POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

A Study on Secondary Flows in Turbine Linear Cascades Using a Data-Driven Approach

MSc Thesis in
Mechanical Engineering - Propulsion and Power

Author: **Matteo Bonanni**

Student ID: 251651
Advisor: Prof. Paolo Gaetani
Co-advisor: Prof. Sergio Lavagnoli
Academic Year: 2025-26

Abstract

The demand for highly efficient turbomachinery requires the continuous optimization of their design, which traduces, among other things, in making efforts to mitigate loss sources. Among these there are the endwall losses, including the loss contribution due to secondary vortical structures formation through the blade's passage, which typically accounts for one third of the overall turbine's efficiency reduction. While Computational Fluid Dynamics (CFD) provides insights into 3D vortical structures, its high computational cost heavily limits extensive design space exploration. This thesis presents an automated, data-driven workflow for the rapid prediction of secondary losses in linear turbine cascades, significantly reducing computational bottlenecks.

To achieve this, a Python-based Blade Processing Tool (BPT) was developed to fully automate the generation, meshing, and solving of turbine linear cascades. By leveraging a GPU-accelerated CFD solver, simulation times were reduced by a factor of 14. The BPT was used to build a database of 864 RANS simulations following a Full-Factorial Design of Experiments (DoE). The numerical setup's reliability was validated against experimental data from the SPLEEN cascade, extensively tested at the von Karman Institute for Fluid dynamics.

Finally, the generated database was used to train two Machine Learning surrogate models: a Gaussian Process Regression (GPR) and Radial Basis Function Network (RBFN). Both successfully mapped the correlations between six design parameters and endwall losses with good accuracy ($MSE \approx 10^{-6}$), confirming flow deflection as the primary endwall loss driver. GPR proved to be the superior model due to its uncertainty quantification capabilities. Ultimately, this work demonstrates that coupling GPU acceleration with machine learning provides a promising foundation for ultra-fast, simulation-free turbomachinery optimization.

Keywords: CFD, Turbomachinery, Turbine Design, Database, Surrogate Models for Aerodynamics

Abstract in Italiano

La richiesta di turbomacchine ad alta efficienza ne impone la loro continua ottimizzazione, che si traduce, tra le atre cose, nel cercare di mitigare il più possibile le fonti di perdita aerodinamica. Tra queste vi sono le perdite di endwall, che includono il contributo di perdita associato alla generazione di strutture vorticose all'interno del canale interpalare, tipicamente stimate come un terzo della riduzione d'efficienza totale nelle turbine. Sebbene la fluidodinamica computazionale (CFD) fornisca una buona comprensione di tali strutture tridimensionali, il suo elevato costo computazionale limita fortemente l'esplorazione estensiva dello spazio di progetto. Questa tesi propone un flusso di lavoro automatizzato e basato sui dati (data-driven) per la rapida stima delle perdite secondarie nelle schiere lineari di turbine, riducendo significativamente i colli di bottiglia computazionali.

Per raggiungere questo obiettivo, è stato prodotto un sofware interamente sviluppato in linguaggio Python, il Blade Processing Tool (BPT), al fine di automatizzare completamente la generazione, creazione della mesh e simulazione fluidodinamica di schiere lineari di turbine. Sfruttando un solutore CFD accelerato tramite GPU, i tempi di simulazione sono stati ridotti di un fattore 14. Il BPT è stato utilizzato per costruire un database di 864 simulazioni RANS seguendo un approccio di progettazione degli esperimenti di tipo Full-Factorial (Full-Factorial Design of Experiments). L'affidabilità del setup numerico è stata validata rispetto ai dati sperimentali della schiera SPLEEN, testata innumerevoli volte presso il von Karman Institute for Fluid Dynamics.

Infine, i dati estratti dal database sono stati impiegati per addestrare due modelli surrogati: una Gaussian Process Regression (GPR) e una Radial Basis Function Network (RBFN). Entrambi hanno mappato con successo le correlazioni tra i sei parametri di progetto e le perdite di estremità con un'elevata accuratezza ($MSE \approx 10^{-6}$), confermando la deflessione del flusso come causa principale delle perdite. La GPR si è dimostrata il modello superiore grazie alle sue capacità di quantificazione dell'incertezza. In definitiva, questo lavoro dimostra che combinare l'accelerazione GPU con il machine learning fornisce una base promettente per un'ottimizzazione ultra veloce delle turbomacchine, senza la necessità di eseguire simulazioni CFD.

Parole chiave: CFD, Turbomacchine, Design di Turbine, Database, Modelli Surrogati per Aerodinamica

Contents

1 | Introduction

In an historical period such as the one we are living, the need of propulsion and power conversion systems is increasing exponentially day by day. When there is the need to convert energy, there is, obviously, also the need for dedicate systems to do that. In the framework of this work, the core system considered is turbomachinery. They can be briefly described as very complex and expensive rotating machinery, with a very high technological level in terms of materials and function principles. In a standard turbomachinery there are always two components and a medium that hosts thermodynamic transformations, namely "working fluid". The components are: "Compressor", which increases pressure and internal energy of the working fluid through a rotating motion, and "Turbine", that uses the working fluid's high pressure to generate a rotating motion, hence a torque that can be used for both propulsion (aeronautical applications) and electrical power generation (power plants).
Going deeper in the heart of this work, the attention shifts on turbines, for which a lot of research has been conducted during World War 2, in order to build high performance aero engines that could have helped to win the war. In the past, important research topics were focused to develop high temperature materials and improve the thermodynamic cycle efficiency. Nowadays, regardless of the fact that turbine's technology has matured a lot, research efforts are still needed to address important aspects, like cost reduction and performance increase due to their boundless presence in the aforementioned sectors. Turbines are composed of two parts, a static one called "Stator" and a moving one called "Rotor". Their names are quite evocative with respect to their scope, and hence it is simple to understand that these machinery's parts rotate around an axis. For this reason they are complex to be studied and enhanced, and a substantial amount of physics and math is required to understand how they work, starting by the fluid's interaction with static and moving components, fluid's particles curvature through the blades passage, formation of a strong velocity gradient region close to the physical walls in the stage (rotor + stator), temperature effects, etc...
This thesis work is divided in three steps, executed sequentially. The first regarded the development of a *Blade Processing Tool* (BPT), with the aim to process an high number

of turbine cascades using a GPU-accelerated CFD software. The second involved the creation of a *database* (n-dimensional space), collecting relevant data for an high number of simulations performed through the BPT. Finally, the third step was focused on the study of the correlation between design input parameters, used to create the cascades database, and the related secondary losses produced through a *data-driven approach*, leading to the use of Machine Learning strategies.

In a world dominated by the progress, it is relevant to keep up with it. The complexity of the systems among us leads to a massive use of data, in many sectors, to investigate phenomena. This is also the case of the cutting-edge topics inspiring this work, leading to a data-driven approach for conducting the analysis.

Physically, the work took place in the context of a collaboration between Politecnico di Milano and the Von Karman Institute for Fluid Dynamics (Be), in a Short Training Program (internship position) for the duration of 6 months (Sept. 2025 - March 2026).

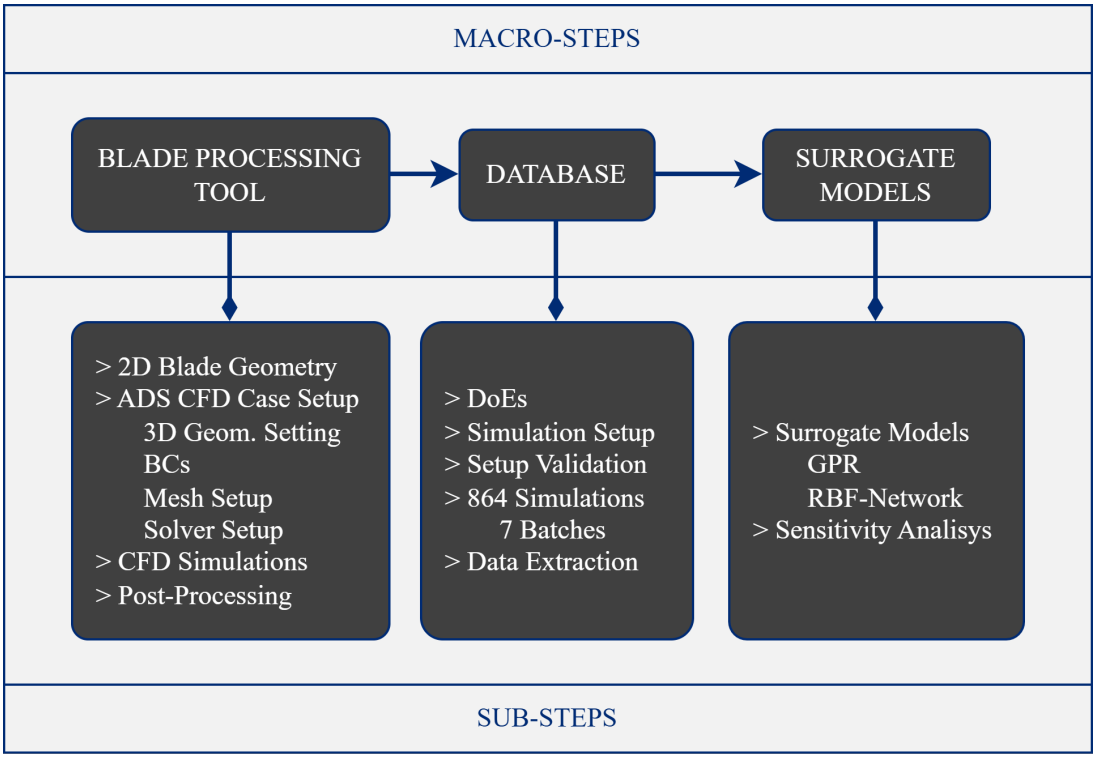


Figure 1.1: Report's workflow.

The following report has been structured with the idea of highlighting, in sequence, all the logical steps followed throughout this work's development. After this brief introduction, at first a detailed literature review covering the main research topics is reported in Chapter 2, such as the state-of-the-art knowledge about data-driven strategies for turbomachinery design, knowledge on database creation and the most recent studies on secondary flow

losses in turbine linear cascades. Then, the Blade Processing Tool is presented in terms of its architecture and main functionalities in Chapter 3, also providing useful information on the virtual environmental setup needed to run it (the reader might find this redundant, but due to the unavoidable complexity of the tool, the author deemed necessary reporting also these aspects). Then, the database creation strategy together with the post-processing results has been reported in Chapter 4, followed by the final implementation of the two Machine Learning surrogate models in Chapter 5), with the objective of finding a correlation between input design parameters and secondary losses in linear turbine cascades. At the end, project's related conclusions and future developments have been addressed in Chapter 6.

2 | Literature Review

In this chapter, a comprehensive literature review about the main theoretical concepts related to this project has been reported. They are presented following the same sequential logic behind the work development. For instance, the main purpose of the work was to create an extensive database collecting several different turbine linear cascade simulations, also referred as cases or **points** from now on, using a custom-made tool that entirely, and automatically processes them (Chapter 3). The idea is to, first create a **benchmark** to be used as a reference, second for training surrogate models useful for very early design purposes.
Once the database has been created, the cases contained inside have to be processed in order to collect physically relevant data. There are several strategies to process these data, depending on what phenomenon has to be studied. This is why the second part of the survey is dedicated to reporting data-driven design strategies used in fluid-dynamics applications, and hence also for turbomachinery design.
The processed data, at the end, have been used to study the correlation between database input design parameters and secondary flow losses. That is why a theoretical background on secondary losses has been reported in the final part of this survey.
Resuming, the first part of this survey regards the state-of-the-art of database creation strategies for fluid-dynamics applications, then the principal surrogate models applicable in the same context, and finally a review on secondary flow losses in axial turbines.

2.1. Design of Experiments

This section describes the principal methodologies used for the creation of a database. Yondo et al. [18] presented in a survey the state-of-the-art mathematical methods required to efficiently collect data for surrogate models development.
Before proceeding, let us introduce some technical terminology about data management. Each **point** in the database represents a unique combination of the input design parameters, and the collection of all their possible combinations is named **design space**. For

what concerns this project, all the variable parameters are bounded by a maximum and a minimum limit, therefore, also the design space will be limited. The space extrema get the name of **corner points**, while all the other configurations are the **inner points**. The action of identifying a point in the design space is named **sampling**. The strategy for sampling an entire database gets the name of **design of experiments** (DoE). The term "experiments" is generic, and takes into account the fact that points can be the result of laboratory or purely numerical activities, using for example a CFD software. Hence, considering the numerical nature of this work, "experiments" refers to CFD simulations. In the following are listed the principal sampling strategies adopted to create a database (db) for engineering applications.

2.1.1. Sampling Strategies

The design of experiments can be approached as a mathematical problem. It accepts more results, which means that more DoE strategies exist. Some of them are clever and efficient but complex, others are less efficient but very straightforward and fast to apply. There is no DoE absolutely better than the others, the choice depends on the designer's purposes.

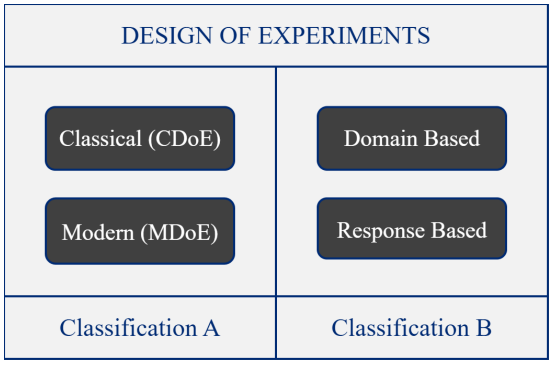


Figure 2.1: Classification of DoE strategies as reported in [18].

As depicted in fig.2.1, mainly two classes of DoE strategies can be identified: on the left, the distinction between **Classical DoE (CDoE)** and **Modern DoE (MDoE)**. CDoE have been defined and used before MDoE, implying that the latter strategies result to be more efficient but also more complex than the former ones. On the right, the distinction between **Domain-based** and **Response-based** strategies is reported. Domain-based ones are also defined as offline, since their main purpose is to create a uniformly distributed grid of points, sampling them apriori of the surrogate model implementation. The response-based ones, instead, can be defined as online strategies because they sample

first a small amount of points, then, thanks to prediction algorithms based on the concepts of exploration and exploitation (like "Expected Improvement") sample iteratively other points, sequentially building a strong training set for the surrogate model. Online DoE allow to develop high accuracy surrogate models with few points in the design space, substantially reducing computational efforts (high when CFD simulations are required). However, these are more complex to be applied than offline ones.
In the following is reported a series of DoE strategies, sorted following the classical/modern classification.

Classical DoE

A classic approach is the factorial one, that can be full or fractional. In the following the descriptions:

- **Full-Factorial Design**: sampling strategy giving all the possible combinations of L levels for v variables. The formula giving all the combinations is L^v, which means that, for example, if $L = 2$ (one level for the maximum, and one level for the minimum of a variable) and $v = 6$, the sampled points through this strategy will be $2^6 = 64$. The growth of the sampled points number is exponential, limiting the applicability of this method for the creation of extensive database. In literature [18] is stated that full-factorial design works fine up to a maximum of 4 levels, than the number of sampled points becomes too big. Nevertheless, in this project the experiments are CFD simulations, which computational cost is generally high. Thanks to the usage of GPU (Chapter 4) and for project-related requirements, it was possible to create an extensive database with the full-factorial strategy.
- **Fractional-Factorial Design**: they work following the same principles of full-factorial ones, but instead of considering the entire set of configurations, is considered only a sub-set, so that the formula to compute the number of sampled points becomes L^{v-r}, where r represents the "restriction" of the full set of sampled points. This method allows using the simplicity of full-factorial design, maintaining low computational efforts. However, its main constraint is that they can focus only on certain regions of the design space, keeping uncertainty on the unexplored regions.

Another classic DoE is the **Central Composite Design** (CCD). It allows to extend the capabilities of factorial strategy by adding points inside the design space (for example along its axis) with a criteria. The new samplings are called star points, and they are used to selectively augment the amount of information in selected regions of the design space.

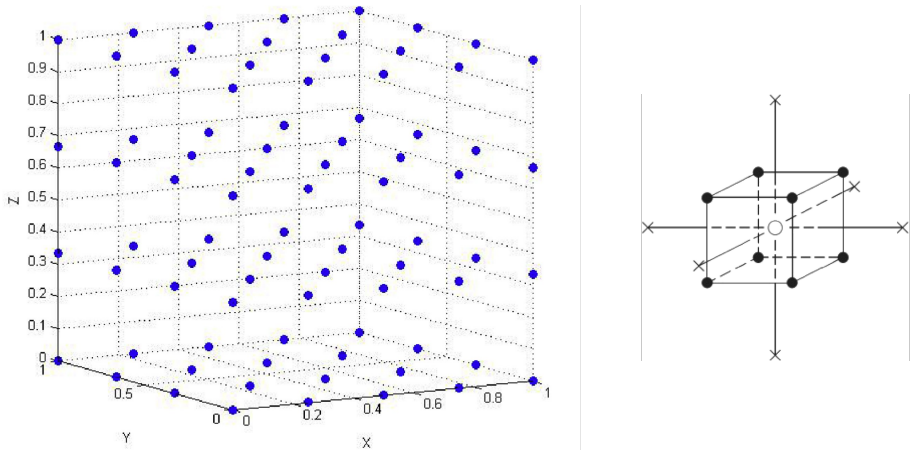


Figure 2.2: On the left, a full-factorial design representation, where the sampled points fill in the design space creating an ordered grid. On the right, a representation of a CCD. Both images are adapted from [18].

A mention goes also to the **Optimal Design** strategy, showing good performances only when working with problems for which the objective function is known apriori. Schemes for some classical DoE mentioned above are depicted in fig.2.2.

Modern DoE

While classical designs concentrate their efforts in trying to fill in the entire design space, modern ones aim in concentrating the samples only in specific regions of it. Below, some of the most relevant modern DoE.

- **Projection Based Design**: useful when implementing or willing to work with a regression based surrogate model.
- **Uniform Design**: actually one of the most used strategies to create a database. It generates the samples and scatters them throughout the design space, creating a uniform dispersion of points.
- **Statistical Measures-based Design**: they foresee the use of statistical analysis so that to collect, for example, entropy data. Probability distributions over these data are created and evaluated to schedule an efficient sampling strategy. To use this type of design, statistics and probability theory foundations must be well known, making them more complex-to-use from an inexperienced designer.
- **Miscellaneous Design**: in this class are enclosed all the strategies that cannot be classified in the other aforementioned groups. To make an example, among miscellaneous design there are the **Latin Hypercubes** (LHS).

2.2. Data-Driven methods for Turbomachinery

In all engineering branches, to design components and systems, a good practice is to first perform numerical simulations, giving insights on how a physical phenomenon works in a relatively inexpensive way, then experiments to investigate simulation's reliability and how the investigated phenomenon behaves in real life. In turbomachinery design, numerical simulations have the purpose of studying interactions between internal components and working fluid (air in this project), entering in the realm of CFD.
Considering an high amount of CFD simulations performed to investigate a target phenomenon (secondary losses), each one related to different input design parameters, data can be extracted for data-driven purposes. The term **data-driven**, in this context, indicates a cluster of strategies able to reach a target goal, like understanding underlying physical correlations, using high amount of data. Unfortunately these strategies cannot always be used, since they require very big datasets, generally not available because of the high complexity or high cost in the data collection process. Besides, data cannot be randomly collected, but a rationale has to be followed in order to efficiently sample the design space.
This section reports up-to-date information on data-driven design methods used in the context of CFD applications, with a particular focus on aerodynamic-related problems, paving the way for the use of **Machine Learning** to design turbomachinery.

2.2.1. Surrogate Models for CFD Applications

The use of Machine Learning (ML) to design turbomachinery components is a new research topic, meaning that literature and scientific community show a lack of knowledge about it. For this reason, first could be relevant to report the general understanding in the use of ML for general CFD applications.
ML results particularly useful to improve some current CFD related aspects, such as improving computational performance and reducing simulation time by reshaping the definition of numerical solvers and algorithms, or even the definition of new turbulence models. Another application regards its use for post-processing CFD simulation data.
The survey conducted by Wang et al. [17] provides a meaningful classification for the lately used ML strategies for CFD applications such as atmosphere science, plasma studies, and aerodynamics. Contextually to this work there are principally two kinds of interesting ML models: **Data-Driven Surrogate Models** and **Physics-Driven Surrogate Models**. The former use CFD data to train algorithms for physical phenomena modeling; some examples are the DD (Discretization Dependent) methods and DI (Discretization Inde-

pendent) methods, distinguished relatively to how the domain generating data is made. However, as stated above, these models need high amounts of data, presenting issues in terms of robustness and generalization. In these terms, the latter type of models represent an enhancement, providing the opportunity to include in the model also the physics of the problem, enhancing its reliability.

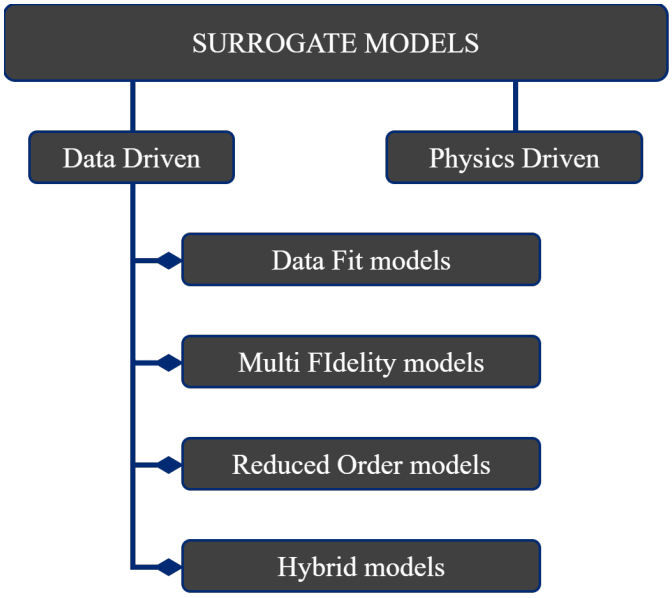


Figure 2.3: Surrogate models classification for engineering applications, summarized in [18] and [17].

The problem of implementing Physics-Driven Surrogate Models is to provide them with the needed information about the physics of a certain phenomenon, task that would go beyond the scopes of this project. Hence the focus falls on Data-Driven Surrogate Models, for which deeper insights are given in the work of Yondo et al. [18]. The term **surrogate** refers to the considered model as a substitution of the real one, capable to give reliable and representative results of the studied phenomenon. They are used when, for a certain phenomenon, true correlations are not known yet or because of their intrinsic speed in outputting results. They can be classified following the **nature of design variables** (parametric, semi-parametric, non-parametric) or by their **operating methodology** (interpolation, regression, projection). As depicted in fig.2.3, four classes of data-driven surrogate models can be identified: **data fit (DFM)**, **multi fidelity (MFM)**, **reduced order (ROM)**, **hybrid (HSM)**.

Data Fit Models

These have resulted to be the most interesting ones concerning the purposes of this work, due to their intrinsic simplicity and fewer input data required to return relevant results.

- **RMS (Polynomial Regression)**: it shows computational flexibility and simplicity, but, on the other side, poor prediction for highly non-linear full order models, considering also the need for large training sets with respects to other DFM;
- **MARS (Multivariate Adaptive Regression Splines)**: they divide the dataset into smaller partitions, and for each partition a linear regression model is applied. They can also work well without cost function;
- **GPR (Gaussian Process Regression)**: also named as "Kriging", they are generally very much used in the ML field. Let us imagine two curves: an objective one, representing a real trend, and the GPR, or surrogate curve. The GPR model undergoes a sampling phase, after which it will represent a very similar shape to the real trend. The sampling phase is done making reference to some fixed training points, allowing to sample the surrogate curve by considering a spatial correlation. For instance, if in the space design the sampled point is really close to the training point, their function's image will be highly correlated, and vice-versa; this is the concept of kernel function. GPR have the advantages to be capable to mimic non-linear models with a good approximation, and they are capable to work with up to 50 variable parameters (high dimensionality of the model).
- **ANN (Artificial Neural Networks)**: these are very powerful non-linear regression models that try to reproduce the functionalities of human brain. A neural network can be composed by multiple layers. Each layer can have more neurons (NN's operative unit). Neurons of different layers can exchange information, and these "information bridges" are weighted. The more complex the NN's structure is, the better it will predict complex correlation, but the higher its computational cost will be. One of the most efficient ways to define a surrogate model for aerodynamic related problems is to use RBF (Radial Basis Functions) as activation functions in the NN's neurons, shaping the concept of RBFN. This kind of model shows good flow features predictions for both steady and unsteady phenomena [18].

Schematizations of the previously cited surrogate models are shown in fig.2.4 below.

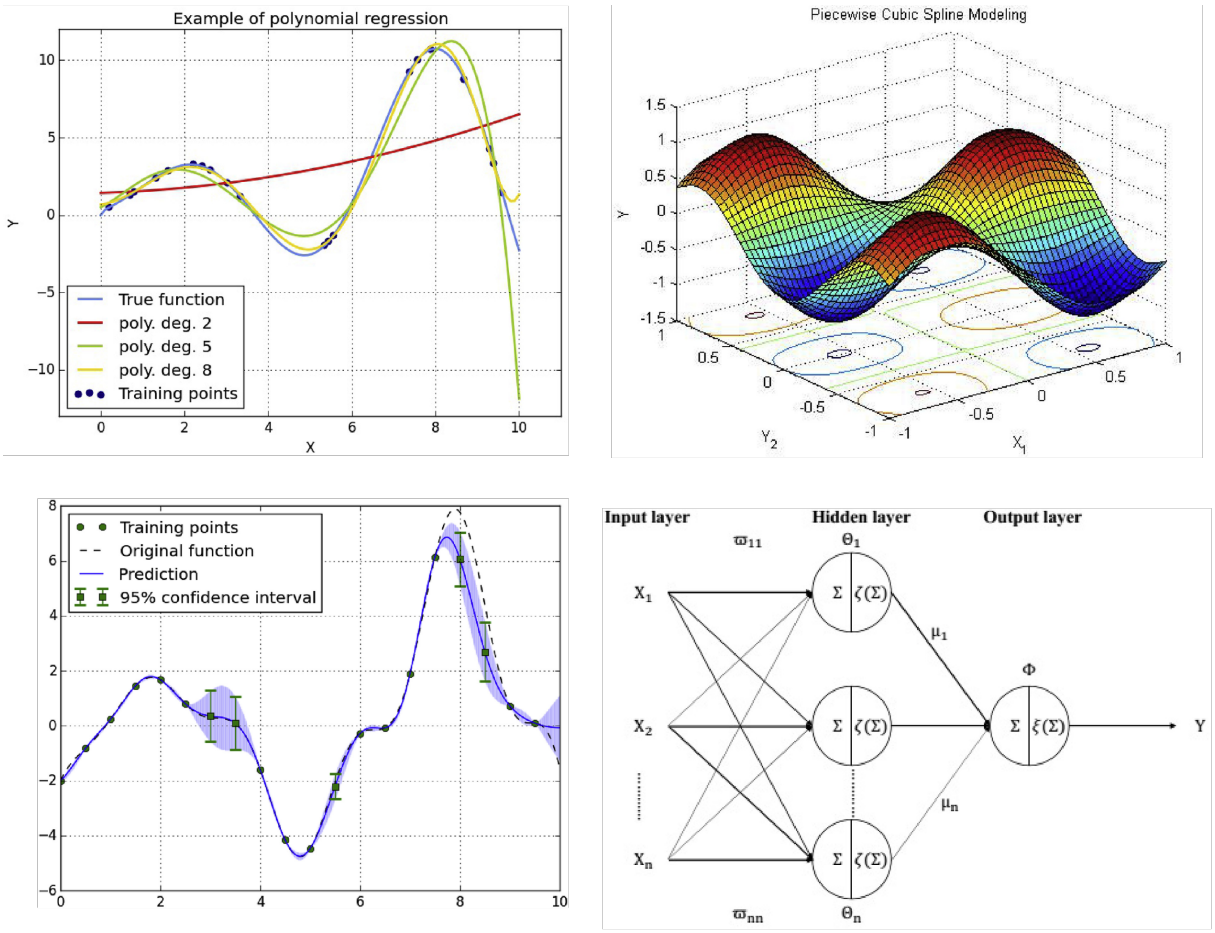


Figure 2.4: On top left, a representation of the RMS model. On top right, MARS. On bottom left, GPR. On bottom right, ANN. All the graphs have been taken from [18].

