POLITECNICO DI MILANO
Scuola di Ingegneria Industriale e dell'Informazione
Master Degree in Mechanical Engineering
Energy Department

Three-dimensional shape-optimization of a transonic gas-turbine cascade by surrogate-based evolutionary techniques

Supervisor: Prof. Giacomo Persico

Graduation thesis of:
Andrea ROSSI, student number: 883580

Anno Accademico 2017-2018

Contents

List of Figures

List of Tables

Abstract

The aim of the work is to systematically investigate the blade shape optimization process in the turbomachinery field. We starts from an in house tool which has been developed at Politecnico di Milano, and we completely rewritten it with two main targets; the former is to prepare a tool which should be as flexible as possible, to avoid having to change the code, but in case it is necessary, the Python objected-oriented capability makes the implementation of custom function straightforward, since it is not required to rewritten already available pieces of code; the latter instead is to already implement features that can be useful for a designer when dealing with turbomachinery blade shape optimization. Actually the original tool was only capable of managing bi-dimensional blades, while the new version includes the possibility to perform three-dimensional, stator-rotor, multi-points, multi-objectives, multi-constraints, and free-slip no-slip optimizations, other that support features like flexible multi-threading capabilities and post-processing directly during the optimization.

The parametrization process performed with B-spline curves is improved, providing an automatic way of generating both the control points and the knot array. The editing of the points is along direction tangential to the curve, to halve the design variables, and the trailing edge is generated with the most rounded ellipse which guarantees continuity and smoothness of the blade, minimizing the trailing edge issues. The performances of the blades are assessed with a CFD commercial code, Ansys-CFX, computing the entropy production as objective function. Since we are dealing with expensive functions, a surrogate-based algorithm is applied to reduce the computational cost. A Kriging model is built over an initial database of samples and updated every iteration. A genetic algorithm is responsible for finding the optimum over the surrogate model. The great benefit of this heuristic approach compared to analytical techniques is that it is completely not intrusive, treating the objective function as a black box without requiring the computation of the derivatives, with the great advantage of being automatized, from the parametrization of the baseline shape to the post-processing.

The present tool is applied to a three-dimensional gas turbine stator blade, analyzing the influence that the surrogate parameters have on the optimal blade, like the choice between local and global approaches, the size of the design of experiments, and the influence of the randomness in this kind of optimizations. Both unconstrained and constrained optimizations are run to highlight where the a better blade should go.

The objective function and the constraint of the three-dimensional simulations are

managed by an innovative procedure; we separate the contribution of the reference flow from the secondary one, and we use the former for the constraint on the outlet flow angle, and the latter for the entropy production.

Key Words: Blade Shape Optimization; Three-dimensional Optimization; Genetic Algorithms; Evolutionary Techniques; Kriging modeling; B-splines Parameterization; Seed analysis; Gas Turbine.

Sommario

Lo scopo del lavoro è quello di indagare sistematicamente il processo di ottimizzazione di forma di palettature nel campo delle turbomacchine.

Partendo da un software sviluppato internamente al Politecnico di Milano abbiamo completamente riscritto il codice con due obiettivi principali. Il primo è quello di preparare uno strumento che sia il più flessibile possibile, affinchè si eviti, dove non strettamente necessario, di modificare il codice, ma nel caso non si possa fare altrimenti, di rendere l'implementazione della funzione personalizzata alquanto semplice grazie al paradigma di programmazione di Python (Objective oriented OOP ovvero programmazione orientata agli oggetti), in quanto non è piu necessario riscrivere parti di codice già disponibili. Il secondo invece è quello di fornire molte delle funzionalità utili ad un progettista che affronti l'ottimizzazione della geometria di una turbomacchina.

Lo strumento originale era solo in grado di gestire le palettature bidimensionali, mentre la nostra nuova versione prevede la possibilità di eseguire ottimizzazioni tridimensionali, statore-rotore, multi-punto, multi-obiettivo, multi-vincolo e ottimizzazioni del flusso di riferimento, oltre a supportare funzionalità come multi-threading personalizzabile e post-elaborazione durante l'ottimizzazione.

