary to characterise the blade row, which is confined by the inlet and outlet planes. For this purpose, this task requires to open the IGG^{TM} interface and to add the two lines by first creating the hub line, connecting the LE and TE points and performing line extension both for inlet and outlet, in terms of distance from LE and TE respectively.

In order to clarify the reader's comprehension, it is useful to establish a reference system. Hence, the blade domain is defined in cartesian coordinates as (x, y, z) - since no rotary component is involved -, referring to span-wise, pitch-wise and stream-wise directions, respectively, conventionally to IGG^{TM}.

Since the software needs the hub and shroud lines to intersect correctly the blade, it is required them to be shifted slightly span-wisely. Then, the same approach is done for the

shroud line, from a distance h with respect to the hub. A macro Python script has been created to perform this in IGG^{TM} environment: `create_domain_extension_lines.py`.

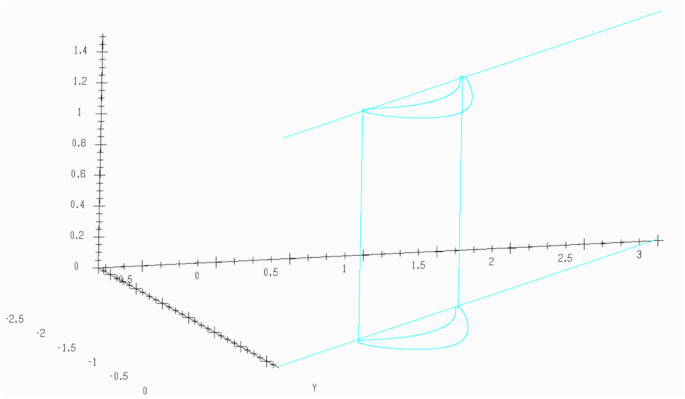
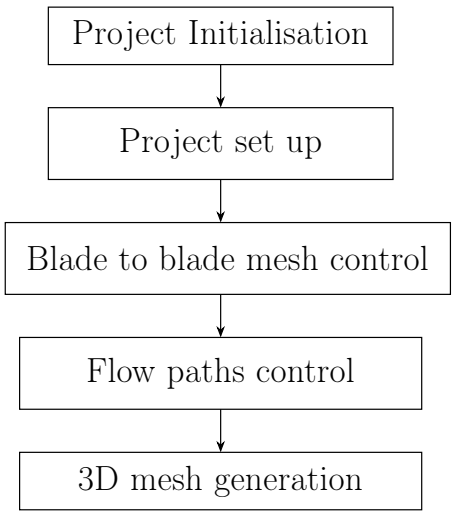


Figure 3.6: Blade with domain extension lines

After saving the new CAD file into *.igg* extension, everything is set up for the mesh generation in $AutoGrid5^{TM}$ environment. To clarify for the reader, this procedure was performed during the training phase. However, in the actual implementation of process automation, it serves as the basis for generating a *.geomTurbo* file. The details will be reported in the next sections.

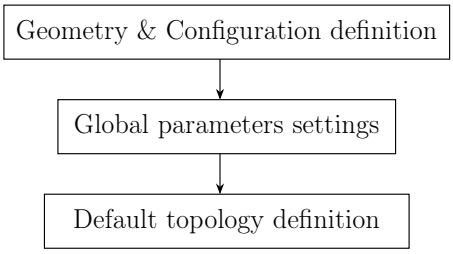
3.6 Mesh



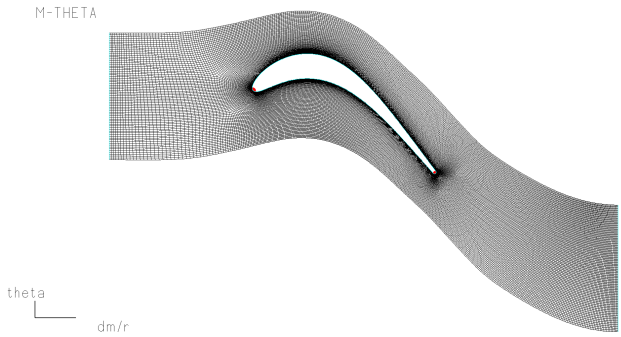
The fundamentals of the mesh generation, especially for turbomachinery components in this environment, are essentaially five. Additional steps can be added if the designer

is interested in comprising technological effects - i.e. seal leakages, purge flows, cooling effects, etc., as suggests the user manual [28]. For the purpose of the project, a **standard turbine single-bladed row of a linear cascade configuration** is considered.
The meshing phase starts with the **project initialisation**, namely the geometry definition. There are two different ways: by importing geometry from *.geomTurbo* file, the in-house software own way of reading and recognising the geometry given, or by initialising the project through an external CAD file. During the initial phase of the project, the latter approach was handled. Whereas, for the tool implementation, the first was adopted.

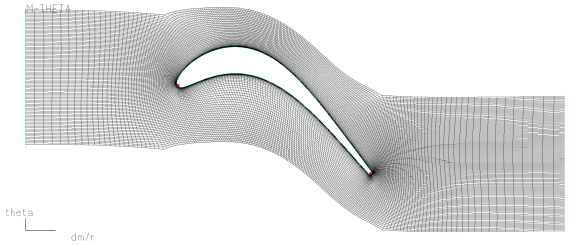
Subsequently, the mesh project set up can therefore be organised into three steps:



The geometry & configuration allows the creation of the *.geomTurbo* file starting from external CAD through *Import and Link CAD* functionality. In particular, it entails the import of the domain file like in Figure 3.6, and the manual assignment to the row features, such as: hub, shroud, LE, TE lines and the blade surface. Once this task is completed, the user can choose between generate the mesh between creating it manually by explicitly specifying all parameters or by appealing to the *Row wizard*, a feature that simplifies the process of generating the structured grid by optimising all of its features based on the user selected parameters. It is also important to consider that in this case optimising means to obtain a certain grid with desired shape and features and concurrently accelerating the generation time process - since it mainly consists in solving elliptic partial differential equations. While creating a new mesh from scratch confers more freedom in choosing nodes distribution, attributes and optimal shapes, thus resulting in a higher quality grid, it can also be highly time consuming. This is fundamental in the automation process point of view.



(a) Blade-to-blade view of manually created mesh: generation time 00:49:58 s



(b) Blade-to-blade *Row wizard* mesh: generation time 00:04:14 s

Figure 3.7: Example: SPLEEN C1 High-Speed turbine cascade blade [25]

In first instance, a *Geometry check* is performed, to ensure that no open curves and correct intersections are present. Consequently, the user is required to specify the machine characteristics, whether the blade row is a wind turbine, axial turbine, Francis turbine, Kaplan turbine, compressor, *etc.* For the scope of the project, axial turbine has been selected. In addition, the value of the pitch p - or periodicity - has to be specified: this is one of the inputs besides blade profiles coordinates, but normalised with axial chord, p/C_{ax}. There is possibility of adding fillets or gaps to the extremities of the blade. However, this step was omitted for sake of simplicity. The following stage of the wizard control regards the choice of the number of flow paths, or span points, and the width of the first cell at the wall. The first represents the three-dimension resolution of the grid, while the latter allows the user to superimpose a certain cell size for every surface/wall first layer within the domain. In most CFD applications, the user must generate a mesh that helps the solver to provide a solution compatible with reality, which necessitates not just spatial resolution within the domain but also, peculiarly, wall resolution since the existence of boundary layers. As explained in the turbulence model section afterwards, in order to reach a certain value of normalised distance from wall y^+ - which normally is linked to the optimal value for its turbulence closure model -, there is the need of adding and/or shaping

the so-called prism layers in the walls proximities. Whereas, *AutoGrid5*^{TM} only needs the user to establish the width of the first wall cell, from which it computes the optimal grid nodes distribution that satisfy important mesh quality requirements, such as:

- **Aspect ratio**:

$$AR = \frac{\max(x, y)}{\min(x, y)}, \quad \text{where} \quad x = \frac{a + b}{2}, \quad y = \frac{c + d}{2}$$

  (see Fig. 3.8a)

- **Expansion ratio**:

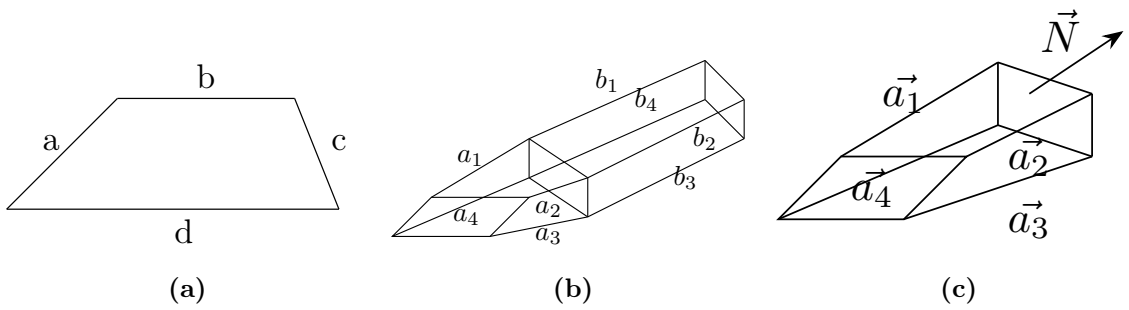
$$ER = \frac{\max(x, y)}{\min(x, y)}, \quad \text{where} \quad x = \frac{a_1 + a_2 + a_3 + a_4}{4}, \quad y = \frac{b_1 + b_2 + b_3 + b_4}{4}$$

  (see Fig. 3.8b)

- **Orthogonality**:

$$\langle \vec{x}, \vec{N} \rangle, \quad \text{where} \quad \vec{x} = \frac{\vec{a_1} + \vec{a_2} + \vec{a_3} + \vec{a_4}}{4}$$

  and $\vec{N}$ is the normal vector to the block boundary face (see Fig. 3.8c)



(a) **(b)** **(c)**

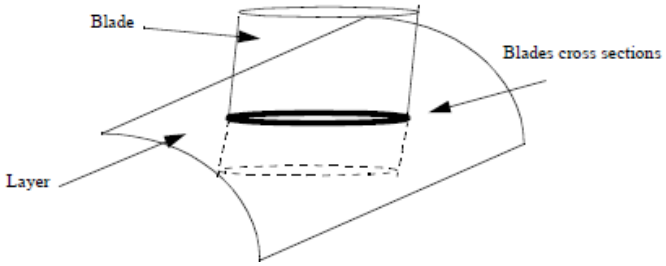
Figure 3.8: Mesh quality metrics

In order to guarantee solution convergence in the CFD solver and avoid numerical diffusion, these quality parameters are essential. Hence, in order to obtained the desired value of y^+, the user manual provides the relation to compute the first cell wall distance y_{wall} through a truncated series solution of Blasius equation ([29]):

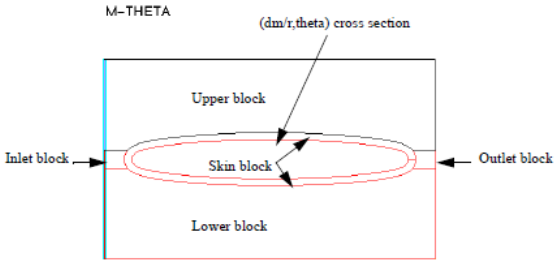
$$y_{\text{wall}} = 6 \left(\frac{V_{\text{ref}}}{\nu} \right)^{-\frac{7}{8}} \left(\frac{L_{\text{ref}}}{2} \right)^{\frac{1}{8}} y^+ \tag{3.12}$$

where V_{ref}, L_{ref} and ν are the reference velocity and length - in this case the inlet free-stream velocity V_1 and the axial chord C_{ax} -, and the local kinematic viscosity, respectively.

Once defined these two important aspects, the last step of the "wizard" requires the *Blade-to-blade* mesh definition, namely the result of cutting the blade domain with an (y, z) plane - or (m, θ) in cylindrical coordinates. (see Fig. 3.13a).



(a) Plane cut sketch



(b) Blade-to-blade blocks layout

Figure 3.9: Sketches figuring mesh blocks layouts. From [28]

Structured grids are normally composed of more blocks "glued" together to shape the blade domain. Each block can be conventionally classified based on its utility. In particular, among them, it is common to talk about **H** blocks, characterised by a cartesian distribution of nodes in a way that faces, and so cells, are **orthogonal** one another, shaping an H-like pattern. They are ideal for far-field regions, where flow quantities convection and gradients do not follow a preferred direction. Therefore, a moderate value of aspect ratio is recommended. Whether **O** blocks describe nodes distribution around walls such that grid lines wrap the body concentrically through a radial distribution. They are conceived to capture near-wall flows, therefore boundary layers and their features. Hence, since this type of flow is predominantly axial, O-blocks cells feature relatively high values of aspect ratios, resulting in elongated rectangular-like shapes.
Since this project deals with axial turbine blades, the default **O4H** topology is used in accordance with research conventions in the field, as in Fig. 3.10.

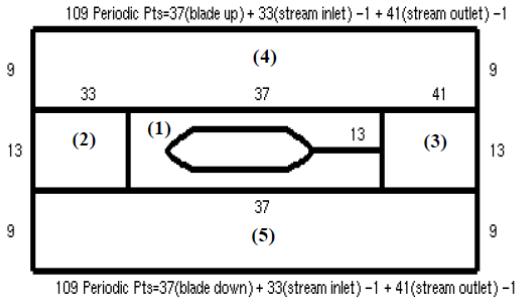


Figure 3.10: Blade-to-blade control layout

Eventually, the last step of the project initialisation on *Row wizard* concerns establishing the blade-to-blade mesh. In particular, the software proposes a default grid nodes plane-wise distribution, that is to say assigning the number of nodes at each block boundary - *i.e.* in Fig. 3.10, the number of nodes to discretise the O-block is 37 for both SS and PS of the blade (1). Successively, the user is allowed to increase or decrease the so-called *grid level*, that is the level of refinement desired. By doing this, it automatically computes the number of points to assign at each block in a way that 3 *multigrid* levels are ensured for each grid level. In other words, the distribution and number of grid points is always set to make sure that when dividing "isotropically" the total number of points in (x, y, z) directions by 2, 4 or 8 - respectively 2, 1, 0 multigrid level - the same ratio between them throughout the blocks is always guaranteed, in order to preserve the grid features.
Afterwards, since the initialisation is completed, the user can choose between creating the 3D mesh or editing the layout proposed by *Row wizard* - *i.e.* the distribution of the span points, the number of points in each block in the blade-to-blade. It needs to be highlighted that, after defining the desired blade-to-blade layout, the user can choose how many optimisation steps the software must perform in order to run the smoothing elliptic algorithm that creates the grid: after some trial and error, a value of 2000 is set - instead of 200 as default - for any mesh created. A good check for an optimal blade-to-blade mesh can be the orthogonality plotting integrated in the software, as in Fig. 3.11.

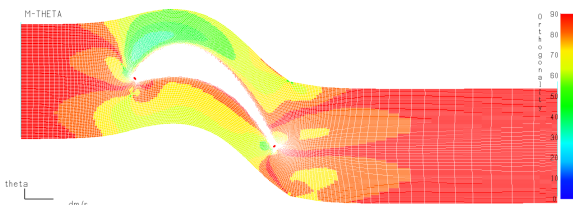


Figure 3.11: Orthogonality cells contour

The latter option is normally preferred in order to better customise the grid topology with a prior knowledge of the flow behaviour, specifically allowing local refinement in domain zones where particular care is needed, or enhance the span points distribution, for instance. This approach has been employed during the training phase and the CFD workflow development of the project. Eventually, when all parameters are set, the user then has to select *Generate 3D mesh*. As previously mentioned, the further the parameters are from those optimally advised by *Row wizard*, the more time 3D meshing phase lasts. At the end of the process, *AutoGrid*5^{TM} creates the *.igg*, *.trb*, *.qualityReport* files associated to the case. In particular, the latter's content is also displayed (see Fig.3.12).

Figure 3.12: Grid quality check displayed after 3D meshing.

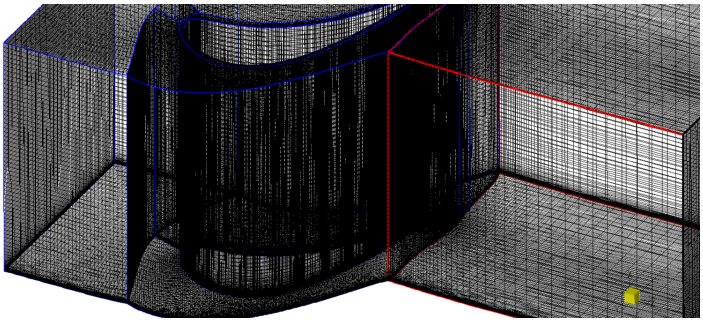
Recommended values by the software community and experts are roughly:

- Negative cells: 0
- Number of Grid Levels: ≥ 3
- Minimum Skewness (the measure of discrepancy between the element and its *ideal* shape: > 15
- Maximum Aspect Ratio: < 15000
- Maximum Expansion Ratio: < 2

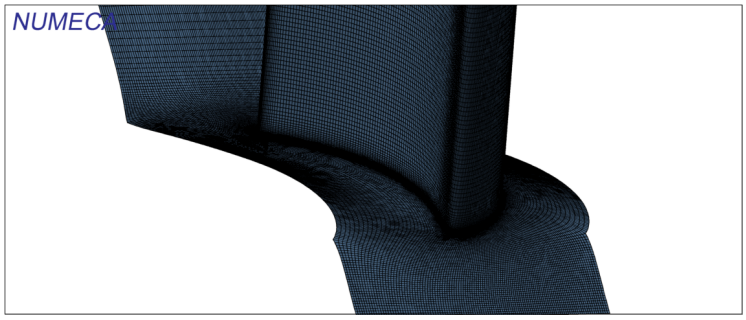
- Spanwise Angular Deviation: < 30 At a later stage, it is possible to edit each grid patch manually. To specify, a patch represent a designated boundary condition that each face contouring the mesh block must have, so that its physical properties are characterised. This leads to a well-posed mathematical problem for the governing equations solved. In turbomachinery applications, the most common patches within CFD solvers are:

- **Wall**: let the block behave like a solid surface, hence mathematically imposing the *impermeability* or *tangency* conditions, if the flow is viscous or inviscid, respectively.
- **Periodic**: it establishes a pairing between two boundaries: the flow quantities in each cell of the first are correspondingly the same for the coupled cell in the second boundary. For instance, if a variable ϕ is evaluated between x and $x + L$ - where L is the periodicity length, such conditions are satisfied: $\phi(x) = \phi(x + L)$ and $\left.\frac{\partial \phi}{\partial x}\right|_{x} = \left.\frac{\partial \phi}{\partial x}\right|_{x+L}$. It is largely adopted in linear cascades.
- **Mirror**: enforces flow field symmetry. It serves as a plane of reflection for the flow quantities.
- **Inlet**: attributed for each block faces that represent the domain inlet. Here, flow quantities inlet values are addressed by the user.
- **Outlet**: works as inlet, but represent the domain outlet.

To leverage the three-dimensional symmetry of the domain span-wisely, the numerical campaign has been conducted by assigning mirroring condition to the block faces that make up the shroud. As a matter of fact, simulating only half-blade domain guarantees reduced computational cost while enhance mesh refinement.



(a) 3D mesh: blocks



(b) 3D mesh: walls

Figure 3.13: 3D mesh example

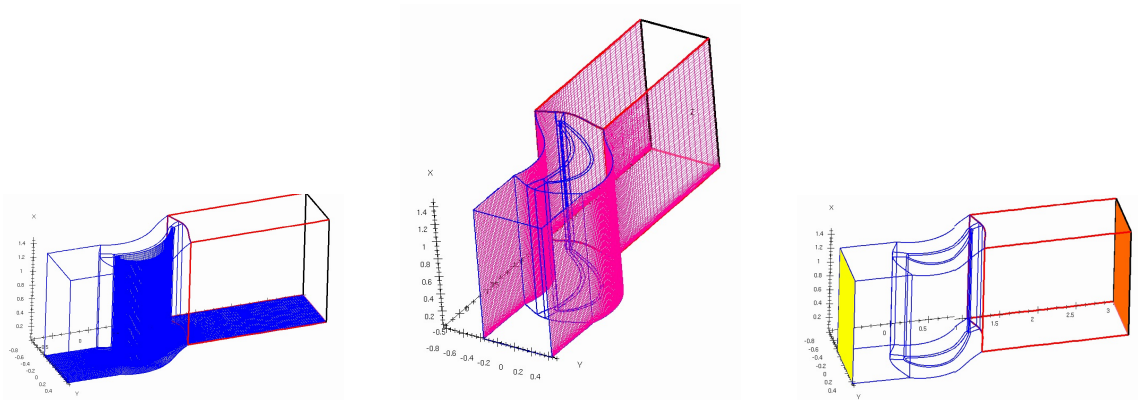
The patch assignment is performed manually, whereas during the automation this is addressed to a Python script comprised in `mesh_processor.py`, the Python class comprised in the main tool script, which execution covers all the steps hitherto mentioned.