Multi Fidelity, Reduced Order and Hybrid Models

Multi-fidelity models result to be interesting for their ability of combining different accuracy level results. For instance, it means that low-fidelity results (from Euler equations and coarse grid simulations) are combined with high-fidelity ones (RANS/LES simulations and experimental tests).

Moving forward, ROMs are described. A model is said to be **full** when it describes a system in the most complete way possible through its equations. Instead, a model is said to be **reduced** when it approximates the real system, typically with a lower number of variables, but retaining its physics. Therefore, reduced order models are capable to provide good quality results at a low computational effort.

Finally, a mention also for the Hybrid models (HSMs), which are created through the combination of more surrogate models, with the aim to combine their advantages and compensate the weaknesses.

By virtue of what said above, to study the correlation among input design parameters and secondary flow losses using the database points, the types of models described in this last rows will not be implemented in this project, due to their higher complexity. Data fit models, specifically GPR and neural networks could better meet project's requirements. All the technical aspects of the implemented ML models are reported in Chapter 5.

2.3. Losses in Turbomachinery

In literature, losses in turbomachinery can generally be expressed as the fraction of lost kinetic energy (Eq.2.1) or lost total pressure (Eq.2.2) through a single cascade, as considered in this report, or the entire stage. Losses magnitude has direct impact on turbomachinery efficiency, due to their inverse proportionality. Hence it is of high interest, from the designer perspective, to understand the generation mechanisms behind these complex phenomena, aiming to find out smart and cost-effective design strategies to mitigate them.

$$\zeta_{tot} = \frac{V_{2-is}^2 - V_{2-mix}^2}{V_{2-is}^2} \tag{2.1}$$

$$Y_{tot} = \frac{P_{01} - P_{02}}{P_{02} - P_2} \tag{2.2}$$

Eq.2.1 [6] shows the ratio among the loss of kinetic energy, calculated like the difference between isentropic and mixed-out velocity at the cascade's outlet plane, and all the available kinetic energy. Eq.2.2 [9] shows the same concept, but using quantities expressed in terms of total pressures. Both formulas are extensively reported and used in literature. To better understand the meaning of their terms, let us imagine a turbomachinery cascade and two planes, each one orthogonal with respect to the axial direction of the machine: the **inlet plane**, referenced with number "1" and generally placed at a fraction of C_{ax} upstream the LE, and an **outlet plane**, referenced with number "2" and placed at a fraction of C_{ax} downstream the TE (in the project's post-processing, the outlet plane was set at a distance of $0.5 \cdot C_{ax}$ downstream TE).
Focusing on axial-flow turbines, [6] states that, making reference to a blade with a large enough AR ($= h/C_{ax}$) to have the mid-span section completely unaffected by secondary flows, it is possible to breakdown total losses into **profile losses** and **endwall losses**.

$$Y_{tot} = Y_{profile} + Y_{endwall} \tag{2.3}$$

Let us suppose to use a CFD simulation as benchmark; the Eq.2.2 allows computing both total and profile losses. For instance, to compute the total ones, quantities used in the formula are averaged on the entire planes at inlet and outlet, while to compute profile ones, the calculation is made by averaging quantities along a line, representing the crossing between mid-span and outlet planes, or a very narrow slab centered to the mid-span section, to improve result's numerical stability. This makes possible, with good approximation, to isolate the endwall effects using Eq.2.3.
However, for preliminary designs, CFD simulations are not always available or affordable in terms of time/computational cost, hence **empirical correlations** are used to compute the single losses contributions separately. These correlations are case dependent and subject to constant revisions by their authors or other researchers, mainly because built on poor experimental database. Historically, the most relevant contributions in terms of correlations to study losses in axial-flow turbines are those of Ainley and Methieson [2], later improved by Dunham and Came [10], Craig and Cox [8], Kacker and Okapuu [11]. But also the works of Sharma and Butler [15], Benner et al. [3] and Coull [5], [6] are highly recognized and used among researchers.
Below are described endwall losses further in details, allowing to better understand their nature and how they are produced, introducing the concept of secondary flows.

2.3.1. Endwall Losses

Secondary losses, commonly referred to in literature as endwall losses, represent a major source of aerodynamic inefficiency in axial-flow turbines, accounting for roughly one-third of the total losses. These arise from the highly three-dimensional nature of the flow near the hub and shroud endwalls. When the incoming boundary layer approaches the blade leading edge, it decelerates and experiences an adverse pressure gradient, rolling up into horseshoe vortexes. As the flow evolves through the cascade, this incoming vorticity is stretched and re-oriented, ultimately generating a strong vorticity component at the exit plane (see fig.2.5).
Furthermore, the pressure gradient established between the pressure side of a blade and the suction side of a neighboring one in the cascade (cross-passage pressure gradient), drives the low-momentum fluid of the boundary layer across the passage towards suction side. This vorticity movement feeds the so called passage vortex, a massive vortical structure inducing significant losses due to downstream mixing. The intensity of these secondary flows is directly linked to the blade aerodynamic design: it increases with higher flow turning, decreases with the outlet-to-inlet velocity ratio, and is heavily influenced by the shape and thickness of the inlet boundary layer [7].

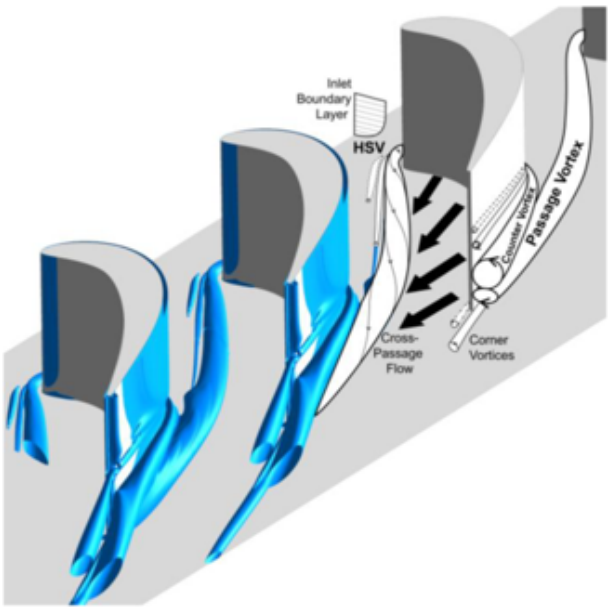


Figure 2.5: Secondary flows formation in a turbine linear cascade. Image taken from [5].

However, vortex-induced mixing is not the only contributor to the endwall losses. A substantial fraction of the total endwall loss is generated directly by the viscous dissipation in boundary layers over the endwall platforms. This specific mechanism gets the name of wetted area loss, that becomes particularly intense in regions of the passage characterized by high freestream velocities.

While the fundamental physics of secondary flows are well-captured by simplified linear cascade models, the environment inside a real turbine is significantly more hostile. Actual turbomachinery features complex geometries, leading to more intense secondary losses formation. The flow field entering a real blade row is spatially complex and highly unsteady: secondary vortices from upstream stages tends to convect downstream, creating a chaotic inlet boundary layer, augmenting losses intensity. This contrasts traditional cascade experiments, typically relying on thin and perfectly turbulent boundary layers. Recognizing this discrepancy is essential, as the mismatch in inlet conditions is a primary reason why traditional cascade-based correlations often fail to predict the much higher endwall losses observed in operating engines.

For the purposes of this work, the correlation developed by Coull in 2025 [6] has been used in comparison with the results outputted by CFD simulations. Hereafter are reported and commented the most relevant steps to compute the endwall losses through the Coull model.

2.3.2. Coull Model for Endwall Losses Prediction

The 2025 Coull model takes a physics-based approach to estimate endwall losses, minimizing the reliance on empirical coefficients [6]. According to this method, the net endwall loss coefficient ζ_{end} can be split into two macro-contributions:

$$\zeta_{end} = \zeta_{wet} + \zeta_{sec} \tag{2.4}$$

The first term, ζ_{wet}, represents the wetted area loss. This accounts for the viscous dissipation occurring within the boundary layers that develop over the hub and shroud surfaces. Coull evaluates this contribution by dividing the fluid domain into three distinct regions: the upstream area, the in-passage area, and the downstream area. The overall wetted area loss is formulated as:

$$\zeta_{wet} = C_D \left(\frac{C_x}{h}\right) \left(\frac{4}{\cos\alpha_2}\left[\left(\frac{V_1}{V_2}\right)^3 \left(\frac{X_{US}}{C_x}\right) + \left(\frac{X_{DS}}{C_x}\right)\right] + 4.363\right) \tag{2.5}$$

In this equation, C_D is a dissipation coefficient typically assumed equal to 0.002, while X_{US} and X_{DS} represent the axial lengths of the upstream and downstream domain's areas. The constant 4.363 is an empirical approximation to account for the complex in-passage dissipation.
The second major term in the model is ζ_{sec}, which expresses the losses induced by secondary flows. This phenomenon is further decomposed into two mechanisms:

$$\zeta_{sec} = \zeta_{sec-mix} + \zeta_{SKE} \tag{2.6}$$

Here, $\zeta_{sec-mix}$ computes the penalty related to the mixing out of the secondary flows, while ζ_{SKE} measures the dissipation of the secondary kinetic energy. Both of these components are heavily driven by the strength of the passage vortex, which is quantified by its non-dimensional circulation Γ^*_{sec}:

$$\zeta_{sec-mix} = 2\left(\frac{p\cos\alpha_2}{h}\right)(0.01442\Gamma^*_{sec}) \tag{2.7}$$

$$\zeta_{SKE} = 2\left(\frac{p\cos\alpha_2}{h}\right)(\Gamma^*_{sec})^2\Pi_{SKE} \tag{2.8}$$

As visible in the formulas, both secondary loss terms scale inversely with the cascade aspect ratio. The secondary kinetic energy component relies also on the parameter Π_{SKE}, which mathematically describes how the thickness and the shape factor of the incoming boundary layer impact the secondary vorticity field.
Finally, the non-dimensional passage vortex circulation Γ^*_{sec} must be evaluated. Coull relates this quantity directly to the aerodynamic loading of the blade mid-span section:

$$\Gamma^*_{sec} = V^* \frac{\Delta T^* C_x}{p \cos \alpha_2} + \left| \frac{V_1}{V_2} \frac{\sin \alpha_1}{\cos \alpha_2} - V^* \tan \alpha_2 \right| \tag{2.9}$$

The crucial parameter in this last step is ΔT^*, representing the non-dimensional transit time difference for a fluid particle traveling over the pressure side versus the suction side of the blade. As the flow turning increases or the pressure side velocity decreases, the transit time difference increases, ultimately feeding more strength into the secondary vortices.

2.3.3. Secondary Flows Penetration Height

Another fundamental aspect related to the presence of secondary flows in 3D turbine passages, is related to how far these vortical structures extend along the blade span. Secondary flows are not strictly confined to the hub and shroud regions; instead, they tend to migrate radially towards the mid-span of the channel. To quantify this phenomenon, Sharma and Butler [15] introduced the concept of **penetration depth**, generally denoted as Z_{TE} at the trailing edge axial plane. Through their investigations, they showed that the fraction of the blade span affected by secondary flows strongly depends on the aerodynamic loading and cascade geometry, such as pitch and chord, as well as on the inlet boundary layer thickness. Accurately estimating Z_{TE} is a crucial step during both design and post-processing phases: it geometrically bounds the region where losses are dominated by three-dimensional endwall effects, separating it from the mid-span area where the flow remains two-dimensional and dissipation are only attributable to profile losses.

$$\frac{Z_{TE}}{b_x} = 0.15 \cdot \frac{\epsilon}{\sqrt{CR}} + f(\frac{\delta_1}{h}) \tag{2.10}$$

This equation represents the Sharma and Butler model, also implemented in the Blade Processing Tool for the optimization of the blades ARs.

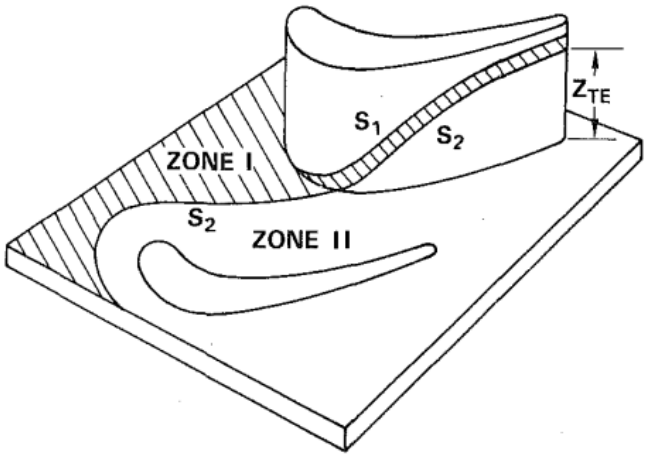


Figure 2.6: Graphical representation of the penetration height, taken from [15].

3 | Blade Processing Tool

3.1. Generalities

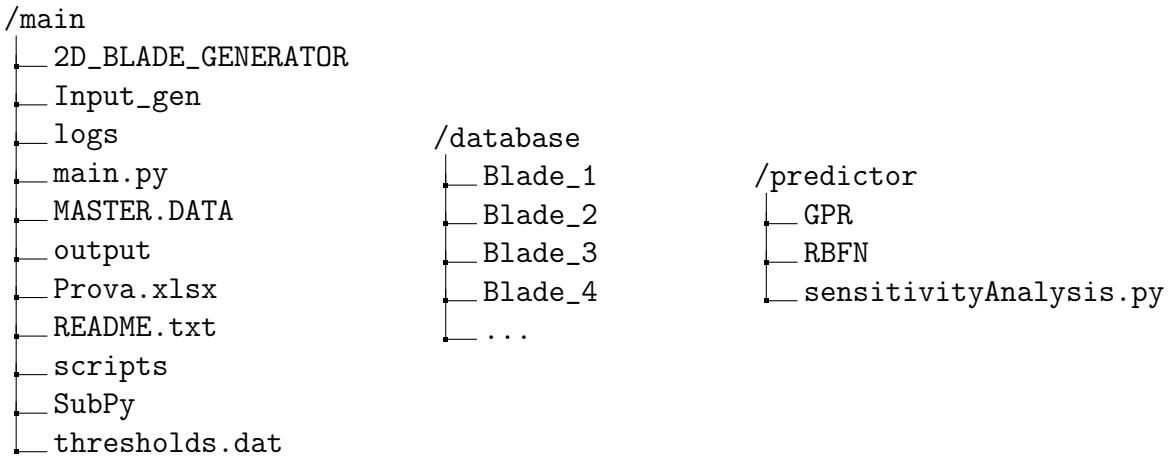
The name Blade Processing Tool has been chosen intentionally, in order to be evocative and to immediately understand what it does. For instance, the term "Blade" refers to the fact that blades (in turbine linear cascades) are the subject of the processing. It is important to specify that the cascades taken into consideration for this work are **linear**, but the tool has been developed to be easily convertible to annular ones. The term *Processing* collects a series of operations that are automatically handled during the run, such as (in the following order): 2D blade generation, Boundary Conditions calculation, inlet Boundary Layer definition, "ADS CFD" case files creation (.AGF, .WAND, .MESHIX, .LEO), "ADS CFD" case automatic run. For more specific information about the processing, the reader should check below in this Chapter. *Tool* because it is intended to be used as it to create a database (Chapter 4). Its short name, also used from here on in this report is **BPT**.

The BPT was thought to be the evolution of Scarimbolo [14], inspired by the works of Clark [4] and Porta [12] . In his paper, Clark defined the theoretical background for an intelligent inverse design strategy for 2D turbomachinery blades. Based on that, Porta developed a 2D Blade Generator capable of creating 2D blades geometries, giving in input aerodynamic load and style for a blade through a set of parameters. Later, Scarimbolo made a step ahead, extending Porta's blade geometry from 2D to 3D, allowing for BCs computation, inlet BL definition, meshing and CPU based simulation, all automatized.

Throughout this project, the old Scarimbolo's version logic has been completely changed, due to the introduction of a newer, more performant software to conduct faster CFD simulations using GPU in place of CPU. To be able to use GPU acceleration a LINUX Operating System is required, which is the reason why the entire BPT has been developed for such OS. To favor in future the tool enhancement, a short description of the virtual environment setup to launch it has been reported.

3.2. Software Development Setup

To develop software, different programming languages could be used. For the BPT, **Python** has been chosen due to its enormous potentiality and flexibility, especially with the aim to plug the tool with Machine Learning surrogate models (easily usable built-in Python packages are available to define and train ML models). The software is fully contained in the `/main` folder, where are collected all the scripts and templates needed for the ADS cases simulation and post-processing. The results of CFD simulations are collected in the `/database` folder, while the ML models have been defined and stored in the `/predictor` folder. These three folders are all contained in the general tool's folder `/BPT`.



(a) `/main` folder (b) `/database` folder (c) `/predictor` folder

Figure 3.1: Internal structures of `/main`, `/database` and `/predictor` folders.

To write, modify and debug the scripts, the IDE "VS Code" was chosen thanks to its capabilities of easily switching through virtual environments and its good adaptability to both Windows and LINUX OS. **Adaptability** has been a particularly important aspect during the software development, with the idea to make it easily portable from a computer to another and to make it usable on different OS. However, the current version of the tool can only be launched on LINUX OS, since the 2D Blade Generator by Porta and ADS CFD require it. BPT was first developed on WSL, using a LINUX OS in a native Windows environment, and then tested in a native LINUX OS.
BPT requires the installation of specific Python libraries to properly function. They have been downloaded and stored in a **Virtual Environment** defined through Miniconda, a specifically dedicated software for programming in Python. In order to avoid wasting

time to solve libraries compatibility issues, an executable file (`BPT_setup.sh`) has been created, such that once run, it automatically set up the needed virtual environment with all the necessary Python packages already installed (see Appendix A.2).

3.3. ADS CFD

For 3D blades modeling, meshing and simulations, the Blade Processing Tool uses the support of a software called **ADS CFD**. This software is mostly suited for performing CFD analysis of turbomachinery. Its main feature, with respect to other simulation software, regards the possibility to perform GPU accelerated CFD simulations, allowing for simulating 20-50 times faster than standard CPU-based simulations [1].
To use it, the user has two ways to proceed: via Workbench (a GUI developed by ADS, making the software more "User friendly" but less flexible at the same time) or via Command Line, making use of a terminal. To have more control on ADS functions, making it compatible for its use in the BPT, the author opted for the second strategy.
For the purposes of this work it's relevant to understand how an ADS Case is made, in order to create study cases relevant for the database creation.

3.3.1. ADS Case

An ADS Case is a folder (see fig.3.2) containing a series of text files with different purposes, such as blades geometry and BCs definition, mesh topology definition, etc... These files will be better described below in this section.

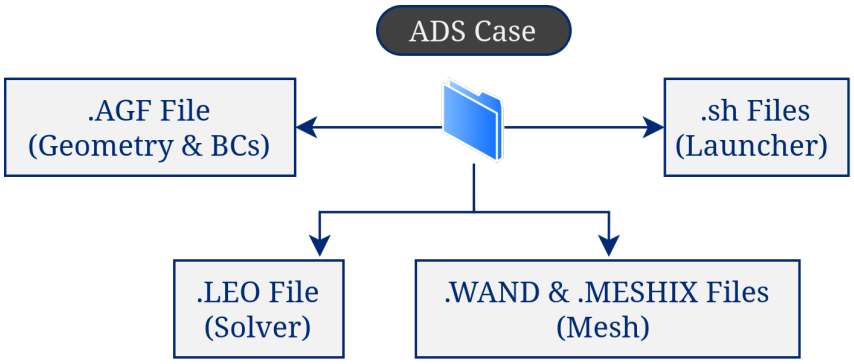


Figure 3.2: Scheme showing the content of an ADS Case folder.

Once clear the ADS case structure, let us have a look at the scheme reported in fig. 3.3 to understand how to mesh and run CFD simulations. At first, the `RunWand.sh` executable file is run launching a command in a terminal. This runs `CodeWAND`, a script that automatically generates the mesh for the study case. In order to be initialized and

launched, `CodeWAND` needs first to read the `.AGF` file, which contains information about blades geometry (2D or 3D, depending on the case that has to be studied) and Boundary Conditions, then it reads the `.WAND` file, collecting the basic mesh setup information. If an advanced control on mesh topology wants to be applied, the `.MESHIX` file is also present in the case folder and therefore read by ADS. Once meshed a cascade, additional text files describing the mesh will appear in the case folder. Finally, the `RunLeo.sh` executable file is run, calling `CodeLEO`, a script that runs simulations by reading the `.LEO` file, where solver setup options are listed. Equivalently to what said for meshing, when a simulation ends results are written in text files located in the case folder.

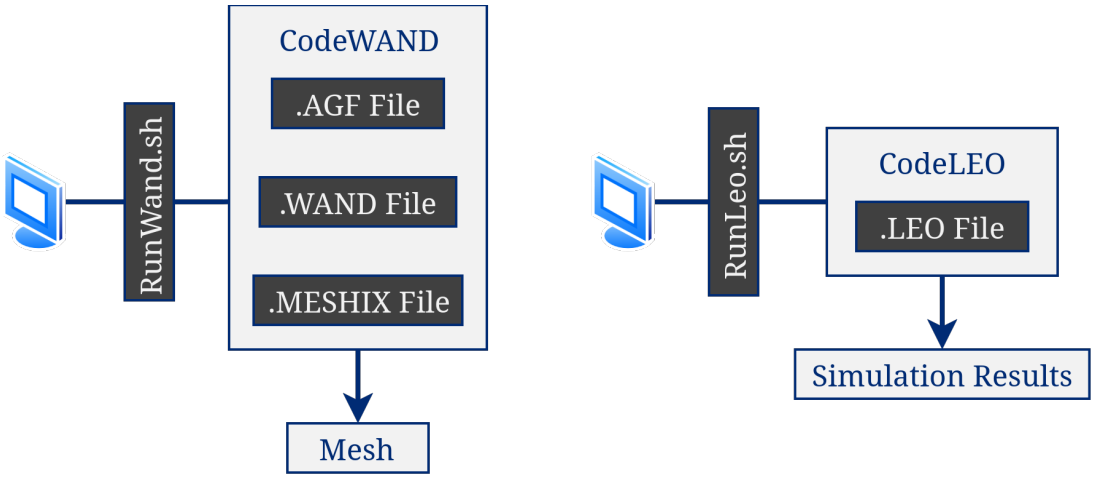


Figure 3.3: ADS Meshing and Simulation flowchart.

In Appendix A.1 are reported, and quickly described, the terminal output logs of meshing and simulation processes, giving the reader an idea of what to expect when processing an ADS CFD case.
The ADS files described and discussed below are organized in **Sections** and **Locations**, as reported in the ADS technical documentation [1]. Each file contains more sections (rows), and in each section there are more locations (columns). This logic creates grid-structured files, easy to be manipulated with Python scripts.

3.3.2. .AGF File

It is the longest ADS case file. It contains information about cascade's **geometry** and **BCs**. In this report, only the most important features of this file are going to be presented. For further insights the author suggests to read the ADS technical documentation [1].

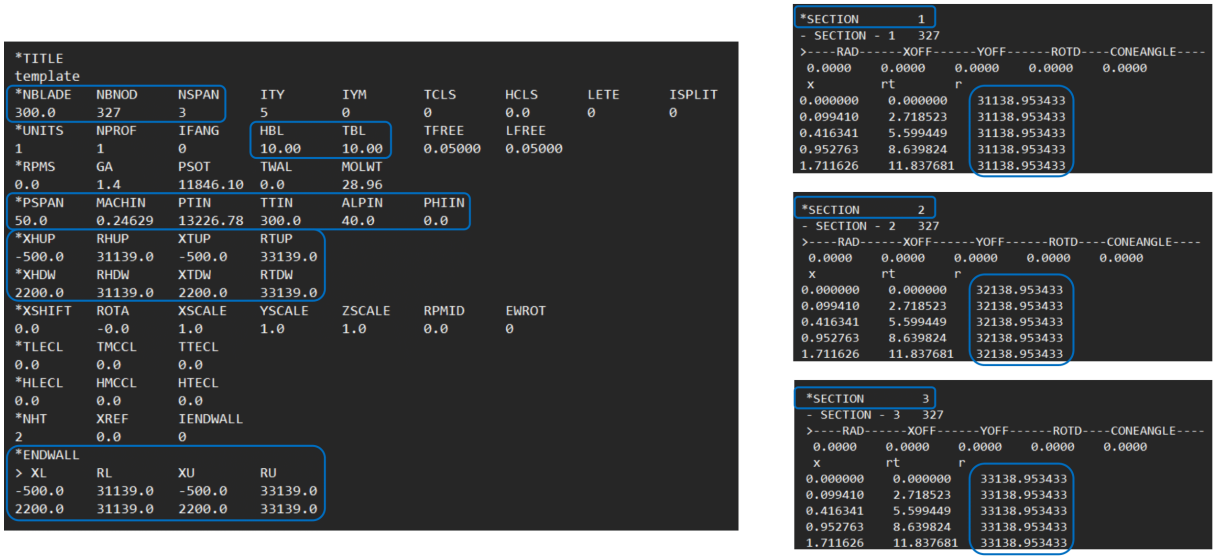


Figure 3.4: `.AGF` file content. On the left, the BCs settings. On the right, the geometry settings.

On the left of fig.3.4 have been highlighted different sections. Proceeding with order, from the top:

- **Sec.2** specifies the number of blades (NBLADE) in the cascade, through which is possible to fix also its pitch and hub radius (see below). It is also specified the number of points of the 2D blade profiles (NBNOD) used to define the span-wise geometry of the 3D blade. Also the number of used radial profiles is reported (NSPAN).
- **Sec.3** allows to set the inlet BL thickness as a percentage of the blade span at the two endwalls: Hub (HBL) and Shroud (TBL). For symmetry these two numbers are the same, and they can vary among two values, depending on the type of BL chosen for a cascade. The 2 BLs considered are the same used in [14] for studying their effects on secondary losses.
- **Sec.5** allows to set the inlet BCs. In fig.3.4, BCs are defined by a single point along the radius (PSPAN). However, being present also the inlet BL, an inlet radial profile of BCs is automatically created.
- **Sec.6 & Sec.7** report the fluid physical domain's limits.
- **Sec.8** defines the endwalls. To be noted that the simulation domain has to be contained inside the endwalls limits.

On the right side of fig.3.4 have been highlighted the three span-wise 2D sections used

to define the 3D blade geometry (by interpolation), pointing out their radial coordinates. With this, let's specify the units of measurement used, that are the European ones: mm for lengths, Pa for pressures and K for temperatures.
Profiles coordinates are of the type (Axial, Tangential, Radial), where the radial one is clearly enormous compared with the other two. In these cases, an annular cascade can generally be approximated as a linear one. Keeping this BPT architecture allows, in future, to extend its functionalities also to annular cascades simply reducing the hub radius. In this sense, the developer will only need to change the blades number in the `.AGF` template file to pass from linear to annular cascades. The analyzed cases, than, are not perfectly linear cascades, but **annular cascades where the hub radius is brought towards infinity, approximating a linear one**. To better explain the concept, let's start analyzing the very simple geometrical relation in Eq.3.1:

$$N_b = \frac{2\pi R_m}{s} \tag{3.1}$$

In a turbomachinery row all the blades have the same geometry, and this is the case also for the cascades used in this work. Taking the midspan plane as reference, blades number (N_b) is calculated as the ratio between the midspan circumference length ($2\pi R_m$) and pitch (s). The pitch is defined as the tangential distance among the same points of two consecutive blades in a row. R_m can be expressed as:

$$R_m = \frac{R_h + R_s}{2} = \frac{R_h + (R_h + h)}{2} \tag{3.2}$$

Therefore, substituting this last relation into Eq.3.1 and performing the calculation, the result is giving:

$$R_h = \frac{1}{2}(\frac{N_b \cdot s}{\pi} - h) \tag{3.3}$$

This relation is particularly relevant, because it states the dependency between R_h and N_b, already mentioned above, since s is fixed (computed by the 2D Blade Generator tool with MISES [12]), as long as h. Between R_h and N_b there is proportionality, which explains that for increasing the hub radius also the number of blades has to augment, and vice versa. A schematic representation of the concept explained above can be seen in fig.3.5.

Figure 3.5: Hub surface when its radius is small (left) or big (right).

Due to the objective of this work (linear cascades analysis), and due to the cascade geometry approximation, the **static pressure variation** along blade span has been checked to exclude the possible presence of flow curvature effects through the blade passage, negatively affecting the analysis, that are typical of annular cascades and are not present in linear ones. The idea was to check through simulations if the above mentioned static pressure difference is small enough, so that to consider negligible flow curvature effects. The control was made considering three points on the hub surface and their projection on the shroud surface, like shown in fig.3.6.