Il processo di parametrizzazione eseguito con curve B-spline è migliorato rispetto alla versione precedente, fornendo un sistema automatico di generazione sia dei punti di controllo che della sequenza di nodi. La modifica dei punti avviene lungo la direzione tangenziale alla curva, in modo da dimezzare il numero di variabili, mentre il trailing edge viene costruito con unéllisse che sia il più arrotondata possibile, il che garantisce continuità e derivabilità della pala, minimizzando i problemi che potrebbero sorgere durante la simulazione fluidodinamica.

Le prestazioni delle pale sono valutate per mezzo di un codice commerciale CFD: Ansys-CFX, il quale calcola la produzione di entropia utilizzata come funzione obiettivo. Poiché si tratta di funzioni computazionalmente costose, si applica un algoritmo basato su funzioni surrogate per ridurre il tempo di calcolo. Un modello di Kriging è costruito interpolando un database iniziale di campioni che si aggiorna ad ogni iterazione. Per trovare l'ottimo del modello surrogato è utilizzato un algoritmo genetico fornito dalla libreria JEGA. Il grande vantaggio di questo approccio euristico rispetto alle tecniche analitiche è quello di non essere completamente intrusivo, trattando la funzione obiettivo come una scatola nera senza richiedere il calcolo delle derivate, con la possibilità di essere automatizzato, partendo dalla parametrizzazione della forma della pala originale

fino al post-elaborazione.
Lo strumento attuale viene applicato a una pala statorica di una turbina a gas tridimensionale, analizzando l'influenza che hanno i vari parametri del surrogato sulla pala ottimale, come la scelta tra approccio locale e globale, la dimensione del design degli esperimenti e línfluenza della casualità in questo tipo di ottimizzazioni. Vengono eseguite sia ottimizzazioni vincolate che libere, per individuare la regione in cui si trovano le pale migliori.
La funzione obiettivo ed il vincolo delle simulazioni tridimensionali sono gestiti da una procedura innovativa; viene separato il contributo del flusso di riferimento da quello secondario, utilizzando il primo per il vincolo sull'angolo del flusso in uscita, ed il secondo per il calcolo della produzione entropica.

Parole Chiave: Ottimizzazione di palettature; Ottimizzazione Tridimensionale; Algoritmi Genetici; Strategie Evolutive; Metamodelli di Kriging; Parametrizzazione con curve B-spline; Analisi del seed; Turbine a Gas.

Chapter 1

Introduction

Computational methods are nowadays spread in almost any engineering field as support tool for the design of many products or plants. In fluid-dynamics and turbomachinery design, the goals of efficiency and performance are becoming more and more challenging, and automatic numeric tools are often necessary to improve the manual design being able to fulfill all the requirements while satisfying the constraints.
CFD software and codes have been available since decades, but in the past were mainly applied as validation tools of the already designed geometry. With the increase in reliability of the fluid-dynamics and turbulence models and of the CFD algorithms, in combination with the progressively availability of more and more computational power at relatively low cost, it is nowadays possible the use of CFD simulations as design tools. In this work we will exploit the field of fluid-dynamics optimal shape optimization, which requires the combination of efficient heuristic optimization algorithms with high-fidelity CFD simulations. The fluid-dynamic phenomena are highly non-linear, making the use of deterministic optimization algorithms often problematic. On the other side heuristic algorithms can have a variability on the solution found, depending on random initial parameters, like the seed of the random number generator.

Even if the computational power has increased, aerodynamic optimization remains, however, a really challenging task from both the algorithmic, the engineering and the computational points of view, being necessary to perform CFD simulations during optimization with a lower level of fidelity.