3.7 CFD workflow

Following the mesh generation process detailed herein, the numerical investigation framework can be outlined. In the domain of CFD, distinct methodological approaches share several common critical components: classifying boundary conditions across designated domain patches; identifying appropriate numerical schemes that guarantee the stability and consistency of the equations solution; defining convergence criteria; and, finally, implementing turbulence models, as previously mentioned.
At this stage of process, it was first necessary to gain familiarity with the *FINE*^{TM}*/Turbo* solver environment. In particular, preliminary test cases obtained from different input blade geometries and mesh configurations have been employed in order to obtain results that ensured both numerical stability and convergence, nonetheless a good quality solution. The approach was largely based on a combination of trial and error and prior knowledge

gained in academic courses.
In particular, RANS steady-flow calculations have been performed, drawing upon the work of Coull in 2021, [5], as it explains that, according to Marconcini et al. [20], these calculations can deliver an endwall loss estimation comparable to the accuracy of LES computations. In addition, the effects of unsteadiness and of the full stage - so rotor and stator combined - are not taken into account, even though Denton & Pullan [8] highlight how steady RANS overestimate losses, such in SKE prediction and entropy generation, and how evaluating the whole stage could lead to higher loss values.
Always Coull [5], [4] states that it is widely acknowledged that inlet plane position can be placed at 0.8 C_{ax} upstream LE and outlet at $1.2 \div 1.5$ C_{ax} downstream TE, since quite all SKE dissipates within this range, according to Denton & Pullan [8]. For the purpose of the project, to ensure that the major quantity of the main vortex structures evolve and dissipate, the outlet plane has been positioned at 2 C_{ax}.

3.7.1 Boundary conditions



(a) Wall patches **(b)** Periodic patches **(c)** Inlet and Outlet patches

Figure 3.14: Boundary Conditions setup

A fundamental step in the CFD computation setup is defining appropriate boundary conditions, namely mathematical constraints placed in certain zones of the domain - typically at the inlet and outlet - that guarantee a solution for PDE equations system and ensure it to be a *well-posed* problem. Hence, their nature appears to be mainly mathematical, but when dealing with flow governing equations, it becomes also physical: adequate values are needed to describe a realistic scenario. Each problem can require its own types of boundary conditions that change in function of the type of fluid treated and its application - *i.e.* internal or external; reacting or non reacting; supersonic or subsonic flows, and so on. A

detailed treatment on the topic, although, goes beyond the scope of this work.
For what extents the application of this project, mainly Low Speed turbine blades, the configuration of the flow field at the boundaries follows the standard approach for subsonic flows, thus for parabolic equations. Specifically, two specific flow variables at the inlet, while only one is applied at the outlet.
Inlet conditions are specified in terms of total quantities, thus total pressure p_1^o and total temperature T_1^o, including flow angles (V_x, V_y, V_z normalized with the module of midspan free stream velocity $|V|$). In addition, as previously mentioned, since dealing with RANS model, further turbulence quantities are to be given. When a two-equation turbulence model is employed, two values are to be given in addition. While, when dealing with a single equation model, only one value in input is sufficient. Velocity magnitude is then extrapolated by those input quantities.
Outlet conditions, as common practice, are specified through a pressure outlet condition - from a subcategory of conditions known as *reflecting*. It requires that the outlet plane is everywhere subsonic, otherwise if locally supersonic, unphysical field reflections and numerical instabilities could occur. In addition, since the software allows it, a backflow control is enabled: the solver automatically adapts total temperature values in order to avoid local reverse flow, that brings to negative values of pressure and/or density, and so to numerical divergence. In both inlet and outlet, a first-order values extrapolation is applied. (See Fig:3.14c).
To complete the setup, periodic boundary conditions are assigned to the patches on the other two domain boundaries (Fig:3.14b); wall type to the patches representing the cascade endwall and the blade surface (Fig:3.14a).
The calculation of aero-thermodynamic quantities is performed by a subroutine based on isentropic flow relations, as common practice. By doing so, a flow field physically coherent with desired inputs is obtained. This is crucial for the numerical stability and, also, for the quality of the solution perspectives. This routine has been conceived to match the tool processing procedure, namely computing the flow quantities given certain inputs.
As previously mentioned, the ensemble of input airfoil geometries is a generated output of an optimisation tool. Since it realises a desired blade loading, it also needs a computational means to verify if the predicted matches the measured M_{is} distribution along the airfoil. In addition, its validation criteria includes also the outlet flow angle α_2 esteem: the prediction error $\Delta\alpha_2 = \alpha_{2_{PRED.}} - \alpha_{2_{MEAS.}}$ and its Root-Mean-Square (RMS) error. The measured flow quantities are obtained numerically by fast-generated simulations run

on MISES, a coupled viscous-inviscid flow solver developed by MIT (*Massachusetts Institute of Technology*). It means that couples the inviscid effects of Euler equations and the viscosity effects through boundary layer integral equations [10]. For that specific design-case purpose, inlet flow angle α_1, inlet Reynolds measured on axial chord Re_{x_1} with total temperature T^o, outlet Mach number M_2 and TE Kutta conditions - since open, the flow must obey $\vec{V}_{TE_{PS}} = \vec{V}_{TE_{SS}}$, both in modulus and direction. Hence, each blade geometry is coupled with a report file `outlet_XX datablade VALIDATION` on which every inlet/outlet computed flow quantities ratios, also including blade pitch p.

These represent the starting point for boundary conditions subroutine `calculate_flow_quantities`. Coherently, it reads values of: p, $\alpha_{1,2}$, T^o, M_2 from each MISES output file and accepts as input the desired value of $Re_{2_{is}}$. With these quantities prescribed, all aero-thermodynamic quantities at outlet can be computed: static temperature T_2 and velocity V_2 values from Mach and Total Temperature, and then density ρ_2 and static pressure p_2 with Reynolds. Subsequently, through mass conservation between inlet and outlet planes constraint, inlet flow quantities can be determined. This methodology needs an iterative procedure on guessing M_1 value, compute the 0D-model mass flow, and compare it to the outlet one, until their difference is lower than a tolerance value.

By doing so, the tool structure allows the user also to investigate the influence of the Reynolds number Re_2 within the design space already composed of loading parameters and M_2. It is important to emphasise that the expected flow regime in analysis is high-Re fully turbulent, hence transition effects are not taken into account.

At this point, the solver project file *.iec* can be filled in with the boundary conditions values if the user is only interested in an "uniform" inlet condition. In this case, imposing constant flow values at the inlet plane, the computation allows the numerical development of the flow, letting the endwall boundary layer evolve towards the blade passage, just before encountering its potential flow effects. Alternatively, one can be interested in investigating the effects of an "artificial" boundary layer imposed at inlet plane, as highlighted in the review section.

Whereas, *FINE^{TM}/Turbo* solver allows the user to input boundary condition quantities in terms of profiles allowing the reproduction of experimental conditions: angles, total pressure, total temperature and turbulence closure values are accepted as span-wise distributions.

Since this study aims to investigate the effect of the inlet boundary layer among the parameters that effect secondary flow and endwall induced losses, a boundary layer at

inlet is imposed through span-wise velocity ratio profile according to Blasius formulation:

$$\frac{u(x)}{U} = \left(\frac{x}{\delta}\right)^{\frac{1}{n}}, \quad \text{for } x < \delta$$

$$\frac{u(x)}{U} = 1, \quad \text{for } x \geq \delta$$

Consequently, the velocity distribution is expressed as total pressure profile through incompressible flow relation:

$$\frac{u(x)}{U} = \sqrt{\frac{\frac{2}{\rho}(P^0(x) - P)}{\frac{2}{\rho}(P^0 - P)}} \Rightarrow$$

$$P_1^0(x) = P_1 + \left(\frac{u(x)}{U}\right)^2 (P_1^0 - P_1)$$

It is noticed that three-dimensional boundary layer effects, such as skewness due to incidence, are not investigated, despite their significant contribution to secondary flow effects in turbine cascades.

Following the approach of [5], choosing parameters that shape boundary layer becomes important for the data analysis. For this reason, the tool implementation is designed to receive them as inputs. First, the thickness - normalised to the axial chord, useful for design purposes - δ_{98}/C_{ax} has to be specified. In addition, two input configurations can be handled: power-law exponent n, which allows to compute boundary layer parameters δ^*, θ. Or, it is possible to give parameters ratios δ^*/δ and θ/δ and then a dedicated subroutine computes the value of the Blasius power-law exponent. In both cases, shape factor is then computed $H_{12} = \delta^*/\theta$. As Coull defends in his works, both δ and H_{12} are highly impacting secondary flow, hence these values will be part of the data acquisition.
It is worth highlighting that the velocity ratio profile is sampled making sure that the first point out of the wall, hence the second array point, is constrained roughly within the first cell of the walls. This ensures that the solver performs accurate interpolation, as the quality of inlet boundary layer profile sampling can influence loss distribution at the outlet, so affecting their assessment - as Giovannini et al. outline [20].
Every aforementioned step is performed by `generate_radial_bc.py`, summing up: handles two types of boundary layer parameters, computes midspan boundary condition values through isentropic relations and generates inlet total pressure boundary layer profiles.

3.7.2 Turbulence model

The last step of boundary conditions setting is choosing the suited turbulence model. As specified in the previous sections, the choice can be case-dependent when comes to run

RANS computations. In particular, flow regime, its steadiness or unsteadiness, flow domain extension and properties, for instance, can vehicle to one model despite another. As a matter of fact, literature offers experts points of view for any turbulence model implemented in own-built or commercial solvers, for a wide range of flows and their applications, supported by experimental campaigns. As a consequence, a conventionally preferred model can be employed.

To what extents steady RANS computations of turbine linear cascades, two equations shear-stress-models - as $k-\omega$, [20] or SST [5], [3]- and one equation Spalart-Allmaras (S-A, [26]) [8], [17] are commonly adopted.

The two equations models, namely, add one transport equation for **turbulent kinetic energy** k $[\frac{m^2}{s^2}]$ and one for the **turbulent kinetic energy dissipation rate** ε $[\frac{m^2}{s^3}]$, meaning that their expression mimics NS mathematical structure: a time-variation term, a convective term and a diffusive term - where, in particular, turbulent viscosity μ_t expression appears modelled as $\mu_t = \bar{\rho} C_\mu \frac{k^2}{\varepsilon}$ - combined with empirical coefficients, like C_μ. Furthermore, as $k-\omega$ model name suggests, the energy dissipation rate is replaced with **specific dissipation rate** of turbulent kinetic energy $\omega = \varepsilon/k$ $[\frac{1}{s}]$. Without diving deeply in the topic, historically $k-\varepsilon$ model have been adoperated for external flows, whereas $k-\omega$ for internal ones, each individually better performing in restricted cases. As a matter of fact, since the latter has been better treating near wall flows - and its behaviours like transition and separation -, an hybrid model called SST (Shear-Stress-Transport) was conceived by Menter [19] introducing a blending function that alters coefficients in case of wall proximity, consequently obtaining turbulent viscosity. Hence, turbulence quantities to prescribe at inlet plane are conventionally expressed in function of turbulence intensity (Tu) $[-]$ defined as:

$$Tu = \frac{\sqrt{\frac{1}{3}(u'^2 + v'^2 + w'^2)}}{V_{ref}}$$

for internal flows like turbine application, normally accounts for 3-5% . And integral length scale (ILS) $[m]$:

$$ILS = c\frac{k^{\frac{3}{2}}}{\varepsilon}$$

from literature typically expressed as a percentage of axial chord, often chosen as 10% C_{ax}, as recommended by [4, 5]. Then, they are linkable to turbulent quantities:

$$k = \frac{3}{2}(V_{ref} Tu)^2 \qquad \varepsilon = C_\mu^{\frac{3}{4}} \frac{k^{\frac{3}{2}}}{ILS}$$

Whereas, to what extents S-A model, a single transport equation in function of an *eddy*

viscosity $\tilde{\nu}$ $[\frac{m^2}{s}]$ that is linked to turbulent viscosity by multiplying by flow density and field corrective function f_{v_1}, hence the "eddy" meaning. In this case, the inlet value can be linked to the former quantities as:

$$\tilde{\nu} = \sqrt{\frac{3}{2}}(V_{ref}\ Tu\ ILS)$$

At this stage, SST model has been chosen and a value of $y^+ = 1$ has been imposed in order to resolve the viscous sub-layer of the boundary layer, as the Author suggests. First wall cell width, then, has been computed with 3.12. In addition, an expansion ratio of ≈ 1.2 has been kept for cells on blade wall surfaces.

3.7.3 Numerical scheme and convergence criteria

In order to achieve numerical stability, choosing the best suited spatial and temporal discretization schemes is crucial. For steady RANS computations, only a spatial numerical discretisation is needed. For this purpose, the commercial solver proposes two main approaches: **centered** and **upwind** schemes. Given that all test cases in this work pertain to low speed turbine blades, the flow fields in exam are expected to be subsonic everywhere. Hence, due to their nature described as elliptic equations and since there is no priviliged direction for wave propagations, both schemes are suitable. Although centered schemes guarantee numerical stability, ensuring convergence, through "injection" of artificial dissipation terms, they could pay in solution quality. Whereas, upwind discretisation schemes deliver high quality solution but are known to be less numerically stable, even though stability can be controlled through solution slope limiters, as as summed up in [27]. Anyway, the latter is most suitable with transonic and supersonic flow regimes, governed by hyperbolic equations, which are characterised by a unique direction where waves propagate. In such, compressibility effects become crucial and, hence, the presence of shock waves and other types of flow discontinuities must be accurately treated because of steep gradients of aero-thermodynamic quantities can produce numerical spurious oscillations compromising the solution. For this reason, upwind schemes deliver good solutions and numerical behaviour by leveraging characteristic theory.
So said, in the first stages of numerical investigation, a trade off between two numerical schemes proposed by the software is performed: second-order symmetric *Total Variation Diminishing* (TVD) upwind scheme, with convective fluxes computed through Roe's linearised approach, [22], along with entropy fix correction by Harten, [13], and coupled with Van Albada limiter; and centered scheme, employing the approach by Jameson & Martinelli, [15], which propose a blended second and fourth-order discretization schemes in

order to introduce numerical viscosity terms which avoid excessive numerical dissipation. In addition, FINE^{TM}/Turbo allows to select CFL number, which scales the time-step sizes used for time discretisation scheme's iterations. Its value is crucial when solving unsteady CFD computation, while in steady ones only indicates how fast the stationary solution must be reached through numerical iterations, namely a fictitious time-step. For these particular conditions, the software offers the possibility to run with *CPU Booster*, an optimised numerical algorithm employed to solve implicit equation systems - in other words, a computational-effective matrix inversion algorithm. Consequently, convergence acceleration is guaranteed. To fulfill this purpose, also *Multigrid Initialisation* is available: it allows the solver to run first iteration steps with different coarseness mesh level, from the level "0" - coarsest, obtained dividing the number of nodes isotropically in (x, y, z) direction by the total number of levels - to "2" the actual mesh.
Initially, a preliminary numerical campaign has been performed to choose the most suited discretization scheme, that ensured numerical stability most of all. Furthermore, a CFL number of 1000 - with booster - has been set, with three levels multigrid initialisation.
To what extents convergence criteria, residuals threshold has been set to 10^{-7} with a total number of 1500 iterations permitted, as arrest criteria. These boundaries were determined *a posteriori*, given the simulations carried out.

3.8 Grid convergence analysis

Once the computational setup is established, the numerical investigation can be performed. As outlined in Section 3.1, minimising the source of error generated by the mesh is crucial in order to obtain a convincing numerical solution. Therefore, performing a grid convergence analysis is widely acknowledged to be an indispensable practice when numerical results are proposed in literature. In particular, it assesses whether the solution obtained approaches the "true" mathematical solution - although itself affected by discretisation error - as the computational grid is progressively **refined**. In this case, if a certain measured quantity gets independent from the mesh density - *i.e* the number of cells - it can deliver a proper estimation of numerical uncertainty, hence considered reliable. In addition, this practice helps to establish a trade-off between the grid error and the mesh refinement level that balances sufficient accuracy for the desired quantity estimation and, especially, computational cost.
As for turbomachinery applications, especially turbine cascades, mesh quality and density are parameters that highly depend on the flow regime, the operational Re and complex

geometries induced flow topologies, as purge flows, cavity flows *etc.*; but also the type of numerical analysis that wants to be performed.
In this work, as hitherto mentioned, steady RANS computations with fully turbulent flow regime are realised, hence no complex flow phenomena are expected - such as high level of turbulence and vorticity near transition and/or separation zones - and then a modest number of grid cells is sufficient to reproduce the mean stationary flow field.

The independence study has been carried out bearing conclusions from Coull [4] and Coull & Clark [6],[5] works. In the first, among the extended research on design sensitivity to aerodynamic losses, a strong dependency of turning angle and blade shape against endwall losses is outlined. In particular, the amplification factors, coming from "classic" secondary flow theory - and then correlated with predicted endwall losses - strictly depend on flow angles - hence turning - and velocity ratio across the passage, as for ΔT^*. It comes out that, independently from inlet conditions, for higher turning blade designs, AF values tend to grow significantly with a near-proportional relation with secondary flow-induced loss. In addition, from the second work, a correlation between AF and inlet boundary layer thickness δ with SKE production is investigated from a design sensitivity perspective. It emerges that SKE scales with the square of AF: consequently, high AF designs - hence mainly high turning blades - are expected to produce higher magnitude of SKE, hence "bigger" secondary flows. Moreover, these are also keen on being more sensitive to inlet boundary layer properties, as illustrated in section 2. In fact, Coull [5] highlights that laminar-like profile show higher peaks of inlet vorticity - and velocity-vorticity products - far from wall, with respect to turbulent profiles which show this trend in near wall region. As a consequence, levels of maximum vorticity generation result to be away from endwall, hence greater span-wise penetration and heading towards the center of main passage flow.

Based on the points discussed thus far, a grid independence study has been carried out for **two contrasting test cases** that could represent the **extremes** of a potential data-set accounting for blade airfoil optimisation tool, and considering secondary flows behaviour:

- **Laminar-like thick boundary layer with high turning blade**
- **Turbulent thin inlet boundary layer with low turning blade**

The choice of boundary layer characteristics has been performed accounting for vorticity distribution coefficient Π_{SKE} in function of normalised boundary layer thickness $\delta_{98}/p\cos\alpha_2$ as in Fig.2.5, [5]. Through secondary flow theory calculations, their relation is parametrised with power law exponent n and shape factor H_{12} of inlet boundary layer profile. Here,

three regimes are distinguished by the effects on secondary flows: thin boundary layer, buffer zone, thick boundary layer regimes. The first involves low Π_{SKE}, this means that vorticity is mainly relegated near the endwall, hence low secondary vortices penetration ($\delta_{98}/p\cos\alpha_2 \lesssim 0.1$); and the latter shows the opposite trend, and therefore deleterious for secondary losses ($\delta_{98}/p\cos\alpha_2 \gtrsim 0.5$). Two values of thickness have been selected from the figure plot values available. As thick laminar-like profile, de-normalised $\delta_{98}/C_{ax} = 0.3178$ with $n = 1.4$, yielding a computed $H_{12} = 2.43$. Whereas, as turbulent thin, the corresponding value chosen is $\delta_{98}/C_{ax} = 0.05$ with $n = 7$, resulting in $H_{12} = 1.29$. Since dealing with steady computations, the sensitivity analysis can be performed by evaluating a field quantity useful for the analysis. Therefore, the plane-wise value and radial pitch-wise profiles of mixed out averaged kinetic energy gross loss coefficient ξ extracted at 0.5 C_{ax} downstream TE and, in order to guarantee a bi-dimensional flow at midspan, a span-to-chord ratio $h/C_{ax} = 3$ has been chosen, [5]. **Half-blade calculations** have been performed since the flow structure is expected to be symmetrical, in order to save computational cost. In addition, a central scheme has been employed for spatial discretisation, and S-A as turbulence model with computed $\tilde{\nu}$ as inlet boundary condition. The remaining numerical parameters have been set as previously stated.

As commonly argued, the analysis is first conducted for 2D *B2B* mesh, constituting one "layer". Once independence is reached, the structured 3D mesh is built up by a systematic span-wise addition of these mesh layers. In order to capture the complex flow structures near the endwall, the refinement is performed by adding layers - or *flow paths* - only relegated in wall proximity. This is allowed by keeping constant *Percentage of Mid-flow Cells* value at 23% in *AutoGrid5*^{TM} environment. This parameter expresses the constant-sized flow paths designated to occupy the main passage region, in percentage of the total number of span layers - intended to be towards midspan in this particular case. The value has been chosen through trial and error in previous "training" computations, observing numerical stability by limiting the span-wise cells expansion ratio that avoided computations residuals to diverge.