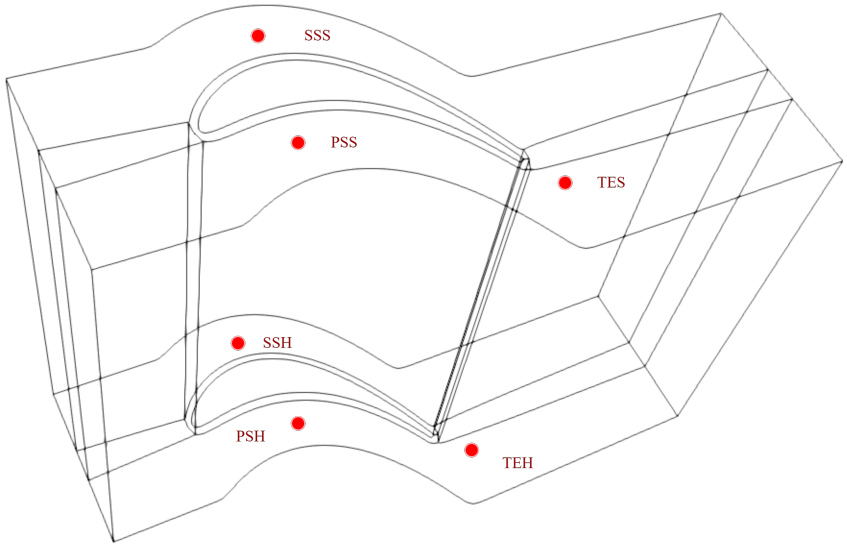


Figure 3.6: Radial ΔP_s control points. SS: Suction Side; PS: Pressure Side; TE: Trailing Edge; S: Shroud; H: Hub.

Without curvature effects, the static pressure should remain constant along blade span, therefore ΔP_s is expected to be as close to 0 as possible. A random database cascade has been check, and the static pressure difference was evaluated among the pairs of points

(SSS-SSH), (PSS-PSH), (TES-TEH) shown in fig.3.6. It has been noticed that for all the pairs, the static pressure difference was below 10^{-2}. Considering the propagation of possible numerical errors through simulation results, this outcome can be considered as good, even if not perfect. A ratio of around 200:1 between radial and the other two profile coordinates generates a $\Delta P_s < 10^{-2}$, ensuring that for each case in the database, and for what stated above it can be considered satisfying for **linear cascade approximation**. An idea to further reduce ΔP_s could be to make the radial coordinate of the hub profiles tending to even higher values, but ADS presents some computational limitations: precisely, if the radial coordinate is too big compared to the other ones, numerical errors arise and simulations don't start.

3.3.3. .WAND & .MESHIX Files

They control both standard (`.WAND`) and advanced (`.MESHIX`) mesh settings. `.MESHIX` is not one of those files that must be present in an ADS case folder, since the `.WAND` contains all the needed settings to fully create a mesh, but for the purposes of this project, a more careful mesh topology control has been preferred, and to do so, the `.MESHIX` file has been used.

```
*ITYPE    OUTYPE    IGRID     IPOSS     ICLS      IWAL      IFAN      ILEO
-3        8         1         1         2         0         3         0
*MBLD     KSPAN     FUTURE
-5        2         1
*IUP      IDW       JPITCH
13        13        41
*NFILES   IPLOT
1         204
*FILENAME
Blade_1.AGF
*IUP      IDW       JPITCH
-13       13        41
GRID-CONTROL ON
DWAL-OMESH 0.00003
OMESH-MULT 1.00000
```

```
NBLOCK    ADVANCED
5         -20
BCK#      IM        JM        KM
1         365       25        42
2         365       25        42
3         241       25        42
4         129       13        42
5         129       13        42
LE-RFA, LE-MOV, LE-ANG, TED-RFA, TE-MOV, TE-ANG
1.00000   0.00000   1.00000   1.00000   0.00000   1.00000
```

Figure 3.7: `.WAND` file above, `.MESHIX` file below.

In fig.3.7 is reported the content of the `.WAND` and `.MESHIX` files. For the `.WAND` file can be highlighted **Sec.1**, defining the type of mesh used (OHH), the coordinates system for mesh generation (cylindrical) and other controls such as the fanouts shape at LE and TE. **Sec.2** allows to choose the preset mesh refinement (MBLD). If it is set to specific values, `.WAND` ignores all the other settings that control the mesh number of nodes along system coordinates, defined below in the file. **Sec.7** allows to set the $y+$ value. For the `.MESHIX` file, instead, can be highlighted **Sec.1**, where is specified the number of blocks in which the mesh is divided (5 blocks, as shown in fig.3.8) and, for each block, the number of divisions along each direction of the coordinates system: IM is the axial direction, JM is the tangential direction, KM is the radial direction. When used, `.MESHIX` has the full control on mesh topology settings and it overrides the corresponding controls in the `.WAND` file.
However, the specific meaning of all settings of both files is extensively reported in the ADS technical documentation.

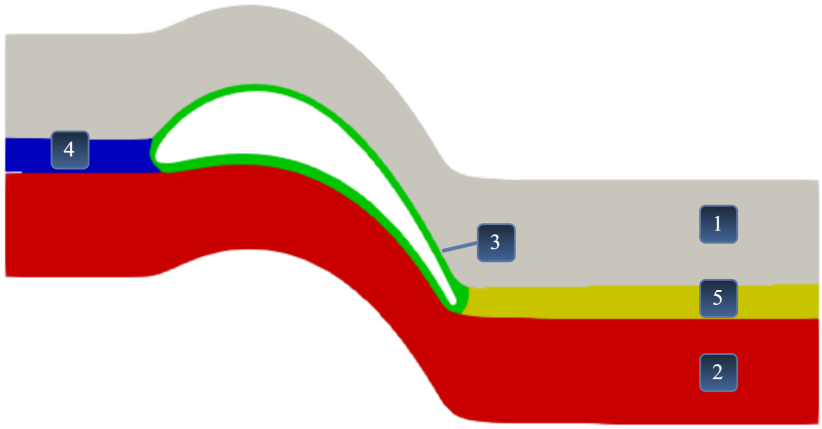


Figure 3.8: Mesh blocks.

Mesh Sensitivity Analysis

As a good practice, a mesh sensitivity analysis should always be done before performing CFD simulations. It allows to find the minimum number of elements in a mesh, namely the minimum refinement, above which simulation results do not depend anymore by the mesh number of elements. In other words, this analysis identifies a mesh for which simulation results are independent by further grid refinement. This procedure allows to maintain a good trade-off between low computational power and high precision of the results. Due the nature of this project, a template mesh adaptable for all the points in the database is

required, hence the worst case scenario in the entire database was considered to perform this analysis (blade with highest deflection and loading). The object mesh should provide the following features: as low elements as possible, ensuring low computational time per simulation; reliability, such that it is sure that outputted results have physical meaning. Nevertheless, using a single mesh template for hundreds of blades with different geometry requires a trade-off.
The procedure reported in [13] was followed, using Y_{tot} as control parameter. Results are presented in table 3.1 and fig.3.9.

Data	Values	Units
N_F, N_M, N_C	3512, 2020, 1153	x1000 el.
r	1.3	/
Y_{tot}^F, Y_{tot}^M, Y_{tot}^C	7.04%, 7.07%, 6.83%	/
p	12.187	/
GCI_{C-M}, GCI_{M-F}	0.481%, 0.051%	/

Table 3.1: Results of the procedure reported in [13] applied to the most deflected and loaded cascade of the database, considering 3 different refinement levels.

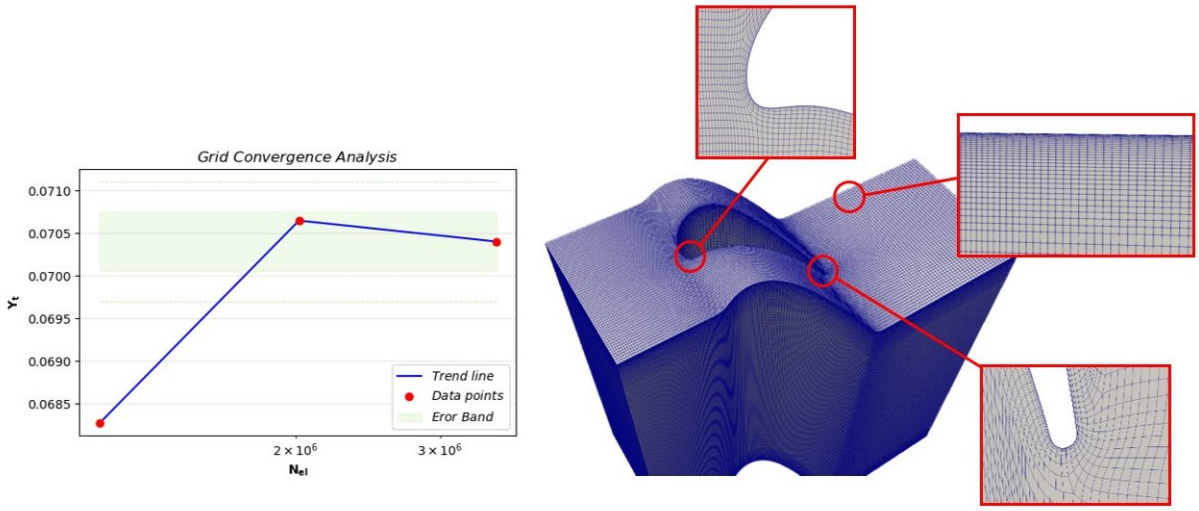


Figure 3.9: Mesh sensitivity analysis results. On the left, trend of Y_{tot} as function of mesh number of elements. On the right, principal features of the chosen template mesh for database creation.

Table 3.1 shows that three mesh refinement levels have been used to perform the sen-

sitivity analysis. Looking at the results, the **medium (M) mesh has been chosen as template**. The two GCI values have been evaluated comparing Y_{tot} for the couple Course-Medium and Medium-Fine.
The acquisition planes where Y_{tot} was calculated are placed upstream LE and half C_{ax} downstream TE, both orthogonally oriented with respect to the machine axial direction. Studies in literature, generally, report different values of the outlet plane's distance from TE, depending on the phenomena object of the study. For the purpose of this work, half C_{ax} downstream TE can be considered a fair distance, allowing the flow to mix out and hence to correctly compute Y_{tot} with the following formula [9]:

$$Y_{tot} = \frac{P_{01} - P_{02}}{P_{02} - P_2} \tag{3.4}$$

P_{01} is the upstream fluid total pressure, while P_{02} and P_2 are the fluid total and static pressures downstream the blade (half C_{ax} downstream TE), respectively. Total quantities have been averaged with the mass flow rate, while static ones with the area. The BPT post-processing mode (see below) was used to directly compute Y_{tot}.

3.3.4. .LEO File

This file gives `CodeLEO` the information on how to setup the solver. It contains a series of settings, and uses the same logic of the other files described above.

```
*RESTART-FILE-NAME
Blade_1.REST
*NITER    MPID     PRUN     IPRT     NIRST    IPLOT
4000      101      0        20       1000     4
*ITURB    ITIME    EMODEL   IFAST    IGAMMA   IRUN
1         0        0        1        0        21
```

Figure 3.10: Example of `.LEO` file used to setup the ADS solver.

The `.LEO` file reported in fig.3.10 shows only two sections: **Sec.1**, where is set the number of iterations for each simulation (NITER), the number of iterations between results printing in the relative text files (IPRT) (e.g. results are written once every 20 iterations), as well as the results printing options (IPLOT). **Sec.2** reports the turbulence model chosen (ITURB), that is the *Wilcox* $k - \omega$ 98), the simulation time resolution (ITIME), which is steady, and the measure system used to write out simulation results (IRUN), set to European units.
As optimal number of iterations ADS suggests to use 6000, because convergence is generally reached in the most of the cases after such number of iterations, but it was decided to

reduce it to 4000 after some trial, since it was noted that simulations converge anyways for the study cases, allowing a reduction in computational time and therefore making faster the database creation. For the purposes of this work, only **RANS simulations** have been performed.

3.3.5. .sh Files

To conclude this first part dedicated to the ADS case files description, let us discuss the `.sh` executable files. ".sh" means that the file in question is a text file executable through a UNIX-like shell. It requires permissions to be executed. In each of these two files there is written a single line:

```
wand < Blade_X.WAND (a)
leo_acc80 Blade_X.LEO (b)
```

(a) is the line in `RunWand.sh`, which synthesized means: "use the ADS executable file called `wand` (`CodeWAND`) to create the mesh with the specifics indicated in `Blade_X.WAND`". This command needs also the `.AGF` file to function, and it considers the `.MESHIX` if present in the case folder. (b) is the line in `RunLeo.sh`, that synthesized means: "use the ADS executable file called `leo` (`CodeLEO`) to run the simulation with the solver specifics indicated in `Blade_1.LEO`". The second line can function only after a mesh is present in the case folder, and it is not the only ADS executable file to run the solver that can be used. Actually, `leo` runs simulations with CPU. There are other executable (see [1]), like `leo_acc80`, that run simulations with GPU. This second modality requires specific hardware configurations, but enables for very fast simulations. To speed up calculations even more, jobs parallelization can be implemented.

3.4. BPT Architecture

Now that is clear how ADS CFD works in the context of this work, its use and implementation in the BPT can be reported. Together with that, in the following section will be shown the complete architecture of the Blade Processing Tool. For instance, the reader will find the logic behind the most relevant scripts, the hierarchy of folders and files and comments on final outcomes.

To start, the scheme in fig.3.11, reporting the entire tool workflow is presented. It shows how the tool is made internally, and it will be taken as reference to explain all its fundamentals. What was reported in the scheme represents the `/main` folder's content, which contains the tool. The scheme is divided in two parts: **Main Files**, on the right, where are

listed all the files of general purpose. **Folders and Scripts**, on the left, is the part showing the core of the tool, describing conceptually its internal tasks performed to process turbomachinery cascades.

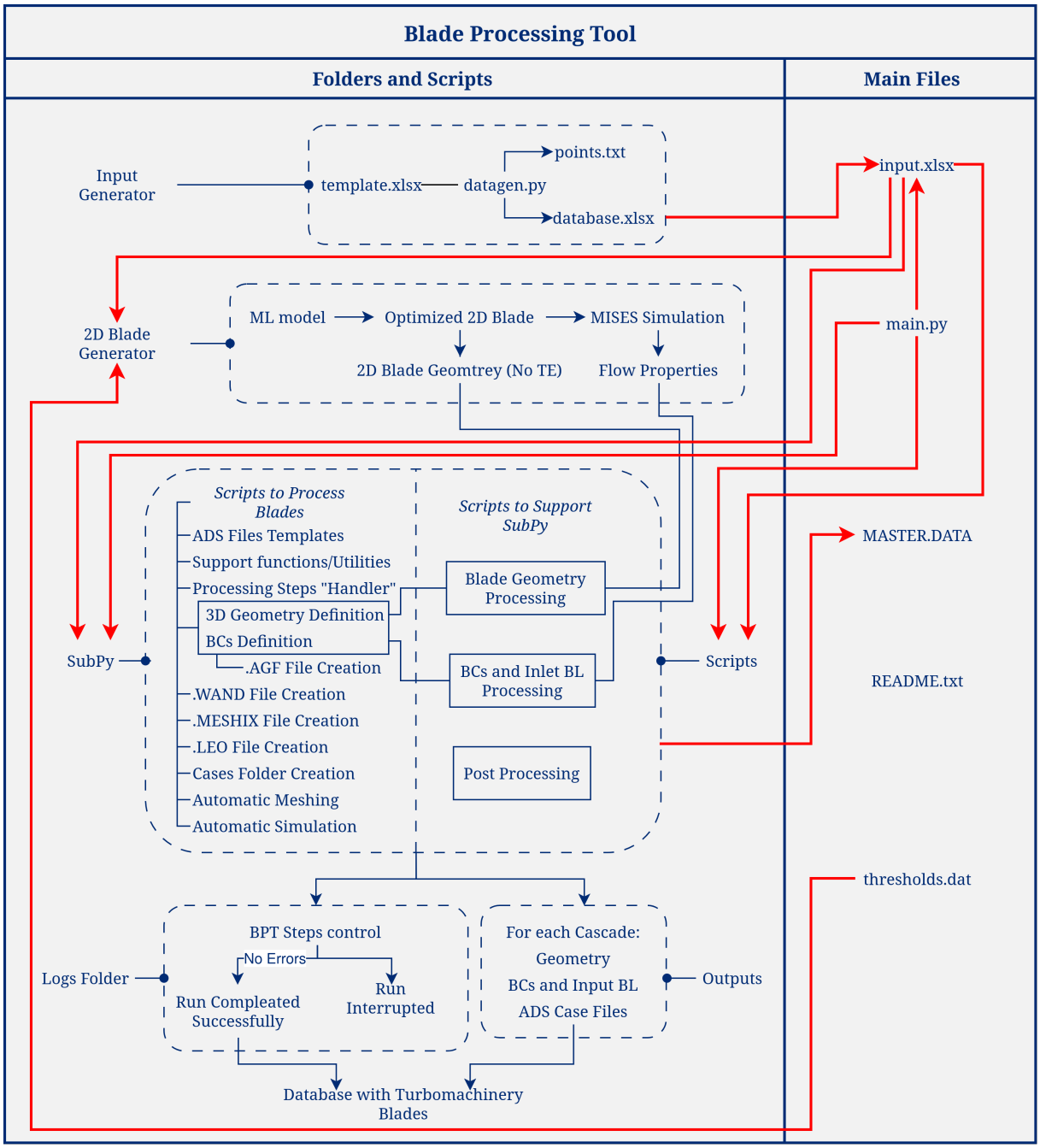


Figure 3.11: Scheme of BPT Workflow. With dark blue arrows/lines are reported the logical connections among BPT internal tasks, on the left part. Following the red arrows, instead, is possible to understand BPT workflow.

To better understand the scheme above, let's focus on main.py in the Main Files part, than

follow the the red arrows moving initially towards input.xlsx, than proceeding downward. Everything is controlled by the main.py script, that first read and load the .xlsx file, containing design input data for each blade of the database, than it launches 2D Blade Generator (by Porta [12]) N times, where N is the number of cascades that populate the final database, than proceeds with blade processing, implementing 2D blades profile finalization (TE closure), inlet BL and BCs computation from the previous version of the tool (by Scarimbolo [14]), after which blade processing continues up to the database creation.

3.4.1. Main Files

input.xlsx

The first thing that should be discussed also before main.py, following tool's logic, is the creation of the file `input.xlsx`, but it will be extensively reported in Chapter 4. Concerning the scopes of this chapter, it is important only to know that in this file, for each blade to be processed are reported a series of data, some of them needed by 2D Blade Generator, and others needed by SubPy and Scripts functions (see red arrows in fig.3.11).

blade #	alpha1 [deg]	alpha2 [deg]	M2 [/]	lpeak_ratio [/]	mpeak_ratio [/]	h/Cax [/]	Cax [m]	Re2_is [/]	BL Type	δ [/]	δ*/δ [/]	θ/δ [/]	Tu [%]	Mesh	Simulation
6	-45	65	0.4	0.6	1.2	4	0.05	1200000	2	0.05	0.143	0.111	5	0	0

Figure 3.12: A single row in the `.xlsx` file representing Blade 6 input data.

In fig.3.12 is shown an example of input data to be provided through the `.xlsx` file to the BPT. Once it is generated from the dedicated script (in Input Generator, fig.3.11), in order to be read and load it must be moved in the `/main` folder at the same level of `main.py`. Its name can be changed, the important is to maintain extension (`.xlsx`) and internal structure, since the functions in SubPy and Scripts are programmed to work with excel input files having this specific internal structure. The reason why such methodology was used to provide BPT the input data are two: this way the tool is more User-friendly, avoiding the need of prompting all data (such as in the previous version of the tool), allowing to reduce possible errors while defining the inputs. The second reason is that tool's workflow gets faster. For the sake of simplicity, to refer to this kind of logic the expression **input-output** should be used.

thresholds.dat & README.txt

These are not files of primary importance, but they are kept at the main level for the following reasons: `README.txt` contains a small guide on what the tool does and how to launch it, so it should be easily accessible and visible for the Users. `thresholds.dat` is a file containing, as understandable by its name, two threshold values for two control parameters (errors), used to filter out the 2D Blade Generator outputted blades. The author decided to keep it where it is, to give the User the possibility to control how accurately blades will be generated.

```
================================================================================
                          BLADE PROCESSING TOOL (BPT)
================================================================================

Author: Matteo Bonanni (matteo.bonanni@vki.ac.be)
Organization: Von Karman Institute for Fluid Dynamics (Be)
Department: TURBINE REG (ref: prof. Sergio Lavagnoli)

================================================================================
1. DESCRIPTION
================================================================================
The Blade Processing Tool (BPT) is a fully automated Python pipeline designed
for the generation, optimization, and CFD analysis of linear turbine cascades...
```

```
RMS = 0.08
angle_error = 1.1
```

Figure 3.13: `README.txt` content above, and `thresholds.dat` content below.

The thresholds in fig.3.13 are higher with respect to the ones reported by Porta in its work ($RMS_{th} = 0.04$ and $\Delta\alpha_{2,th} = 0.1$). This was intentional, allowing to process all the corner points of the database object of this work, since they are clearly more difficult to be predicted by the 2D Blade Generator. Indeed, the blades associated to these points have higher values of RMS and Angle Error, which means that they would be filtered out by the BPT if the thresholds were set to the values of Porta. The reader might find the simulation of blades outside the generator's validity range a forced strategy, but it has to be considered that in a database, the corners not always represent feasible points. However, also these points have been considered interesting in the frame of this work, expecting them to give less reliable results in the post processing phase.

main.py

The BPT version developed and used along the course of this work reports very important changes with respect to its previous version. One of these is related to the structure of the main script. By definition, the main script orchestrates the entire software's workflow, recalling all the modules, files, executable needed during program execution, leading to the tool's output. In its old version, main.py contained an initial section with all the supporting/helping functions, and a second very long section where blades were processed. With the old logic, the tool was subjected to a series of problems that now have been solved, such as:

- **Low Robustness and Modularity**: meaning that with minor codes variation, the tool used to break easily;
- **Hard-Coding**: meaning that the paths internal to the codes to make communicate scripts/files among each other were **absolute**. In other words, all the times the tool was moved from a computer to another, the tool stopped, since it wasn't capable of finding all the needed files in the new environment.

The first problem was strictly related to `main.py`. To solve it, in its new version was decided to break the very long blade processing procedure in more steps, and moving them in a SubPy module called `WORKFLOW_STEPS.py` (described in Subsection 3.4.3), so that `main.py` currently contains only the calls to these steps, simplifying debugging when needed. Than, all the supporting/helping functions have been moved in another SubPy module called `UTILITIES.py` (described in Subsection 3.4.3), in order to make `main.py` cleaner and more straightforward. That was how the robustness issue was solved. Clearly, better implementations are possible, but it can be considered a good result in terms of tool's logic enhancement. Hard-Coding, instead, was a problem common to all the scripts, and it was solved through the introduction of a path variable, at the beginning of each code, pointing to the main folder, making possible to refer all the other paths in the script to it. This way, moving the main folder will not cause path issues anymore. These problems and their resolution are addressed and discussed here for two reasons: they were noted while working on `main.py`, they prevented main.py to be easily modifiable, inducing considerable loss of time while debugging it/adapting it to ADS CFD.

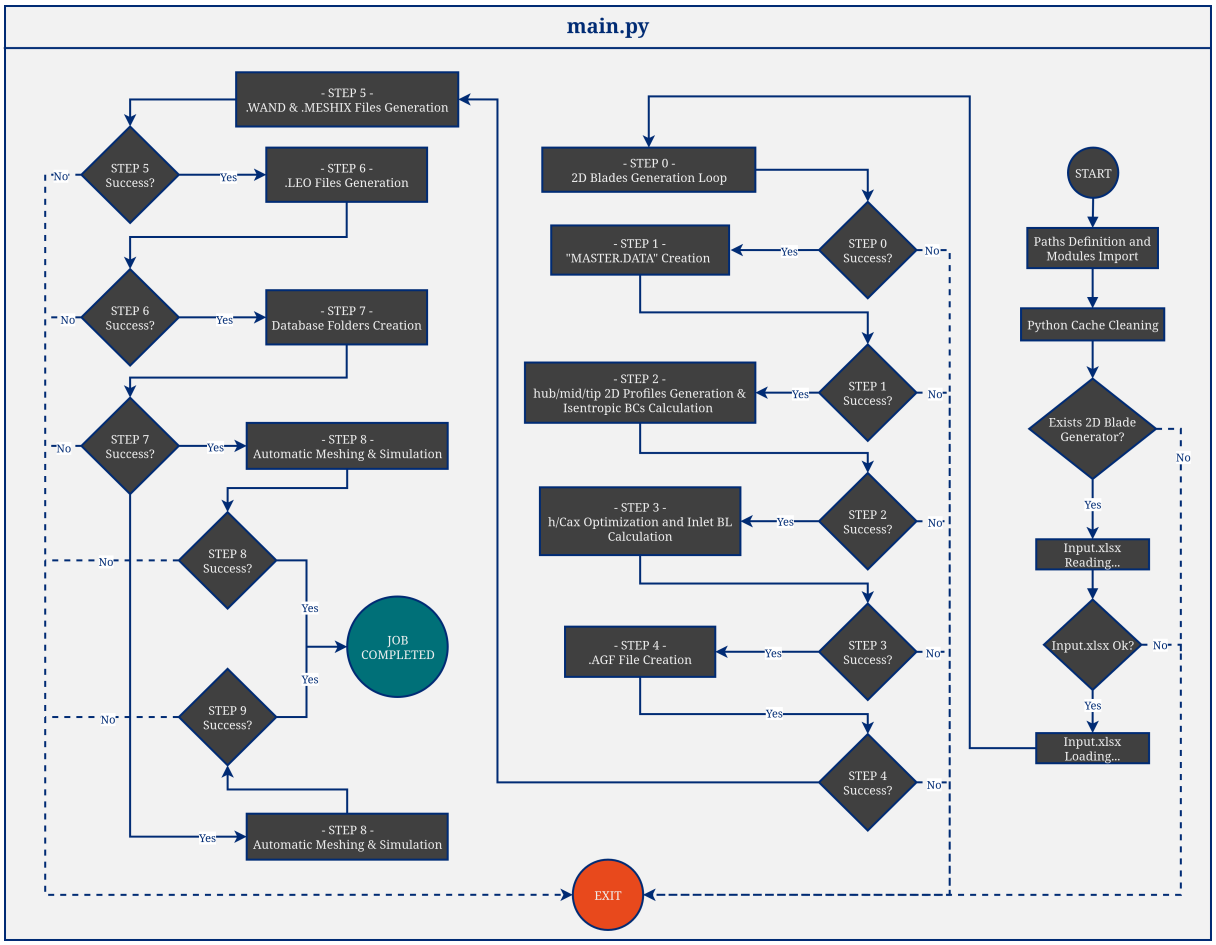


Figure 3.14: Scheme showing `main.py` logic.

In fig.3.14 is schematized the logic of `main.py`. It seems apparently intricate, but it was designed to be as much straightforward as possible. `main.py` is launched by command line through a LINUX terminal connected to a dedicated virtual environment. First, all the needed paths to files/modules/etc... are created, and than all the needed modules are imported. After that, the User is required to select if the Python cache has to be cleaned. This option is present because cache has little effects on BPT execution performance (see `README.txt`), and default choice is no clean. Proceeding, a check on 2D Blade Generator presence in the main folder is done, since it is essential for the BPT. If no errors occur, `Input.xlsx` is first read and checked, if no errors are present in the document it is loaded and printed on terminal. Here starts the core of `main.py`, where the **STEPS** are recalled leading to blades processing. From the first to the last step, the execution logic is always the same: perform the step, check the output log generated by that step, than if no errors occurred go ahead to the next step. Success for the last step means that all the simulations and hence related meshes, for all the blades specified in `Input.xlsx`, have

been correctly executed/generated with with no errors. This identifies the status of JOB COMPLETED. Vice versa, when 2D Blade Generator is not detected, or `Input.xlsx` contains any error, or the output log for any step contains any error, the BPT execution is stopped (EXIT in the scheme).

MASTER.DATA

During tool's execution, for each cascade are imported/calculated several values and created files. It could be needed to check them, therefore a text file collecting these data is created in STEP 1. Its content is reported in the following figure:

```
BLADE N.       Mpeak / M2      Lpeak   alpha1 [deg]    alpha2 [deg]    M2      Angle error [deg]       RMS     h/Cax   Re2_is  Cax_m   Flag    Stagger_angle   M1      T1      P1      rho1
       V1      T2      P2      rho2    V2      T0      P0      BL_Type Delta   Delta_Star_Ratio        Theta_Ratio     Tu_Percent      BL_Method       Pitch   delta   n       H12
       delta_star_ratio        theta_ratio     cell_width      Tu      ILS     AGF_File        WAND_File       MESHIX_File     LEO_File

1      1.10000 0.40000 40.00000        -60.00000       0.40000 0.67000 0.27577 2.00000 1200000 1       ✓       -26.17286       0.24629 296.40408       12680.15015     0.14906
       84.9953 290.69767       11846.09595     0.14199 136.70542       300.00000       13226.78172     1       0.20000 0.37900 0.14500 5       2       0.80774 0.20000 1.72461 2.15970
       0.36703 0.16994 4.1900e-05      0.05000 0.10000 Blade_1.AGF     Blade_1.WAND    Blade_1.AGF.MESHIX      Blade_1.LEO
```

Figure 3.15: `MASTER.DATA` file structure. Two blocks can be noticed for each processed cascade: one on top and one on bottom.

Can be observed that in fig.3.15 two blocks are present, one on top and one on bottom. The top block reports all the variable names/kind of files, while the bottom one reports their values/names. These are the data for a single cascade, hence the final `MASTER.DATA` file will report this two blocks for each processed cascade in the database. At the beginning, just after being created in STEP 1 it is void, than at the end of each step it is updated with new imported/calculated data. To be noted the presence of the ADS case files names. These have been put in the file because it is intended to be a list collecting all the important outcomes of the tool, and among these there are, obviously, also the case files needed for cascade processing.
`MASTER.DATA` was present also in the old version of the tool, but with a different name (`blade_data_file.DATAinput`).

3.4.2. Folders and Scripts: 2D Blade Generator

Let's move, now, on the left part of the scheme in fig.3.11. Here are shown all the folders containing custom made python modules used in the BPT workflow and also the folder containing the 2D Blade Generator, object of this section. Here, for brevity will be mentioned only its main functionalities contextualized in the BPT workflow.
Porta started from the work of Clark [4], from which the concepts of Aerodynamic Load and Style are used to identify turbomachinery blades. Specifically, to a set of five parameters describing aerodynamic style and load is assigned a 2D blade geometry. The

generated blades, however, are subjected to a certain error with respect to the real blade geometry associated to that specific aerodynamic load and style. The error is measured in terms of load and outlet flow angle. A batch of blade geometries has been generated with the idea to keep, for each blade, these errors as low as possible, considering a range of variability for each one of the five parameters mentioned above. Therefore this process generates a database of optimized blades, later used to train a ML model that gets in input the five parameters, delivering in output the 2D blade geometry (with no TE) that matches aerodynamic style and load associated to the five input parameters.

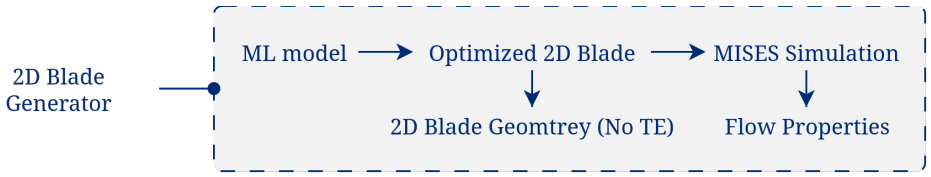


Figure 3.16: Recalling BPT scheme: this figure shows in a gross way how the 2DBG works in the context of the BPT workflow.

Clearly, the 2DBG is more complex than how is represented here, but this schematization can be considered enough for this work's purposes. The reader can find the complete explanation on how the 2D Blade Generator works reading the work of Porta [12].

3.4.3. Folders and Scripts: SubPy and Scripts

This part of the tool is the most important one, containing all the python modules used by the BPT. It also accounts for the biggest programming effort of this project, considering the relevant amount of files produced. **SubPy** and **scripts** are two folders: the first can be considered the one containing principal modules, while the second contains supporting functions used for the codes in SubPy. The logic behind this two folders is the same of the old version of the tool, what changes are the python scripts in them. Due to the use of a different CFD simulation software and enhancements made on the tool, in terms of modularity and robustness (discussed above), the most of the old version modules have been removed/deeply changed to match with the new BPT version.

WORKFLOW_STEPS.py

To start, let's introduce the module `WORKFLOW_STEPS.py`, inside SubPy. It replaces the definition of the blade processing logic in the old `main.py`, that was fully contained in a long FOR cycle making the tool slow and weak, with a series of **STEPS** reproducing the same blade processing logic, but tailored to work with ADS CFD.

Figure 3.17: Schematic representation of `WORKFLOW_STEPS.py`, including the processing STEPS.

Looking at fig.3.17, can be observed that `WORKFLOW_STEPS.py` is called by `main.py`. Specifically, it is called for eight times, since eight is the number of STEPS in which the blade processing procedure has been divided, but also the number of functions contained in the file (a function for each STEP). The STEPS numbering is not random, it represents the order in which they are called by `main.py`.
Hereafter, STEPS are described more in detail, showing their dependencies with the other modules in SubPy and scripts folders.

» STEP 0

This is the function calling the 2D Blade Generator. In fig.3.14 is shown that this call can be performed after `Input.xlsx` loading, since it contains input data necessary to launch the blade generator. `main.py` calls STEP 0 recursively N times, where N is the number of blades to be processed defined in the database. Results are automatically saved inside the 2D Blade Generator folder, in a sub-folder named "Generated_Blades".

» STEP 1

At the end of STEP 0, the blade generator creates also a file with extension `.dat` (`"blade_data_file.dat"`), that contains, for each blade generated, the five input parameters required by the blade generator and the two errors: on the load (**RMS**) and of the outlet flow angle (**Angle error**). In STEP 1, at first is created `MASTER.DATA`, file already presented above, where are copied inside all the data present in the `.dat` file. After that, a **filter** is applied to each generated blade, taking the two values reported in `thresholds.dat` (on the left of fig.3.13) as reference to filter out blades. For instance, if a blade has RMS and Angle error smaller than the threshold, it assumes the status of **valid blade**, so that a blade that can continue the processing. All the blades that do not satisfy the thresholds are considered not valid, hence they are discarded and not processed. This is done because if the blade's errors are too high, it is not representative of the load and style used to define it. Finally, the `blade.XdatabladeVALIDATION` file

(where X is the blade identification number in `Input.xlsx`) that reports the coordinates of the generated 2D profile, which is contained in the specific blade's folder in Generated_Blades (STEP 0 outputs), is copied and pasted in the tool output folder, which will be extensively commented towards the end of this chapter in section 3.4.5. Once pasted, files name are changed into `blade.POINT_X` for simplicity reasons. A schematic view of STEP 1 can be seen in fig.3.18.

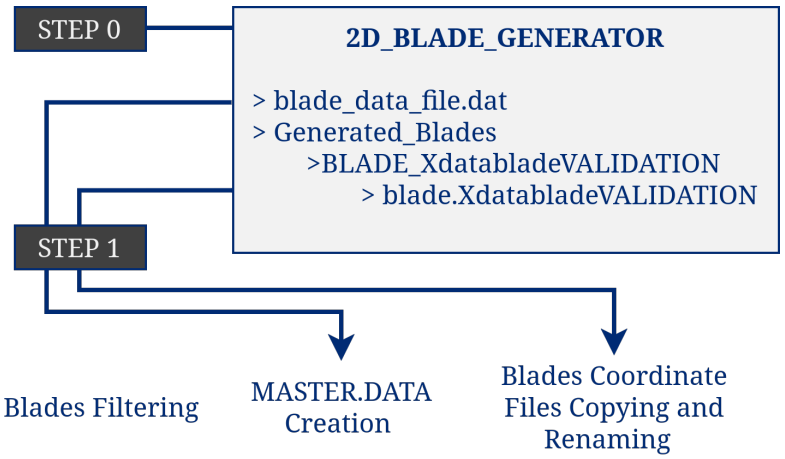


Figure 3.18: STEP 1 schematic representation.

The logic behind this STEP was present also in the old version of the tool. In this new version it was enhanced and tailored for the purposes of this work.

» STEP 2

Arrived at this point, the BPT workflow continues making two things: **3D blade sections** definition and **isentropic BCs** computation. Let's start reporting the first one.
Remember that the geometrical environment the tool has to reproduce is the one of linear cascades of turbomachinery blades (subsonic turbine blades considered as reference). This requires the definition of a blade as a 3D object in space, placed side by side. As can be seen in fig.3.4, to recreate the mentioned geometry ADS CFD requires to specify: number of blade's spanwise sections (three has been chosen, specifically hub, midspan, tip sections, all with the same coordinates but placed at a different radius, since the cascade is linear) with their coordinates, and the number of blades in the entire cascade to determine the pitch. In STEP 2 are created the three text files (hub, midspan and tip sections) containing the profile coordinates for each blade. Hence, if N valid blades are processed by the BPT, there will be created $3 \cdot N$ text files, opportunely saved in the outputs folder of the tool. These three profiles points will be interpolated by ADS through the `.AGF` file, to create the final 3D blade geometry. Later on, thanks to a sub-routine

already implemented in the old version of the tool, but modified and enhanced in this new version, the second objective of STEP 2 is addressed, computing the isentropic BCs. These are not representing the final flow conditions used for the database creation, but are preliminary calculations made to perform STEP 3 described below, specifically to be able to perform the h/C_{ax} optimization.

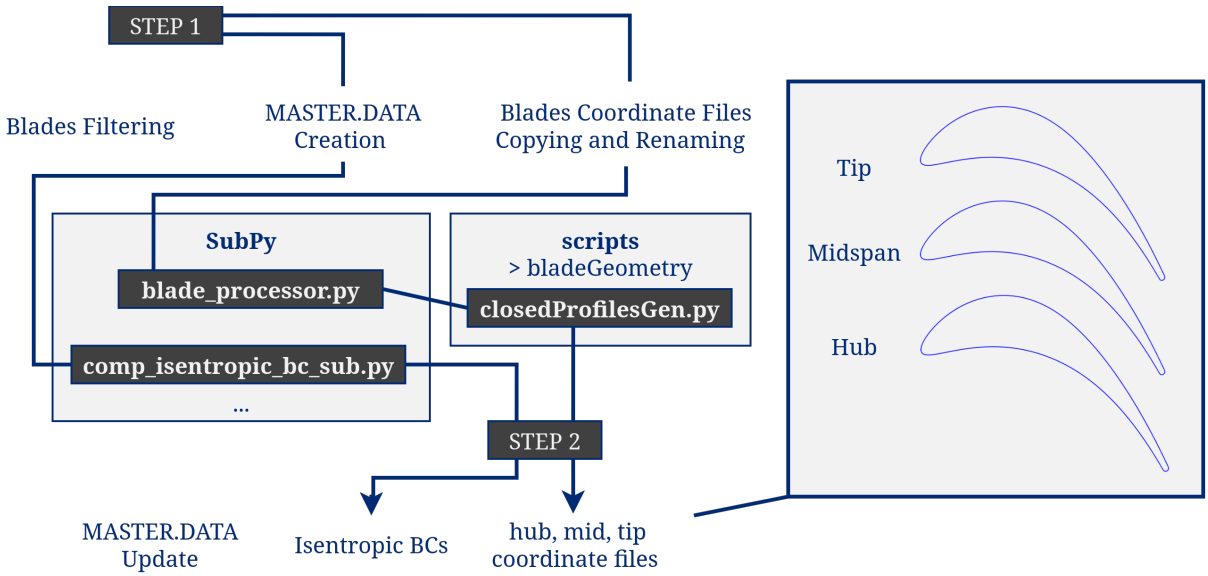


Figure 3.19: STEP 2 schematic representation.

In fig.3.19, on the left, can be identified the scheme of the custom made modules dependencies for STEP 2 execution, while on the right, the three profiles (hub. mid, tip) that are interpolated to create the 3D blade geometry. Profile's TE are closed with a NURBS curve [14], and the code to do that is contained in `closedProfilesGen.py` script.

» STEP 3

This STEP performs two phases relevant for the study of secondary flows: first, using data defined in `Input.xlsx`, the cascade's **input BL** parameters are calculated. As reported in literature, a BL can be defined using ratios calculated using some relevant quantities $(\delta, \delta^*, \theta)$, or either using a power law. In this work, the first mentioned strategy for BL definition (ratios) has been adopted, using two sets of ratios in the `.xlsx` file, allowing for the definition of two BLs, a thick one and a thin one. The effect of different inlet BLs is relevant when studying secondary flows in turbomachinery cascades, as referenced in literature. After that, the h/C_{ax} **optimization** loop is executed. This serves to find an admissible value of h/C_{ax} such that h, the blade span, is higher than the secondary flows spanwise penetration at blade's TE, given by z_{TE_Cx} (called "Penetration Height").

Such parameter was calculated using the Sharma & Butler model [15], but also other correlations are available in literature, such as the one presented by Benner et al. [3].

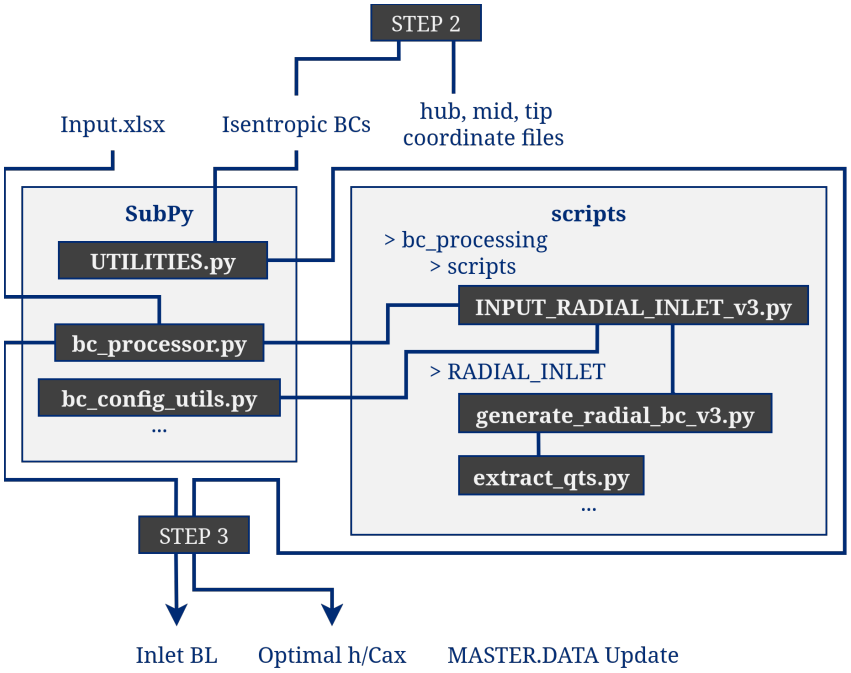


Figure 3.20: STEP 3 schematic representation.

Fig.3.20 shows that `bc_processor.py` calls `INPUT_RADIAL_INLET_v3.py` (a sub-processor that orchestrates inlet BL and BCs radial profile creation), which in turn calls `generate_radial_bc_v3.py` and `bc_config_utils.py`, which calculates the power law exponent to define the inlet BL starting from the ratios, in case it has/wants to be used for future implementations. Also, `generate_radial_bc_v3.py` recalls `extract_qts.py`, a script that reads MISES simulation outputs, which are used to compute BCs in the BPT. Isentropic BCs are directly used inside STEP 3 for z_{TE_Cx} computation, through a function defined in `UTILITIES.py` (explained below). Once Penetration Height is computed, the optimal h/C_{ax} is given. The computed inlet BL profile is reported in the tool's output folder (described below) as `Ptot.p`, a file collecting the inlet total pressure radial distribution (100 points) defining BL shape.
The logical content of this STEP, such as related scripts/folders hierarchy, has been kept the same of the one present in the old version of the tool, differently from the two previous STEPS, where modifications have been done.

» STEP 4

Proceeding, STEP 4 performs some preliminary operations, and finally creates the `.AGF` file for each blade being processed in the tool. Specifically, the preliminary operations consist in re-defining the final blades geometry thanks to the optimized h/C_{ax} value computed above, and in re-calculating the BCs for the last time. Even if they seem repetitions, these two operations, at this point of the tool's workflow, are necessary for the successful creation of an ADS case as well as for the database. Than, the `.AGF` file is compiled with the aforementioned data, creating the first file needed for the ADS case definition. This file is particularly important, carrying information about cascade's BCs and geometry as reported in Section 3.3.

In fig.3.21 is reported the schematization of STEP 4, where three modules and their interactions are highlighted. To re-define the final blade shape, `blade_processor.py` is used at the same way of what reported for STEP 2, while for computing the final BCs at midspan position, `comp_isentropic_bc_sub.py` is used. The outputs of these two modules, together with the information of the inlet BL from STEP 3, are sent in input to `AGF_processor.py`, a script designed to create a specific `.AGF` file for a given blade modifying a template file. Specifically, the processor makes a copy of the template file, and than substitutes all the values that are blade dependent (parameters for defining BCs, inlet BL and blade profiles coordinates). This script is recursively called by `main.py` a number of times equal to the number of valid blades under processing.

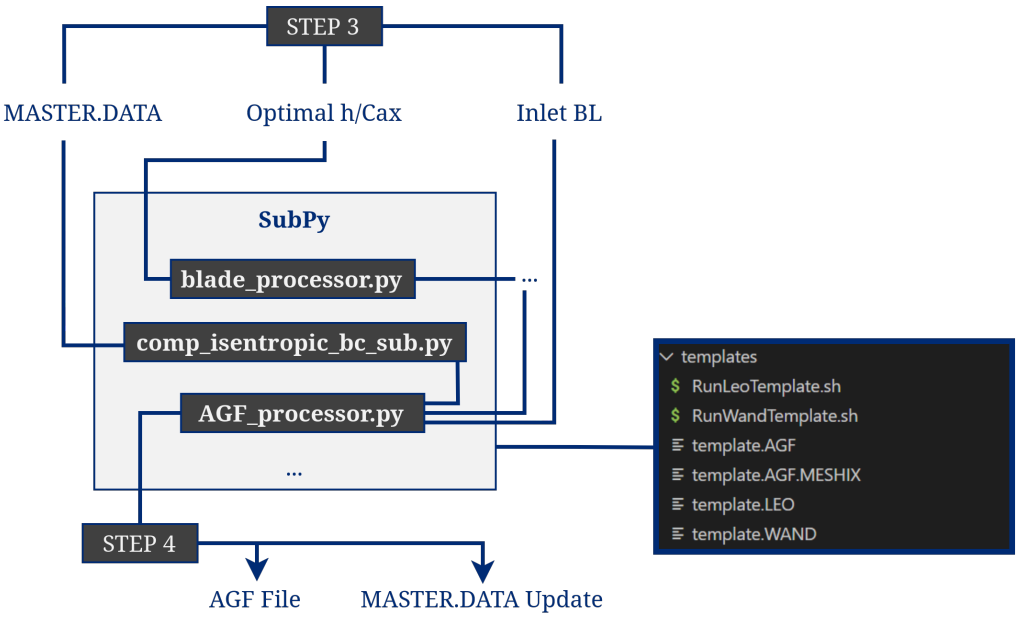


Figure 3.21: STEP 4 schematic representation.

Templates are shown on the right side of the figure, in the red box, and as can be noticed there is one of them for each file needed in an ADS case. They are collected in a SubPy sub-folder, and they are not necessarily unique, in the sense that more templates for an ADS File can be defined, depending by User needs.

» STEPS 5 & 6

At this point, the meshing files are created in STEP 5 through the use of an additional module, called `MESH_processor.py`. It handles both `.WAND` and `.MESHIX` files creation for each valid blade of the database. After that, an additional module named `LEO_processor.py` is used in the BPT workflow to create the `.LEO` file, needed to setup the solver. Differently from the AGF processor, `MESH_processor.py` and `LEO_processor.py` are not blade dependent, making a copy of `.WAND`, `.MESHIX` and `.LEO` templates without modifying any of their data. Therefore, similarly to what said also for the `.AGF` template file, to change mesh properties and solver settings the related templates have to be substituted.

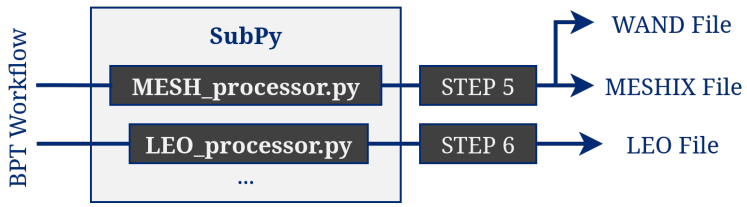


Figure 3.22: STEPS 5 and 6 schematic representation.

» STEP 7

After all ADS case files are ready for all the blades processed, the BPT workflow executes STEP 7 to create the **database cases folders**. At the very beginning of this chapter, in fig.3.1b is shown `/database` folder containing the cases populating the database itself (`Blade_1`, `Blade_2`, etc...). Each of these case folders contains four text files, which are the ones above created, plus two executable files needed to run meshing and simulation, already explained in detail above, for a total of six files. The latter two files are the same for each case folder, hence they are simply copied from their templates. The database folders creation is handled by a module named `DATABASE.py`, as reported in fig.3.23.

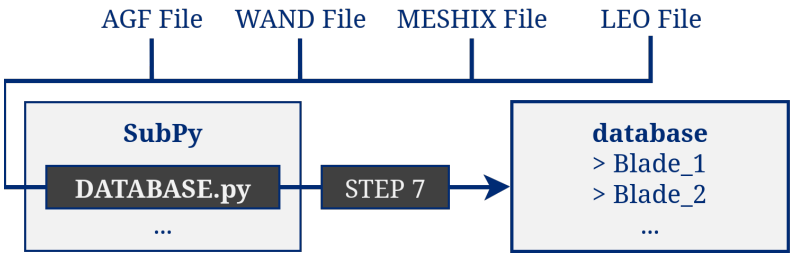


Figure 3.23: STEP 7 schematic representation.

» STEP 8

This STEP closes the cascades processing procedure by performing automatic meshing and simulation. These operations are handled by choice, properly setting the related flags in `input.xlsx` (1 means do mesh/sim. and 0 means do not do mesh/sim.; check fig.3.12) for each blade. It is quite obvious, but at the same time important to highlight that a simulation can be performed only after having the case related mesh, hence meshing and simulation procedures are handled sequentially for each blade. For instance, the BPT executes STEP 8 by first generating the mesh, and than performing simulation for `Blade_1`, than it does the same for `Blade_2`, and so on for all database points considering the flags in `input.xlsx`. The possibility to control meshing and simulation procedures activation with a flag enables the BPT usage also as a **mesh generator**.

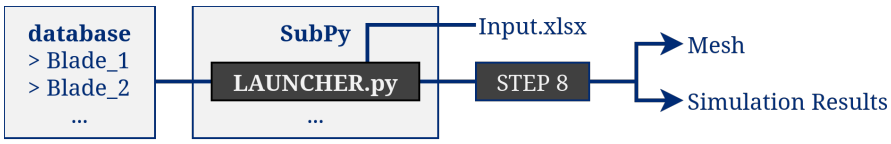


Figure 3.24: STEP 8 schematic representation.

Mesh and **Simulation Results** are both generated in the relative blade folder, according to how ADS works.
Another functionality has been thought in this step, that is a check for the presence of the ADS software in the Linux environment from which the BPT is launched. This functionality was implemented to give the tool an additional level of robustness.

» STEP 9 (Post Processing)

Finally, the last phase of the BPT workflow is the Post Processing. Once simulations are completed, the flow fields generated by the CFD solver contain a large amount of data.

To use these data for the subsequent Machine Learning training phase, a robust data reduction procedure is necessary. This step extracts the relevant quantities of interest, summarizing the complex aerodynamic behavior of each processed blade into discrete parameters.

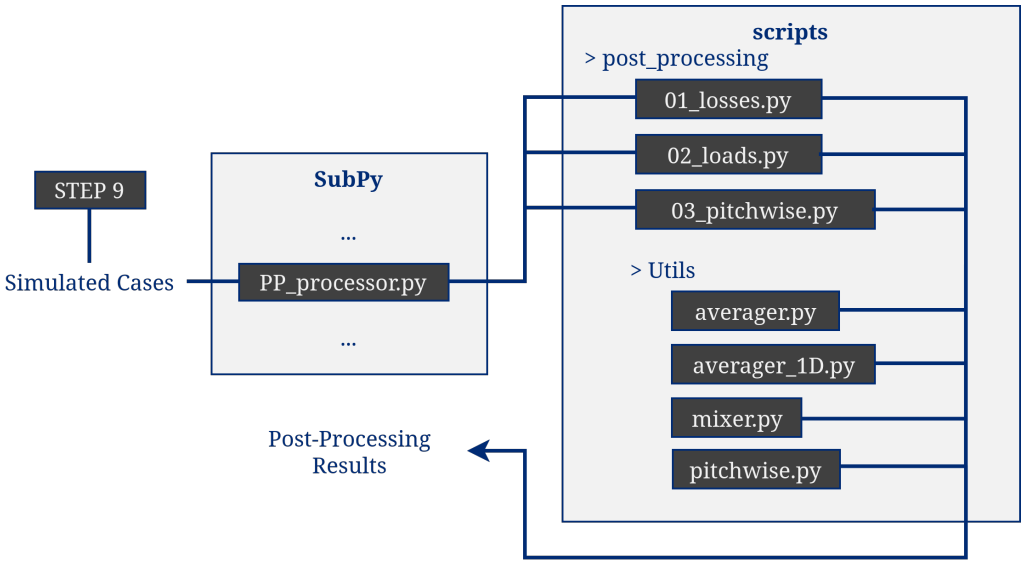


Figure 3.25: STEP 9 schematic representation.

By default, the BPT workflow concludes its operations after the execution of STEP 8 (meshing and simulation). To avoid excessive storage requirements, and to allow the User to eventually perform customized data extraction at a later moment, the post-processing phase is handled as an independent function of the tool. To activate it, the main script must be executed via command line using a specific flag:

```
python3 main.py -pp
```

If this flag is omitted, the tool will stop after the cases simulation, leaving the output files untouched in their respective folders.
When the `-pp` flag is used, the BPT accesses the modules dedicated to data extraction, which are located in the `/main/scripts/post_processing` directory. To maintain high modularity throughout the tool, post processing operations have been divided into three scripts, each one dedicated to specific quantities to be evaluated:

- `01_losses.py`: This module extracts all the needed quantities to evaluate the cascade's aerodynamic losses. Specifically, it reads the flow variables at the inlet and outlet planes (see fig.3.26) in the fluid domain to compute the losses. In the context of this work, the relevant losses evaluated with CFD calculations are total

(Y_{tot}, ζ_{tot}) and profile losses ($Y_{profile}$, $\zeta_{profile}$), both in terms of total pressure and kinetic energy, as well as endwall losses ($Y_{endwall}$, $\zeta_{endwall}$, $\zeta_{endwall-Coull}$), also in this case evaluated in terms of both total pressure and kinetc energy. Endwall losses have been evaluated also using the Coull model [6], in order to validate the results extracted by CFD. Together with the losses, also the most relevant fluid-dynamic quantities, at both inlet and outlet planes, averaged with respect to mass or area (depending on the quantity). All the extracted quantities, for each blade, are then reported in the `losses.txt` file, a snapshot of which is reported in the Chapter 4 section where database post-processing results are shown.

- `02_loads.py`: This script focuses on the blade surface, extracting from it some of the most relevant quantities giving info about its aerodynamic load distribution. The load is quantified through specific dimensionless parameters, namely the Zweifel coefficient, isentropic Mach number distribution, pressure coefficient, diffusion factor and the circulation coefficient C0, a parameter extracted from [5]. Among these, for each blade, the isentropic Mach number distribution and pressure coefficient are plotted, while the others are reported in the `loads.txt` file. Below, the equations used to compute such quantities:

$$C_p = \frac{P_{01} - P}{P_{01} - P_2} \tag{3.5}$$

$$DF = \frac{V_{SS}^{max} - V_{TE}}{V_{TE}} \tag{3.6}$$

$$Zw = \frac{\left|\oint P\ dx\right|}{C_x \cdot \Delta P} = \frac{\left|\int_{SS} P\ dx\ -\ \int_{PS} P\ dx\right|}{C_x \cdot (P_{01} - P_2)} \tag{3.7}$$

$$C_0 = \frac{\Gamma}{V_{2,is} \cdot L} = \frac{\left|\int_{SS} V\ dl\ -\ \int_{PS} V\ dl\right|}{V_{2,is} \cdot L} \tag{3.8}$$

Eq.3.6 defines the **Diffusion Factor** (DF), which quantifies the flow deceleration on the final part of the suction side (V_{SS}^{max} to V_{TE}) to assess the risk of boundary layer separation. Eq.3.7 introduces the **Zweifel Coefficient** (Zw), a dimensionless number comparing the actual aerodynamic load ($\oint Pdx$) against the ideal maximum load achievable by the cascade. Finally, Eq.3.8 represents the **Circulation Coefficient** (C_0), which normalizes the kinematic circulation Γ by the pitch (L) and isentropic exit velocity, providing a direct measure of the cascade's flow turning

capability.