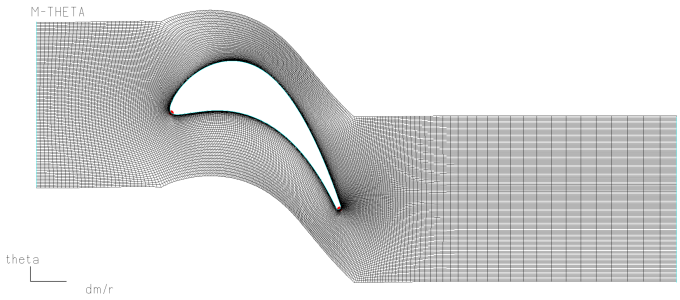
3.8.1 *LTHT* (Laminar Thick High Turning blade) case

Turning	α_1	α_2	$Re_{2_{is}}$	M_2	H_{12}	$\delta_{98}/(p\cos\alpha_2)$	n
105.125°	**40°**	**−65.125°**	$\mathbf{1.1 \times 10^6}$	**0.5**	**2.43**	**0.76466**	**1.4**

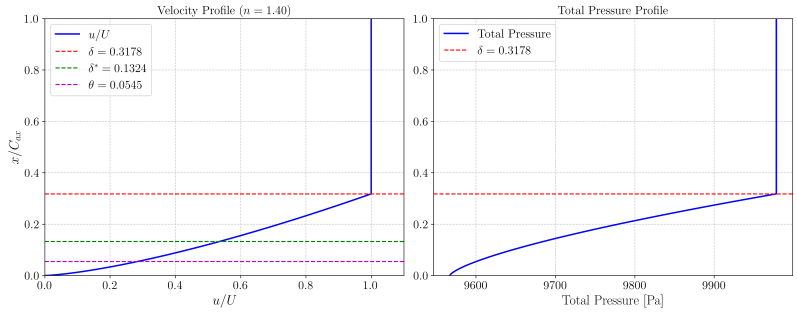
Table 3.1: *LTHT* blade parameters and flow boundary conditions

Among the blade shape generation parameters, flow angles and M_2 are reported in Table 3.1 along with inlet boundary layer parameters. As in Fig.3.15a, the blade shape suggests also that it is intended to withstand high loading since its pronounced thickness. Moreover, this characteristic, in conjunction with the high turning angle, confirm the suitability of this test case for examining significant expected secondary flows induced loss, since the passage blockage is increased, pressure side velocities are lower, and therefore higher values of ΔT^* are reached.

Fig.3.15b illustrates the span-wise boundary layer profile imposed at inlet. In particular, it is noteworthy that total pressure distribution presents an inflection point towards the wall, and this is explained mathematically by the value of the Blasius power law exponent $n = 1.4$.



(a) B2B LTHT blade mesh



(b) Inlet boundary layer profiles and parameters

Figure 3.15: LTHT configuration

3.8.1.1 2D: Blade to Blade

As previously mentioned, the sensitivity analysis has been performed by leveraging the B2B mesh refinement function through *Levels* offered by *Row Wizard* tool in *AutoGrid5*^{TM}. Therefore, the baseline mesh has been set to Level 4, as lower refinement levels delivered a

very coarse mesh. Subsequently, the increase of Levels mainly focused on suction side and pressure side discretisation points. It is worth to notice that a higher amount of points is destined to SS, since blade's high thickness and high turning lead to higher suction surface length with respect to PS. Inlet and outlet points exhibited minimal sensitivity to mesh refinement, and the number of cells shaping the *O-type* mesh for boundary layer capturing has been fixed to 29, ensuring a near-wall expansion ratio of ≈ 1.19. In addition, a mesh cells **relaxation** has been adopted after the stream-wise position of $z = 1.7\ C_{ax}$, since the evaluation of the loss is obtained by extracting flow quantities at $z = 1.5\ C_{ax}$, and therefore saving computational cost. Tab. 3.2 synthesises the mesh properties of the progressive refinement.

Level	**SS points**	**PS points**	**N° cells**
4	251	79	43302
5	289	89	56582
6	305	101	62926
7	313	107	68900
9	329	109	75878
10	366	119	87566
12	379	123	92340
14	409	137	120533

Table 3.2: B2B Mesh properties for LTHT refinement

2D RANS calculations have been carried out employing analogous methodological approaches with respect to boundary conditions and numerical methods, as previously exposed.

The quantity evaluated for the sensitivity analysis is the **profile loss** in terms of kinetic energy loss, as expressed in the Eq. 2.1. The profile loss is obtained by averaging static and total pressure at the outlet plane and total pressure at the inlet plane along the pitch. A mixed out constant area flow averaging technique has been employed, with a procedure provided by *SAE* (Safran Aircraft Engines) that will not be shared in this thesis. However, it is founded on an analogous approach outlined by Amecke in 1970s, hence solving four flow governing equations (conservation of mass, momentum parallel to blade row, momentum perpendicular to blade row and energy) but using a 1st order Taylor expansion. It is preferrable to mass-averaging approach because the analysis can take main flow mixing with uniform flowfield generated loss -*mixing*- into account.

As Fig.3.16 outlines, profile loss values apparently plateau around $\approx 3\%$ since the first refinement levels. Instead, an unexpected "under-shoot" occurs for the last two level of refinement, accounting for a $\approx 0\%$ of profile loss. Therefore, this mis-behavior has been investigated looking at mass-flow trend between inlet and outlet for each test case (see Fig. 3.17).

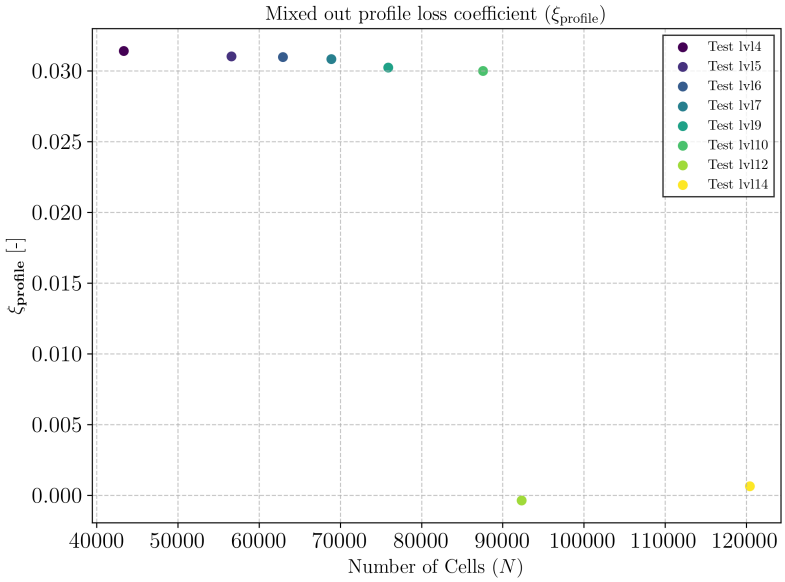
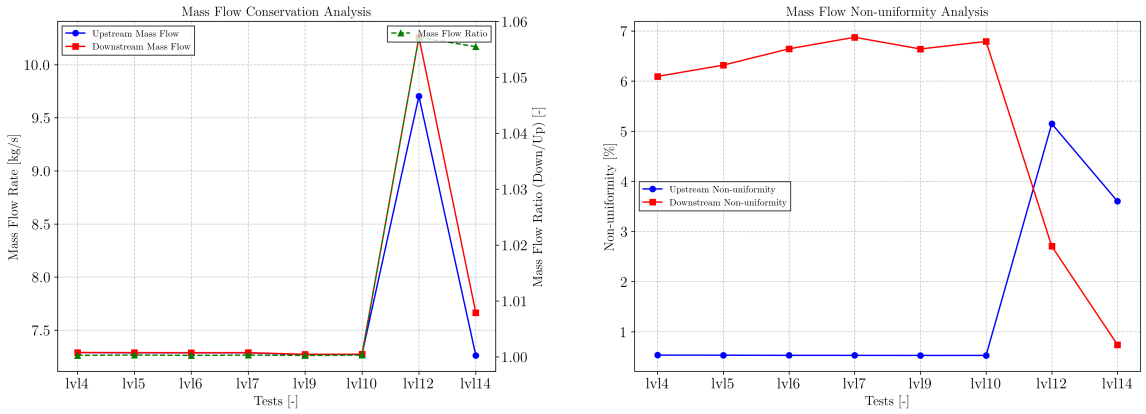


Figure 3.16: ξ_{profile} evaluation for different LTHT configurations



(a) Inlet/outlet mass-flow conservation.

(b) Percentage of mass-flow non-uniformities.

Figure 3.17: Mass-flow investigation for 2D LTHT test cases.

In particular, Fig. 3.17a shows inlet and outlet averaged mass-flow values for each test-case. Levels 4-10 show identical upstream and downstream mass flow values, suggesting

no issue related to convergence status of the computations. Whereas level 12 account for a severe increase of mass-flow average value (from $\approx 7.5kg/s$ to nearly $11kg/s$) and a subsequent drop near baseline value for the finest mesh. Despite this behaviour, a marked discrepancy between upstream and downstream values can be highlighted in the latter two cases, suggesting for an incomplete convergence.

Moreover, non-uniformity of mass flow values extracted in the two planes can be analysed (see Fig. 3.17b). As expected, for the first levels of refinement, a remarked heterogeneity in mass-flow values is seen in the downstream plane - due to the wake. Whereas, an uniform field is measured at the inlet plane: the discrepancy is recorded with mass-flow values from $\approx 0.55\%$ at inlet to $\approx 6-7\%$ downstream TE. However, from level 12 this trend is inverted: higher upstream mass-flow non-uniformity is overseen at inlet plane with respect to downstream. This inversion is **non-physical** and suggests the presence of numerical issues. This analysis provides essential evidence to conclude that the two finest mesh configurations show perplexing flow behaviour because of numerical instabilities that affect convergence and flow properties. Hence, they are **discarded** for the sensitivity analysis.

Fig. 3.18 illustrates the final sensitivity analysis on profile losses: as the mesh gets refined - number of total cells increases - the coefficient tends to plateau around $\approx 3\%$ as previously observed. Evaluating the relative error:

$$\varepsilon_{\text{rel}} = \frac{|\xi_{\text{current}} - \xi_{\text{finest}}|}{|\xi_{\text{finest}}|}$$

delivers a more tangible metrics for the discrepancy between the current coefficient and the highest-mesh resolution one, in percentage (Fig. 3.19). Actually, level 7 represents the best trade-off configuration between accuracy and computational cost. With a relative error of 2.5% with respect to the finest grid configuration, it accounts for 20,000 fewer cells, which in building up the three-dimensional grid it is not negligible.

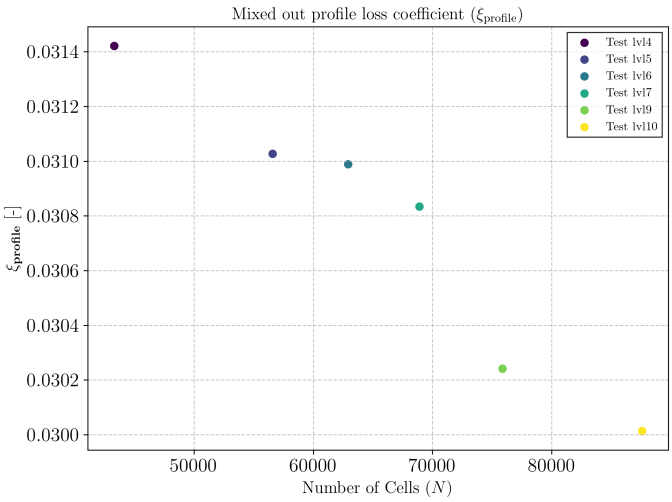


Figure 3.18: LTHT: ξ_{profile} mixed out values against number of cells in different mesh configurations.

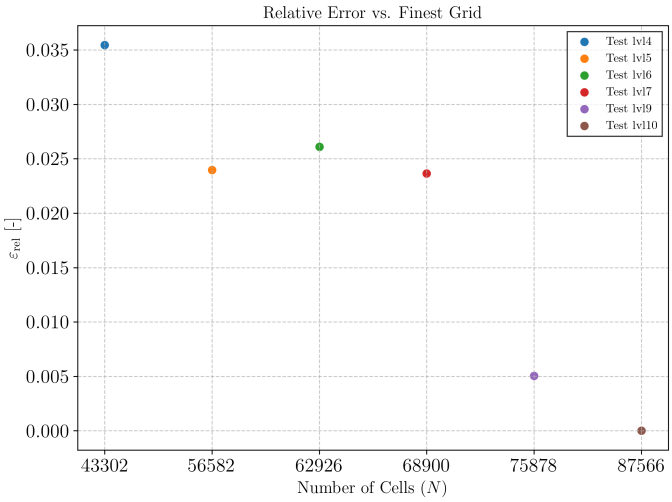


Figure 3.19: LTHT: ξ_{profile} relative error.

Fig.3.20, eventually, shows the mass-averaged pitch-wise distribution of the kinetic energy loss coefficient representing the wake of the profile.

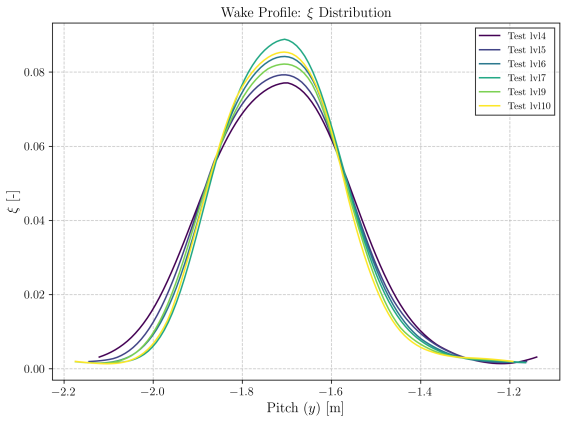


Figure 3.20: LTHT: wake profile (pitch-wise distribution of ξ)

3.8.1.2 3D: Span layers

Once the B2B mesh configuration has been defined, the independence study can be carried out on 3D grid by adding layers span-wisely, as aforementioned. 3D RANS computations were conducted with **four-cores** parallelisation to enhance computational efficiency. CPU time values reported, although, are not to be considered as fully reliable due to their inherent dependence on several numerical factors. These comprehend the choice of numerical schemes as well as the chosen value of CFL or the methodology employed for multigrid initialisation. Further conditioning factors include the number of cores, the number of mesh blocks and the scalability of parallel processes handled by the solver. A rigorous evaluation of these goes beyond the duty of the present work. However, the reported CPU times can serve as a qualitative indicator of computational cost.

Span points	**Nº cells**	**CPU Time [s]**
53	1.96M	5698 (-58.00%)
73	2.55M	6568 (-51.57%)
93	3.44M	9658 (-28.78%)
121	4.47M	10778 (-20.55%)
159	5.28M	13564

Table 3.3: 3D Mesh properties for LTHT refinement

The metrics to detect grid independence is analogous to 2D case, through kinetic energy loss coefficient. However, in the context of 3D domain, the extracted flow quantities are expressed as distributions along a plane. Consequently, both pitch-wise radial distribution - as [20], [12]- and a plane-wise averaged values of ξ will be reported. Whereas, a distribution

of area-averaged exit flow angle $\alpha(x) - \alpha_{\mathbf{is}}$ is not reported, despite its widespread adoption as a metric sensitivity metrics. Likewise, a mixed-out averaging technique is employed to ensure consistency in the methodology.
The kinetic energy loss measured at the outlet plane represents the **overall gross loss**, which includes the inlet boundary layer profile energy loss, the effective loss generated within the passage, the profile loss - computed in the last section - and, eventually, the mixing loss, such as the entropy-generating mechanism of the high momentum mixing with the uniform flow downstream. Hence, by employing a mixed-out averaging approach, the mixing loss is also included in the evaluation, while this information is lost when mass-averaging.

Fig. 3.21 illustrates the radial profiles of ξ at 0.5 C_{ax} downstream TE, and the effects of secondary flows are noticeable. In fact, the "wiggling" behaviour towards higher values of kinetic energy loss is indicative of the presence of vortexes cores and complex flow structures. Specifically, this is the effect of the interaction between Trailing Shed Vorticity (TSV) and PV coming from blade's upper SS, as well as the CV generated at the endwall. It is noteworthy that the zones of high vorticity, and high losses, become more distinct by refining the mesh, and this is noticeable in contour plots in Fig. 3.22. The first two grid levels are apparently not sufficient to fully capture these features, while the last two, representing the finest resolutions, plateau towards the definitive profile. In addition, to highlights the contribution of the three main structures that increase the local vorticity values, a contour plot of stream-wise vorticity ω_x is shown in Fig. 3.24. Near the endwall, CV clock-wise motion dominates the field. Whereas, climbing up along the suction surface of the blade, the main kinetic energy loss core are visible, produced by the interaction of TSV, always moving clock-wisely, with the adjacent PV moving counterclock-wise. Plane-wise averaged vales of ξ in Fig. 3.23 confirm the trend: coarser grid configurations account for a relative error of $\approx 2\%$ with 2.5M cells, whereas the most accurate solution is achieved by doubling the resolution. The best compromise is represented by the mesh configuration built up with 93 span-wise points - mostly relegated towards endwalls, and with a total of 186 span cells if the entire blade is considered, summing up ≈ 3.44 million cells. It delivers a relative error of $\approx 0.5\%$ with respect to the finest grid, and therefore is chosen as the final mesh for this test-case analysis, which eventually registers for a kinetic energy loss averaged value of $\xi = 0.0840$.
It is interesting to notice that the main core vortex is located at $x = 0.6\ C_{ax}$ and $y = -0.5 \div 0.3\ C_{ax}$ approximately towards the center of the passage. Moreover, it is evident how the vortex cores tend to split into two, as the mesh gets refined. This phenomenon

is supposed to be a numerical induced artifact, potentially linked to the RANS turbulence model employed - due to the Boussinesq linear approximation. At this point, a different model, that does not depend on this assumption, can be tested in order to confirm what stated -*i.e.* EARSM (Explicit Algebraic Stress k-omega Turbulence Model).

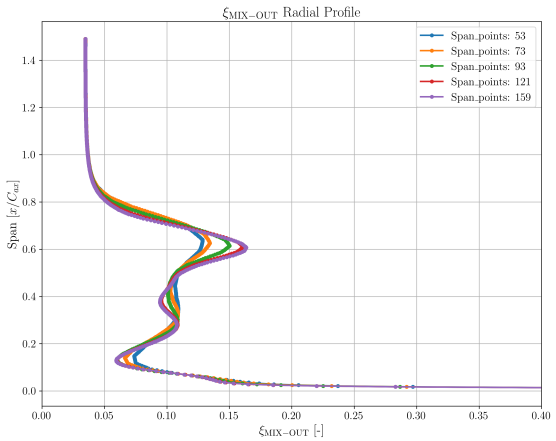


Figure 3.21: LTHT: Pitch-wise radial ξ profile.

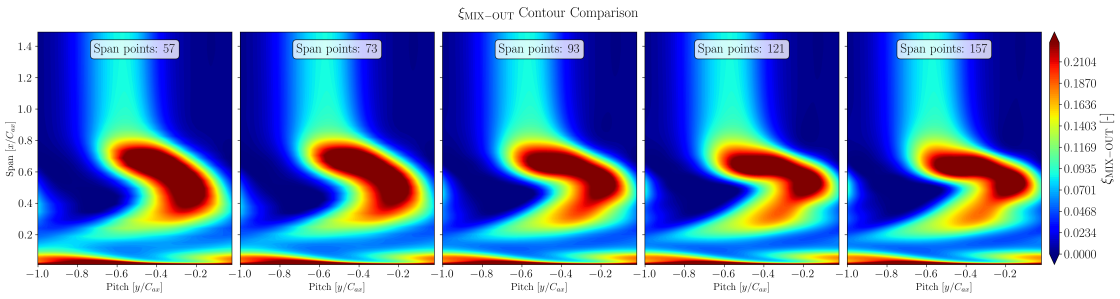


Figure 3.22: LTHT: ξ contour plots.

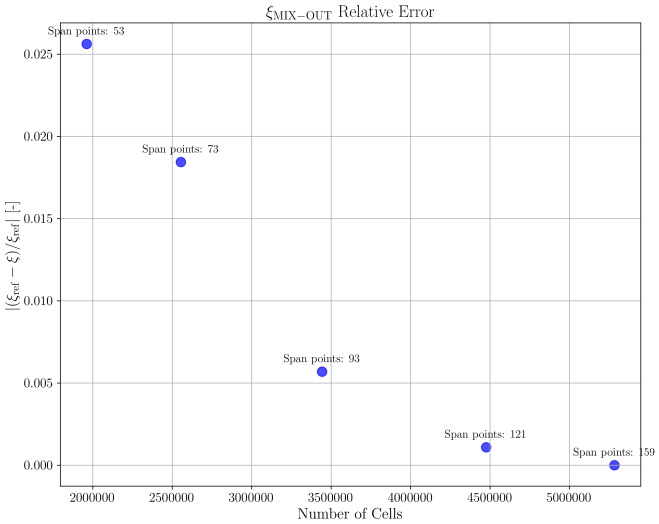


Figure 3.23: LTHT: Relative error for plane-wise averaged values ξ.