- `03_pitchwise.py`: This third script is dedicated to the evaluation of the spanwise distribution of relevant flow quantities. Since the cascade is linear, but 3D effects (like secondary flows) are present, it is crucial to understand how the flow behaves moving from the hub to the shroud. This script extracts the desired physical quantities across the passage and computes their pitch-wise averaged values at several spanwise locations, providing a clear 2D representation of the 3D flow structures. The trend of some quantities, such as the outlet flow angle, is reported in dedicated plots located in the post-processing folder described below. These pitch-wise distributions are also reported in the `pitchwise.txt` file, stored in the same folder of the plots.

Each of these scripts can work integrated in the BPT natural workflow, or in the so called "stand-alone" mode, meaning that they can be run alone. This second operative mode was foreseen primarily for debugging purposes during BPT design, but also to allow the User the freedom to post-process the simulation results in a second moment.

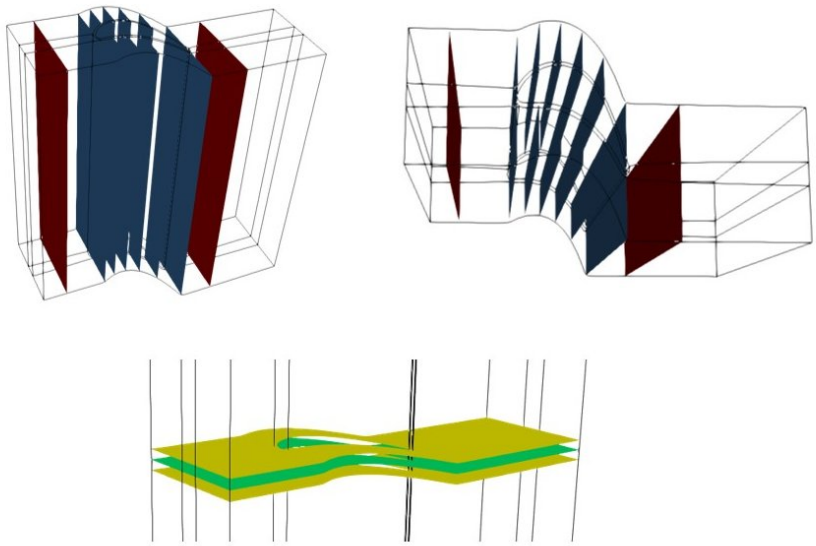


Figure 3.26: Planes used to extract post-processing quantities. On top are show for clarity two views of the axial planes: the red ones are the **inlet and outlet planes**, while the blue ones are "intermediate" planes, used to capture secondary flows evolution along blade passage. On bottom are reported the mid-span plane, in green, and the two limiting planes (enclosing the 5% of the blade span, centered with respect to the mid-span) used to define the **inlet and outlet slabs**, as the crossing plane given by the intersection of inlet/close TE and the yellow planes.

A key aspect of this architecture is the `/post_processing/Utils` folder. The three main scripts described above act as "orchestrators": they manage the general computational logic behind all the formulas, but the actual mathematical operations and numerical integrations used to compute the physical quantities are inside the `/Utils` directory, collecting four scripts: `averager.py`, `averager_1D.py`, `mixer.py`, `pitchwise.py`. These sub-modules represent the mathematical tools of the post processing phase. This design choice ensures avoiding every time to re-define their logic in different scripts, while any future mathematical implementation or update can be injected directly into them.

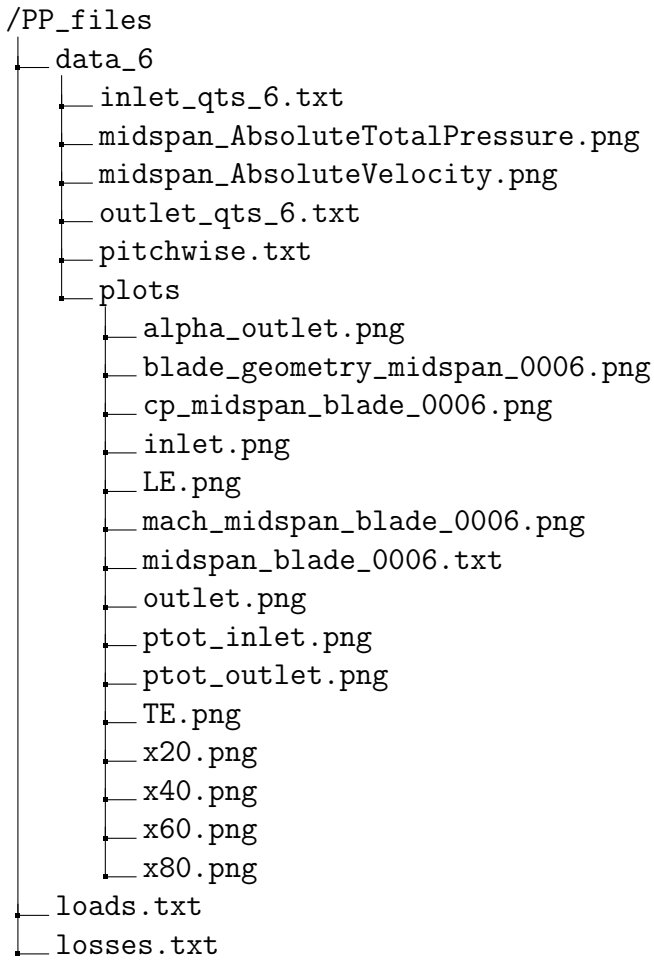


Figure 3.27: Internal structure of the `/PP_files` folder, containing the post-processing of a single blade.

The outcomes of post-processing are organized inside the `/main/PP_files` directory. This folder avoids saving heavy simulation directories and gathers all the results in a single place. Inside `PP_files`, the tool generates two global text files: `losses.txt`, which lists the outcomes of `01_losses.py` for every single blade in the database, and `loads.txt`,

which collects all the dimensionless load coefficients, always for every single blade in the database. Alongside these global files, the tool generates a dedicated sub-folder for each processed blade. Inside it, the BPT saves plots of different kind. A detailed representation of the `/PP_files` folder's content is shown in fig.3.27, where `inlet_qts_X.txt` and `outlet_qts_X.txt` collect the values of the most relevant fluid-dynamic quantities in each cell of the inlet and outlet planes, `midspan_AbsoluteTotalPressure.png` and `midspan_AbsoluteVelocity.png` that, respectively, represent the relative contours, and all the images in the `plots` folder, well described by their names.
The fully post-processed data obtained through this procedure represents the final, synthesized output of the Blade Processing Tool and constitutes the core of the database used for the Machine Learning model. A comprehensive review of these results, along with the detailed analysis of the databases produced, will be extensively discussed in Chapter 4.

» STEP 9 (Post Processing - Averaging Methods and Slabs)

All the quantities needed for the computation of discrete global parameters, such as losses and fluid-dynamic quantities, have been averaged with respect to the mass flow rate, except for the static pressure that has been averaged with respect to the area, due to its intrinsic physical meaning.

$$\bar{\phi}_A = \frac{\int_A \phi \, dA}{\int_A dA} = \frac{1}{A} \int_A \phi \, dA \tag{3.9}$$

$$\bar{\phi}_m = \frac{\int_A \phi \rho \vec{v} \cdot \hat{n} \, dA}{\int_A \rho \vec{v} \cdot \hat{n} \, dA} = \frac{1}{\dot{m}} \int_A \phi \, d\dot{m} \tag{3.10}$$

These two equations provide output the area-average (Eq.3.9) and the mass-flow-avarage (Eq.3.10) of a generic quantity ϕ. The CFD domain is discrete, so that composed by small pieces that, together, try to approximate the real, continuous world. Therefore, these equations will turn into finite summations when used in the CFD context.
For what concerns the computation of total and profile losses, both have been computed using the same expression (Eq.2.2), but using different planes. Specifically, the inlet and outlet planes (in red in fig.3.26) were used for the total losses, while two slabs, at inlet and suddenly downstream trailing (5% of C_{ax} from TE) were used for the computation of profile losses. Slabs are used to avoid numerical instabilities due to the non perfect mesh refinement, that would instead be visible among extracted data if instead of slubs, simple segments on the mid-span section were considered.

UTILITIES.py

This module contains all the supporting and helping functions used by the other modules, mainly `main.py` and `WORKFLOW_STEPS.py`. In the old version of the tool, some of the functions contained in it were in `main.py`, than they have been separated in order to enhance tool's robustness and modularity. Recalling some of the functions, there is "inputDB", used to load `input.xlsx` in the BPT, "signal_handler" that is used to kill tool's execution with the `Ctrl+C` command, and also "saveStepOutput", a function called at the end of each STEP in `WORKFLOW_STEPS.py`. This creates a text file that contains all the most relevant outputs of the STEP for which it is called. Substantially, it creates a `MASTER.DATA` file with the data of a single STEP.

Figure 3.28: Working principles of "saveStepOutput" in `UTILITIES.py`

The main objective of the BPT is to process many blades, therefore is important to have a strategy to check step by step what results the tool is producing. "saveStepOutput" was implemented to satisfy this need, specifically in terms of computed parameters values, allowing the User to control if the STEPS are producing what expected. This check could be performed also by only considering `MASTER.DATA`, but due to the high amount of data contained in it at the end of a run, it would be difficult to diagnose a STEP related error. Paying attention to the scheme in fig.3.28, only STEPS from 1 to 7 have an output file, while STEP 0 and 8 no. This is because STEP 0 already generates, automatically, an output file with all the results, and in STEP 8, ADS already generates text files containing the results of meshing and simulation. Therefore, for STEPS 0 and 8 the step output file would result being a repetition.

3.4.4. Folders and Scripts: Logs Folder

Another tool's diagnostic strategy has been implemented through the **output log files** creation. These files report the STEPS output logs, where are printed messages giving information on how the BPT is performing the job and what tasks have been performed successfully or not. It is relevant to highlight the difference between log and step output files, seen above, since they could seem representing the same thing: the first ones are reporting what is happening in the STEPS by actual text messages, while the second ones

report only the values of the variables/names of produced files, so that the results. Besides, having two different files typologies (log and step output) for diagnostics purposes, should favor tool's workflow control.

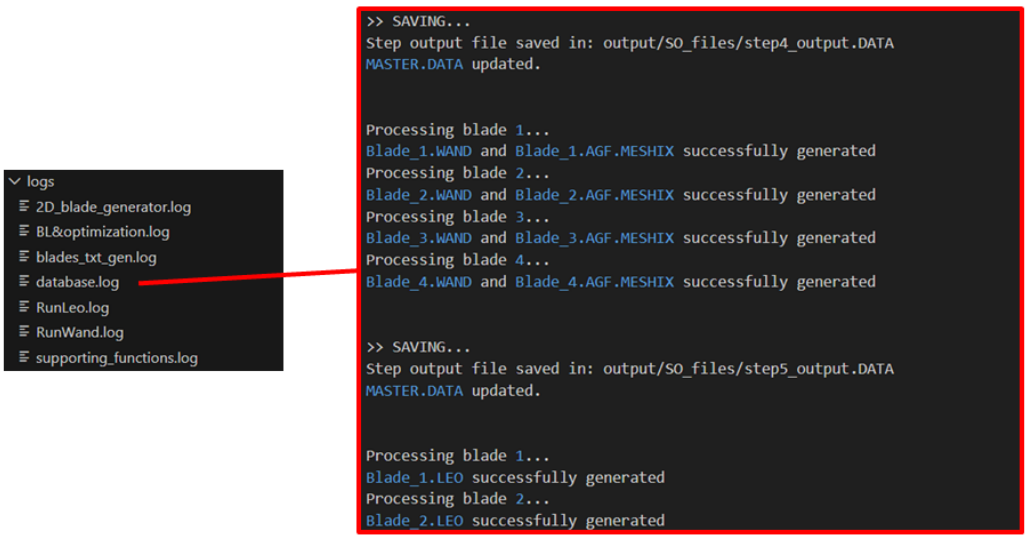


Figure 3.29: STEPS log files in the `/logs` folder.

Making reference to fig.3.29, log files are collocated in the `/logs` folder. Here, in a single log file there could be the logs for also more than one STEP. This was done for logical reasons, enclosing in a single log file the logs of the STEPS logically linked in the processing procedure. For example, in the `database.log` file there are the logs of the STEPS 4,5,6,7, that enclose the generation of the entire ADS case folder. `supporting_functions.log`, instead, contains the logs of the supporting/helping functions in `UTILITIES.py`.

3.4.5. Folders and Scripts: Outputs

Finally, the `/output` folder is the one containing all the files generated along the tool's workflow, which a schematic representation is reported in fig.3.30.
Following the tool's workflow, the first folder generated and that will be discussed is `BLADE_files`, containing two sub-folders: `2D_BG` that stores the blades coordinates text files generated by the 2D Blade Generator, and `txt`, a folder storing the three blades coordinates text files used by ADS to create the 3D blade shape, therefore, for each blade, hub, midspan and tip profiles. Another folder in `/output` is `BC_files`, storing a series of sub-folders and a text file (`bl_config.dat`) containing all the specifics about the inlet BL generated, for each blade. In each sub-folder there is a `Ptot.p` file, which means

Figure 3.30: Scheme of the `/output` folder - Internal structure.

a document reporting the spanwise total pressure profile at cascade's inlet. Proceeding there is `AGF_files`, a folder storing the `.AGF` files for each blade processed, `MESH_files`, a folder with inside both `.WAND` and `.MESHIX` files for each blade processed, `LEO_files` storing `.LEO` files for each blade processed, and finally `SO_files`, where the step output files described above are saved.

3.5. BPT Outcomes and Performances

The BPT has been developed such that it could be easily moved from a machine to another, for this reason it is contained in its completeness in a single folder, collecting three sub-folders: `/main`, `/database` and `/predictor`. The first folder has already been extensively discussed along this chapter, while the other two only mentioned.
`/database` is the folder where the CFD simulations outputted by the BPT are stored. It stores a number of sub-folders equal to the number of blades processed by the tool, for a run. The general sub-folder's name is `Blade_X`, where X is a number identifying a cascade in the database. Each one of these folders, after meshing and simulation procedures took place for the relative cascade, will contain a series files and folders generated by ADS CFD, plus the six initial files generated by STEP 7 in the tool's workflow (see 3.4.3). Therefore,

`Blade_X` plus all the files contained in the `/PP_files` folder (described in section 3.4.3) have to be considered as the totality of the tool's outcomes.
For what concerns the scripts defining the tool, they have been intentionally omitted in this report, since the most of them were very long and their precise explanation goes beyond the scopes of this work. Nevertheless, the reader can contact the Von Karman Institute for Fluid Dynamics - Turbomachinery and Propulsion Department - to personally control the source codes of the entire BPT.

3.5.1. BPT Validation and Time Performance

At the conclusion of this Chapter, the BPT validation procedure is reported. It consisted in launching the tool to process a mini-batch of four blades. The run was successful, hence showing the tool's capability to automatically process several blades. At first, the design input parameters of the four blades have been defined in the `.xlsx` file. Than, when the tool was launched, all the workflow STEPS described before in the Chapter have been executed sequentially resulting in the successful simulations of the four cases.

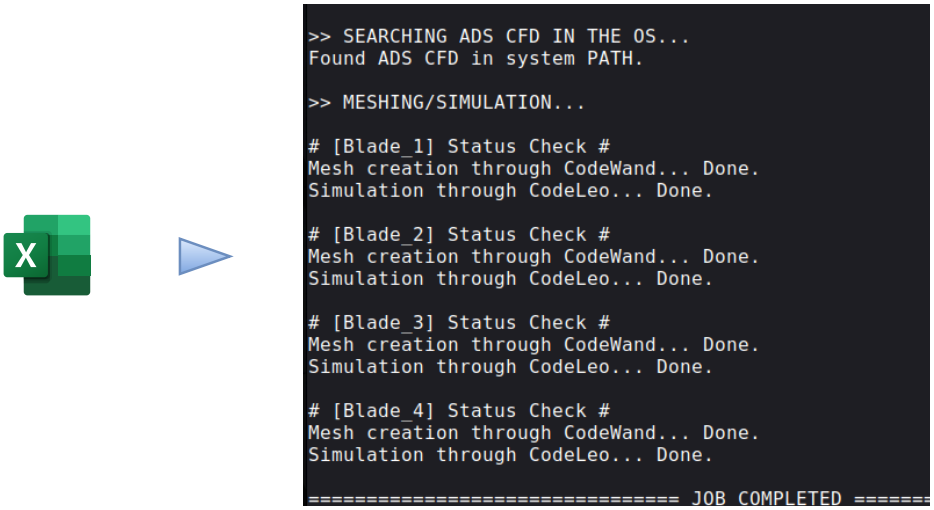


Figure 3.31: Representation of the BPT functionalities in normal mode (no post-processing is active), showing its automatic multiple simulations capabilities.

Thanks to the success of this test, and the final confirmation given by the simulation of the entire database, it is possible to state that the BPT correctly addresses its duties. On the strength of this, an extensive database of cascades was processed. Database creation and post-processing details are extensively described in Chapter 4.
Another important information, relevant more for tool's performances assessment pur-

poses, is its required time to execute each STEP. To investigate on such scope, an additional "functioning mode" has been implemented. Specifically, if the tool is launched with one of the flags `-t` or `-time`, it will run in **timer mode**. The two commands to activate such a modality are:

```
python3 main.py -time
python3 main.py -t
```

In the following is shown the table with all the most relevant STEPS execution times, associated to the creation of the database referenced in this report.

STEPS	Timing [s] (x1 Cascade)	Timing [s] (Full Database)
0	0.3587	309.92
1	0.0190	16.42
2	0.0091	7.86
3	0.4472	386.38
4	0.0224	19.35
5	0.0036	3.11
6	0.0026	2.25
7	0.0043	3.72
8	306.86	265122.63
9	11.11	9601.63
Total	318.83	275473.27

Table 3.2: BPT STEPS timings for a single cascade and for the full database (864 cascades).

Performing the calculation, it results that the total time in seconds is 275473.27, which corresponds to approximately 3 days, 4 hours and 30 minutes.
STEPS timing could differ depending on where the BPT is installed and run. For example, the author experienced a substantial difference in its execution time if launched in local (so that directly on the machine where the tool is installed) or in remote (so that on a machine that is physically different from that where the tool is installed), noticing that the run in local is substantially faster that that on remote.

4 | Database

4.1. CPU vs GPU

To fill in the database, a substantial amount of simulations have been performed. CFD software rely on solvers to compute the flow governing equations results. The old and well known solver technology uses CPU processors, while the new one makes use of GPUs. Nowadays, the trend in CFD research indicates a clear shift towards GPUs replacing old CPUs, paving the way for ultra fast simulations. Below, a quick comparison between two simulation runs for the same case by using CPU and GPU, using the software ADS CFD.

```
IT=  4000 CPU=    4300.01(SEC)  OVERALL TIME=    4298.74(SEC)  CPUeff=     100.03%  ITS/HR=      3350.
  4000     0.02748     0.00761     0.00948     0.04840     2.07178    1587085

  END OF JOB WITH CODE Leo ON 2026 02/22 18:56:07 ELAPSED TIME=    1HRS 11M 46S  452MS
```

```
IT=  4000 CPU=     290.48(SEC)  OVERALL TIME=     290.53(SEC)  CPUeff=      99.98%  ITS/HR=     49565.
  4000     0.02748     0.00761     0.00948     0.04840     2.07180    1587085

  END OF JOB WITH CODE Leo ON 2026 02/22 17:40:35 ELAPSED TIME=    0HRS  4M 56S  892MS
```

Figure 4.1: Above the `.OUT` file for the CPU run, and below the `.OUT` file for the GPU run.

Looking at the comparison shown in fig.4.1, it appears quite clearly that a simulation run using GPU ends much faster than using CPU. For instance, the simulation run with CPU lasts 1 hour and 11 minutes, while the one held with GPU lasts around 5 minutes. The acceleration factor depends by different aspects, but the most important one is the type of GPU used. To perform the simulations interested for the purposes of this project, the author used his personal computer GPU (NVIDIA GEFORCE RTX 4050), allowing to achieve an acceleration factor of around 14, where the acceleration factor was computed as:

$$acc.\ factor = \frac{Simulation\ Time_{CPU}\ [s]}{Simulation\ Time_{GPU}\ [s]} \tag{4.1}$$

Important to highlight that, in the frame of this project, the database creation would have not been possible without using the GPU technology, as gathering results for so many cases would not have been feasible in so small amount of time.

4.2. Database Mesh and Solver

To guarantee consistency across the entire database, standardized mesh (from mesh sensitivity analysis in Chapter 3) and solver templates have been adopted. This section details these templates, giving the reader all the needed information about mesh and solver specifics, but also to allow for the reproduction of the same setup.

4.2.1. Mesh Topology

As already reported in Chapter 3, the mesh topology is controlled and defined by the `.WAND` and `.MESHIX` files. The template mesh has been chosen by looking at several aspects, such as the results of the grid sensitivity analysis (details in Chapter 3), but also considering the general knowledge cases in literature. Mostly, the idea has been to try reaching a good trade-off between result's quality and computational time. It should be clear to the reader that due to the high number, and very different blade geometries contained in the database, a simple mesh sensitivity analysis performed on a single blade is not enough to satisfy the highest quality results requirements. However, the mesh hereafter described can be considered a "good player", giving good quality results at a reasonable computational cost.
In fig.4.3 is clearly visible the high refinement along all the three dimensions of the space. To be noted that it was not possible to mesh half the blade span, due to the ADS software's limitation. The mesh appears to be highly refined close to the walls (hub, shroud and blade surface), enabling the capture of the fluid near-wall behavior.

```
NBLOCK    ADVANCED
5         -20
BCK#      IM          JM        KM
1         212         25        150
2         212         25        150
3         175         13        150
4         68          11        150
5         68          11        150
LE-RFA,    LE-MOV,    LE-ANG,    TED-RFA,   TE-MOV,    TE-ANG
1.00000    0.00000    1.00000    1.00000    0.00000    1.00000
```

```
*ITYPE     OUTYPE     IGRID      IPOSS      ICLS       IWAL       IFAN       ILEO
-3         8          1          1          2          10         3          0
*MBLD      KSPAN      FUTURE
-5         2          1
*IUP       IDW        JPITCH
13         13         41
*NFILES    IPLOT
1          204
*FILENAME
Blade_31.AGF
*IUP       IDW        JPITCH
-13        13         41
GRID-CONTROL ON
DWAL-OMESH 0.00003
OMESH-MULT 1.00000
```

Figure 4.2: Above the `.MESHIX` file, and below the `.WAND` file for database creation.

To understand the meaning of each parameter, the reader should make reference to the dedicated section in Chapter 3, where they have been introduced.

4.2.2. Solver Configuration

```
*RESTART-FILE-NAME
Blade_31.REST
*NITER     MPID       PRUN       IPRT       NIRST      IPLOT
4000       101        0          20         1000       4
*ITURB     ITIME      EMODEL     IFAST      IGAMMA     IRUN
1          0          0          1          0          21
```

Figure 4.4: `.LEO` file used to setup the ADS solver to simulate the entire database.

Also in the case of the `.LEO` file, the reader should make reference to the dedicated section in Chapter 3.

4.2.3. Validation with an Experimental Case (SPLEEN)

Once the mesh was built and the related simulation results available, their reliability cannot be stated apriori. Therefore results need to be validated, and a common practice in engineering, when dealing with computational models, is to use experimental results as reference. That was the strategy reported also in this report to confirm CFD predictions. The experimental case taken into consideration is the one of SPLEEN, a linear turbine cascade extensively tested at the von Karman Institute to accurately investigate aerody-

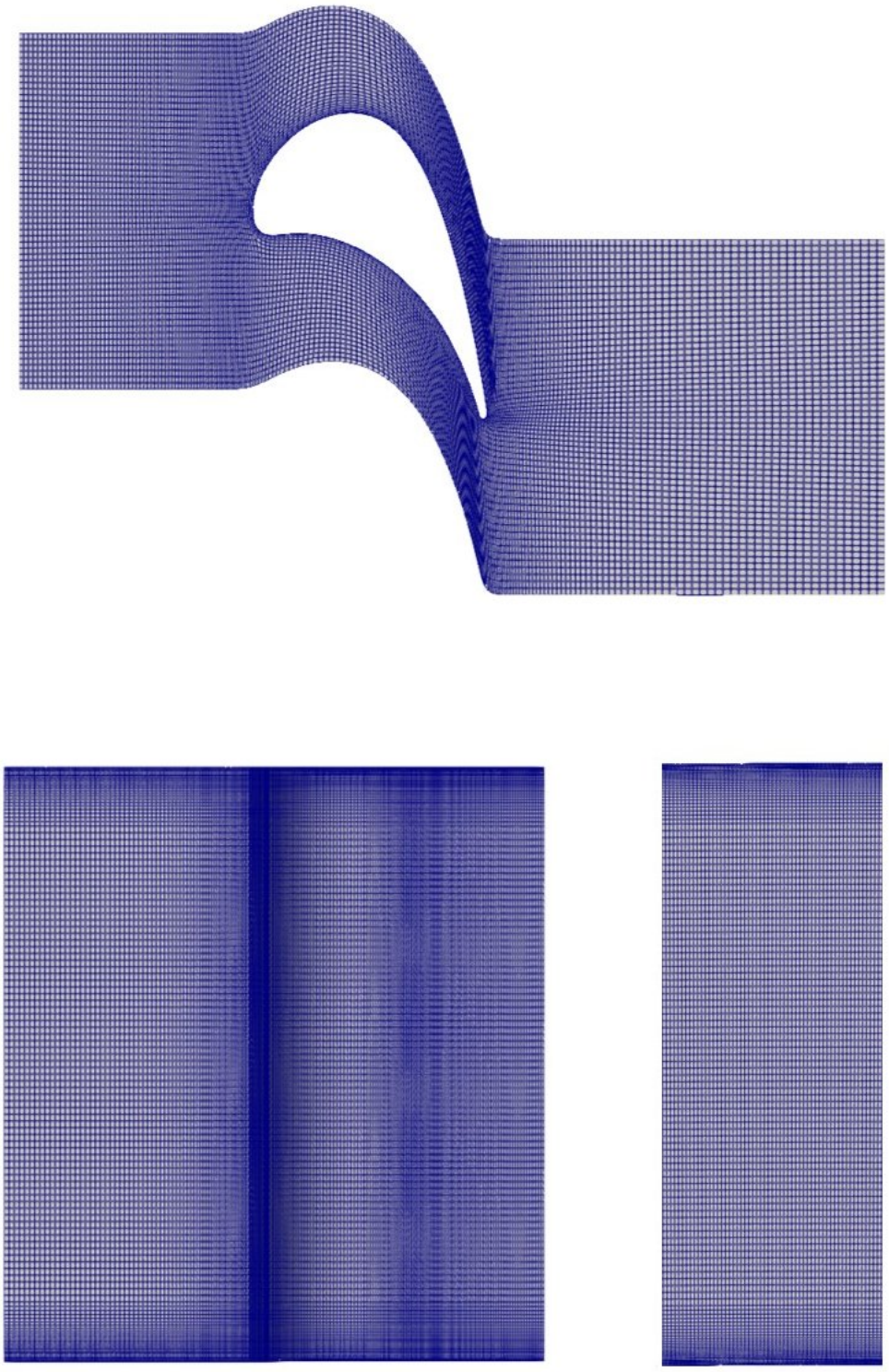


Figure 4.3: Above the blate-to-blade view, and below the two lateral views of the template mesh ($\approx 2M$ elements).

namic loads and secondary flow losses produced.
Hereafter is what has been done to make the comparison:

- **CFD Setup Preparation**: At first, the ADS case folder for a random blade (`Blade_308`) has been prepared. Then, the SPLEEN 2D blade profile and the proper boundary conditions, identified by the operative point: $M_2 = 0.8$, $Re_{2,is} = 1200000$, has been substituted inside the `.AGF` file of the selected case, while all the other files have been kept the same.
- **Measurement Data Extraction**: from specific `.CSV` files, measurements data for the previously identified operative point have been extracted and processed through a python script. The main quantities extracted from the `.CSV` files are: P_{06}/P_{01}, ζ_{tot} and the blade loading distribution, where P_{06} is the fluid total pressure evaluated at half C_{ax} downstream the TE, P_{01} is the fluid total pressure at the inlet plane, and ζ_{tot} represent the total kinetic energy losses.
- **Comparison**: CFD data have been extracted using the BPT post-processing functionality, allowing for their comparison with the experimental data through the same python script mentioned in the previous point.