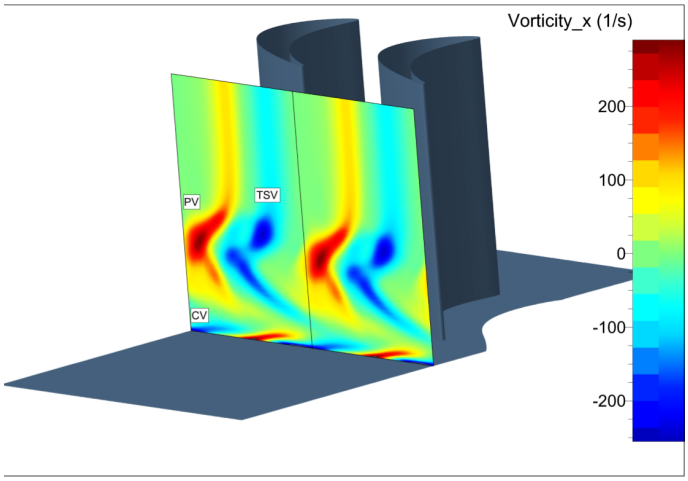


Figure 3.24: LTHT: Stream-wise vorticity ω_x contour plot.

To ensure that the inlet boundary layer characteristics remained unchanged in its evolution path between inlet and blade LE, an additional computation of the same case was performed, maintaining identical numerical methods and boundary conditions but shifting the inlet position closer to the blade at $z = -0.25\ C_{ax}$ upstream LE. Fig. 3.25, hence, illustrates pitch-wise averaged profiles of Total pressure planar fields extracted at the desired axial coordinate z for the foreseen case and for the "shortened" one, both compared with the actual total pressure profile from Blasius power law (Fig. 3.15b).

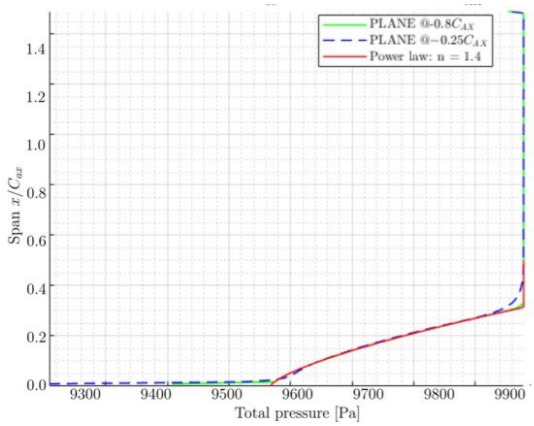
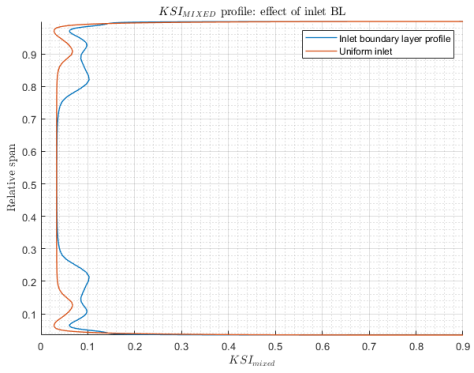


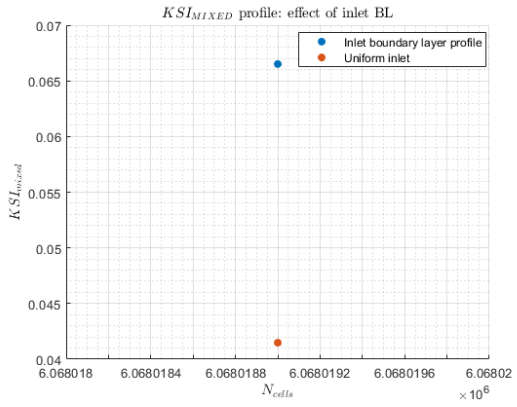
Figure 3.25: LTHT: Total pressure profiles comparison between extracted profiles at two different axial positions and theoretical Blasius profile.

3.8.2 Effects of inlet conditions

In favour of highlighting the effects of inlet boundary layers, as outlined in literature, the same test case was subsequently analysed with an uniform total pressure field at inlet. Midspan value of stagnation pressure is prescribed as input, allowing the uniform distribution to naturally evolve along the endwall as the flow approaches the blade. This time, three-dimensional CFD calculations are performed considering the entire blade, hence with two endwalls - hub and shroud. Results confirm that imposing an "artificial" boundary layer at inlet surely affects kinetic energy losses, since higher values are measured with respect to an uniform inlet, as in Fig. 3.26b. In addition, also secondary flows result to be more developed, and this contributes to the higher loss, Fig. 3.26a.



(a) $\xi_{\mathbf{MIX-OUT}}$ pitch-wise distribution for uniform and non-uniform inlet.



(b) $\xi_{\mathbf{MIX-OUT}}$ plane-wise averaged value for uniform and non-uniform inlet.

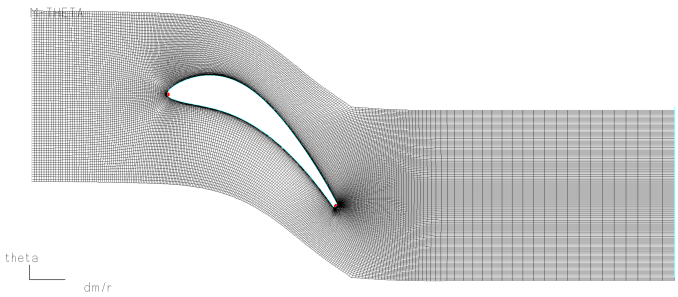
Figure 3.26: Comparison of uniform and non-uniform inlet conditions on $\xi_{\mathbf{MIX-OUT}}$.

3.8.3 *TTLT* (Turbulent Thin Low Turning blade) case

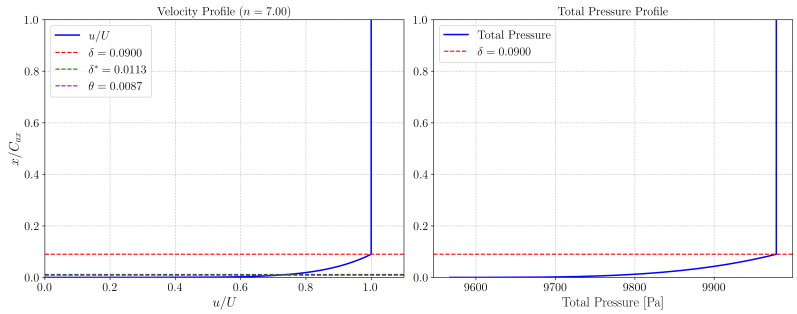
Turning	α_1	α_2	$Re_{2_{is}}$	M_2	H_{12}	$\delta_{98}/(p\cos\alpha_2)$	n
64.11°	**8°**	**−56.11°**	$\mathbf{1.1 \times 10^6}$	**0.6**	**1.29**	**0.16332**	**7**

Table 3.4: *TTLT* blade parameters and flow boundary conditions

For this case analysed, the same methodology has been applied. The mesh properties have been selected to ensure consistency with those of the preceding case.
The blade shape has also been chosen looking at its thickness: due to the reasons explained in the previous case, low thickness blade is desired -as in Fig. 3.27a. Table 3.4 summarizes the main features for this test, outlining a lower value of turning angle - $\approx 40^o$ less - and a turbulent boundary layer characterised by a low values of thickness and shape factor, but higher value of vorticity near wall, (see Fig. 3.27b), with the objective of evaluating a low vorticity distribution coefficient.



(a) B2B TTLT blade mesh



(b) TTLT: Inlet boundary layer profiles and parameters

Figure 3.27: TTLT configuration

3.8.3.1 2D: Blade to Blade

As previously stated, the mesh properties have been chosen to be analogous for those of *LTHT* case. In order to avoid generating two grids with remarkably different characteristics, particularly concerning the anticipated automated process, the 2D B2B grid shows a higher refined discretisation of the SS and PS than a low-thickness and low-turning blade is expected, for the reason exposed previously for *LTHT* case. Indeed, it is arguably known that fewer points for a solution-independent mesh are needed (Table 3.5).

Level	**SS points**	**PS points**	**Nº cells**
3	193	85	40678
5	209	101	46406
6	217	103	55726
7	233	109	61630
9	249	117	70230
10	261	121	78622
14	281	137	96130

Table 3.5: B2B Mesh properties for TTLT refinement

Looking at the $\xi_{\textbf{profile}}$ averaged results, despite the trend seems to be less monotonic with respect to *LTHT* case as in Fig. 3.28, the grid error introduced by those configurations roughly follows its trend in values. As a matter of fact, Fig. 3.29 highlights that levels 3-6 register an error of $\approx 3-4\%$, while the independence is reached for the last two levels, starting from 70.000 cells mesh.

The adequate trade-off between computational cost and relative grid error is identified in level 7 configuration, accounting for 61.630 cells, and yields a relative error of around $\approx 2\%$, consistent with the previous case.

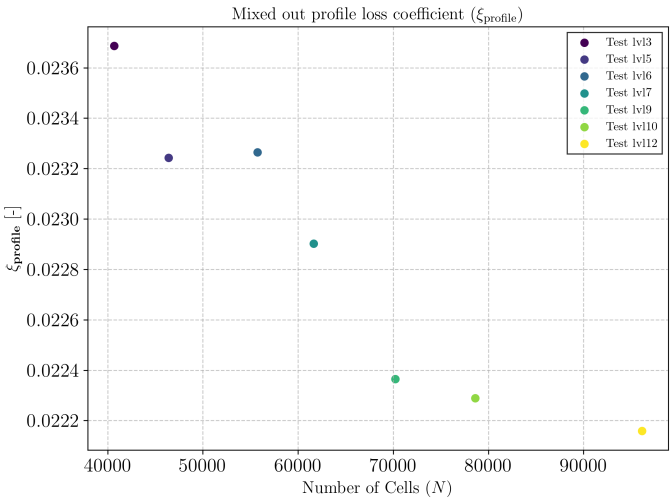


Figure 3.28: TTLT: $\xi_{\textbf{profile}}$ mixed out values against number of cells in different mesh configurations.

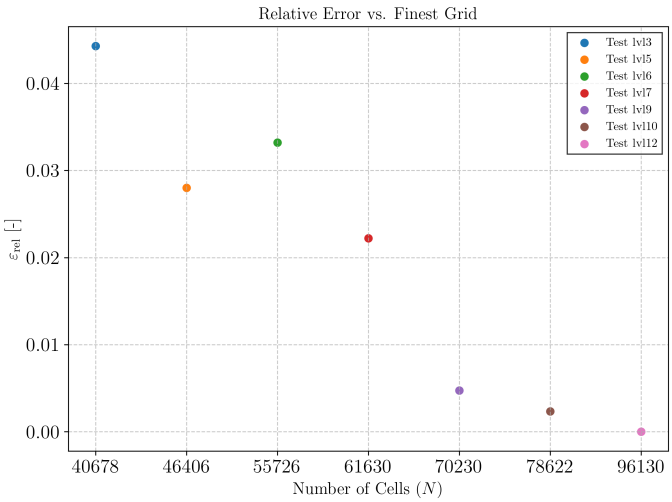


Figure 3.29: TTLT: ξ_{profile} relative error.

Fig. 3.30, similarly to high turning case, portrays the mass averaged distributions of $\xi_{\textbf{profile}}$, hence wake profiles. Although refined meshes do not reach a plateau, they approach it sufficiently closely.

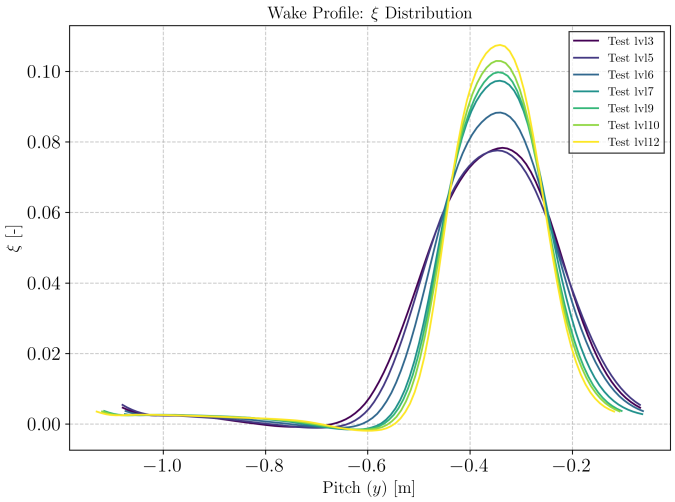


Figure 3.30: LTHT: wake profile (pitch-wise distribution of ξ)

3.8.3.2 3D: Span layers

The structured three-dimensional grid is then built following analogous criteria, summarized in Table 3.3. It is worth to notice that, on average, despite the number of cells constituting the configurations are comparable, the computational time requested to reach numerical convergence is slightly lower, given the same numerical methods.

Span points	**N° cells**	**CPU Time [s]**
57	1.92M	3141 (67.47%)
73	2.46M	4616 (52.17%)
93	3.13M	6523 (32.43%)
121	4.08M	7800 (19.23%)
157	5.29M	9655

Table 3.6: 3D Mesh properties for TTLT refinement

A substantial difference for the radial distribution of kinetic energy loss, in the case of turbulent and thin inlet boundary layer, is observable, as in Fig. 3.31. As a matter of fact, the presence of vortical structures that affect kinetic energy loss coefficient distribution is mostly relegated to wall, according to theory. In addition, the observable vortex core is only one - with the two cores observed in *LTHT* case - and it is significantly smaller with respect to the previous case. This is confirmed by plane-wise averaged values of ξ in Fig. 3.33, always expressed in terms of relative errors. Initial refinement levels, in fact, already deliver an acceptable error value of $\approx 1\%$, indicating a sufficient mesh density throughout

the domain, except outside the wall region. Also by analysing contour plots in Fig. 3.22, it is evident that the mesh refinement has more moderate influence on the flow field.
As a consequence, despite the first two coarsest levels represent a sufficient result, consistency in mesh properties is favoured. Hence, the third configuration is selected as most suitable candidate since it registers a relative error of 0.5% with respect to the finest grid, with a span-wise resolution of 93 points, with a total of ≈ 3.14 million cells. In addition, a value of loss $\xi = 0.0323$ is obtained.

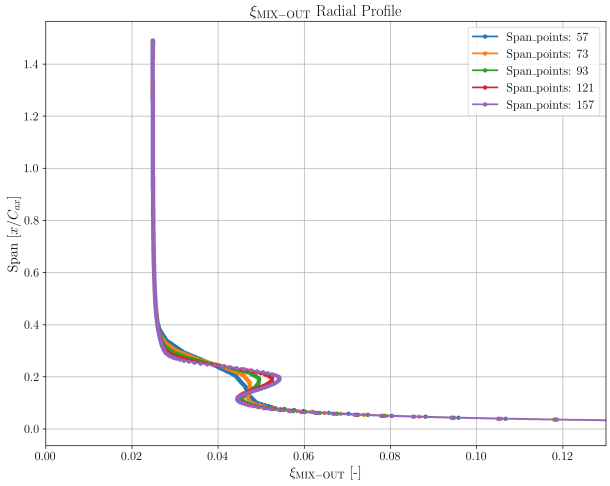


Figure 3.31: TTLT: Pitch-wise radial ξ profile.

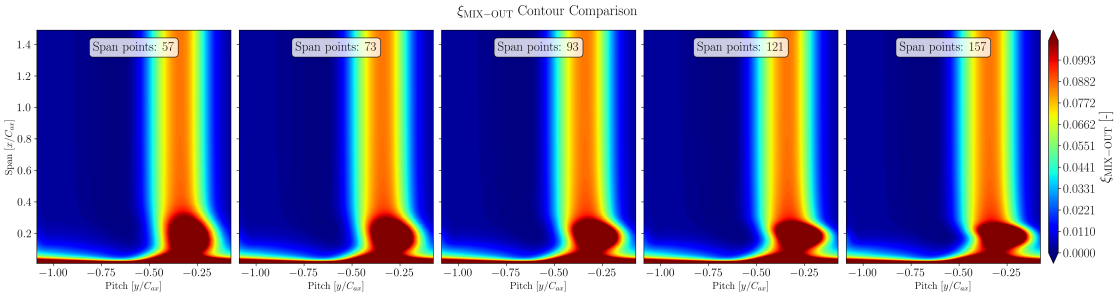


Figure 3.32: TTLT: ξ contour plots.

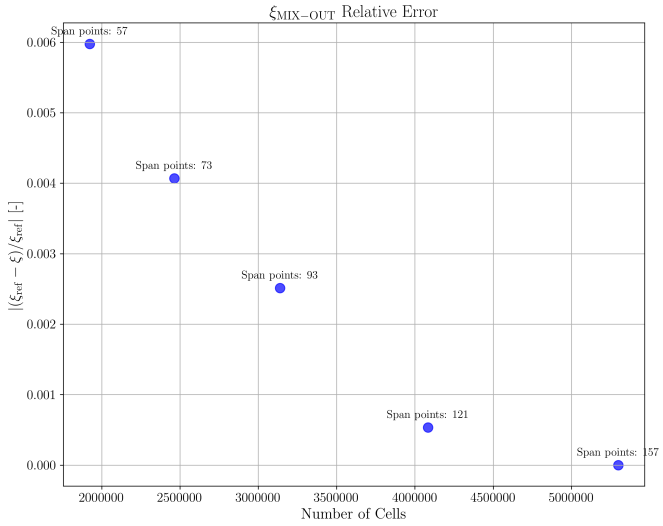


Figure 3.33: TTLT: Relative error for plane-wise averaged values ξ.

The results obtained are coherent with theory: thin and turbulent inlet boundary layer on low-thickness and low-turning blades are "beneficial" if secondary flow-induced losses are analysed. Vortex cores at the outlet plane, in fact, result to be significantly closer to the endwall at a span height of $x = 0.2\ C_{ax}$ and less driven towards the passage at a $y = -0.2\ C_{ax}$ pitch-wisely. Therefore, this confirms that when inlet boundary layer is thinner, therefore characterised by higher level of normal vorticity than a thicker one, but relegated closer to the endwall, less fluid with normal vorticity enters the cascade and then being proccessed. As a consequence, smaller HSV is formed at blade LE, hence lower momentum PV developing in the passage and "climbing" the blade surface, hence smaller overall secondary flows developed.
In Fig. 3.34, the total pressure contours extracted in the observed stream-wise plane cuts are portrayed, through post-processing in-house software $CFView^{TM}$.

3.8.4 Final mesh configuration: test

This preliminary numerical investigation primary objective is to choose the most suitable mesh configuration to serve as mesh template and further to be employed as reference when comes to mesh a certain dataset of blades. Therefore, it needs to deliver a numerical solution within a certain range of acceptability for almost every point constituting the dataset. As this study has outlined, high turning, high-thickness blades with thick laminar-like inlet boundary layer require higher mesh density due to amplification and wider spatial distribution of secondary flow structures throughout the domain. Consequently, *LTHT*

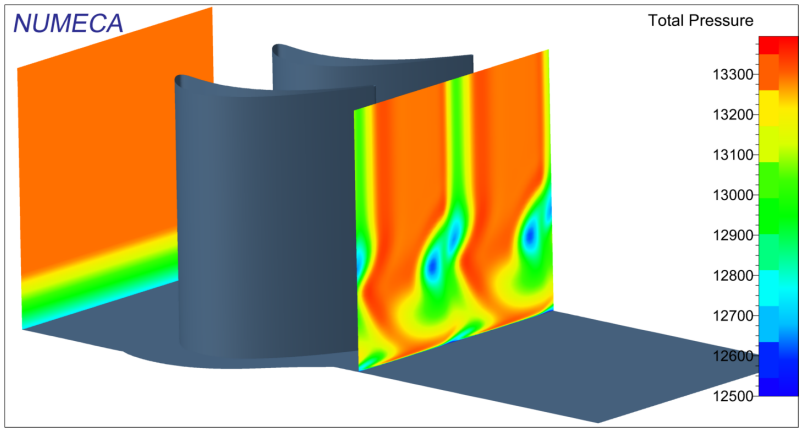


Figure 3.34: Total pressure contour of inlet plane and $z = 1.5$ $C_{\mathbf{ax}}$ in $CFView^{TM}$.

mesh configuration resulted to suit most the final mesh template, as it represents one extreme of a potential dataset generated through the blade airfoil tool optimizer, whereas $TTLT$ figures to be its opposite counterpart.

For this reason, to conclude this part of the project, a "cross-test" has been performed in order to evaluate how the $TTLT$ case geometry behaved on to the template mesh, despite their parameters has been chosen in order to show a consistent similarity between them. For this purpose, a Python script that creates a *.geomTurbo* file has been created, enabling its integration into the meshing process scripts of the tool. Once obtained a functional geometry file, the cross-test mesh has been created by leveraging an optimisation-oriented functionality of IGG^{TM} which favours the generation of a mesh *.igg* file using the geometry file of the desired blade and the *.trb* project file of the mesh used as template, only by a command line process. This approach will be further detailed and implemented in the automated workflow.

Fig. 3.35 shows the results of the $TTLT$ original mesh against the low turning blade with the template mesh. Overall, it performed satisfactorily: kinetic energy loss averaged value got over-estimated of an $\approx 2.5\%$ if compared to the original configuration (see Fig. 3.36). In addition, no remarkable differences are detected in terms of ξ profile distribution along the span, as in Fig. 3.35.

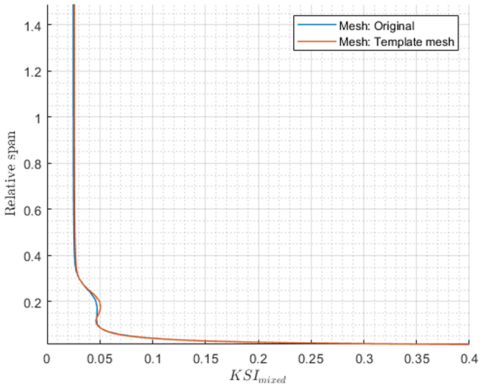


Figure 3.35: Cross-test: ξ (KSI) pitch-wise averaged profiles

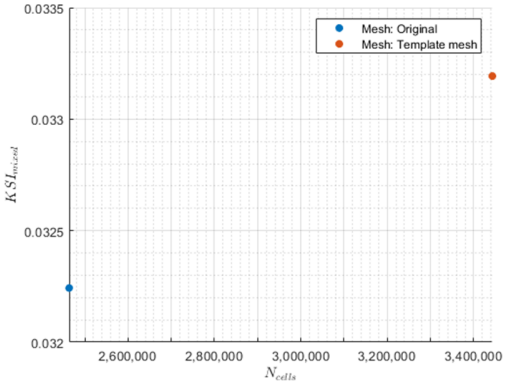


Figure 3.36: Cross-test: ξ (KSI) plane-wise averaged values

3.9 Automated process

The definition of the final template mesh consisted in the last step before the tool could effectively be deployed. In fact, as previously discussed, the main objective of the tool is to conduct RANS three-dimensional simulations and post process them, starting with just 2D blade airfoil shapes, generated by an optimization tool. It performs this task with selected design parameters as inputs, that realise the desired blade loading. In particular, these are classified in terms of aerodynamic "duty" - α_1, α_2, M_2, Re, hence flow quantities that play a role in determining the machine performance-; and "style", whose values shape the loading distribution. Among these, $M_{\mathbf{peak}}/M_2$ - the non-dimensional maximum Mach number reached on SS-, and $L_{\mathbf{peak}}$ - the stream-wise position of the velocity peak. In addition, a coupled inviscid-boundary layer integral resolution simulation on *MISES* is performed after the geometry is created. It delivers the real loading distribution, along with the real outlet flow angle α_2, compared with the expected value through *angle error* and its RMS value. Their values measure the new blade's loading prediction accuracy.
Since this blade optimizer is embodied into the main tool, ranges of these parameters are requested as first user inputs, and will build up the dataset structure. Afterwards, each blade airfoil coordinates file is generated along with the output results file of the related *MISES* calculation, and placed in a dedicated directory.

```
========= GENERATING BLADE DATA =========
Running optimized 2D blade generator...
=======================================
BLADE GENERATOR - PARAMETER CONFIG
=======================================
Enter parameter ranges
Alpha1 [deg] range:
  Min: -40  Max: -20
Alpha2 [deg] range:
  Min: 50   Max: 60
M2 range:
  Min: 0.4  Max: 0.6
Lpeak range:
  Min: 0.4  Max: 0.6
Mpeak / M2 range:
  Min: 1.1  Max: 1.4
```

```
Selected parameter ranges:
Alpha1 [deg]: [-40, -20]
Alpha2 [deg]: [50, 60]
M2: [0.4, 0.6]
Lpeak: [0.4, 0.6]
Mpeak / M2: [1.1, 1.4]
```

Moreover, a table file resuming all parameters chosen for each blade is produced automatically - `blade_data_file.dat` - as the example below. This builds up the basis of the final dataset structure.

```
BLADE N. Mpeak / M2 Lpeak alpha1 [deg] alpha2 [deg] M2 Angle error [deg] RMS
1 1.100 0.400 20.0 -60.0 0.400 0.178 0.262765
2 1.400 0.400 20.0 -60.0 0.400 0.589 0.275358
3 1.100 0.600 20.0 -60.0 0.400 0.540 0.290181
4 1.400 0.600 20.0 -60.0 0.400 0.197 0.315698
5 1.100 0.400 20.0 -60.0 0.600 0.000 0.259629
.
.
32 1.400 0.600 40.0 -70.0 0.600 0.366 0.274152
```

This configuration settles the starting point for the actual tool workflow. In particular, starting from a "raw" airfoil coordinates file, performing a three-dimensional CFD analysis and post-processing data by evaluating losses and loading parameters proposed by literature in order to assess secondary-flow-induced losses correlation with the starting parameters early mentioned. This goal is pursued by the tool ability to automate this

entire process, being able to elaborate a combination of different blades configurations, and arrange a dataset that could be handled with any data-driven algorithm to fulfil this objective. The tool implements a modular architecture based on Python processors, each thought to handle every phase of the process hitherto outlined. This arrangement enables separation of duties and enhances maintainability.
The main script that orchestrates the entire workflow is `blade_generator_main.py`, in particular its function `process_blades()`. It begins with loading the table containing blades configurations. Then all modules involved are initialised:

- *BladeProcessor*
- *BCProcessor*
- *IGGProcessor*
- *FileProcessor*
- *MeshProcessor*
- *ComputationProcessor*
- *PostProcessor*
- *ProcessLogger*

Each module functionality is resumed in Table 3.7. The preliminary step before the process begins is capturing **user inputs**. In fact, the blade height normalised with axial chord h/C_{ax} is set as input, along with the actual axial chord C_{ax} value in meters - to give dimensionality to the problem. Moreover, the outlet Reynolds number is also set as external parameter, as the output example shows.

```
=== User input values  ===
Insert a value for half blade span normalized to Cax (MIRROR!): h/Cax =  1.5

Insert a value for Re2_is: Re2_is = 1200000

Insert a value for axial chord Cax [m]: 1
```

Afterwards, the user has to specify the desired inlet boundary layer parameters. The tool is thought to elaborate the profiles from both n - power law exponent - or integral parameters, expressed as ratios, as in [5], δ^*/δ and θ/δ, along with the thickness δ,. In

Table 3.7: Summary of Blade Generator Tool Modules

Module	Main Functionalities	Key Methods
`bc_processor.py`	Computes boundary conditions, creates boundary layer profile, creates the new project file *.iec*.	`handle_bc_selection()`: Determines boundary condition type based on boundary layer input configuration. `process_boundary_conditions()`: Executes BC computing dedicated script.
`blade_processor.py`	Processes selected blade geometries, closes TE, creates CAD *.IGES* files in dedicated directories.	`get_valid_blade_file()`: Reads each blade geometry file (previous tool output). `load_script()`: Dynamically loads blade *.IGES* CAD creation script.
`computation_processor.py`	Creates *.run* files, sets up and launches parallel computations, monitors *.std* residuals files and handles errors (logging).	`generate_run_file()`: Creates simulation *.run* files for *FINE/Turbo*^{TM} solver. `set_parallel_computation()`: Configures MPI settings for parallel execution. `monitor_std_file()`: Checks computation progress and detects convergence status. `retry_computation()`: Attempts to recover from failed simulations with proper restart.
`file_processor.py`	Generates *.geomTurbo* files, manages directories, monitors disk space.	`generate_geomturbo()`: Creates *.geomTurbo* files from templates and *.dat* files. `prepare_geomturbo_files()`: Sets up all necessary *.geomTurbo* files and directories. `check_disk_space()`: Ensures sufficient machine disk space for operations.
`igg_processor.py`	Modifies IGG scripts, runs domain creation, manages blade output files for mesh creation.	`modify_igg_script()`: Updates IGG scripts with current geometry parameters. `run_igg_processing()`: Executes domain creation and geometry processing operations.
`mesh_processor.py`	Runs mesh generation, sets boundary patches through *IGG*.	`generate_mesh()`: Creates computational mesh using *AutoGrid5* with templates. `set_mir_shroud()`: Sets MIRROR boundary conditions on shroud surfaces. `set_mesh_to_project()`: Updates project files with mesh information.
`post_processor.py`	Executes post-processing, computes performance metrics, performs losses breakdown.	`execute_post_processing()`: Coordinates extraction of flow field data in *CFView*. `compute_loading_parameters()`: Calculates Zweifel coefficient, diffusion factor, circulation coefficient etc. `compute_loss_analysis()`: Determines profile, and gross kinetic energy loss coefficients. `compute_SKE_analysis()`: Analyses secondary kinetic energy distribution along the passage.
`bc_config_utils.py`	Reads configuration files, validates parameters, provides boundary layer settings.	`validate_config()`: Ensures parameters exist and are within valid ranges. `get_bl_method_parameters()`: Returns appropriate boundary layer method parameters. `read_bc_config()`: Reads configuration for specific blade from file.
`process_logger.py`	Handles logging of blade generation process steps, errors, and warnings with timestamps and blade identification.	`log_step()`: Records process steps with success/failure status and timestamps. `log_main_error()`: Records detailed error information with blade identifiers. `log_warning()`: Documents warning messages during processing.

addition, the value of turbulence intensity is requested as input for the successive project boundary conditions setup:

```
=== Boundary Layer Configuration Setup ===
Boundary layer type (1 for laminar, 2 for turbulent): 1
Enter delta thickness (normalized with Cax): 0.2
Express boundary layer using [1] n (power law) or [2] ratios? (1/2): 2
Enter delta*/delta ratio: 0.379
Enter theta/delta ratio: 0.145
Enter Tu value (default 5.0): 5
```

It is also possible to read all these inputs from a file, in order to reduce the user I/O interaction.

The following step involves generating boundary conditions inputs for each blade within the group. More specifically, as detailed further, *BCProcessor* scripts accept a specific input list of values to compute boundary conditions and create the computation project. Therefore, `bc_config.py` is called to write `bc_config.dat` file containing inputs to perform correctly the subsequent steps. Furthermore, each blade is subject to a "quality-check" in terms of angle error and RMS values produced by the blade optimizer. In order to accomplish this, a file containing user-predefined threshold values is read and compared against each blade's respective values during an initial inspection. If either the angle error or RMS value exceeds the threshold, a flag '`X`' is assigned, meaning that configuration is being discarded. Otherwise, the blade is labelled with an '✓', meaning that it can be processed. The blade configuration table is then updated by generating the `.DATAinput` file, formatted to include relevant aero-thermodynamic quantities, user-defined inputs, and, where applicable, the quantities computed during post-processing. This structured dataset corrisponds to the one thought to be suitable for data-driven analyses and optimization processes. The specific thresholds are advised by blade optimizer's tool author basing on tool reliability limiting errors propagation that can lead to an unfeasible test case:

```
RMS = 0.5
angle_error = 1.0
```

The selected configurations are those considered suitable to proceed with the analysis. Thus, the initial processing phase get started, and it is operated from *BladeProcessor*. In an iterative process within the selected blades, it manages the Python scripts realised to close TEs, perform a linear extrusion of the blade by the user-specified h/C_{ax}, optionally scale the dimension of C_{ax} and generates the CAD geometry converting it into an *.IGES*

format. Each processed blade is then stored in its respective directory. Moreover, during the airfoil handling, the value of the stagger angle γ is computed and stored into the `.DATAinput` file, since essential to compute isentropic flow quantities based on the real chord instead of the axial one.

Afterwards, another cycle is performed within the group in order to establish the feasibility of the chosen span-to-chord ratio in terms of secondary flows penetration height Z_{TE} according to the empirical relation formulated by Sharma & Butler in 2.8. Hence, inlet and outlet flow quantities are computed for every blade, and the expected penetration at TE is then evaluated taking into account a safety coefficient of 1.5. If at least one case does not satisfy this constraint, the maximum value of $h/C_{ax} = 1.5 \cdot Z_{TE}$ within the configurations is kept. Hence, all the process of 3D blade generation hitherto outlined is then repeated. Otherwise, the workflow proceeds, entraining the **geometry and mesh generation phase**.

It is coordinated by *MeshProcessor*, *IGGProcessor* and *FileProcessor*, with the main objective of generating the mesh file in *IGG* and *AutoGrid*5 environment. This process is anew carried out within an iterative cycle over the set of selected blades. In particular, it is designed to first check whether the directory containing the mesh files includes a folder corresponding to each blade number, because in this case it detects that a mesh file already exists. If the folder is not present, the mesh generation process starts; otherwise the workflow proceeds skipping meshing process. It is noteworthy that if the mesh needs to be created from scratch, the procedure followed is analogous to that outlined in the previous sections. The CAD *.IGES* file is imported into *IGG*, domain extension lines are created and, since the objective is to generate the *.geomTurbo* geometry file, all CAD "entities" - curve, lines and surfaces, (*i.e.* hub and shroud lines, leading edge and trailing edge lines, suction and pressure side curves, and the relative blade surfaces-, are stored in a proprietary formatted file marked with *.dat* extension and saved in a dedicated directory for these type of files using the `save_geometry_entities()` software-built function. This is performed by `create_domain_script.py` within `igg_processor.py`, which is run via command line, thanks to the *-script* functionality, through the batch subprocess Python function:

```
command = igg -niversion 182 -real-batch
-script {create\_domain\_script.py} -print

process = subprocess.run(
    [command],
    shell=True,
    capture_output=True,
    text=True,
    executable='/bin/bash'
)
```

The file obtained is necessary to create the dedicated *.geomTurbo* file. Since no built-in function allowed to fulfil this, a Python script was then realised. After an accurate analysis of the proprietary file structure, a similarity with the already generated file *.dat* was identified. In particular, after a series of header lines identifying the *.geomTurbo* file and the specification of the geometry properties - such as the cascade pitch value - the file proceeds with listing the geometry coordinates of all entities that compose the blade geometry, just like the *.dat* file. Leveraging this similarity, the developed code takes a *.geomTurbo* template file as input, replaces the pitch value (which is read from the $MISES$ file located in the input geometry directory), and substitutes the entities coordinate section with the data from the previously created *.dat* file. As a result, a new *.geomTurbo* file is generated, which can then be used, together with the template, to generate the final mesh file. Everything above-mentioned is executed with *FileProcessor* in `file_processor.py`, in particular with the Python functions `generate_geomturbo()` and `prepare_geomturbo_files()`. The last step leverages the software functionality, specifically *AutoGrid5*, to generate a mesh *.igg* file from a template project file *.trb* - which is the one chosen from the grid convergence study - , the new geometry contained into the *.geomTurbo* file and the designated directory for saving the generated mesh file. In particular, launching via command line:

```
igg -autogrid5 -real-batch -trb {mesh_templ_trb}
    -geomTurbo {geomturbo_file}
    -mesh {self.new_mesh} -print
```

This procedure is performed within the blade configurations in order to obtain mesh files for each of the group component. The command `-print` is employed to display the entire process in real-time, allowing the user to monitor progress and identify any potential failures. Once the mesh file is successfully created, a dedicated Python script checks for

the presence of all blade mesh files into its designated directory. To finalise the process, a macro script on *IGG*, `set_mir_shroud()`, is then executed. It serves to apply the "MIR-ROR" boundary conditions to those domain patches linked with the shroud, ensuring that only half-blade domain is analysed. When the whole set of configurations is processed, the meshing phase is considered completed. It is important to specify that it has been performed in a separate iterative cycle in order to optimize time-efficiency. In fact, performing mesh generation sequentially - one blade at a time, followed by computation - would have significantly increased the processing time.

The successive step involves boundary conditions calculation, by means of *BCProcessor* in `bc_processor.py`, and the *FINE/Turbo*^{TM} calculation project setup. In particular, the cycle re-iterates within the configurations by reading the input values needed for boundary condition processing from the configuration file initially created. As previously outlined, it contains: inlet boundary layer type, thickness δ normalised with axial chord, velocity ratio profile parameters defined as power-law exponent n or integral boundary layer values, expressed as ratios with thickness, and turbulence intensity value, for each blade. The tool efficiently manages multiple configurations through the function `handle_bc_selection()`. Afterwards, `process_boundary_conditions()` function is executed. It recalls the aforementioned Python scripts useful to compute all inlet and outlet quantities (`generate_radial_bc.py`), by giving input values: $T^o, Re_{2_{is}}, M_2, \alpha_1, \alpha_2$. Furthermore, the integral boundary layer quantities are computed with distinguished approaches based on inputs, and then the shape factor is computed $H_{12} = \delta^*/\theta$. Additionally, the user is given the option to define initial conditions either by specifying velocity components, pressure, and temperature values or by copying the already computed boundary condition values. This flexibility is particularly beneficial for unsteady simulations, where initial conditions strongly influence solution stability, whereas for steady computations, they primarily affect the rate of convergence. Afterwards, the velocity ratio profile is built, and subsequently the profile of $P^o(x)$ is obtained. It is imposed that the first point outside the wall corresponds to the first wall cell center, determined by the computed *y cell width*. It is then stored as *Ptot.p* coordinate file, a format readable by the solver. Upon storing all relevant quantities and profiles, the script `input_radial_inlet.py` is executed to generate the computation project file relative to the case considered. It serves to compile a *.iec* project template file, which results to be the *LTHT* computation file. This is achieved by leveraging a prior analysis that determined the appropriate placement of boundary condition values, turbulence parameters, and initial conditions within the file. Hence, a dedicated function has been developed for this purpose, `modify_file_with_coordinates()`.

This code snippet helps to understand how this is performed. The computation name is built by incorporating: **blade number**, the **boundary layer type**, the **value of boundary layer thickness**.

For instance, in the case of blade number 22, with a laminar inlet boundary layer characterised by $\delta/C_{ax} = 0.2$ is labelled as *22_Lam_02000_radial*. This unique identifier is consistently used throughout the analysis to reference this specific case. Subsequently, a dedicated folder for the case is created inside `Project_files` directory. Then the template file *RADIAL_INLET_TEMPL.iec* is accessed, and all necessary modifications are applied to generate the new blade computation project.

```
#Prepare and Modify the Template File
lines_to_modify_ptot_bc = {
    2368: read_coordinates(coord_files['pt'])
}
lines_to_modify_bc = {
    173: V1,
    175: rho1,
    1770: P2,
    2061: Vx_V,
    2091: Vy_V,
    2224: Vz_V,
    2480: Ttot,
    2613: k,
    2634: epsilon,
    2676: Tu,       #Tu for SST

    2655: nu_t,
    2802: Tu        #Tu for S-A
}
lines_to_modify_ic = {
    1731: t_init,
    1733: v_y_init,
    1734: v_z_init
}
delta_formatted = f"{int(delta * 10000):05d}"
computation_name = f"{case}_{bl_type}_{delta_formatted}_radial"
lines_to_modify_name = {3: computation_name, 20: computation_name}
output_file = modify_file_with_coordinates(
    TEMPLATE_DIRECTORY, TEMPLATE_FILE_NAME, output_folder_path,
    computation_name, lines_to_modify_bc,
    lines_to_modify_ptot_bc, lines_to_modify_ic,
    lines_to_modify_name
)
```

Once the *.iec* file is successfully created, last function of `mesh_processor.py`, `set_mesh_to_project()`, executes a macro Python script that accesses the project file through *FINE/Turbo*^{TM} and links the project to the relative mesh previously created. This is a key passage for the correct execution of the analysis. Afterwards, *ComputationProcessor* is first called to duty in order to generate the *.run* file, starting with the project and the mesh of the blade configuration in exam, via analogous command line feature :

```
# Create computation path with correctly formatted name
self.computation_path = os.path.join(self.project_path,
f"{self.file_name}_computation_1")
command = f"fine -niversion 182 -batch -project
{self.output_path} -print
```

The *.run* file is created within the project dedicated directory; however, configured with standard computational settings, employing a sequential process (*e.g.*, a single-core execution). Given the three-dimensional analysis and the available computational resources, a parallelised computation has been preferable. As a consequence, `set_parallel_computation()` function executes an additional command line process to ensure that the *.run* file is configured for parallel execution, leveraging Intel MPI technology.

```
self.computation_path = os.path.join(self.project_path,
f"{self.file_name}_computation_1")
self.run_file_path = os.path.join(self.computation_path,
f"{self.file_name}_computation_1.run")
# Set comp_file before using it
self.comp_file = f"{self.file_name}_computation_1"

# Number of processors for parallel computation
nproc = 7
print(f"\nNumber of processors: {nproc}")
command = f"fine -niversion 182 -batch -partition -load_balancing
0.85 -computation {self.run_file_path} -nproc {nproc} -print"
```

Specifically, the user has the flexibility to select the desired number of processors, but for the scope of this thesis a fixed configuration of **seven** processors per computation has been adopted, with no further scalability analysis conducted.