SPLEEN 2D geometry and experimental measurements have been taken from the dedicated online database, available here [16]. Further details about validation can be found in Appendix B. Comparison results are shown in fig.4.5.
Let us break down the most important features visible in the graphs shown in fig.4.5. To start, it can be noticed that on the maps axis there are two dimensionless quantities: the blade radial coordinate normalized with respect to the span (vertical axis), and the tangential (pitch-wise) coordinate normalized with respect to the pitch (horizontal axis). The piece of blade span considered goes from 30 to 50%, because on one side the experimental results are available only for half blade's span, and on the other side it has been chosen to avoid the capture of secondary flows influence on the maps, since the SPLEEN inlet boundary layer was not known, hence the two resulting maps would be different close to the hub.
This validation has the main goal to prove that CFD simulations are correctly predicting the physics of the processed blades in the database; for this reason, this validation strategy has been considered. In the two rows of maps, the ones in the middle are those obtained with the mesh and simulation setup used for simulating the database. As clearly appears, on the right-hand side of the middle maps there is a region that slightly differs from experimental results, and for this reason a third case was simulated. This latter case simply consists in a more refined mesh ($\approx 6M$ of elements) with respect to the db setup ($\approx 2M$ of

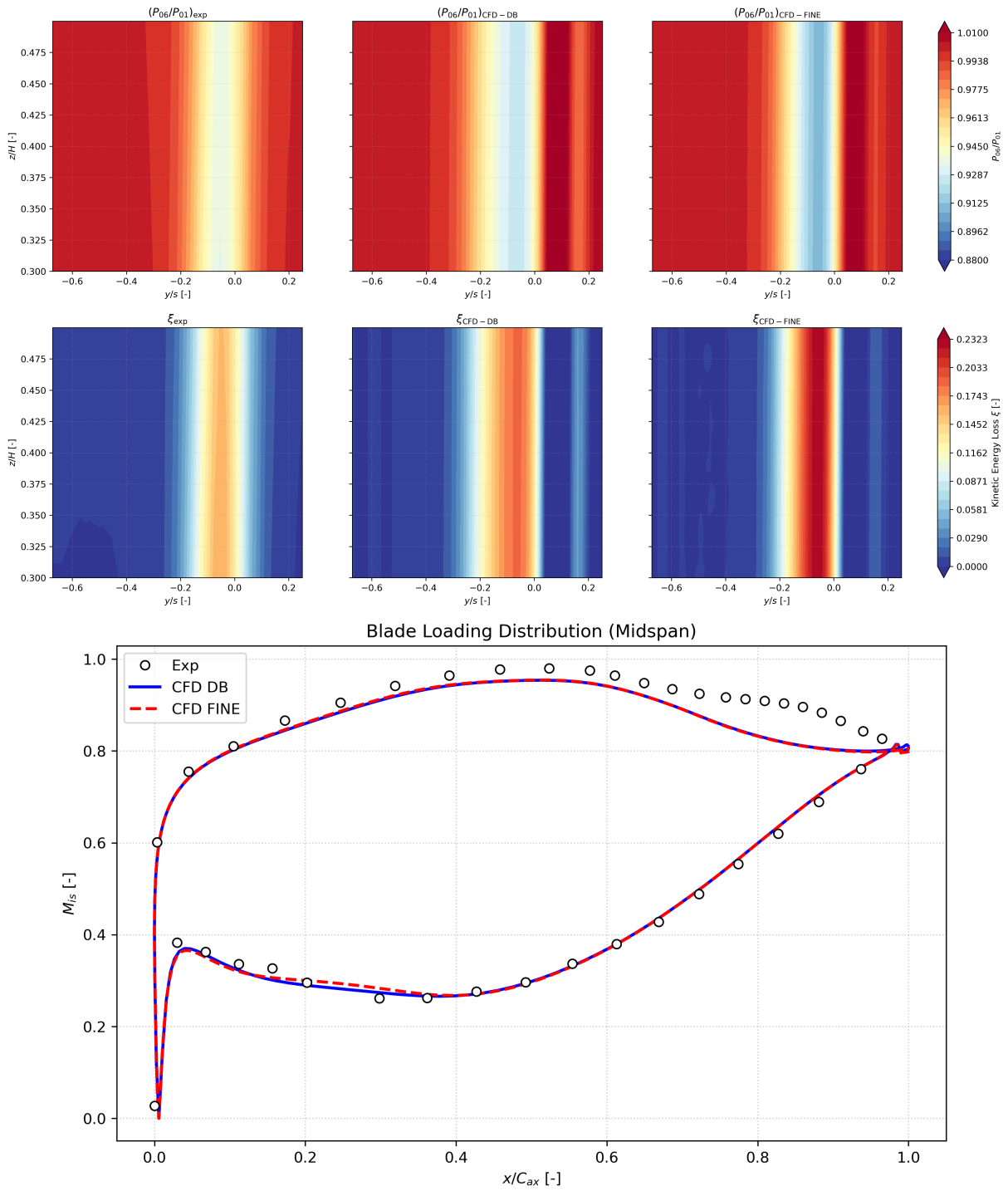


Figure 4.5: Above, the maps of P_{06}/P_{01} (154 experimental points for the map on the left). In the middle, the maps of ζ_{tot} (154 experimental points for the map on the left). Both kinds of map show the quantity trend over an outlet axial plane placed at half C_{ax} downstream TE. Below, the isentropic mach distribution along blade profile, being an index of the aerodynamic loading (41 points for the experimental profile).

elements). Even with higher refinement, the non-uniformity on the right-hand side of the map remains, and the distance from experimental results appears wider in the center of the wake. This aspect touches one of the biggest limitations encountered along this work, that will be further discussed in the conclusions of this report, in Chapter 6. Last but not least, the isentropic mach number distribution along mid-span blade surface is shown on the bottom of fig.4.5. Blade loading is well captured from both CFD simulations along the entire surface, except for a localized region on suction side close to the trailing edge. This discrepancy is due to the presence of a laminar bubble that cannot be detected with the turbulence model used for the database creation ($k-\omega$ Wilcox 98), which is a fully turbulent model. A transition model should be used to fill in that discrepancy.

4.3. Database Structure

The database structure gets the shape from the DoEs theory, reported in the literature survey of this report. For project reasons, a full-factorial design strategy has been chosen to create it. Its implementation was held through the creation of a `.xlsx` file, using a setup file for storing the database specific information, and a python script that translates the setup file information into the actual database excel file.

4.3.1. DB_setup.xlsx & DB.xlsx

Blades to be Processed		Aerodynamic Style and Duty					Blade Geometry		Outlet Reynolds	Boundary Layer					Processing	
# blades to Generate	range	alpha1 [deg]	alpha2 [deg]	M2 [/]	lpeak_ratio [/]	mpeak_ratio [/]	h/Cax [/]	Cax [m]	Re2_is [/]	BL Type	δ [/]	δ*/δ [/]	θ/δ [/]	Tu [/]	Mesh	Simulation
864	min	-45	65.00	0.40	0.50	1.20	4.00	0.05	1,200,000	1	0.20	0.379	0.145	5	0	0
# blades to Process	max	-20	75.00	0.65	0.60	1.40	4.00	0.05	1,200,000	2	0.05	0.143	0.111	5	0	0
864	points	4	4	3	3	3	1	1	1	2	2	2	2	1	0	0

Figure 4.6: Database setup file (`DB_setup.xlsx`).

In fig.4.6 is shown an excel file used as main setup to create the final database. It has four rows. Starting from the "range" column: the first row contains name and units of the variables, the second and third rows, respectively, contain minimum and maximum bounds in which the specific variable vary, and the fourth row contains the number of divisions in which the variable's interval (min-max) is split. The very first column on the left displays the total number of combinations of all the variables (highlighted in the blue boxes), so that the total number of points in the database. It is a function of the variables splits, and follows the rationale of the full-factorial DoEs.
All the unframed quantities are constant. On the very end of the file, to the right, are visible the two columns "Mesh" and "Simulation". They accept two options: 0 if the User

doesn't want to perform that action, 1 if the User wants. Obviously, if the User wants to simulate, the mesh has to be created beforehand (1/1 is accepted, 1/0 is accepted, 0/0 is accepted, but 0/1 is not accepted).

Blades to be Processed		Aerodynamic Style and Duty					Blade Geometry		Outlet Reynolds	Boundary Layer					Processing	
# blades	blade #	alpha1 [deg]	alpha2 [deg]	M2 [/]	lpeak_ratio [/]	mpeak_ratio [/]	h/Cax [/]	Cax [m]	Re2_is [/]	BL Type	δ [/]	δ*/δ [/]	θ/δ [/]	Tu [%]	Mesh	Simulation
864	1	-45	65	0.4	0.5	1.2	4	0.05	1200000	1	0.2	0.379	0.145	5	0	0
	2	-45	65	0.4	0.5	1.2	4	0.05	1200000	2	0.05	0.143	0.111	5	0	0
	3	-45	65	0.4	0.5	1.4	4	0.05	1200000	1	0.2	0.379	0.145	5	0	0
	4	-45	65	0.4	0.5	1.4	4	0.05	1200000	2	0.05	0.143	0.111	5	0	0
	5	-45	65	0.4	0.6	1.2	4	0.05	1200000	1	0.2	0.379	0.145	5	0	0
	6	-45	65	0.4	0.6	1.2	4	0.05	1200000	2	0.05	0.143	0.111	5	0	0
	7	-45	65	0.4	0.6	1.4	4	0.05	1200000	1	0.2	0.379	0.145	5	0	0
	8	-45	65	0.4	0.6	1.4	4	0.05	1200000	2	0.05	0.143	0.111	5	0	0
	9	-45	65	0.65	0.5	1.2	4	0.05	1200000	1	0.2	0.379	0.145	5	0	0
	10	-45	65	0.65	0.5	1.2	4	0.05	1200000	2	0.05	0.143	0.111	5	0	0
	11	-45	65	0.65	0.5	1.4	4	0.05	1200000	1	0.2	0.379	0.145	5	0	0
	12	-45	65	0.65	0.5	1.4	4	0.05	1200000	2	0.05	0.143	0.111	5	0	0
	13	-45	65	0.65	0.6	1.2	4	0.05	1200000	1	0.2	0.379	0.145	5	0	0
	14	-45	65	0.65	0.6	1.2	4	0.05	1200000	2	0.05	0.143	0.111	5	0	0
	15	-45	65	0.65	0.6	1.4	4	0.05	1200000	1	0.2	0.379	0.145	5	0	0
	16	-45	65	0.65	0.6	1.4	4	0.05	1200000	2	0.05	0.143	0.111	5	0	0
	17	-45	75	0.4	0.5	1.2	4	0.05	1200000	1	0.2	0.379	0.145	5	0	0
	18	-45	75	0.4	0.5	1.2	4	0.05	1200000	2	0.05	0.143	0.111	5	0	0
	19	-45	75	0.4	0.5	1.4	4	0.05	1200000	1	0.2	0.379	0.145	5	0	0
	20	-45	75	0.4	0.5	1.4	4	0.05	1200000	2	0.05	0.143	0.111	5	0	0
	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 4.7: Database complete file (`DB.xlsx`).

Running the python script `datagen.py`, contained in the folder `/main/Input_gen` of the BPT (see Chapter 3), that reads the `DB_setup.xlsx` file, the final `DB.xlsx` is created. Therefore, the final form of the database is the one displayed in fig.4.7. This is the file that, once copied and pasted in the `/main` folder, is used by the BPT to process the database cascades.

4.3.2. Corner and Inner points

With the simulations outputted by the tool it was possible to build an extensive database, which materially is a collection of a relevant number of points (cases). At each case in the database is associated an identification number. The entire db accounts for **864 points**, among which 64 are its **corner points**, while the rest are **internal points**. The corners are defined making all the possible combinations between the extremity values of each input design parameter range. Making reference to fig.3.12, the input design parameters that varies while defining db points are six: α_1, α_2, M_2, l_p, M_p, *bl type*, all between two values, a maximum and a minimum. This practically means that:

$$\#Corner\ Points = 2^6 = 64 \tag{4.2}$$

The internal number of points has been set to 800 for the following reasons: due to the kind of database structure chosen (grid) and considering the high number parameters

to be varied, augmenting the number of divisions for a single range with respect to the current situation would mean to increase the number of cases up to a value higher than 1000, going beyond the scope of the project. 800 was considered to be a reasonable amount. Another reason is time dependent. Precisely, for a single simulation are required 5 minutes, which means that for 120 simulations are required 10 hours. Preparing six batches of 120 simulations and a last batch of 80 simulations, and running them during night, it was possible to fill in the inner points of the database in 7 days.

CP	B1	B2	B3	B4	B5	B6	B7
64	120	120	120	120	120	120	80

Table 4.1: Batches for database simulations.

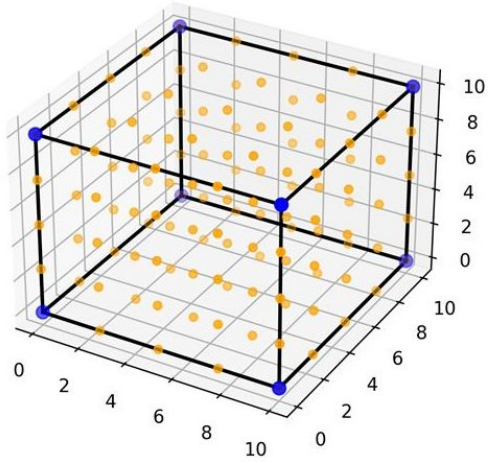


Figure 4.8: Example of the graphical representation of a full-factorial DoEs in three dimensions, hence where only three input parameters are variable. The blue dots are the corner points, while the yellowish ones are the inner points.

4.4. Database Post-Processing Results

Once the database has been fully simulated and stored in an external storage, it was post-processed using the "stand-alone" mode of the BPT post-processing system.
For the sake of brevity, but also because of the repetitive nature of the files outputted by post-processing all the database blades, it has been chosen to show the results in this way: main plots and trends only for the **most loaded** and **least loaded** cases in the database, while loss and load maps have been shown for all the cascades in the database.

4.4.1. Main Plots and Trends

When talking about blade loading, the author refers to the pressure applied on the blade's surface by the fluid. Looking at the database corner points, where all the design space extremes are collected, the most loaded configuration is the Blade number 31, due to its highest deflection ($|\alpha_2 - \alpha_1|$), outlet Mach number (M_2), and dimensionless peak Mach number (M_{peak}), while the least loaded is the Blade number 34 for the opposite reasons.

Blades to be Processed		Aerodynamic Style and Duty					Blade Geometry		Outlet Reynolds	Boundary Layer					Processing	
# blades	blade #	alpha1 [deg]	alpha2 [deg]	M2 [/]	lpeak_ratio [/]	mpeak_ratio [/]	h/Cax [/]	Cax [m]	Re2_is [/]	BL Type	δ [/]	δ*/δ [/]	θ/δ [/]	Tu [%]	Mesh	Simulation
864	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
	31	-45	75	0.65	0.6	1.4	4	0.05	1200000	1	0.2	0.379	0.145	5	0	0
	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
	34	-20	65	0.4	0.5	1.2	4	0.05	1200000	2	0.05	0.143	0.111	5	0	0
	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 4.9: Input design parameters of the most and least loaded blades

In the following are compared and discussed the most relevant plots and trends for these two configurations.

```
--- BLADE 31 ---
Zweifel Coeff        : 0.7268
Circulation C0       : 0.4692
Diffusion Factor     : 0.4129
```

```
--- BLADE 34 ---
Zweifel Coeff        : 0.9149
Circulation C0       : 0.3425
Diffusion Factor     : 0.2224
```

Figure 4.10: Dimensionless loading coefficients for the most and least loaded blades

Fig.4.10 shows the loading coefficients computed for both the most and least loaded blades. The initial boundary conditions computed for these two blades were different, affecting the coefficients calculation.
Fig.4.11 shows the computed losses for the two cases. Let us make reference to the kinetic energy losses, namely the ζ coefficients. The most loaded case shows in general higher losses. The Coull model seems to predict quite good the endwall losses, in both cases. The AVDR, shown in the lower part of the logs, is an index representing how much the secondary vortices affect the mid-span section flow. Its value should be really close to one, indicating that the blades aspect ratio (h/C_{ax}) is high enough to consider the mid-span section in a quasi-2D flow condition. This aspect is relevant, because CFD predicted profile losses are computed using a very narrow slab centered to the mid-span section, and in this slab there must not be secondary flows influence.
Fig.4.12 displays the "absolute" velocity contour, since the cascades processed throughout this project are statoric. Paying attention to the color scale of both graphs can be noticed that the most loaded blade generates a higher velocity excursion through the passage.

```
--- BLADE 31 ---
[Zeta Breakdown]
Zeta Total                : 0.077627
Zeta Profile              : 0.062186
Zeta Endwall Total        : 0.015441
 |-- Mixing               : 0.008509
 |-- Endwall Core         : 0.006932
Zeta Endwall Coull        : 0.106218

[Y Breakdown]
Y Total                   : 0.092865
Y Profile                 : 0.073176
Y Endwall Total           : 0.019689
 |-- Mixing               : 0.009413
 |-- Endwall Core         : 0.010276

[CFD Measurements]
Pt_In                : 150670.599947
Pt_Out               : 147609.414647
Ps_Out               : 113471.816301
Alpha_In             : 44.956907
Alpha_Out            : -72.582748
X_us                 : -50.000000
X_ds                 : 134.999985
AVDR                 : 1.084345
```

```
--- BLADE 34 ---
[Zeta Breakdown]
Zeta Total                : 0.066775
Zeta Profile              : 0.055923
Zeta Endwall Total        : 0.010852
 |-- Mixing               : 0.005734
 |-- Endwall Core         : 0.005119
Zeta Endwall Coull        : 0.012596

[Y Breakdown]
Y Total                   : 0.074385
Y Profile                 : 0.061622
Y Endwall Total           : 0.012763
 |-- Mixing               : 0.005983
 |-- Endwall Core         : 0.006780

[CFD Measurements]
Pt_In                : 239904.106421
Pt_Out               : 238223.271209
Ps_Out               : 214930.789631
Alpha_In             : 19.991298
Alpha_Out            : -63.715435
X_us                 : -50.000000
X_ds                 : 134.999985
AVDR                 : 1.031602
```

Figure 4.11: Loss coefficients for the most and least loaded blades

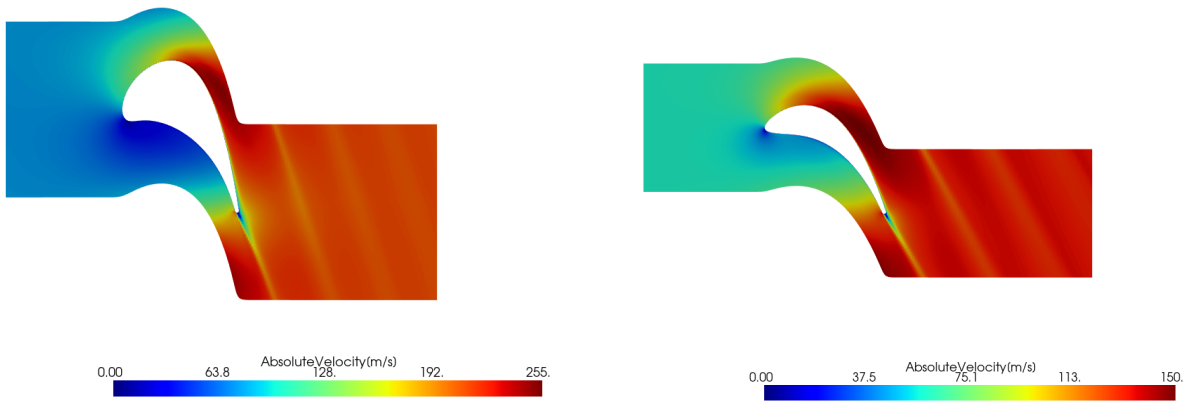


Figure 4.12: Absolute velocity contour at mid-span for `Blade_31` (left) and `Blade_34` (right).

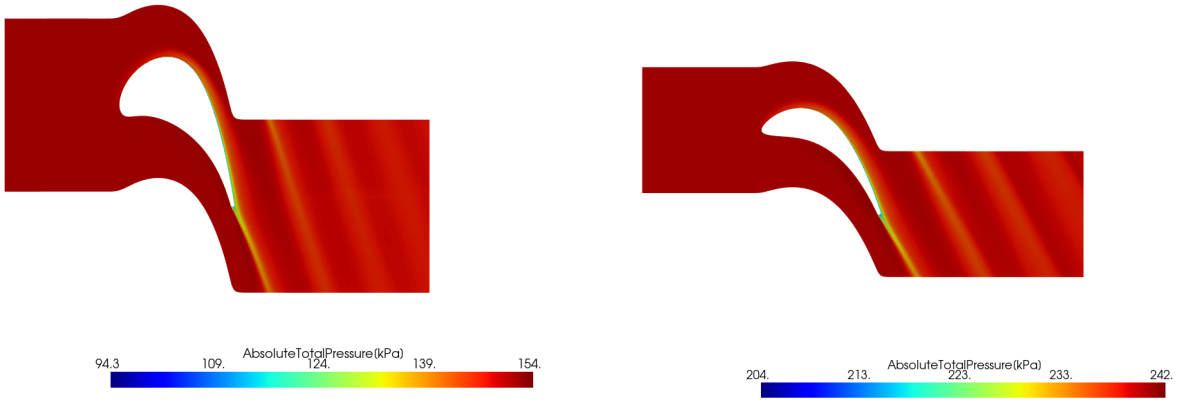


Figure 4.13: Absolute total pressure contour at mid-span for `Blade_31` (left) and `Blade_34` (right).

Fig.4.13, instead, shows the absolute total pressure contours for the two blades. These figures are particularly useful to promptly individuate where are the loss sources in the flow field. As expected, the big loss locations (lighter spots) are concentrated close to the blade's suction side and in the wake, generated downstream trailing edge.

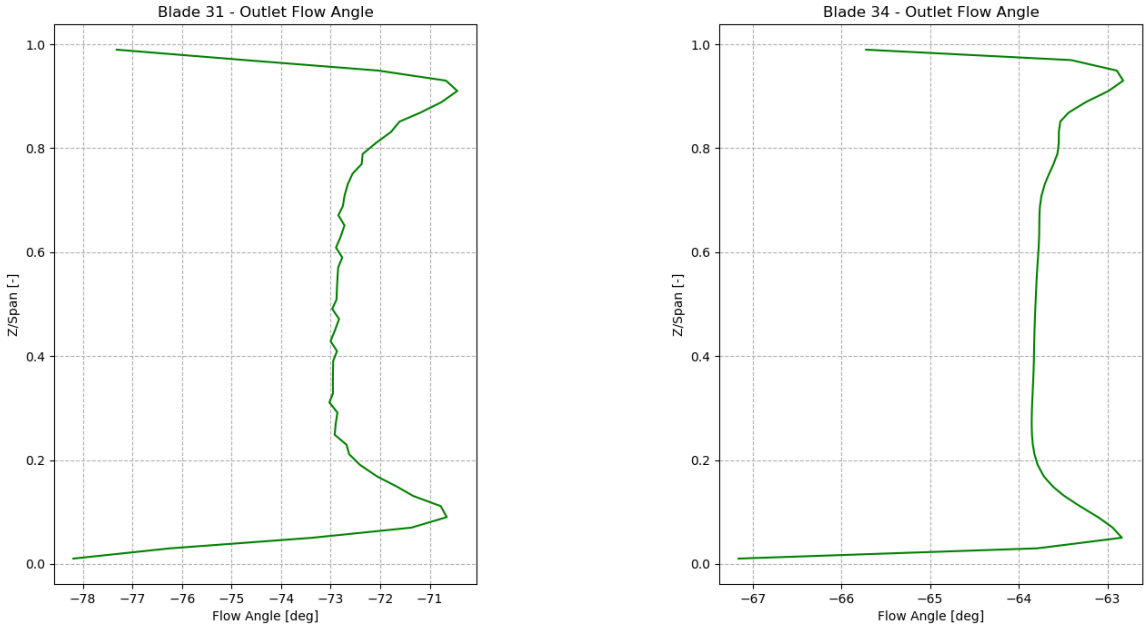


Figure 4.14: Radial pitch-wise averaged outlet flow angle distribution.

Fig.4.14 shows the radial trend of the pitch-wise averaged outlet flow angle for the two

cases. This is a clear indication of both presence and intensity of secondary flows in the linear cascade. At the two extremities (hub and shroud) an increase in the outlet flow angle is detected, indicating the concentration of secondary flows. Close to mid-span the trend is nearly constant to a fixed value of the angle, which is, however, bigger than that imposed in the `.xlsx` file. This aligns with theory, indicating the presence of flow deviation at the outlet (very high for some blade's deflections, due to the nature of the parametrization used to obtain the 2D profile shape - 2D Blade Generator -). Nevertheless, both plots are displaying a correct trend, in general.

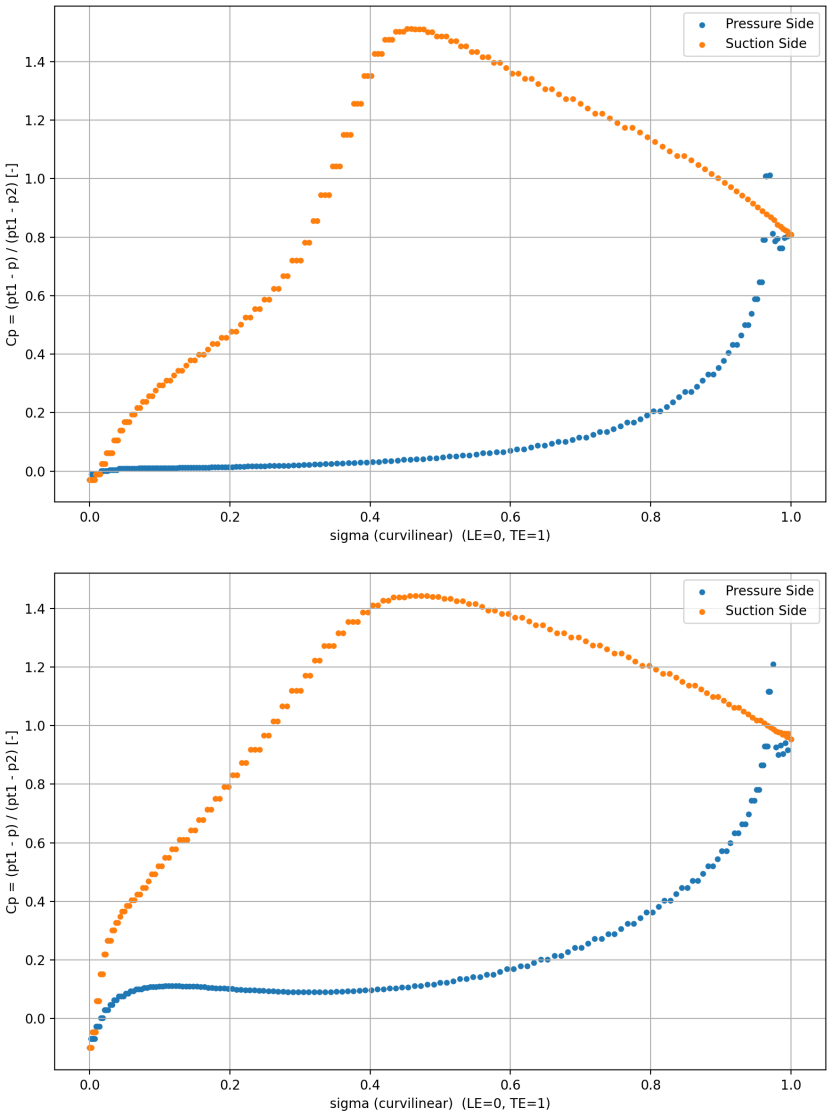


Figure 4.15: C_p distribution along blade surface: for `Blade_31` above, while for `Blade_34` below.