This iterative process terminates once every blade within the group has been successfully assigned its corresponding project. Before entraining the computational phase of the automated process, particular mention must to be given for *ProcessLogger*, a processor thought to provide a detailed error logging throughout every step of the aforementioned workflow, including the subsequent computational steps. In particular, it generates a *blade_process.log* file, which systematically records feedbacks on each process, aiming to make the user aware and to address any failure. This can be valuable when comes to handle a large number of cases, where individual monitoring becomes impractical. Hence,

within the iterative cycles, every successfully completed or failed process is logged, as the example below:

```
[2025-03-04 09:46:03] [Blade 1 - blade_1] Mesh Generation: ✓ SUCCESS
[2025-03-04 09:46:09] [Blade 1 - blade_1] Patch Setting: ✓ SUCCESS
[2025-03-04 09:46:09] [Blade 1 - blade_1] Project Setup: ✓ SUCCESS
[2025-03-04 09:46:21] [Blade 1 - 1_Lam_02000_radial_computation_1]
                                Computation Setup: ✓ SUCCESS
```

The core of the automated process lies in the computational analysis, performed by *ComputationProcessor*. With the objective of optimising efficiency, this phase does not follow an iterative approach; instead, all configurations are divided into groups. The grouping criteria are determined by the *ResourceManager*, a class that is designed to monitor the available disk space in the machine - giving that every computation requires $\approx$ 5 GBs, considering *LTHT* case as reference - and to track the available CPU cores in the machine. These are compared against the pre-defined parallelisation settings for the project, and eventually it limits the number of parallel computations expected to run simultaneously. If no cores are available at a given moment, the script is able to wait for sufficient resources before launching a group of computations, so preventing system overload. Afterwards, each group is processed. In particular, `launch_computation_with_monitoring()` is executed. Here, it leverages the functionality of the solver to launch CFD computations via command line thanks to the execution of a *.batch* file generated along with the *.run* file previously mentioned. In fact, the function is thought to identify every group's blade batch file path, then it ensures that the *.batch* file is built up in order to execute the process in parallelised Intel MPI framework, and successively launches the process through command line subprocess:

```
batch_file = os.path.join(self.computation_path, f"{self.comp_file}.batch")
self._update_batch_file(batch_file)
command = f"cd {self.computation_path} && ./{self.comp_file}.batch"
```

Soon after the computation is launched, `monitor_std_file()` function is invoked. It is thought to give a real-time feedback on the computation residuals. In fact, when the simulation is launched, the solver automatically generates an *.std* file, whereto instantaneously writes the updates of the computation ongoing, showing the residuals. Hence, the function opens separate terminal emulators consoles per each computation ongoing, that serve to visualize and monitor the related *.std* file:

```
# Try using different terminal emulators
terminal_commands = [
f"gnome-terminal -- /bin/bash -c 'tail -f {std_file}; read -p
\"Press enter to close...\"'",
f"xterm -e 'tail -f {std_file}; read -p \"Press enter to close...\"'",
f"konsole -e 'tail -f {std_file}; read -p \"Press enter to close...\"'"
]
```

Moreover, `launch_computation_with_monitoring()` performs a tracking on the residuals for each process by parsing the *.std* file and printing on the terminal the current iteration count with the relative residual value providing periodic updates status. In addition, a convergence detection routine is implemented: when the current iteration reaches **700** - a threshold determined based on prior experience with computations of similar "nature", employing the same numerical methods and settings-, the routine checks the last 50 lines of the file, parses the residual value and calculates their standard deviation. If it remains lower than 0.1 for three consecutive checks, the computation is considered converged, forcing its termination. This procedure ensures time-saving efficiency by preventing further iterations once the stationary solution is reached. Whereas, if the computation fails due to a numerical divergence - normally driven by negative pressure/density and mass flow values in the domain - the function `retry_computation()` is recalled. It specifically deletes the computation directory, re-executes the aforementioned steps for its creation, and launches the process for a second time. By the time that the second attempt fails, a warning log is written in *blade_processing.log* file. In addition, a macro script in *CFView* is then executed to generate and save contour screenshots of Total pressure and velocity vector fields in the corresponding folder within `Results` directory. This is thought to provide the user a visual feedback on the simulation's failure, aiding in the identification of potential issues such as flow separation on the blade, incorrect incidence angle, or other flow anomalies that could contribute to numerical divergence. Since, based on the experience during training, this

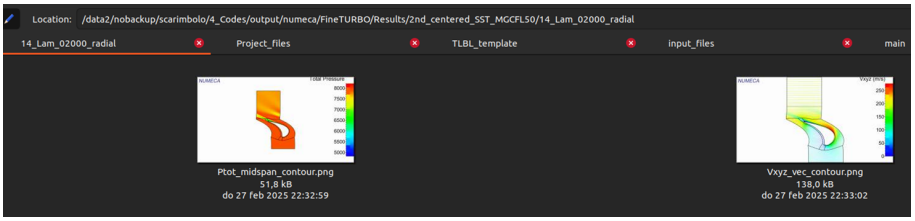


Figure 3.37: Example of contour plots saved in computation results folder for debugging.

kind of problem statistically had mesh-related problem sources, hence the log provided:

```
[2025-03-04 18:35:51] [Blade 26 - 26_Lam_02000_radial_computation_1]
2nd attempt: CHECK MESH!!!: ✗ FAILED
[2025-03-04 18:35:54] [Blade 26 - 26_Lam_02000_radial_computation_1]
Contour debug after 2nd attempt failure completed successfully: ✓ SUCCESS
[2025-03-04 19:03:56] [Blade 27 - 27_Lam_02000_radial_computation_1]
Computation reached final_output: ✓ SUCCESS
```

When groups are processed, each computation gets categorized based on its outcome, either as **successful** - `success_results()` - or **failed** - `failed_blades()`. This classification allows the tool to efficiently track which computations have been successfully completed and are ready for post-processing.

```
if result == "FINAL_OUTPUT" or result == "CONVERGENCE":
    # Successful completion
    logger.log_step(logger.main_log_file, f"Computation reached
    {result.lower()}{residual_info}", True)
    success_results[blade_num] = True
elif result == "ERROR":
    # Failed computation
    failed_blades.append(blade_num)
```

Once one group has been correctly processed, the last step of the tool begins: **post-processing**. For this purpose, *PostProcessor* Python class has been created. It is specifically designed to extract key flow quantities by means of a macro script in *CFView* environment. Subsequently, Python scripts elaborate these data by computing blade loading coefficients and averaged losses, then stored and saved in dataset *.DATAinput* file, ensuring a comprehensive and organised record for further analyses.
The flow fields are extracted at various planes within the computational domain. A meridional cut at mid-span is performed to visualise the two-dimensional flow field. Additionally, multiple span-wise plane cuts are performed:

- **Inlet plane:** $z = -0.8C_{ax}$
- **Outlet observation plane:** $z = 1.5C_{ax}$
- **Blade passage planes:** 10 cuts from the leading edge ($z = 0C_{ax}$) to the trailing edge ($z = 1C_{ax}$), with a step of 0.1.
- **Downstream planes:** 5 additional cuts from $z = 1.05C_{ax}$ to $z = 1.5C_{ax}$.

At each of these measurement planes, the following flow plane surface contours are extracted:

- Velocity components (V_x, V_y, V_z)
- Velocity magnitude
- Static pressure
- Total pressure
- Static temperature
- Total temperature
- Density
- Mass flow
- Flow angles (yaw and pitch)

Additionally, on the blade surfaces, the following quantities are extracted:

- M_{is} and $P_{\mathbf{wall}}$ distributions along PS and SS at mid-span
- M_{is} and $P_{\mathbf{wall}}$, and entropy contour fields on the **endwall**.

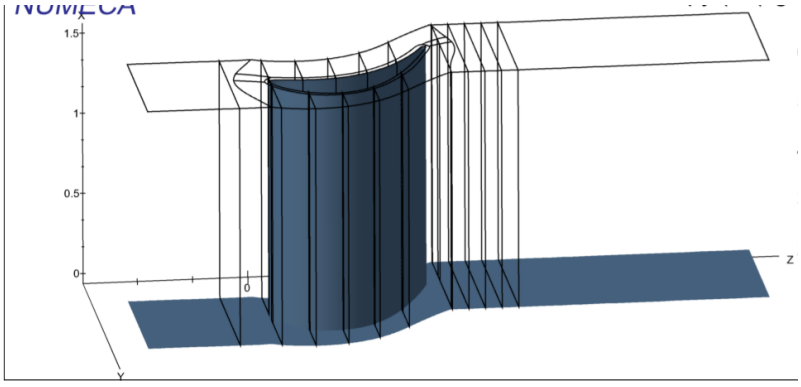


Figure 3.38: Plane cuts visualised in *CFView*.

Every mentioned field data are stored in *.dat* files, which are saved in the relative computation folder within `Results` directory. Analogously to other processes executed in *IGG* or *FINE/Turbo*, the macro script is executed by the function `execute_post_processing()`, via terminal command process in *CFView*:

```
cfview -macro {macro} -batch -niversion 182 -print
```

Afterwards, a logging feedback is hereto also implemented to provide the user with status updates regarding the extraction process. Once the extraction is completed, the script verifies the presence of all expected output files within the folder. If at least one of the expected files is missing, a warning message is issued to alert the user. Furthermore, in the event of a failure during macro execution, the error is reported, ensuring that the user can address any issues effectively.

```
----------------------------- POST-PROCESSING -----------------------------
Extracting quantities...
CFView macro execution completed.
Checking results in: /data2/nobackup/scarimbolo/4_Codes/output/numeca/
FineTURBO/Results/29_Lam_02000_radial
All required files are present in the results directory.
```

Subsequently, the performance analysis is then performed. In particular, it starts with the computation of 2-D blade loading parameters, outlined in [4]:

- **Diffusion factor**:

$$DF = \frac{V_{\mathbf{SS_{MAX}}} - V_{\mathbf{TE}}}{V_{\mathbf{TE}}}$$

  it accounts for ratio of fluid deceleration along SS and when comes to TE. Higher values mean stronger decelerations, leading to high adverse pressure gradient, hence risk of separation;

- **Zweifel coefficient**:

$$Z_w = \oint C_p \, d\left(\frac{x}{C_{ax}}\right)$$

  expressed as the integral value of the pressure coefficient along the blade. It is employed in turbomachinery design to account for blade loading, since it provides a measure of the tangential force on a blade compared to an ideal case. Though, Coull observed that it is actually highly dependent on flow angles, varying of $\approx 40\%$ within the (α_1, α_2) design space with similar SS C_p distribution but different PS velocities.

- **Circulation coefficient**:

$$C_o = \oint \sqrt{C_p} \, d\left(\frac{S}{S_{SS}}\right)$$

proposed as an alternative loading coefficient, since the square root establishes more connection to the velocity field and it appears to give more appropriate weight to SS aerodynamics, recording a less marked variation with flow angles.

- **Non-dimensional transit time**: defined in Eq.2.3, Coull [4] highlights, by recalling Hawthorne's secondary flow theory, that it plays a key role in secondary flows formation, hence contributing to endwall losses. It accounts for flow particles paths along PS and SS differences in time. When this difference is large, hence mainly when pressure velocities are markedly different from suction ones, more significant flow misalignment occurs when particles merge downstream. In addition, for higher values, PS velocities are low: the flow is then more susceptible to blade-to-blade pressure gradient, leading to stronger secondary flows.

For every computation processed within the group, the function `compute_loading_parameters()` is called from *PostProcessor* class. It computes the aforementioned parameters and saves them into the dictionary `loading_params`.
Subsequently, kinetic energy loss coefficients are computed. As proposed in the grid convergence study, as widely acknowledged in literature, a valuable way of assessing machine performances in design phase is the enthalpy (or energy) loss coefficient, which can be expressed in terms of kinetic energy through isentropic relations, as in 2.1. This way, compressibility effects are taken into account, rather than only evaluating the difference of inlet and outlet stagnation pressure. Following the approach of [12], the overall gross loss $\xi_{\mathbf{gross}}$ is computed at the outlet plane with mixed-out averaging technique, to account for mixing losses. Since it encompasses the effects of inlet boundary layer flow momentum defection, the loss generated inside the passage and the profile loss $\xi_{\mathbf{profile}}$ at midspan, all these contributions are then computed in order to isolate the net endwall loss. In particular, the inlet boundary layer loss $\xi_{\mathbf{BL}}$ is computed similarly to 2.1, but instead of considering the isentropic value of the stagnation pressure, the one at the outlet is selected. In this case, the planar field contours are then mass-averaged - but only the convective quantities, such as P^o, whereas static pressure P is area-averaged. Eventually, the net endwall loss - comprising with downstream mixing loss - is obtained by subtracting these contributes from overall gross loss:

$$\xi_{\mathbf{net}} = \xi_{\mathbf{gross}} - \xi_{\mathbf{BL}} - \xi_{\mathbf{profile}}$$

Hence, a comprehensive breakdown of the losses is provided. Every aforementioned step is performed by `compute_loss_analysis()`. It executes the loss quantities calculation script,

which processes flow data contour files of interest by organising them into easily manageable *DataFrames*, structured with field coordinates (x, y, z) alongside the corresponding quantity values. Subsequently, it performs averaging operations, storing every mentioned loss coefficient into the dictionary `losses`. Additionally, the script is thought to provide a visual representation of the results by creating a dedicated subdirectory within the single computation `Results` folder, and saving a pitch-wise averaged radial profile of $\xi_{\mathbf{gross}}$ and a contour plot of its field at the outlet plane.

Eventually, the last post-processing step involves the execution of *compute_ SKE_ analysis()* function, which runs a Python script designed to compute SKE coefficients $\zeta_{\mathbf{SKE}}$, given Eq.2.2 and mass-averaging velocities. This analysis is applied to each of the previously mentioned plane sections at different axial positions, in order to quantify the magnitude and the distribution of SKE throughout the passage and in the downstream region beyond the TE. This approach follows the methodology proposed by Denton [8], where the local SKE decay - hence dissipation driven by viscosity - that occurs from the TE plane onward is thought to be an indicator of the contribution of secondary flows to the overall stagnation pressure loss. Therefore, the values of the coefficients at different planes are stored into the dictionary `ske_results`. Specifically, the value of $\zeta_{\mathbf{SKE}}\big|_{z=\text{outlet}}$ is selected for the dataset filling. Additionally, plot representing the coefficients distribution is generated (see Fig. 3.39) and saved within the same aforementioned subdirectory, along with the contours and plots, as the example shown in Fig. 3.40. Subsequently, all quantities computed in

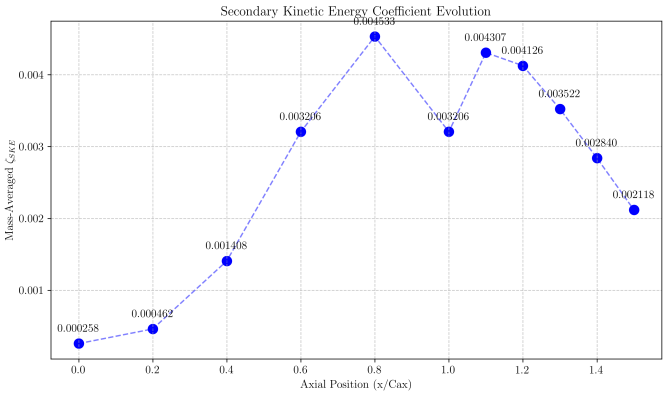


Figure 3.39: Example of $\zeta_{\mathbf{SKE}}$ coefficient distribution plot generated in post-processing.

post-processing, contained in the aforementioned dictionaries, are inserted into the dataset table within the in *.DATAinput* file, as the code snippet below shows:

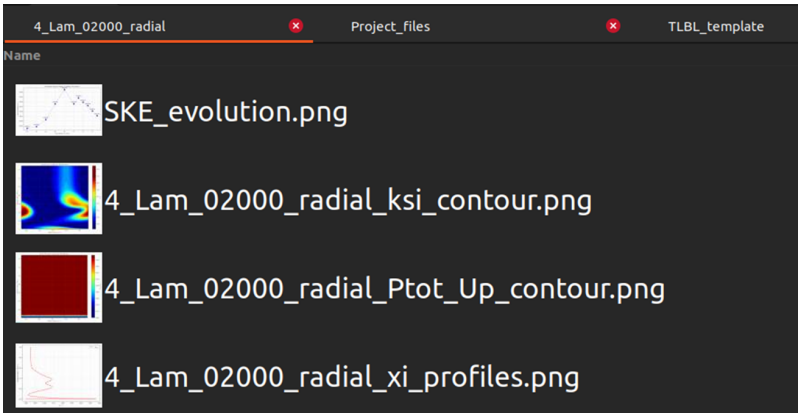


Figure 3.40: Example of post-processing plots and contours saved in computation `Results` subdirectory.

```
if loading_params:
    loading_data = {
        'Zw': loading_params.get('Zw'),
        'DT*': loading_params.get('DT_star'),
        'DF': loading_params.get('DF'),
        'C_0': loading_params.get('C_0')
    }
    update_data_input_file(input_file, loading_data, blade_row_index)
    print(f"Updated .DATAinput with loading parameters for blade {blade_num}")
if losses:
    # Update .DATAinput with KSI values
    ksi_data = {
        '\xi_gross': losses['ksi_mass_planewise'],
        '\xi_gross_mix': losses['ksi_mix_planewise'],
        '\xi_bl': losses['ksi_bl_mass_planewise'],
        '\xi_profile': losses['ksi_profile'],
        '\xi_net': losses['ksi_net']
    }
    update_data_input_file(input_file, ksi_data, blade_row_index)
    print(f"Updated .DATAinput with KSI losses for blade {blade_num}")
else:
    print("Warning: KSI losses computation failed")
# Update SKE results if available
if ske_results and 'ske_plane_15' in ske_results:
    ske_data = {
        '\zeta_SKE': ske_results['ske_plane_15']
    }
    update_data_input_file(input_file, ske_data, blade_row_index)
    print(f"Updated .DATAinput with SKE value (\zeta_SKE) for blade {blade_num}")
else:
    print("Warning: SKE analysis computation failed or plane 15 data not available")
```

Similarly to any other process, the execution of post-processing steps is also equipped with logging. If any fatal error arises due to traceback exceptions in Python executions, they are immediately printed in the main terminal and recorded into the raised from traceback errors in Python executions, they are reported by printing them into the *.log* file. If no error arises and if every required file is found into the `Results` subdirectory, the post-processing is considered **completed**. Eventually, the computation directory is **cleaned up** by removing the simulation files no longer needed in order to save storage usage. The tool is then ready to process the following computations group.
In the following page is reported an output example of the post-processing for a single case and in Table 3.8 is portrayed an example of *.DATAinput* file dataset structure, divided into four parameters subsets.

Table 3.8: Example of *.DATAinput* file: output dataset structure

BLADE	$M_{\mathbf{peak}}/M_2$	$L_{\mathbf{peak}}$	α_1 **[deg]**	α_2 **[deg]**	M_2	**Angle err.**	**RMS**	h/C_{ax}
1	1.1	0.4	40	-60	0.4	0.7	0.2773	3
2	1.4	0.4	40	-60	0.4	0.862	0.3242	3
3	1.1	0.6	40	-60	0.4	0.234	0.3094	3
4	1.4	0.6	40	-60	0.4	0.438	0.233	3

BLADE	$Re_{2_{\mathbf{is}}}$	C_{ax} **[m]**	**Flag**	**Stagger [deg]**	M_1	T_1	P_1	ρ_1	V_1	p/C_{ax}
1	1.2E+06	1	✓	25.578	0.2463	236.4	12744	0.1498	84.935	0.8077
2	1.2E+06	1	×	—	—	—	—	—	—	—
3	1.2E+06	1	✓	24.76	0.2463	236.4	12830	0.1508	84.935	0.8046
4	1.2E+06	1	✓	22.433	0.2463	236.4	13060	0.1535	84.395	0.8344

BLADE	δ/C_{ax}	n	H_{12}	δ^*/δ	θ/δ
1	0.2	1.7246	2.1597	0.3670	0.1699
2	—	—	—	—	—
3	0.2	1.7246	2.1597	0.3670	0.1699
4	0.2	1.7246	2.1597	0.3670	0.1699

BLADE	Z_w	DT^*	DF	C_0	$\xi_{\mathbf{gross}}$	$\xi_{\mathbf{bl}}$	$\xi_{\mathbf{profile}}$	$\xi_{\mathbf{net}}$	$\zeta_{\mathbf{SKE}}$
1	1.0126	1.5763	0.2253	0.3823	0.07251751	0.00506	0.01593	0.05152751	0.0044
2	—	—	—	—	—	—	—	—	—
3	1.0153	1.6023	0.2276	0.3827	0.07183567	0.00513	0.01391	0.05279857	0.0043
4	1.2635	1.7167	0.5001	0.4641	0.07715826	0.00514	0.02782	0.04419826	0.0058

```
Starting post-processing for blade 2...

-------------- POST-PROCESSING ----------------

Extracting quantities...
CFView macro execution completed.

Computing loading parameters...
Parsing computation name: 2_Turb_00500_radial

Loading parameters for blade 2:
  Zweifel Coefficient (Zw): 1.1710
  Circulation Coefficient (C_0): 0.4182
  Non dimensional transit time (DT*): 1.6597
  Diffusion factor (DF): 0.49585
Loading parameters computed successfully

Computing KSI losses...
Parsing computation name: 2_Turb_00500_radial
Plots saved.

KSI Loss Results:
  KSI Mass Planewise: 0.029704
  KSI Mixed out Planewise: 0.030418
  KSI BL Mass Planewise: 0.000407
  KSI Profile: 0.0229545
  KSI Net: 0.00634234

Computing Secondary Kinetic Energy (SKE)...
SKE evolution plot saved.

SKE Analysis Results:
  SKE at plane 15: 0.000158
Updated .DATAinput with loading parameters for blade 2
Updated .DATAinput with KSI losses for blade 2
Updated .DATAinput with SKE value (\zeta_SKE) for blade 2

Checking results in directory...
All required files are present in the results directory
Cleaned up computation directory.
```

Chapter 4

Validation and Data Analysis

In this final section of the Tool development analysis, the focus shifts to the validation stage, where all the hitherto mentioned functionalities are tested, to ensure their effectiveness and bug-less execution. In addition, an evaluation of time performance is conducted to assess the potential impact of the Tool on design phase.
Eventually, an initial data correlation analysis is performed. In fact, due to constraints in time and computational resources, a comprehensive parametric study has not been performed, as exploring a sufficiently large and meaningful design space would have required running thousands of configurations- an impractical endevour. Instead, the preliminary analysis has focused on the following aspects:

- Evaluating the predicted secondary flow-induced loss from Coull [4] model.
- Identifying potential correlations between blade optimization parameters, flow characteristics, and blade loading coefficients with the measured losses.
- Verifying the findings of de La Rosa Blanco [12] and Coull [5] regarding the influence of inlet boundary layer characteristics on endwall losses.

4.1 Dataset corner points: input parameters

The validation process has been carried out by analysing a limited number of blade configurations. This approach allows to raise awareness of any potential bug in the codes, malfunctions or inaccuracies in the results when running different cases, while avoiding an excessive computational cost and a feedback time delay, unnecessary in the development process.

As a matter of fact, a first validation attempt has been performed by selecting **two values** for each parameter defining the blade optimising tool inputs:

- $\alpha_1 = [20^o, 40^o]$;
- $\alpha_2 = [-60^o, -75^o]$;
- $M_{\mathbf{peak}}/M_2 = [1.1, 1.4]$;
- $L_{\mathbf{peak}} = [0.4, 0.6]$;
- $M_2 = [0.4, 0.6]$.

These ranges are advised by the author of the blade optimizer, [21], which deliver feasible blade shapes and minimise the error in predicting loading distribution - $RMS\ error$ - and the outlet flow angle - *angle error*.
Therefore, the combination of these gives as result a total of

$$N_{\text{values}}^{N_{\text{parameters}}} \longrightarrow 2^5 = 32 \text{ cases}$$

Additionally, a single boundary layer configuration has been considered, specifically replicating the turbulent velocity profile from [5]. The corresponding integral boundary layer parameters are given by $\delta^*/\delta_{98} = 0.143$, $\theta/\delta_{98} = 0.111$, hence with a computed shape factor $H_{12} = 1.29$. The boundary layer thickness is set to $\delta_{98}/C_{ax} = 0.05$, similarly to $TTLT$ case. Other user inputs remained consistent with those employed in the grid convergence study, as well as unmentioned numerical methods and boundary conditions - like turbulence quantities. Two different spatial discretization schemes are employed in order to evaluate where better global stability is reached. Table 4.1 resumes the input dataset parameters.

α_1 [deg]	α_2 [deg]	M_{peak}/M_2	L_{peak}	M_2	$Re_{2\text{is}}$	δ_{98}/C_{ax}	δ^*/δ_{98}	θ/δ_{98}	H_{12}	Num. Sch.	$TOT.$
[20, 40]	[-60, -75]	[1.1, 1.4]	[0.4, 0.6]	[0.4, 0.6]	1200000	0.05	0.143	0.111	1.29	Centered	32
"	"	"	"	"	"	"	"	"	"	Upwind/2nd	32

Table 4.1: First dataset parameters for validation

The outcome of the first attempt has already given satisfactory results. On the total number configurations, only **five** geometries failed to mesh, due to drastically different blade geometry obtained compared to template mesh one. Whereas, from the remain valid geometries, the centered scheme delivered better performances in terms of numerical stability, if compared to second-order upwind scheme with Van Albada derivative limiter. In

fact, the first has given 24 cases successively converged, against the 20 of the latter, which resulted to be more subject to numerical instabilities. As a consequence, centered schemes has been employed from this point onward.

A second attempt has been conducted using the same configuration, but with **multigrid initialisation deactivated**. As a result, no computation has encountered numerical divergence, and therefore a total of 27 blade geometries were successfully analysed. Removing multigrid initialisation has lead to more stable simulations, and this can be due to several reasons, both numerical and physical. In fact, initialising with a coarse mesh may cause initial numerical errors to propagate through the domain too rapidly. Consequently, considering high turning blades, with high inlet flow angles, for example, the initial solution may capture non-physical flow separations or mis-predicting complex flow structures, leading to numerical instabilities and affecting the real physical solution.

Afterwards, a third validation attempt has followed. In particular, in order to minimise failures in meshing geometries, the corner point values needed to be narrowed. In particular, it has been decided to change the range of values for the outlet flow angle α_2, resulting in the configuration in Table 4.2.

α_1 [deg]	α_2 [deg]	M_{peak}/M_2	L_{peak}	M_2	$Re_{2\text{is}}$	δ_{98}/C_{ax}	δ^*/δ_{98}	θ/δ_{98}	H_{12}	Num. Sch.	$TOT.$
[20, 40]	[-60, -70]	[1.1, 1.4]	[0.4, 0.6]	[0.4, 0.6]	1200000	0.05	0.143	0.111	1.29	Centered	32

Table 4.2: Second dataset parameters for validation

This has brought to a better outcome: no geometries failed to mesh. Moreover, with multigrid initialisation kept disabled, neither none of the computations failed to converge. As a consequence, **all 32** blades have been successfully processed, and this dataset configuration has been kept as the definitive one. In fact, the final comprehensive validation has been carried out with **two** inlet boundary layer configurations: the turbulent/thin already tested **configuration 1** and the laminar/thick, matching the $LTHT$ test case with integral boundary layer quantities $\delta^*/\delta_{98} = 0.379$, $\theta/\delta_{98} = 0.145$, hence $H_{12} = 2.6$, and a thickness of $\delta_{98}/C_{ax} = 0.3$ - **configuration 2**. Adding two parameters, the total number of combinations raise up to **64**.

α_1 [deg]	α_2 [deg]	M_{peak}/M_2	L_{peak}	M_2	$Re_{2\text{is}}$	δ_{98}/C_{ax}	δ^*/δ_{98}	θ/δ_{98}	H_{12}	Num. Sch.	$TOT.$
[20, 40]	[-60, -70]	[1.1, 1.4]	[0.4, 0.6]	[0.4, 0.6]	1200000	[0.05, 0.3]	[0.143, 0.379]	[0.111, 0.143]	[1.29, 2.6]	Centered	64

Table 4.3: Final dataset parameters for validation

4.2 Time performance estimation

The evaluation of the computational performances serves as a qualitative estimation be interpreted as a qualitative assessment, providing the reader with an order of magnitude regarding the time required for the Tool to perform analyses within a design framework. Indeed, many factors can influence the performance in time, among which the number of processors employed in the CFD simulations play a significant role. In fact, comparing the CPU time values from the test cases reported in the grid convergence analysis could be unfounded if the simulations performed in this circumstance has been boosted up with a higher number of processor allocated. In spite of this, hereafter the tool performances are reported for each stage:

- **CAD generation**: ∼1 second per blade
- **Mesh generation**: ∼300 seconds (5 minutes) per blade
- **Computation setup**: ∼5 seconds per blade
- **CFD analysis**: ∼3,000 seconds (slightly less than an hour) per blade on average (using 8 processors per process)
- **Post processing**: 15s per blade.

For this specific purpose, with a dataset comprising 64 cases, the CAD generation time accounts for approximately a minute, the meshing time takes around ~ 5 hours in total, whereas the computations setup lasts about ~ 6 minutes. The main time load is evidently registered by the CFD simulations: the total number of blades is divided in groups of five. Given that each group requires $\sim 3,100 - 3,200$ seconds to complete, on average, the total duration is estimated to reach up to ~ 11.5 hours. Eventually, the post-processing needs ~ 16 minutes to conclude data analysis. Therefore, the total time consumption for the successful processing of the dataset is estimated to last ~ 17 hours. While this estimation is not intended to be accurate, it provides an order of magnitude of the time required for three-dimensional RANS analyses.

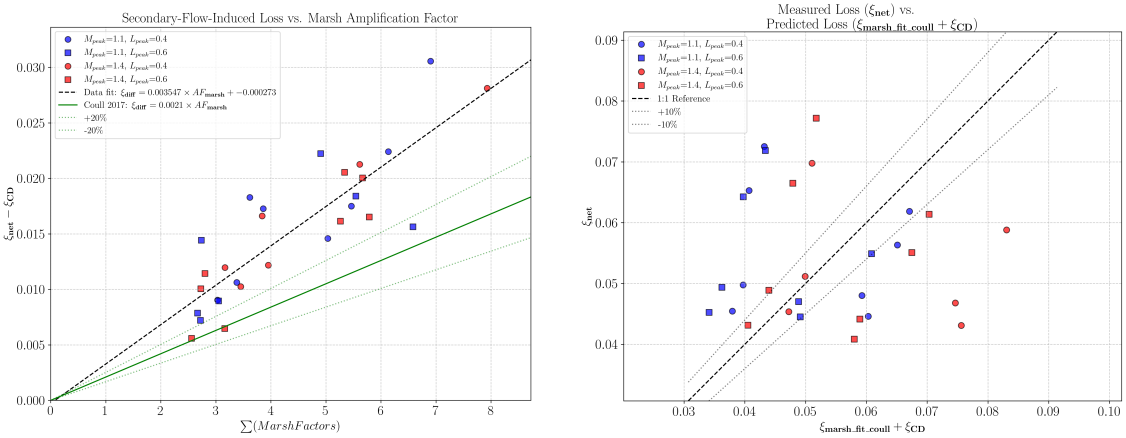
4.3 Data analysis

All the data collected in the structured dataset, contained in *.DATAinput* file at the end of every execution of the tool, is intended to be analysed with the aim of identifying potential correlations between design-oriented parameters - previously identified as key factors in shaping the desired blade loading - and flow quantities against the measured losses. This can be approached through the implementation of machine-learning based stochastic algorithms, capable of building predictive models, performing any type of regression analysis, and in general identifying underlying patterns within the dataset. However, due to time constraints and limited computational resources, conducting a comprehensive study that needed to involve thousands of configurations was not feasible in this work. In fact, it is widely acknowledged that any data-driven analysis is highly dependent on the quality and sparsity of the dataset, in order to enhance its robustness and generalisability. In this particular context, exploring a wide design parameters space is crucial to prevent overfitting to specific cases - when the data employed in training is too "clustered" that the predictive analysis works perfectly with the training points, but its performances drop when using test data. In addition, this is also important to derive meaningful physical and generalisable results.
For the scope of this work. though, the object solely focuses on comparing results obtained in already existent models, identifying any potential correlation between design parameters and measured losses, and eventually justifying the trends of loss coefficients sensitivity to inlet boundary layer profile characteristics.

As outlined in the literature survey section 2, the work of Coull, 2017 [4] offers key insights on the approach to endwall loss and secondary flow-induced loss prediction. In fact, he tries to adapt results found in literature and to find any correlation with design parameters - such as loading coefficients, flow angles, *etc.*

In light of what hitherto mentioned, in the post-processing phase, the computation of the estimated value of background dissipation loss $\xi_{\mathbf{CD}}$ is performed for each blade configuration. This is achieved by extracting the contour fields of isentropic Mach number M_{is} and static pressure P at the endwall, subsequently deriving the free-stream fields $V_{fs},\ T_{fs},\ \rho_{fs}$. These are then integrated over the endwall area and multiplied by an external factor, as in Eq.2.9. Afterwards, for each case processed, the respective value of Marsh's vorticity amplification factor is computed, according to Eq.2.7. It is noteworthy that the configurations considered pertain to **configuration 1** of the dataset, hence with

laminar/thick inlet boundary layer. Fig. 4.1 highlights the comparison of the Coull predicted model - Eq.2.10, reported in green line - and the regression line obtained by the data, classified in terms of parameters $M_{peak}/M2, L_{peak}$, (see Fig. 4.1a). It appears to be $\xi_{\mathbf{MARSH-fit}} \simeq 0.003547 \cdot AF_{\mathbf{MARSH}}$, very close if compared to Coull's model. If, therefore, the predicted loss - expressed as $\xi_{\mathbf{MARSH-fit}} + \xi_{\mathbf{CD}}$ - is compared to the endwall loss measured from RANS - $\xi_{\mathbf{net}}$, a neat discrepancy is observed when compared to the reported model, as the data exhibits more dispersion, deviating from a perfect fit to the reference line. As a result, the deviation in the regression line slope and the sparsity within the observed data can be due to the limited exploration of the design space. In fact, in the cited study, a significantly larger design space was employed, including hundreds of point within the (α_1, α_2) space, classified with different blade thicknesses expressed in percentage of C_{ax}. By expanding the dataset with additional points, perhaps the fit to the observed trends could be enhanced.



(a) $\xi_{\mathbf{net}} - \xi_{\mathbf{CD}}$ *vs.* $AF_{\mathbf{MARSH}}$.

(b) Measured loss *vs.* predicted loss.

Figure 4.1: Comparison of regression line obtained by data against Coull, 2017 [4] model.

In second instance, another analysis of the obtained data has been performed, with the aim of highlighting the parameters that most influence the measured losses. This can be reached by employing a **random forest regressor** with **feature importance analysis**. A random forest regressor is an ensemble machine learning method - meaning that it combines more models to enhance accuracy and reduce the risk of overfitting data. It operates by building multiple decision trees, mathematical hierarchical structures composed by internal nodes, that describe a decision - hence a mathematical operation like sum or difference - based on a feature - in this case a parameter given as input-, and leaf nodes,

which contain the final prediction based on the internal nodes interaction. A decision tree is then trained with data in order to fulfil two types of tasks: **regression**, so outputting the mean prediction, or **classification**, outputting the feature most voted by the majority of internal nodes. In random forest algorithms, many decision trees are employed, each of them containing a random subset of features rather than the full set given as input.

What characterises the algorithm is the **bootstrap sampling**, or "bagging", of the decision trees. In fact, each of them is trained with a random subset of input data in order to create a predictive model. The training is based on the implementation of the feature importance, where the most important contribution of each input feature to the model's predictive performance is evaluated. The training is set to minimise of the Mean Square Error (MSE) at each node:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

where n is the number of samples in the node, y_i is the value of the sample i and $\bar{y}$ is the mean value of all the samples in the node. There are two approaches for assess a feature importance analysis:

- **Mean Decrease in Impurity (MDI)**: It evaluates how much each feature reduces impurity across all trees in the forest. For regression tasks like this case, the impurity is measured as the total reduction of variance (MSE) brought by each feature. Mathematically, the importance of feature X_j is calculated as:

$$Importance(X_j) = \frac{1}{B}\sum_{b=1}^{B}\sum_{n \in N_b} w_n \Delta i(s_n, j)$$

  Where B is the total number of trees, N_b is the set of nodes in tree b, w_n is the weighted number of samples reaching node n, and $\Delta i(s_n, j)$ is the impurity decrease at node n where feature j is used for splitting.

- **Permutation Importance**: This method measures feature importance by randomly shuffling the values of each feature and observing how much the model's performance drops. A feature is considered important if the model prediction error increases by randomly shuffling the feature values. This highlights how the model heavily relies on that particular feature for accurate predictions. For each feature X_j, the permutation importance is calculated as:

$$I(X_j) = E[L(Y, f(X_{perm,j}))] - E[L(Y, f(X))]$$

Where $E[L(Y, f(X))]$ is the expected loss of the model on the original data, and $E[L(Y, f(X_{perm,j}))]$ is the expected loss when feature X_j is permuted.

MDI methods for feature importance analysis can over-estimate the importance related to high cardinality features - hence attributing more importance than they actually have. Moreover, it does not account for feature interactions. Whereas, Permutation Importance better accounts for feature contributions in the presence of correlations, directly evaluating model performances - accounts for prediction errors.
For the scope of this work, input design parameters, flow quantities and loading coefficients are chosen as features, while the measured loss $\xi_{\mathbf{net}}$ chosen as target. Through the `RandomForestRegressor` function in Python library `scikit-learn`, a feature importance analysis has been conducted. The results are shown in Fig. 4.2.

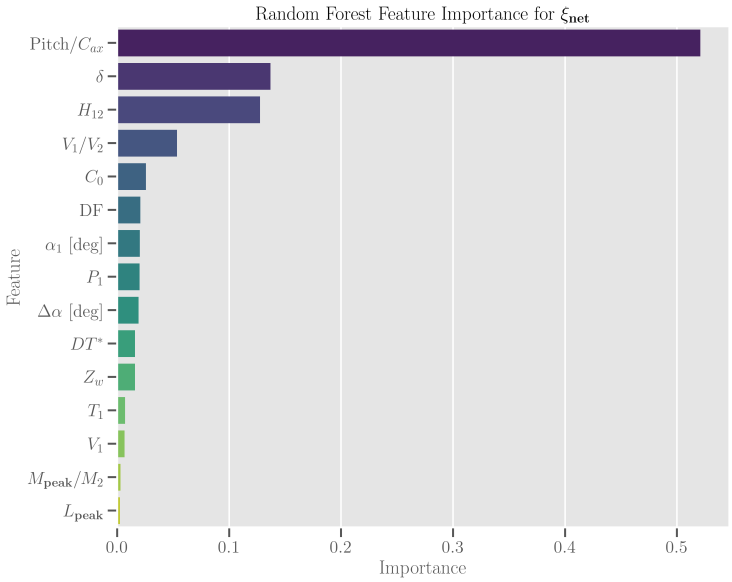
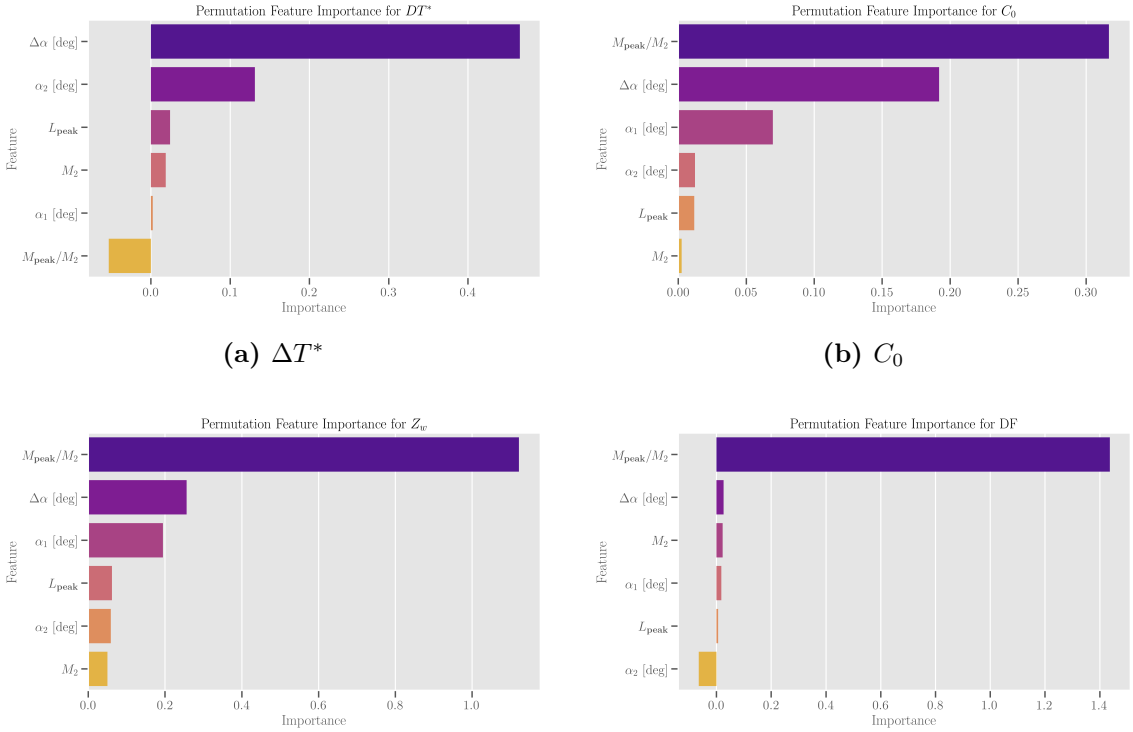


Figure 4.2: Random forest feature importance analysis on $\xi_{\mathbf{net}}$ as target.

The parameters that most influence the endwall losses are the pitch-to-chord ratio p/C_{ax}, the boundary layer thickness δ and shape factor H_{12} - already highlighted by the pitch-wise averaged losses in the grid convergence study. Other features that contribute to the losses, but less remarkedly, are inlet flow angle α_1, flow turning $\Delta\alpha = |\alpha_1 - \alpha_2|$, velocity ratio V_1/V_2, ΔT^*, C_0, Z_w, etc.
These show some similarities with Marsh amplification factors expressions, which strengthens the study previously exposed. If, therefore, the influence of blade optimising design parameters against loading coefficients - which lead to some influence also on losses - wants

to be evaluated, another analysis can be performed. In fact, this time a permutation importance analysis is conducted, this time considering blade loading coefficients are only features, as in Fig. 4.3. Results highlight a high influence of M_{peak}/M_2 and $\Delta\alpha$ on loading - Z_w and C_0, since have similar definitions. M_{peak}/M_2 also highly affects diffusion factor. Whereas, as expected, $\Delta\alpha$ and α_2 are the only features highly influencing non-dimensional transit time values, with a minor importance for L_{peak} and a negative one for M_{peak}/M_2. Both analyses have been run with a number of 100 decision trees (`n_estimators`), as default. In addition the test size has been set as 0.2, hence the subdivision of the whole set of data into 20% test and 80% train subsets.