In fig.4.15, paying attention at the orange and blue curves in both graphs, it appears that

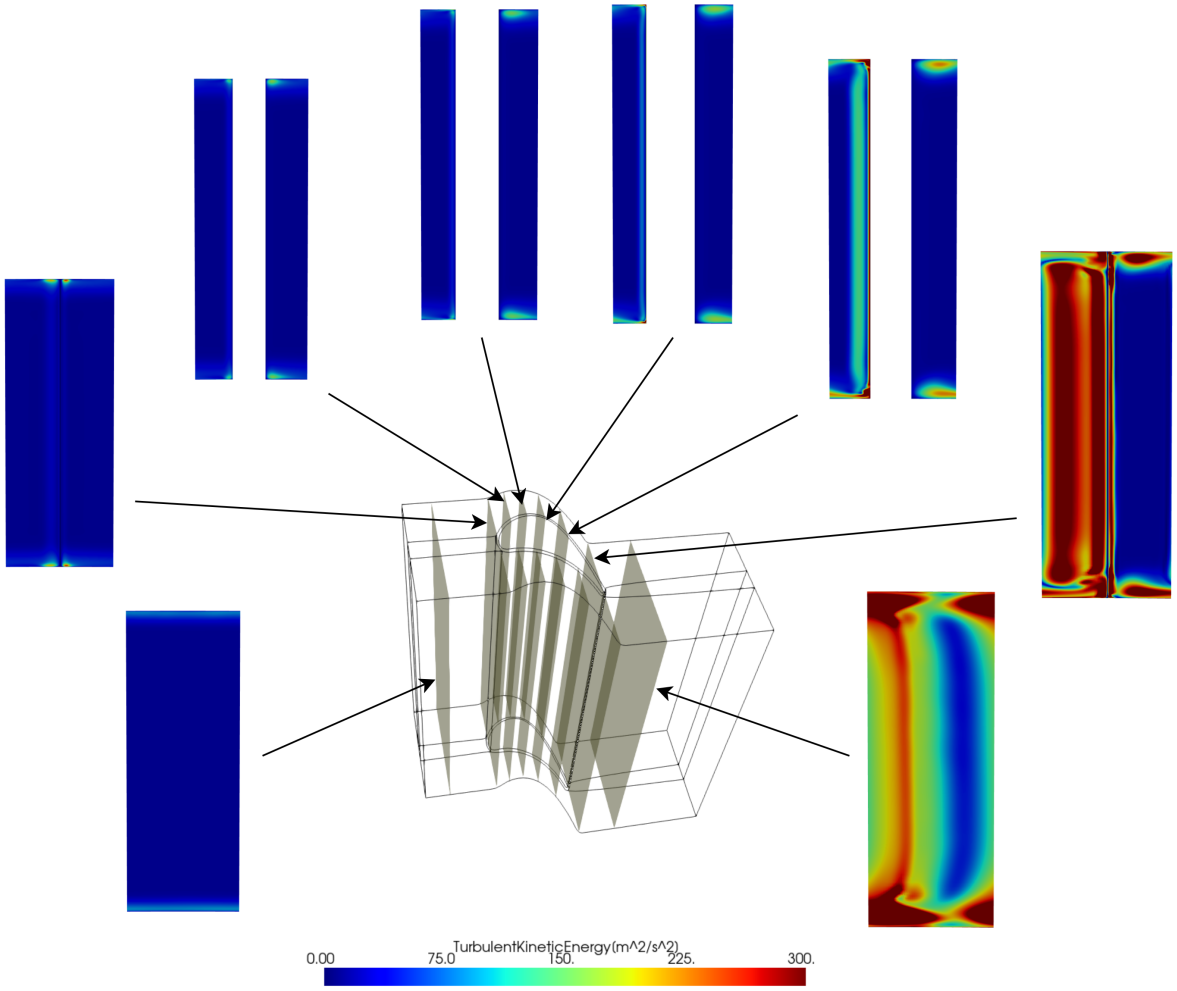


Figure 4.16: Turbulent Kinetic Energy (TKE) contour for several axial planes. In sequence: inlet, LE, 20%, 40%, 60%, 80% C_{ax} from LE, TE, and outlet planes for `Blade_31`.

the C_p close-loop area for `Blade_31` is slightly different than that of `Blade_34` on the suction side.

Figures 4.16 and 4.17 show the evolution of secondary flows trough a series of axial planes, from inlet to the outlet (see definitions of inlet and outlet planes in 3.4.3). The first thing that can be noticed is that for two different blades, the strength of secondary flows varies. It is expected, since the boundary and loading conditions are different. Specifically, `Blade_31` creates more dissipative and bigger vortical structures. The Turbulent Kinetic Energy (TKE) color scale was set such to have minimum and maximum values equal for all the cascades post-processed, in order to have comparable images from a cascade to another. Another feature that the reader might find not straightforward, is the difference in the axial planes thickness among the two blades, due to the fluid domain's automatic generation by `PyVista`, the python library used in the post-processing scripts that loads

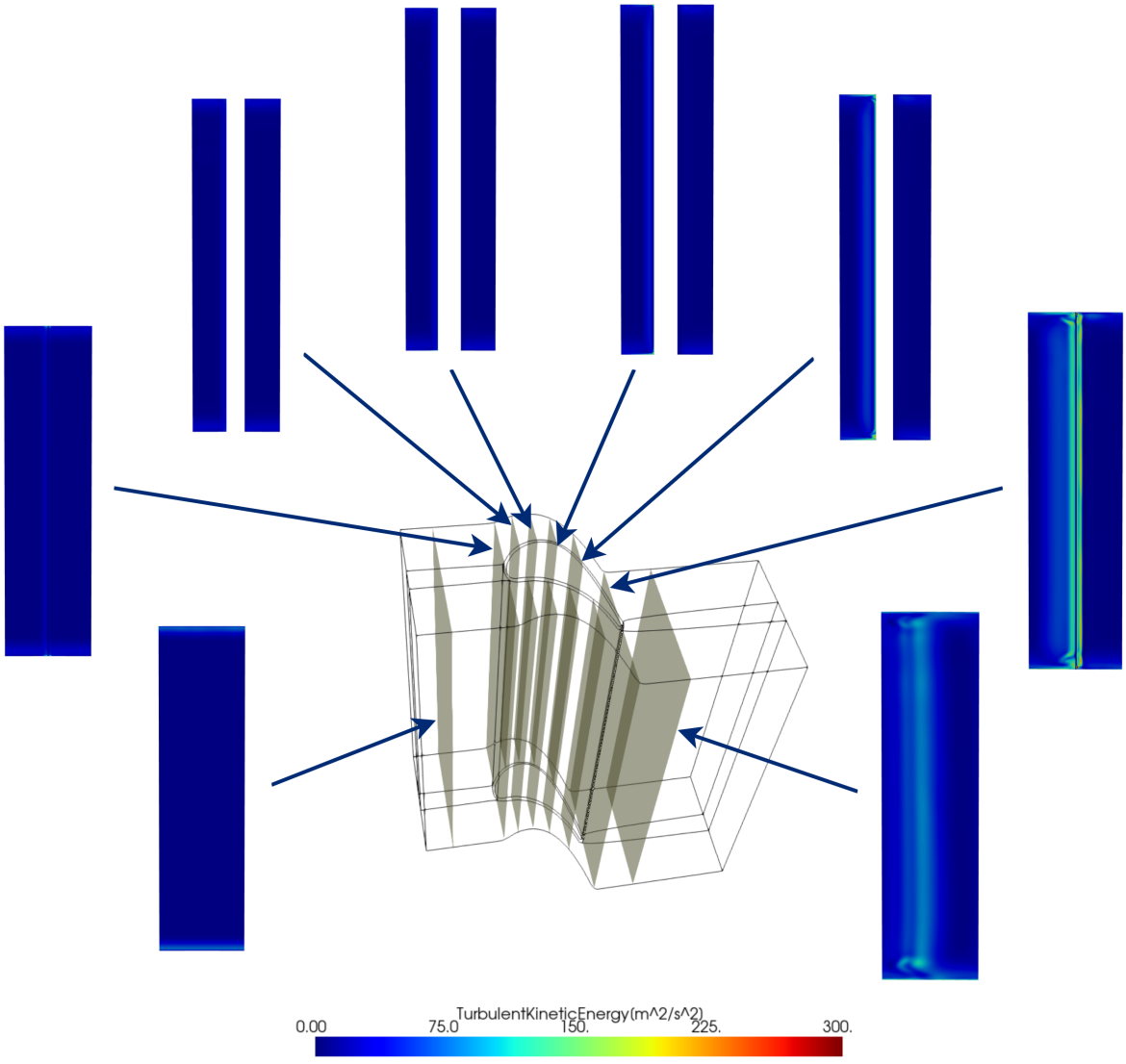


Figure 4.17: Turbulent Kinetic Energy (TKE) contour for several axial planes. In sequence: inlet, LE, 20%, 40%, 60%, 80% C_{ax} from LE, TE, and outlet planes for `Blade_34`.

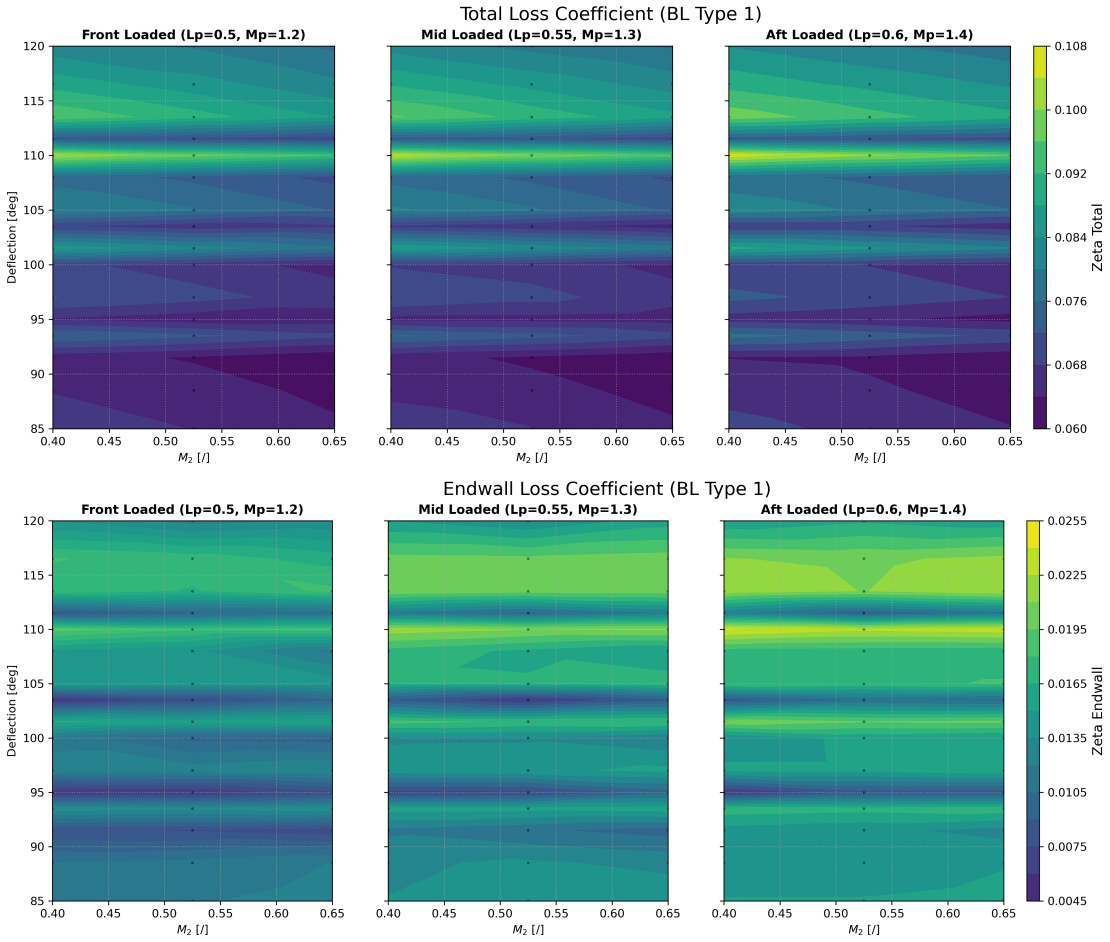


Figure 4.18: ζ_{tot} and $\zeta_{endwall}$ as function of deflection, outlet mach number, and a fixed inlet boundary layer (config. 1) for three different blade loading configurations: front-loaded, mid-loaded, after-loaded. Maps are evaluated on an axial plane at a distance of half C_{ax} downstream TE. M_p and L_p are dimensionless parameters that, respectively, refer to the maximum suction side Mach number and its position with respect LE.

the mesh in the PC RAM and extracts all the needed quantities for calculations.

4.4.2. Loss and Load Dimensionless Coefficients

Finally, loss and load dimensionless coefficients trends throughout the entire database are plotted in figures 4.18 and 4.19. For each monitored quantity, three maps have been reported: the one on the left for front-loaded blades, the one in the middle for mid-loaded blades, and the one on the right for after-loaded blades. The loading position is given by L_p, representing the axial distance between maximum section side Mach number and leading edge. From theory it is expected that front-loaded blades mitigate losses thanks to their geometry, producing lower loadings, while the opposite is for the after-loaded

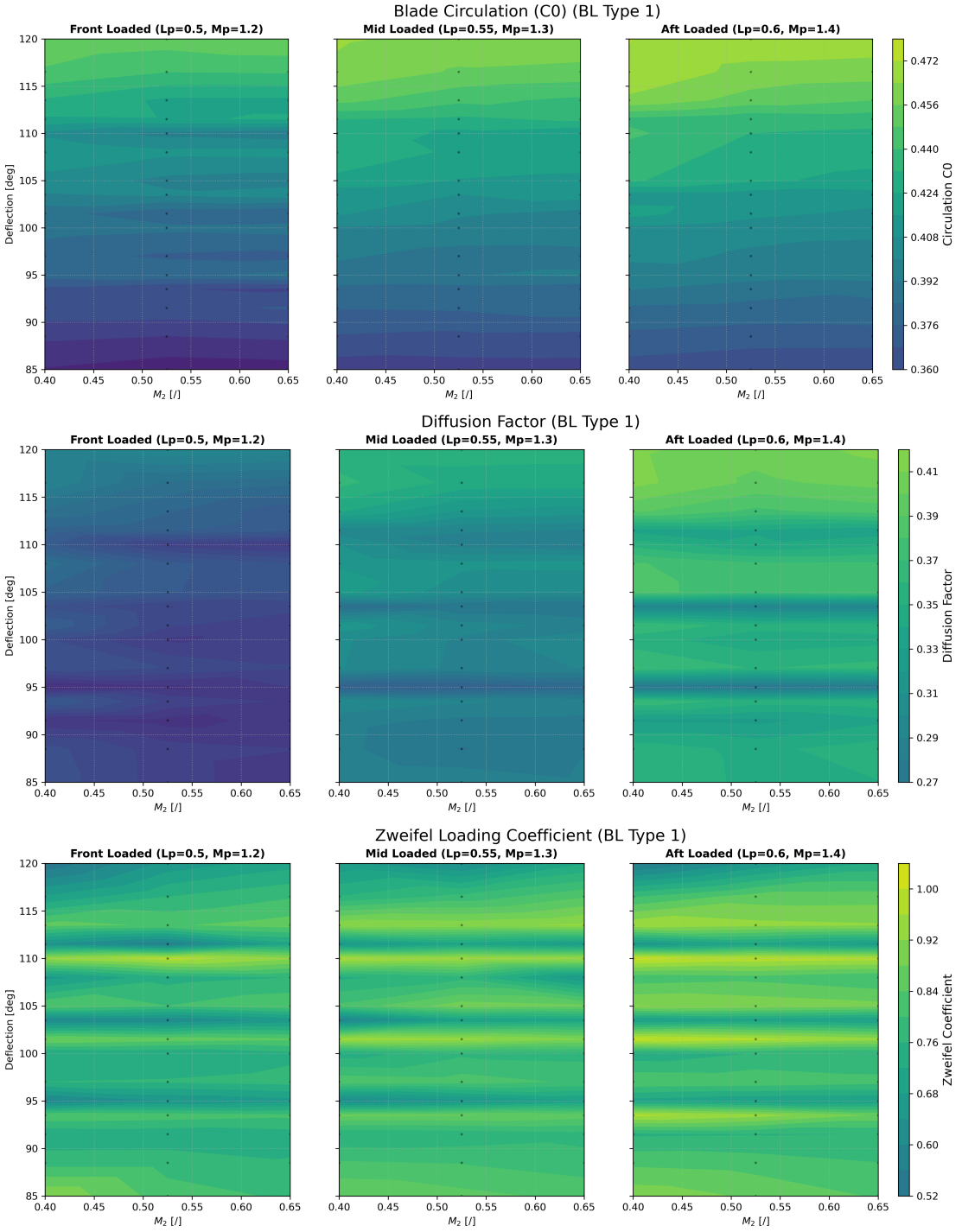


Figure 4.19: Circulation (C_0), diffusion factor (DF), and Zweifel coefficient (Z_w) as function of deflection, outlet mach number, and a fixed inlet boundary layer (config. 1) for three different blade loading configurations: front-loaded, mid-loaded, after-loaded. Maps are evaluated on an axial plane at a distance of half C_{ax} downstream TE. M_p and L_p are dimensionless parameters that, respectively, refer to the maximum suction side Mach number and its position with respect LE.

ones. This trend can be noticed for all the quantities, but with major intensity for the endwall losses ($\zeta_{endwall}$), circulation (C_0), and diffusion factor (DF). Another trend worth highlighting is the one given by blade's deflection, defined as $|\alpha_1 - \alpha_2|$. Also in this case, the global expected trend is captured for all the quantities: a general increase of blade loading, and hence losses, is expected when increasing flow turning (deflection). Nevertheless, loss maps show a discontinuous trend along the vertical direction. This could be attributed to different causes, among which the most important is the blade's parametrization strategy (2D Blade Generator), that due to its "nature" generates profiles geometry that match the imposed loading conditions (in the `.xlsx` file) with good approximation, neglecting the geometrical optimization required for obtaining reasonable loss values. In this sense, for specific deflection angles profile losses grow, and hence the endwall ones decrease (with fixed total losses), causing the above mentioned discontinuous trend. Other causes could be due to the use of a non-perfect mesh, inducing numerical dispersion in the CFD results. The poor database sampling (only 864 RANS simulations) is also not helping in properly shaping the maps trends. Finally, it can be highlighted also the influence of the outlet Mach number on the monitored parameters, which is minor with respect to that of deflection, but still present.

In the next chapter, the implementation of two surrogate models trained and tested on the extracted $Y_{endwall}$ values for the database cases will be discussed, closing the project's work.

5 | Surrogate Models

5.1. Why a Surrogate Model

Although Computational Fluid Dynamics (CFD) is an essential tool for the detailed analysis of turbomachinery, its computational cost remains a significant bottleneck when exploring large design spaces. With the aid of the BPT and GPU technology it was possible to run 864 RANS simulations in a reasonable amount of time. However, 864 points represent only a discrete and limited sampling of the infinite possible configurations. If we wanted to use this database for an automatic optimization process or to instantly evaluate new geometries, running a new CFD simulation for each iteration would require unacceptable time expenses. This is where surrogate models come into play.
The goal of a surrogate model is to "learn" from the existing CFD database correlations among some interested parameters. In this way, it is possible to build a fast, reliable mathematical function that links the input parameters to the quantities of interest in the output (in this specific case, the endwall losses). In other words, the surrogate model acts as a smart bridge: once trained, it can predict the aerodynamic performance of a configuration that has never been simulated before, returning the result in fractions of a second rather than hours of computation. In this chapter, the application and effectiveness of two specific Machine Learning techniques for this purpose will be discussed: Gaussian Process Regression (GPR) and Radial Basis Function Networks (RBFN).

5.2. Gaussian Process Regression

The Gaussian Process Regression (GPR) model was implemented using the `scikit-learn` library, providing both non-linear predictions and uncertainty quantification based on a Bayesian approach. Unlike deterministic regression, the model defines a distribution over possible functions thanks to a composite kernel, designed to reflect the physics of the dataset:

$$k(x, x') = C \cdot \mathrm{RBF}(x, x') + \mathrm{WhiteKernel}(x, x') \tag{5.1}$$

The *RBF* component enforces the principle that similar geometric configurations yield similar losses, while the *WhiteKernel* explicitly models the uncertainty introduced by the numerical discretization and mesh topology limitations observed in the RANS results. The 6-dimensional input space is heterogeneous, meaning that the magnitude of different variables varies significantly (e.g: angles span close to 100 deg, while Mach numbers span between 0 and 1). Hence, a `MinMaxScaler` was applied to normalize all features to the [0,1] range. After that, the database was split into training (70%) and test (30%) sets. During training, hyperparameters were tuned using 20 optimizer restarts (`n_restarts_optimizer=20`) to avoid local minima and ensure robust convergence. The overall training time is a few seconds.

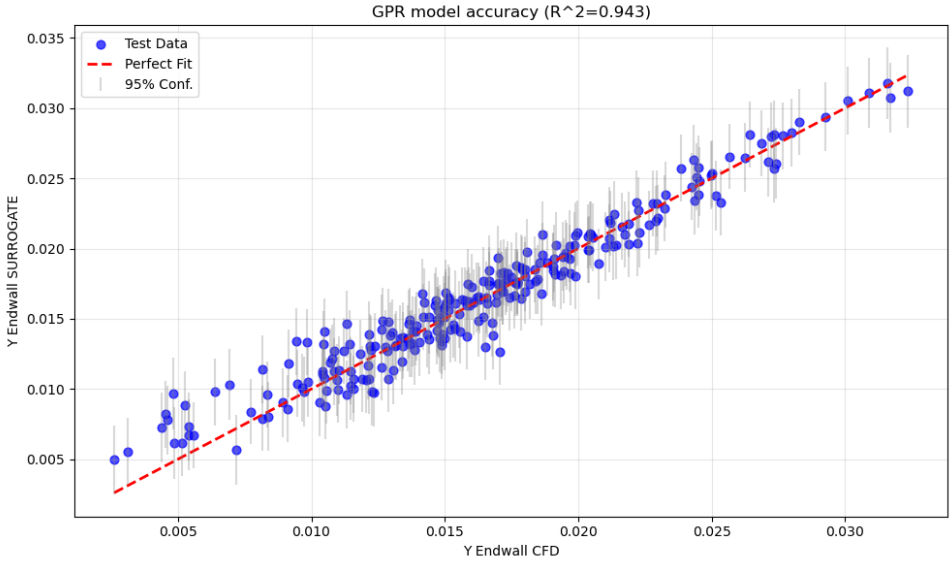


Figure 5.1: Accuracy estimation of the GPR model.

Fig.5.1 shows an evaluation of the model's accuracy. Each blue point is localized by the value of endwall loss computed by CFD, taken from the test set, and by the value of endwall loss predicted by the GPR model. The fact that points are close to the red dashed-line refers to a good accuracy for the model. Notice that each point is coupled with a gray bar, showing the 95% confidence interval for that specific prediction of the surrogate model.

$$R^2_{GPR} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} = 0.9432 \tag{5.2}$$

$$MSE_{GPR} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = 0.000002 \tag{5.3}$$

Two metrics have been computed: R^2 that represents how good the model prediction is with respect to a simple average, computed on the available data in the dataset, and MSE, namely the mean squared error between model prediction and "real", CFD acquired data.

5.3. Radial Basis Function Network

As an alternative to the Gaussian Process Regressor, a Radial Basis Function Network (RBFN) was implemented. RBFNs are a specific class of feed-forward artificial neural networks that are particularly effective for multidimensional interpolation and function approximation problems. Unlike deep neural networks, which may require complex iterative training procedures, RBFNs feature a simpler three-layer architecture and often allow for much faster training times.

5.3.1. Architecture and Mathematical Formulation

The architecture consists of three functional layers: an input layer, a single non-linear hidden layer, and a linear output layer.

- **Input Layer:** It distributes the input vector $\mathbf{x} \in \mathbb{R}^D$ (where $D = 6$, and represents the design parameters) to the hidden layer without applying any weight or transformation.
- **Hidden Layer:** This is the core of the network. It consists of M neurons (defined as `n_components` in the model), each characterized by a radial basis function ϕ. These functions act as local detectors: they produce a significant output only when the input $\mathbf{x}$ falls within a local region of the input space.
- **Output Layer:** It computes the final prediction as a weighted linear combination of the hidden layer's outputs.

In this work, the Gaussian function was selected as the radial basis function. For a generic neuron j, the activation is given by:

$$\phi_j(\mathbf{x}) = \exp\left(-\gamma||\mathbf{x} - \mathbf{c}_j||^2\right) \tag{5.4}$$

where γ is a hyperparameter controlling the width (or spread) of the Gaussian "bell". A

low γ results in a broad response (smooth model), while a high γ leads to highly localized responses (risk of overfitting)

The final approximation of the endwall losses, $y(\mathbf{x})$, is obtained by the linear combination:

$$y(\mathbf{x}) = \sum_{j=1}^{M} w_j \phi_j(\mathbf{x}) + b \tag{5.5}$$

where w_j are the weights for the connections between hidden and output layer. b is the bias term.

5.3.2. Training Strategy

Since the mapping from the hidden layer to the output is linear (Eq. 5.5), the optimal weights $\mathbf{w}$ can be determined analytically using **Ridge Regression**, an approach that minimizes a regularized loss function:

$$\mathcal{L}(\mathbf{w}) = ||\mathbf{y}_{target} - \mathbf{\Phi w}||^2 + \alpha||\mathbf{w}||^2 \tag{5.6}$$

Here, α is the regularization parameter (corresponding to `ridge_alpha` in the script). This term penalizes large weights, reducing the model's sensitivity to noise in the CFD data and improving its generalization capabilities.

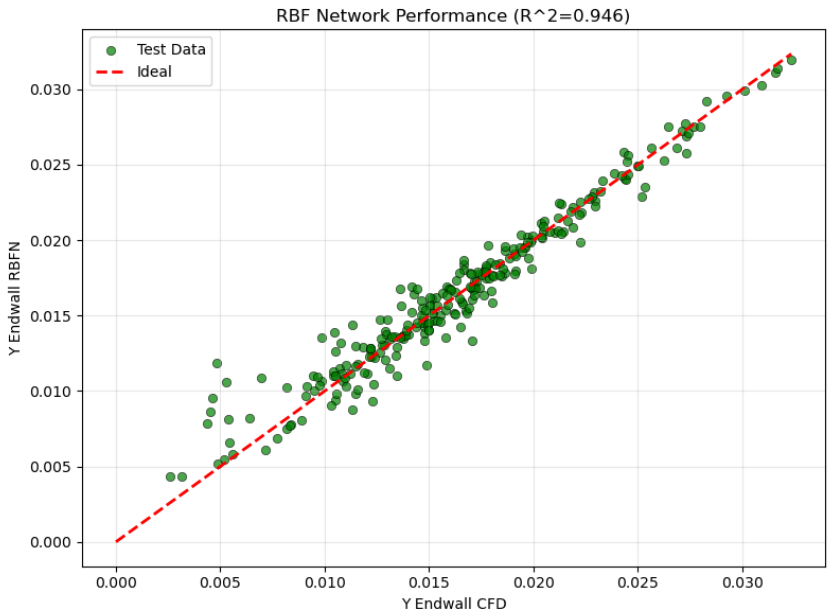


Figure 5.2: Accuracy estimation of the RBFN model.

Similarly to what was discussed for the GPR model, also for the RBFN an accuracy estimation can be performed. Its outcomes are shown in fig.5.2. The MSE and R^2 computed for this model are, respectively, 0.0000018 and 0.9464.

5.4. Results and Comparison

Beyond standard statistical metrics like MSE and R^2, a One-Factor-at-a-Time (OFAT) sensitivity analysis was conducted to verify whether the surrogate models correctly captured the underlying physics of the problem. A reference blade configuration was defined using the mean values of the design space; subsequently, each parameter was varied individually across its full range while holding the others constant. All that was done by generating massive amount of data with the two models, and using them for the sensitivity analysis.