(a) ΔT^* **(b)** C_0

(c) Z_w **(d)** DF

Figure 4.3: Permutation feature importance for blade loading coefficients.

It should be noted that this analysis was performed on a relatively small dataset (approximately 60 data points), which introduces some limitations. With small sample sizes, Random Forest models tend to overfit data, and feature importance estimates may be affected by high variance. This means the confidence intervals in the estimation of importance values would be wide, and repeated runs with slightly different data splits could produce different rankings. Therefore, this analysis is proposed as preliminary evaluation of the data.

To conclude, the last analysis has been focused to the effects of the inlet boundary layer profile properties, comparing the results of **configuration 1** and **configuration 2**. As highlighted by de la Rosa Blanco, [12], breaking down the total kinetic energy loss coefficient into the contributions of profile, inlet boundary layer and net losses can help to evaluate how inlet condition can influence secondary flows penetration towards midspan and the magnitude of endwall loss. From both the inlet configurations processed, profile loss $\xi_{\mathbf{profile}}$, gross mixed out loss $\xi_{\mathbf{gross-mix}}$ and net loss $\xi_{\mathbf{net}}$ coefficients are extracted and averaged across the blade geometries examined, as illustrated in Fig. 4.4. The results clearly indicate that profile loss shows minimal sensitivity to inlet conditions, confirming what stated in [12]. Moreover, high shape factor and thick inlet boundary layer cases exhibited higher gross mixed out and net losses - up to double those observed in turbulent thin cases. Notably, when comparing two same blade geometries under different inlet boundary layers, endwall loss values increased by a factor of $\approx$**2.6**. According to Coull, 2022 [5], indeed, the magnitude of endwall losses for high shape factor profiles could increase by more than **two times** the one measured in low shape factor inlet boundary layers.

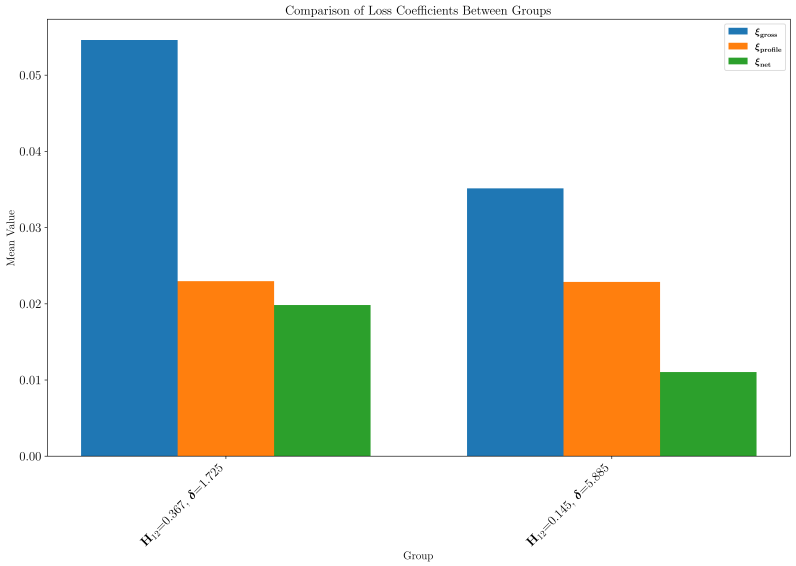


Figure 4.4: Losses breakdown results.

BLADE	N.	M_{peak}/M_2	L_{peak}	α_1	α_2	M_2	δ	n	H_{12}	δ^*/δ	θ/δ	Z_w	ΔT^*	DF	C_0	$\xi_{\mathbf{gross}}$	ξ_{bl}	$\xi_{profile}$	$\xi_{\mathbf{net}}$	ζ_{SKE}
1	1.1	0.4	40	-60	0.4	0.7	0.2	1.725	2.62	0.379	0.145	1.011	1.590	0.225	0.382	**0.059**	0.005	0.016	**0.038**	0.004
1	1.1	0.4	40	-60	0.4	0.7	0.05	5.885	1.29	0.143	0.111	1.011	1.590	0.225	0.382	**0.038**	0.001	0.016	**0.022**	0.001

Table 4.4: Data comparison for blade case 1 between **configurations 1** and **2**.

Chapter 5

Conclusions

In the present work, a Python-based tool designed to develop a computational framework for 3D CFD analysis of turbine cascade blades has been realised. In particular, the main objective was to automate the workflow - adopted in a potential design stage-, which begins with blade profiles optimized by a machine-learning model based on ANNs, [21], that delivers the optimal airfoil that satisfies a certain aerodynamic duty and style -such as loading-, characterised by a specific set of parameters. The tool processes a user-defined dataset of geometries, structured according to the number and combination of the mentioned parameters. The blade profiles are then processed by closing TEs, performing linear extrusion of a quantity h/C_{ax} defined as input and generating a CAD file in *.IGES* format. Subsequently, each blade is integrated in a structured mesh, whose characteristics are derived from a template mesh, itself selected based on a grid convergence study encompassing two potential extreme cases. This process leverages $AutoGrid5^{TM}$ functionalities in mesh generation. Afterwards, the CFD computation projects are created by user-specified boundary conditions, specifically defining an inlet boundary layer in terms of total pressure profile, as it influences secondary flows development in the cascade, [12],[5]. In fact, the tool is designed to take boundary layer parameters - thickness δ; and one between n exponent of Blasius velocity profile or δ^* and θ. Afterwards, once all boundary conditions are set, the CFD analysis begins. The blades dataset is divided into groups of five, sequentially processed. In fact, each group's computation is actively monitored -both by the user, via an emulator terminal console, and by the tool itself, through a convergence detection loop. When all simulations are successfully completed, post-processing is performed on each element within the group. Contour fields at different plane sections are extracted using Python macros executed in $CFView^{TM}$ in *batch* mode. Then, aerodynamic loading coefficients are computed -*i.e.* Zw, C_0, DF, *etc.*

Additionally, the kinetic energy loss coefficients are evaluated and then decomposed into $\xi_{\mathbf{gross\ mix}}$, $\xi_{\mathbf{bl}}$, $\xi_{\mathbf{profile}}$, ultimately isolating the endwall loss contribution $\xi_{\mathbf{net}}$. Furthermore, an evaluation of SKE coefficients distribution in different axial position in the cascade is assessed. At the final stage, the initial dataset is enriched with aero-thermodynamic quantities, boundary layer properties, 2D loading coefficients and computed losses averaged values. A subsequent tool validation is proposed by testing a very narrow dataset. It is created by selecting two values per each blade generator parameters -*corner points*-, and then the tool correct functionality is assessed by analysing two inlet boundary layers, with the related time performance analysis of its execution. In fact, for a dataset containing **64 cases**, the process successfully takes around **17 hours** for a machine with limited core capacities.
Furthermore, although in a preliminary stage, a data analysis is performed. Specifically, Coull physics-based model [4] for predicting endwall losses is compared with results obtained. Afterwards, a sensitivity analysis of blade design parameters against net endwall losses is performed by employing a random forest feature importance approach. Eventually, the effects of inlet conditions to endwall losses are assessed and compared with observations from literature.

The structure of the final dataset generated by the tool is designed to facilitate data-driven machine learning approached analyses. However, due to time constraints and limited computational resources, a comprehensive analysis has not been conducted in this study. As a matter of fact, in order to establish a robust correlation model that links relevant design parameters to endwall losses for predictive use in preliminary design phases —or to isolate secondary flow-induced losses, for example, as proposed by Coull [4] or to evaluate the influence of inlet conditions- a **significantly broader** design space must be explored. This includes the evaluation of **thousands** of configurations, obtained by varying both blade design parameters and inlet boundary layer properties, to capture the full complexity of endwall loss mechanisms.

As a matter of fact, at this stage, the tool is in its early development phase, laying the groundwork for a long-term research initiative. The ultimate objective is to integrate it into the preliminary design process of turbine blades, providing highly accurate loss estimations and potentially coupling it with optimization algorithms. Such an approach could help to settle new loss correlations or refine existing empirical and semi-empirical models, which are still used in design practice. In fact, the tool is thought to let the designer overcome the limitations of traditional loss models that often rely on experimental data fitting rather than fundamental physical principles.

Furthermore, future advancements could introduce additional complexities, such as mesh parameterization based on inlet conditions and blade profile variations, as well as incorporating 3D blade twisting laws. Extending the methodology to full-stage evaluations would provide a more accurate representation of losses, particularly given that rotor inlet conditions differ significantly from those of stators, as highlighted by Denton and Pullan [8]. This distinction is crucial, as the interaction between stator and rotor flow fields strongly influences endwall loss generation and evolution.

Ultimately, this tool represents a first step toward a more rigorous and physics-based approach to 3D turbine cascade design. Its functionalities can practically help designers to contribute significantly to the next generation of turbomachinery design, reducing reliance on empirical correlations and enabling more efficient and aerodynamically optimized blade geometries.

List of Figures

List of Tables

Nomenclature

Abbreviations

α	Flow angle
ΔT^*	Non-dimensional transit time
δ	Boundary layer thickness
δ^*	Displacement thickness
γ	Stagger angle
κ	Curvature
μ	Dynamic viscosity
ν	Kinematic viscosity
ω	Specific dissipation rate / Stream-wise vorticity
ϕ	Generic flow variable
Π	Vorticity distribution coefficient
ρ	Density
τ	Shear stress tensor
θ	Momentum thickness
ε	Error / Turbulent kinetic energy dissipation rate
ξ	Loss coefficient

LIST OF TABLES

ζ	Secondary kinetic energy coefficient
$_1$	Inlet conditions
$_2$	Outlet conditions
$_{bl}$	Boundary layer
$_{fs}$	Free-stream conditions
$_{gross}$	Gross loss
$_{is}$	Isentropic conditions
$_{net}$	Net loss
$_{profile}$	Profile loss
$_{sec-flow}$	Secondary flow-induced loss
$_t$	Turbulent quantity
AF	Amplification Factor
C_0	Circulation coefficient
C_D	Dissipation coefficient
C_p	Pressure coefficient
C_{ax}	Axial chord
DF	Diffusion factor
E	Total energy per unit volume
H_{12}	Shape factor (ratio of displacement to momentum thickness)
ILS	Integral length scale
L_{peak}	Stream-wise position of velocity peak
L_{ref}	Reference length
M	Mach number
M_{peak}	Maximum Mach number on suction side

P	Pressure
P^o	Total pressure
Re	Reynolds number
S_{ij}	Strain-rate tensor
SKE	Secondary Kinetic Energy
T	Temperature
T^o	Total temperature
Tu	Turbulence intensity
V	Velocity vector
V_{ref}	Reference velocity
y^+	Dimensionless wall distance
y_{wall}	First cell wall distance
Z_w	Zweifel coefficient
Z_{TE}	Secondary flows penetration height
.batch	Batch script file
.bcs	Boundary Conditions file
.cgns	CFD General Notation System file
.dat	Data file format
.geom	Geometry file format
.geomTurbo	Turbomachinery geometry format
.iec	Integrated Environment Configuration
.iges	Initial Graphics Exchange Specification
.igg	Interactive Grid Generator file
.py	Python script file

LIST OF TABLES

.qualityReport	Mesh quality report file
.res	Results file
.run	Runtime configuration file
.std	Standard output file
.trb	Turbomachinery template file
AR	Aspect Ratio
AutoGrid5	Automatic Grid Generator Version 5
CAD	Computer-Aided Design
CFD	Computational Fluid Dynamics
CFL	Courant-Friedrichs-Lewy
CFView	Computational Fluid Viewer
CV	Counter Vortex
DNS	Direct Numerical Simulation
EARSM	Explicit Algebraic Stress k-omega Turbulence Model
ER	Expansion Ratio
FINETurbo	Flow INtegrated Environment Turbo
GUI	Graphical User Interface
HSV	Horseshoe vortex
IGG	Interactive Grid Generator
LE	Leading Edge
LES	Large Eddy Simulation
LTHT	Laminar Thick High Turning blade test case
MDI	Mean Decrease in Impurity
MISES	MIT's viscous-inviscid flow solver

MSE	Mean Square Error
NURBS	Non-Uniform Rational B-Splines
PDE	Partial Differential Equations
PS	Pressure Side
PV	Passage Vortex
RANS	Reynolds-Averaged Navier-Stokes
RMS	Root Mean Square
S-A	Spalart-Allmaras turbulence model
SS	Suction Side
SST	Shear Stress Transport turbulence model
TE	Trailing Edge
TSV	Trailing Shed Vorticity
TTLT	Turbulent Thin Low Turning blade test case
TVD	Total Variation Diminishing

References

[1] R. Agromayor et al. "A Unified Geometry Parametrization Method for Turbomachinery Blades". In: *CAD Computer Aided Design* 133 (2021), p. 102987. DOI: 10.1016/j.cad.2020.102987.

[2] J.V. Boussinesq. "Essai sur la théorie des eaux courantes". In: *Mémoires présentés par divers savants à l'Académie des Sciences* XXIII.1 (1877), pp. 1–680.

[3] D. Burdett and T. Povey. "Analysis of Averaging Methods for Nonuniform Total Pressure Fields". In: *Journal of Turbomachinery* 144.5 (May 2022), p. 051011. DOI: 10.1115/1.4053020.

[4] J. D. Coull. "Endwall Loss in Turbine Cascades". In: *Journal of Turbomachinery* 139.8 (Aug. 2017).

[5] J. D. Coull and C. J. Clark. "The Effect of Inlet Conditions on Turbine Endwall Loss". In: *Journal of Turbomachinery* 144.10 (Oct. 2021).

[6] Vazquez R. Coull J. D. Clark C. J. "The sensitivity of turbine cascade endwall loss to inlet boundary layer thickness". In: *Journal of the Global Power and Propulsion Society* 2 (2018), CKB8N6. DOI: 10.22261/OEYMDE. URL: https://doi.org/10.22261/OEYMDE.

[7] H. R. M. Craig and H. J. A. Cox. "Performance Estimation of Axial Flow Turbines". In: *Proceedings of the Institution of Mechanical Engineers* 185.1 (1971), pp. 407–424. DOI: 10.1243/PIME_PROC_1970_185_048_02.

[8] J. Denton and G. Pullan. "A Numerical Investigation Into the Sources of Endwall Loss in Axial Flow Turbines". In: *Proceedings of the ASME Turbo Expo 2012: Turbine Technical Conference and Exposition*. Vol. 8. Turbomachinery, Parts A, B, and C. Copenhagen, Denmark: ASME, June 2012, pp. 1417–1430. DOI: 10.1115/GT2012-69173.

[9] J. D. Denton. "Loss Mechanisms in Turbomachines". In: *Proceedings of the ASME International Gas Turbine and Aeroengine Congress and Exposition* 2 (May 1993), V002T14A001. DOI: 10.1115/93-GT-435.

[10] Mark Drela and Harold Youngren. *MISES: Multiple-blade Interacting Streamtube Euler Solver*. https://web.mit.edu/drela/Public/web/mises/. Accessed: 2025-02-25. 2008.

[11] J. Dunham and P. M. Came. "Improvements to the Ainley-Mathieson Method of Turbine Performance Prediction". In: *Journal of Engineering for Power* 92.3 (July 1970), pp. 252–256. DOI: 10.1115/1.3445349.

[12] R. Vazquez E. de la Rosa Blanco H. P. Hodson and D. Torre. "Influence of the State of the Inlet Endwall Boundary Layer on the Interaction Between Pressure Surface Separation and Endwall Flows". In: *Proceedings of the Institution of Mechanical Engineers, Part A: Journal of Power and Energy* 217.4 (June 2003), pp. 433–441. DOI: 10.1243/095765003322315496.

[13] A. Harten and J.M. Hyman. "Self adjusting grid methods for one-dimensional hyperbolic conservation laws". In: *Journal of Computational Physics* 50.2 (1983), pp. 235–269. ISSN: 0021-9991. DOI: https://doi.org/10.1016/0021-9991(83)90066-9. URL: https://www.sciencedirect.com/science/article/pii/0021999183900669.

[14] W. R. Hawthorne. "Rotational Flow Through Cascades Part I. The Components Of Vorticity". In: *The Quarterly Journal of Mechanics and Applied Mathematics* 8.3 (1955), pp. 266–279.

[15] Martinelli Jameson A., L., and N. Pierce. "Optimum Aerodynamic Design Using the Navier–Stokes Equations". In: *Theoretical and Computational Fluid Dynamics* 10 (Jan. 1998), pp. 213–237. DOI: 10.1007/s001620050060.

[16] L. S. Langston, M. L. Nice, and R. M. Hooper. "Three-Dimensional Flow Within a Turbine Cascade Passage". In: *Journal of Engineering for Power* 99.1 (1977), pp. 21–28.

[17] Errante M. Larocca F. Ferrero A. "RANS Simulation of Secondary Flows in a Low Pressure Turbine Cascade: Influence of Inlet Boundary Layer Profile". In: *AIP Conference Proceedings* 2611.1 (Nov. 2022). DOI: 10.1063/5.0120392.

[18] H. Marsh. "Secondary flow in cascades: The effect of compressibility". In: (1976). URL: https://api.semanticscholar.org/CorpusID:118043716.

[19] NASA Langley Research Center. *SST Turbulence Model*. Accessed: February 17, 2025. 2025. URL: https://turbmodels.larc.nasa.gov/sst.html.

[20] R. Pichler et al. "Large-Eddy Simulation and RANS Analysis of the End-Wall Flow in a Linear Low-Pressure Turbine Cascade, Part I: Flow and Secondary Vorticity Fields Under Varying Inlet Condition". In: *Journal of Turbomachinery* 141.12 (Dec. 2019), p. 121005. DOI: 10.1115/1.4045080.

[21] F. Porta. "Development of a Machine-Learning-Based Tool for Turbomachinery Blades". MA thesis. Politecnico di Milano, von Karman institute for Fluid Dynamics, 2022-2023.

[22] P.L. Roe. "Approximate Riemann solvers, parameter vectors, and difference schemes". In: *Journal of Computational Physics* 43.2 (1981), pp. 357–372. ISSN: 0021-9991. DOI: https://doi.org/10.1016/0021-9991(81)90128-5. URL: https://www.sciencedirect.com/science/article/pii/0021999181901285.

[23] O. P. Sharma and T. L. Butler. "Predictions of Endwall Losses and Secondary Flows in Axial Flow Turbine Cascades". In: *Journal of Turbomachinery* 109.2 (Apr. 1987), pp. 229–236. DOI: 10.1115/1.3262089. URL: https://doi.org/10.1115/1.3262089.

[24] C. H. Sieverding. "Recent Progress in the Understanding of Basic Aspects of Secondary Flows in Turbine Blade Passages". In: *Journal of Engineering for Gas Turbines and Power* 107.2 (1985), pp. 248–257.

[25] L. Simonassi, G. Lopes, and S. Lavagnoli. "Effects of Periodic Incoming Wakes on the Aerodynamics of a High-Speed Low-Pressure Turbine Cascade". In: *International Journal of Turbomachinery Propulsion and Power* 8 (Sept. 2023), p. 35. DOI: 10.3390/ijtpp8030035.

[26] P. R. Spalart and S. R. Allmaras. "A One-Equation Turbulence Model for Aerodynamic Flows". In: (1992). AIAA Paper 92-0439, p. 439.

[27] R. C. Swanson and Eli Turkel. "On Central-Difference and Upwind Schemes". In: *Journal of Computational Physics* 101.2 (1992), pp. 292–306. DOI: 10.1016/0021-9991(92)90007-L.

[28] NUMECA Cadence design systems. *NUMECA AutoGrid5 User Manual*. Available at: https://www.numeca.com/. 2024.

[29] NUMECA Cadence design systems. *NUMECA FINETurbo 14.1 User Guide*. Available at: https://www.numeca.com/.

[30] NUMECA Cadence design systems. *NUMECA IGG User Manual*. Available at: https://www.numeca.com/. 2024.

[31] Open CASCADE Technology. *OpenCASCADE Technology Documentation*. Available at: https://dev.opencascade.org/doc/overview/html/. 2025.

Acknowledgements

I would like to first express my sincere gratitude to prof. Andrea Ferrero for his unwavering support and for allowing me to undertake this project and experience. His guidance has significantly deepened my interest in turbomachinery, and especially in fluid dynamics as applied to propulsion systems. I am particularly thankful for his extensive knowledge and expertise in the field, as well as for his constant availability and dedication towards his students.

A special thanks also goes to Prof. Sergio Lavagnoli for welcoming me into VKI environment and offering me a perspective on work that transcends the academic realm I was used to. His mentorship has taught me the importance of dedication and meticulousness to the work, which will continue to influence my approach to problems in the future.