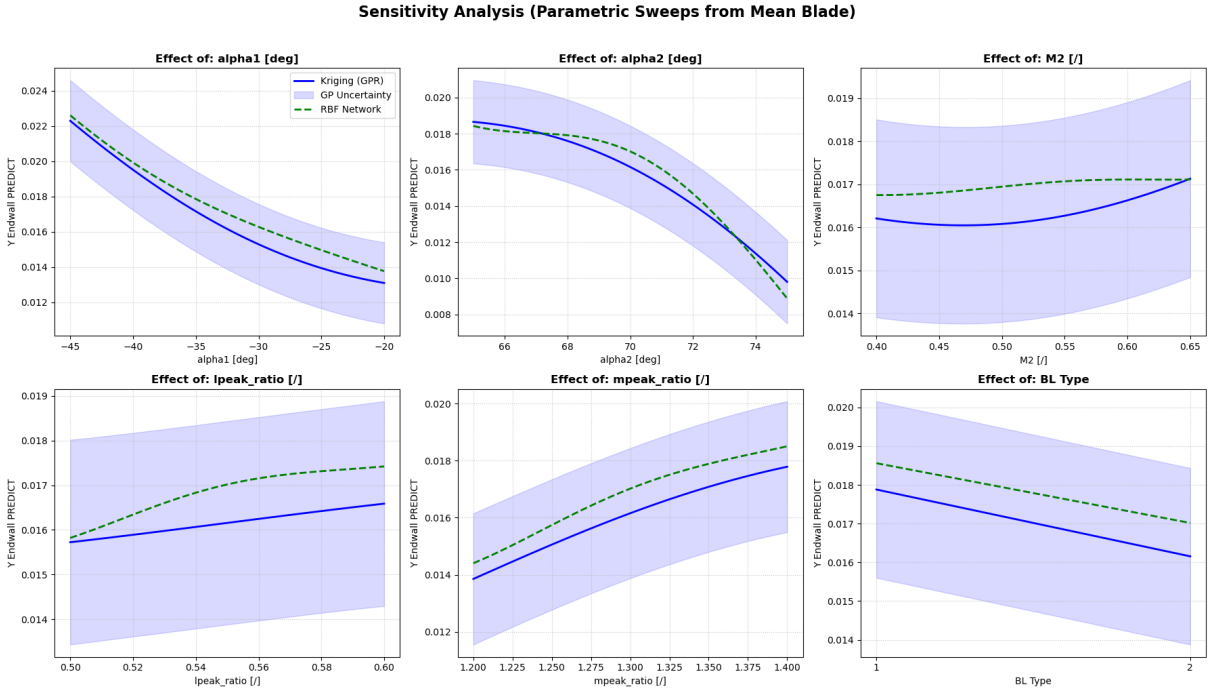


Figure 5.3: Models sensitivity analysis outcomes. Six trends for six input design parameters: each plot shows how models predicted endwall losses vary as function of an input parameter.

The analysis outcomes, depicted in fig.5.3, demonstrate strong agreement between the two models, with both reproducing the expected aerodynamic trends. The key advantage of the GPR is its probabilistic nature: as visualized in the sensitivity plots, it provides a 95% confidence interval (shaded area).

Physically, the models confirm that secondary losses are predominantly driven by blade's geometrical deflection. Given its robust performance and the ability to quantify uncertainty, the GPR can be considered as the most reliable one for possible future optimization phases.
Further information about models theoretical background and assumptions can be found in the literature review (Chapter 2) dedicated section of this report.

6 | Conclusions and Future Developments

6.1. Summary of the Work

The primary objective of this thesis work was to establish a robust, automated workflow for the prediction of secondary flow losses in turbine linear cascades, leveraging the power of Data-Driven design strategies.
To achieve this goal, the work was broken down into three logical steps. First, the development of the Blade Processing Tool (BPT), a Python-based software capable of automating the entire CFD processing, from 2D geometry generation to 3D meshing, solving, and post-processing. Second, the construction of a Database comprising 864 RANS simulations, generated through a Full-Factorial Design of Experiments. Finally, the implementation of two Machine Learning surrogate models (GPR and RBFN) to map the correlations between design input parameters and aerodynamic endwall losses.

6.2. Research Contribution

The results obtained throughout this project highlight the following important achievements.

- **Computational Performance:** Moving from CPU-based to GPU-accelerated solvers (ADS CFD) was a game-changer for this project. As reported in Chapter 4, the simulation acceleration factor of approximately 14x allowed for the simulation of the entire database in roughly one week. This demonstrates that high-fidelity database generation is now possible. Additionally, BPT demonstrated high robustness, successfully handling the automatic processing of hundreds of cases with no User intervention.
- **Aerodynamic Correlations:** From a physical viewpoint, the post-processing of the database confirmed the validity of the approach. Considering the available tools

used for the execution of this work, the CFD results showed good agreement with the SPLEEN case experimental data, particularly in capturing the macroscopic trends of blade loading and loss generation mechanisms.
The sensitivity analysis conducted on the surrogate models confirmed that the flow deflection is the dominant driver for secondary flow losses. Specifically, the investigated maps (Chapter 4) show that increasing the flow turning, and so the aerodynamic loading, leads to stronger passage vortices and, consequently, higher endwall losses. This aligns with the theoretical background and the analytical model (Coull 2025) discussed in the literature review, validating the physics of the generated database.

- **Surrogate Models Application:** Both the Gaussian Process Regression (GPR) and the Radial Basis Function Network (RBFN) achieved good accuracy metrics on the test set. However, the GPR proved to be the better choice for this specific application. Its ability to output a confidence interval together with the predictions offers an advantage for engineering design.

6.3. Encountered Issues and Limitations

Together with the outcome of this project, it is the author's duty to report the issues and limitations encountered along the work, which must be acknowledged to coherently interpret the results and guide future improvements.

- **Mesh Topology Control:** One of the most significant challenges was related to the automated meshing process of the database cascades. The usage of a structured multi-block mesh (O-H-H topology) ensures high quality results when the grid is characterized by high orthogonality. However, automating this mesh topology creation process for so many different geometries (database) required a trade-off, leading to the use of a single mesh template for all the cases, which topology was given as the result of a mesh independence analysis (Chapter 3). For the corner points of the design space, the mesh quality degraded slightly, inevitably introducing numerical noise. This fact is also visible in the loss/load maps as small local "rugosity" in the contours (Chapter 4). While the **global trends remain valid**, a more flexible meshing approach (e.g. unstructured grids) would smooth out this numerical noise.

- **Blade Parametrization:** Despite the robustness and effectiveness of the 2D Blade Generator [12], its resulting blades were not completely suited for the objective of this work. The blade generator produces blades that, with good reliability, respect

the loading conditions imposed by the input design parameters, but do not necessarily respect the required geometrical features to minimize the losses. Additionally, RANS simulations cannot capture the vortical structures with a high level of detail, inducing an additional, cumulative error in the predicted loss. This is the reason why the reader might find the reported loss values as high.

- **Small Training Set in RBFN:** Although the Radial Basis Function Network performed well, literature and experts suggest that for a Neural Network to be fully effective and robust in an n-dimensional space, the training set size should be in the order of 10^n. In the context of this work, with $n = 6$ input design parameters, this would theoretically mean that a training set of one million points is required, meaning that the full database, comprising both training and testing sets, should contain more than a million points.
  With a dataset of only 864 points, the RBFN operates with very little information. Although it managed to interpolate the provided points correctly, its prediction capabilities in the empty regions of the design space, between sampling points, is less accurate than GPR, which is mathematically more suited for smaller datasets. This limitation sets GPR as the primary surrogate model for the objectives of this work.

6.4. Future Developments

Being the strategies for turbomachinery blades inverse data-driven design relatively new, and not so documented in literature, this thesis lays the foundation for such innovative approach. The natural continuation of this work involves the creation of larger datasets, enabling the conception of new, more powerful surrogate models to substitute the current time-consuming CFD-based inverse design approach.
Specifically, following this project trend it will be possible, after the development of a well-established and high-performance surrogate model, to search for optimal blade configurations that minimize secondary losses without running computationally expensive CFD simulations.
Further potential developments of this work include:

- **Annular Cascades:** Extending the BPT to handle fully annular geometries to capture radial pressure gradient effects.
- **Twisted Blades:** Enhancing 3D blade's geometrical complexity, providing higher generalization to the tool.

- **Compressor Blades Design:** Extending the BPT to the generation of compressor cascades to spread the design methodology described in this report also for such machines.
- **Higher-Fidelity Simulation Setup:** Developing a strategy to automatically cluster the database cases based on their most relevant geometrical features, in order to create and assign a proper mesh template for each class of blades populating the database.

In conclusion, this work demonstrated that a data-driven approach, when supported by modern GPU acceleration and robust automation, is not only feasible, but highly effective for the preliminary design and analysis of turbine secondary flows.

Bibliography

[1] I. AeroDynamic Solutions. Ads cfd inc., 2024. URL `https://new.aerodynamic-solutions.com/`.

[2] D. G. Ainley and G. C. R. Mathieson. A method of performance estimation for axial-flow turbines. *Areonautical Research Council Reports and Memoranda*, n.d.

[3] M. W. Benner, S. A. Sjolander, and S. H. Moustapha. An empirical prediction method for secondary losses in turbines—part i: A new loss breakdown scheme and penetration depth correlation. *Journal of Turbomachinery*, pages 273–280, 2006.

[4] C. J. Clark. A step towards an intelligent aerodynamic design process. *ASME Turbo Expo 2019: Turbomachinery Technical Conference and Exposition*, 2019.

[5] J. D. Coull. Endwall loss in turbine cascades, 2017.

[6] J. D. Coull. Turbomachinery technical conference and exposistion, 2025.

[7] J. D. Coull and C. J. Clark. The effect of inlet conditions on turbine endwall loss, 2021.

[8] H. R. M. Craig and H. J. A. Cox. Performance estimation of axial flow turbines, 1970.

[9] J. D. Denton. Loss mechanisms in turbomachines. *Journal of Turbomachinery*, 1993.

[10] J. Dunham and P. M. Came. Improvements to the ainley-mathieson method of turbine performance prediction. *Journal of Engineering for Power*, 1970.

[11] S. C. Kacker and U. Okapuu. A mean line prediction method for axial flow turbine efficiency. *Journal of Engineering for Power*, 1982.

[12] F. Porta. Development of a machine-learning-based tool for turbomachinery blades. Master's thesis, Scuola di Ingegneria Industriale e dell'Informazione, Politecnico di Milano, 4 2023.

[13] P. J. Roache, I. B. Celik, U. Ghia, C. J. Freitas, H. Coleman, and P. E. Raad.

Procedure for estimation and reporting of uncertainty due to discretization in cfd applications. *Journal of Fluids Engineering*, 2008.

[14] C. Scarimbolo. Development of a data-driven design tool for turbomachinery blades: Exploration of secondary flows in 3d turbine cascades. Master's thesis, Collegio di Ingegneria Meccanica, Aerospaziale e dell'Autoveicolo, Politecnico di Torino, 4 2025.

[15] O. P. Sharma and T. L. Butler. International gas turbine conference, 1986.

[16] von Karman Institute for Fluid Dynamics. Ads cfd inc., 2024. URL `https://zenodo.org/records/13712768`.

[17] H. Wang, Y. Cao, Z. Huang, Y. Liu, P. Hu, X. Luo, Z. Song, W. Zhao, J. Liu, J. Sun, S. Zhang, L. Wei, Y. Wang, T. Wu, Z. Ma, and Y. Sun. Recent advances on machine learning for computational fluid dynamics: A survey, 2024.

[18] R. Yondo, E. Andres, and E. Valero. A review on design of experiments and surrogate models in aircraft real-time and many-query aerodynamic analyses. *Progress in Aerospace Sciences*, pages 1–39, 2017.

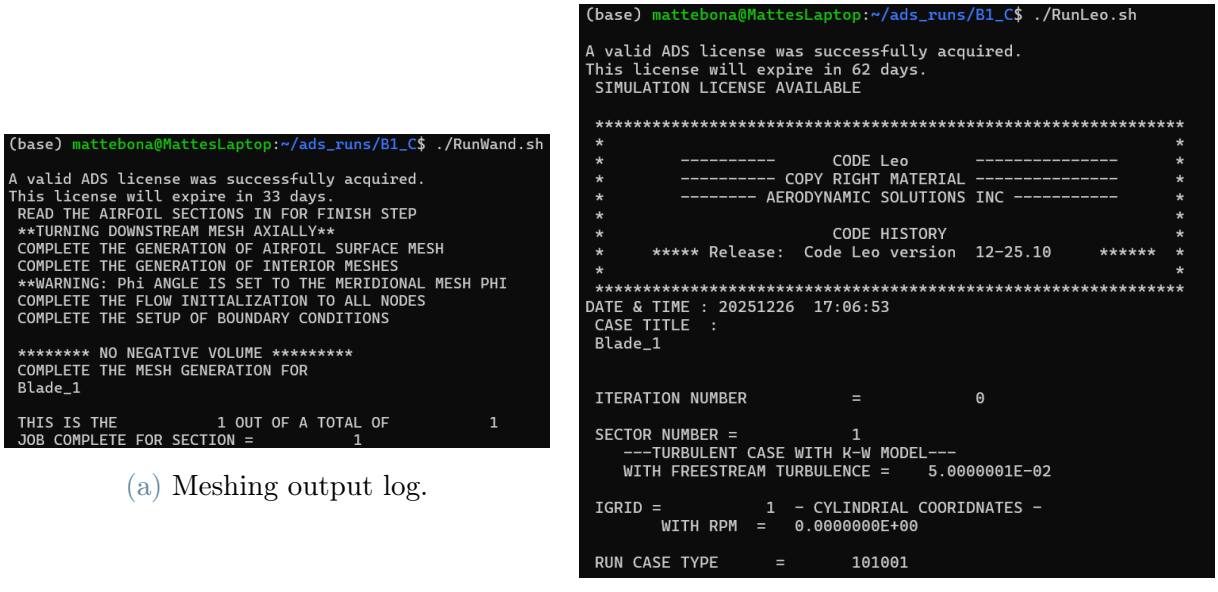
A | BPT Appendix

This appendix reports additional details for the Blade Processing Tool Chapter. The aim is to provide the reader with more information about some aspects, with the hope that a potential User wouldn't be lost in the technical aspects of the tool.

A.1. ADS - Insights

Insights about the ADS CFD software are reported below. `CodeWAND` and `CodeLEO` output logs are shown to give the User an idea of how to check if she/he is correctly launching the codes for debugging purposes. Additionally, the `.MESHIX` files (topology) for the three grids used for the sensitivity analysis are reported for completeness.

- `CodeWAND` & `CodeLEO`:



(a) Meshing output log.

(b) Simulation output log.

Figure A.1: Terminal output logs for the meshing and simulation processes.

On the left-hand side of fig.A.1, the `CodeWAND` output log, showing the general

mesh quality metrics. On the right-hand side, the `CodeLEO` output log, showing the solver's computation progresses.

- **Sensitivity Analysis:**

```
NBLOCK   ADVANCED
5        -20
BCK#     IM        JM        KM
1        195       23        101
2        195       23        101
3        161       12        101
4        63        11        101
5        63        11        101
LE-RFA, LE-MOV, LE-ANG, TED-RFA, TE-MOV, TE-ANG
1.00000   0.00000   1.00000   1.00000   0.00000   1.00000
```

```
NBLOCK   ADVANCED
5        -20
BCK#     IM        JM        KM
1        212       25        150
2        212       25        150
3        175       13        150
4        68        11        150
5        68        11        150
LE-RFA, LE-MOV, LE-ANG, TED-RFA, TE-MOV, TE-ANG
1.00000   0.00000   1.00000   1.00000   0.00000   1.00000
```

```
NBLOCK   ADVANCED
5        -20
BCK#     IM        JM        KM
1        230       27        223
2        230       27        223
3        191       14        223
4        73        11        223
5        73        11        223
LE-RFA, LE-MOV, LE-ANG, TED-RFA, TE-MOV, TE-ANG
1.00000   0.00000   1.00000   1.00000   0.00000   1.00000
```

Figure A.2: `.MESHIX` files used for mesh sensitivity analysis. In the top, the Coarse mesh. In the middle, the Medium mesh. In the bottom, the Fine mesh.

Increasing mesh refinement means increasing the number of cells which discretize the fluid domain. The rationale for creating them has been focused on delivering three meshes whose cells' aspect ratio could be as close as possible to 1. The mesh finally chosen as template for the database generation is the Medium one, giving the right compromise in terms of quality and computational efforts.

A.2. Virtual Environment - Insights

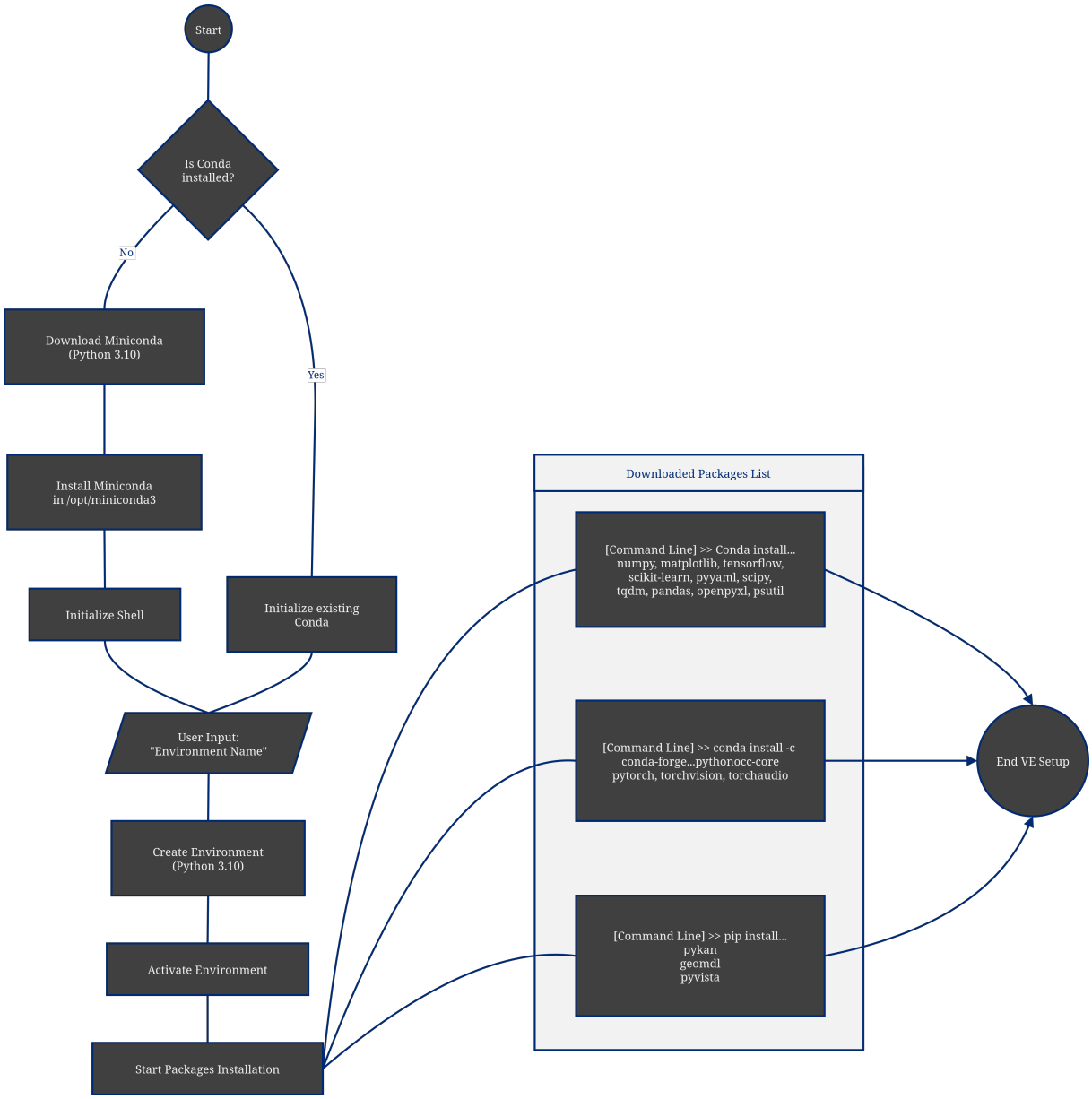


Figure A.3: Virtual environment setup pipeline followed by the `BPT_setup.sh`.

To launch the BPT, a Linux-based OS and a virtual environment for downloading all the needed packages to support the Python modules and sub-modules are required. In order to avoid confusion and loss of time, an executable script (`BPT_setup.sh`) that automatically performs the virtual environment setup has been created. It is delivered together with the BPT. To run it, it must first be made executable from terminal with the command: `chmod +x BPT_setup.sh`, then it can be launched with `./BPT_setup.sh`.

The User should pay attention to the ability, in the Linux-based environment from which the tool is launched, to give MISES (used by the 2D Blade Generator) the capability to run simulations, hence to create virtual screens. In case problems in this sense are encountered, the `Xvfb` package should be installed with the command: `apt install xvfb`.

B | Database Appendix

B.1. SPLEEN Validation

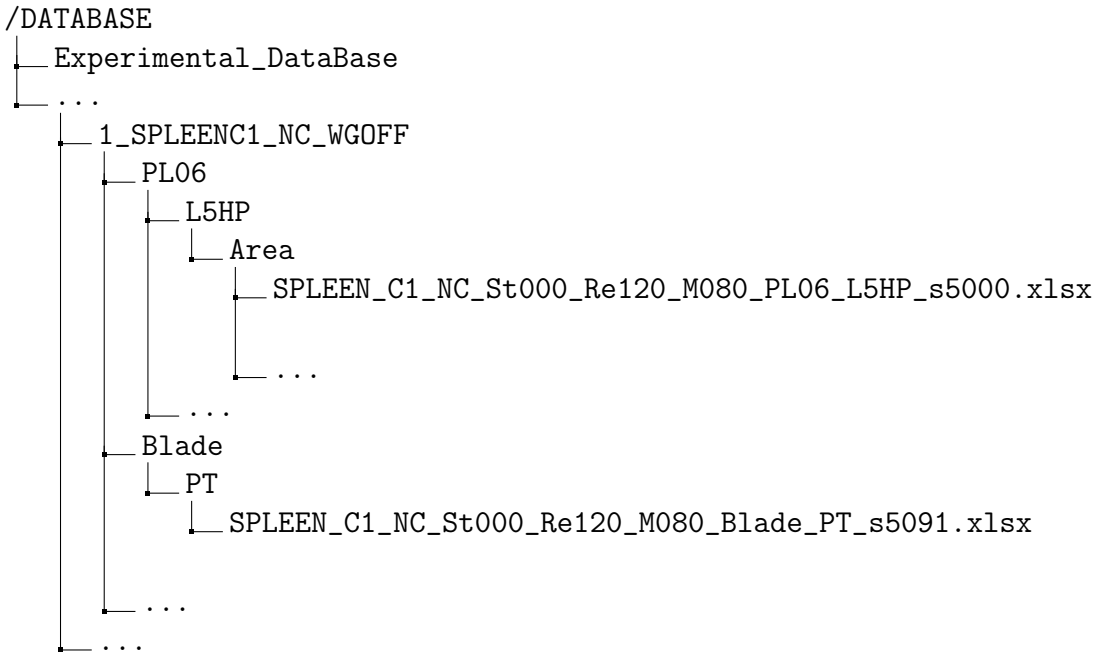


Figure B.1: Internal structure of the SPLEEN database folder, showing the path to the files used for the validation.

Fig.B.1 reports a tree representation of the SPLEEN database internal structure, highlighting the files used for the validation procedure described in detail in Chapter 4. Fig.B.2, instead, shows the SPLEEN 2D blade section on the left, and its CFD simulation with the database setup on the right.

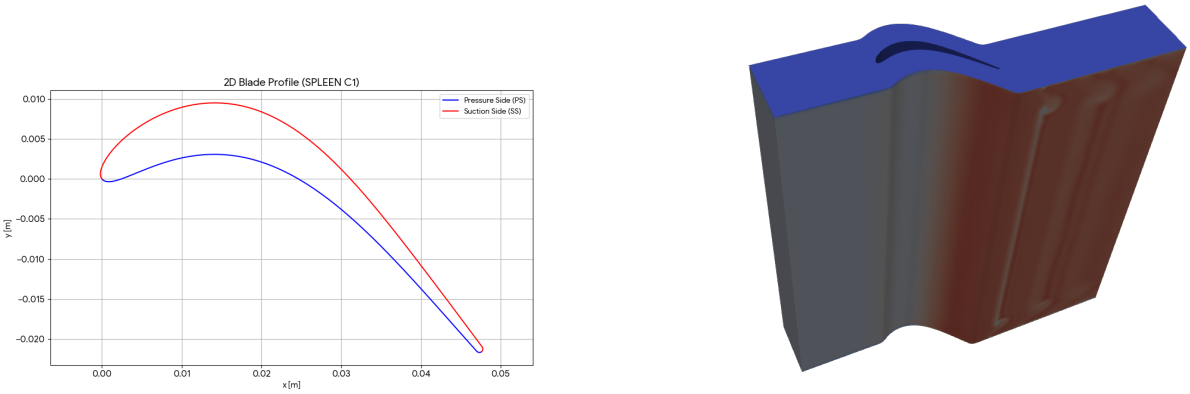


Figure B.2: SPLEEN 2D blade geometry on the left, and the corresponding CFD case with database setup on the right.

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Full Form / Meaning
DoE	Design of Experiments
db	Database
DFM	Data Fit Model
MFM	Multi Fidelity Model
ROM	Reduced Order Model
HSM	Hybrid Surrogate Model
NN	Neural Network
GPR	Gaussian Process Regression
BPT	Blade Processing Tool
GPU	Graphics Processing Unit
CPU	Central Processing Unit
CFD	Computational Fluid Dynamics
ML	Machine Learning
ADS CFD	Aerodynamic CFD Simulation Software
.AGF	ADS case file
.WAND	ADS case file
.MESHIX	ADS case file
.LEO	ADS case file
BCs	Boundary Conditions
BL	Boundary Layer
OS	Operative System
IDE	Integrated Development Environment
WSL	Windows Subsystem for Linux
VE	Virtual Environment
LE	Leading Edge
TE	Trealing Edge
GCI	Grid Convergence Index

Abbreviation	Full Form / Meaning
$RANS$	Reynolds Averaged Navier-Stokes
$2DBG$	2D Blade Generator
TKE	Turbulent Kinetic Energy

List of Symbols

Symbol	Description	SI unit
N_b	Blades Number	/
R_h	Hub Radius	mm
R_s	Shroud Radius	mm
R_m	Midspan Radius	mm
s	Pitch	mm
h	Blade Span	mm
Y_t	Total Pressure Loss Coefficient	/
C_{ax}	Blade Axial Chord	mm
AR	Blade Aspect Ratio ($= h/C_{ax}$)	/
Z_{TE}	Penetration Height	mm
Zw	Zweifel Coefficient	/
C_0	Circulation Coefficient	/
DF	Diffusion Factor	/
Y	Total Pressure Losses (Total Pressure)	/
ζ	Kinetic Energy Losses	/

Acknowledgements

I would like to express my gratitude to my advisor, Prof. Paolo Gaetani, and my co-advisor, Prof. Sergio Lavagnoli, for giving me the extraordinary opportunity to work on my thesis project within the framework of the Short Training Program held at the von Karman Institute for Fluid Dynamics, allowing me to spend six months in a worldwide recognized high-level research institute.

I would like to thank my family. To my parents, Rita and Moreno, for their unwavering belief in my potential and for the sacrifices they made to allow me to reach this milestone. To the rest of my family, for their patience and support throughout this journey. To my grandmother, Raffaella, whose life lessons and charisma sustained my motivation in pursuing this achievement.

Last but not least, I want to thank my friends and colleagues. A special mention goes to my friend and colleague Matteo Aprile: thank you for your valuable help in supporting my thesis with a strong theoretical background and for your patience. To my friends and colleagues Antonio Silvestri and Alessandro Pescosolido, thank you for the nights spent in the office and the laughter we shared together. Finally, to all my friends in my hometown, Prato (Italy), in Milan (Italy), and in Bruxelles (Belgium), thank you for your precious moral support that boosted my motivation.

Ringraziamenti

Vorrei esprimere la mia gratitudine al mio relatore, Prof. Paolo Gaetani, e al mio correlatore, Prof. Sergio Lavagnoli, per avermi offerto la straordinaria opportunità di lavorare al mio progetto di tesi nell'ambito dello Short Training Program tenutosi presso il von Karman Institute for Fluid Dynamics, permettendomi di trascorrere sei mesi in un istituto di ricerca di alto livello riconosciuto in tutto il mondo.

Vorrei ringraziare la mia famiglia. Ai miei genitori, Rita e Moreno, per la loro incrollabile fiducia nel mio potenziale e per i sacrifici fatti per permettermi di raggiungere questo traguardo. Al resto della mia famiglia, per la pazienza e il supporto durante tutto questo percorso. A mia nonna, Raffaella, i cui insegnamenti di vita e carisma hanno alimentato la mia motivazione nel perseguire questo obiettivo.

Infine, ma non per importanza, voglio ringraziare i miei amici e colleghi. Una menzione speciale va al mio amico e collega Matteo Aprile: grazie per il tuo prezioso aiuto nel supportare la mia tesi con solide basi teoriche e per la tua pazienza. Ai miei amici e colleghi Antonio Silvestri e Alessandro Pescosolido, grazie per le notti trascorse in ufficio e per le risate condivise. Infine, a tutti i miei amici nella mia città natale, Prato, a Milano, e a Bruxelles, grazie per il vostro prezioso supporto morale che ha alimentato la mia motivazione.