1.1 Motivations

At Politecnico di Milano, in the Energy department, an in-house tool for shape optimization already exists which combines the Dakota framework for the surrogate model and the genetic algorithm with the Ansys CFX software for the CFD simulations. Even though already successfully applied to a number of different turbomachinery, it has limitations in terms of flexibility, manual pre-processing and blade interpolation, features and future extensibility. For these reasons, the software has been completely rewritten, integrating all the features together in Python code. It is an open source,

dynamic, object oriented, imperative, functional, procedural, multi-threaded enabled and widely widespread programming language [1]. It is particularly suitable for scientific and engineering applications thanks to its simplicity, code readability, extensibility and to the huge amount of libraries of any purpose already available. It interfaces with CFX-Ansys for the fluid-dynamic simulations and with Dakota for the surrogate model building and for the optimization.

The new code improves the already valid B-spline parametrization tool, introducing an automatic blade interpolation algorithm, which chooses the knot sequence of the B-spline through an optimization procedure, a more flexible movement of the trailing edge and a better elliptical shape of it, reducing the user effort just to set some basic interpolation parameters. Then the objective oriented principles of inheritance and polymorphism have been applied to improve the code quality and ensure future extensibility in a much easier way than before; actually introducing a new data processing function is as easy and simply write that specific mathematical operation.

Beside programming features, we have introduced an automatic way to manage 3D parametrization and simulations, multi-point optimizations, a complete stage with both a stator and a rotor, and the chance to run the same simulation in both free-slip and no-slip on hub and shroud walls. And all these different kind of optimizations can be theoretically combined together, so to simulate both the fully 3D flow in cascades as well as reference flow of multi-point stator-rotor simulations. Then post-processing on all the blades that is not directly related to the optimization can be applied with the tool, computing for example the convergence trend of the quantity of interest, the residuals or the blade loading, making much easier to monitor both the optimization and the solutions while it is running.

In terms of performances we have exploited both python and Dakota multi-process capability, giving the opportunity to run multiple high-fidelity CFD simulations in parallel, instead of just one simulation with more cores. This has a great impact on performances since not massive simulations like the ones we are dealing with, do not scale too much with cores count.

After discussing the development of this tool we will focus in particular on 3-dimensional blade shape optimization and we will investigate some of the topics related to surrogate-based optimization that are not fully covered in literature. First at all we have to decide the surrogate parameters, from the kind of strategy between local and global, to the number of sample for the generation of the initial database, and the exact number of control points, then we have rerun multiple times the same optimization, analyzing the effect of the randomness during the choice of the samples of the DOE and then keeping the same database, we have run the optimization with different seeds that govern the genetic algorithm, for both constrained and unconstrained cases. After that, referring to Persico et Al. peper [2], we have decided to perform optimization with different boundary conditions at the hub and shroud walls, computing in this way the reference flow. The idea is that we actually want to constraint the outlet flow angle of the reference flow generated by a blade, because we are not really interested in having

the same secondary flows of the baseline blade, since in general we want to minimize them. This strategy has been applied for two different blades, with different constraints applied. Then a seed analysis has been executed for the most significant case to highlight the difference in the influence of randomness between 2D and 3D simulations.

At the end we have extended the reference flow concept observing that we are interested in the outlet flow angle of the reference flow that a blade generates, but for the entropy drop minimization we are actually interested in that generated also by the secondary flows. For this reason we have combined both simulations extracting the relevant data from the former, the latter or even a combination between them.

1.2 Thesis structure

Chapter 2 presents few among the many examples from the literature ; between chapter 3 and 6 the mathematical steps required by the optimization process are explained, in particular in chapter 3 some of the parametrization techniques available in literature are presented, focusing then on the B-spline model implemented for the current work, detailing the interpolation, the modification of the control points and the management of the trailing edge; in chapter 4 are explained the design of experiments techniques. Chapter 5 will focus on the choice of surrogate models available pointing out also the difference between local and global surrogate based optimization, chapter 6 describes the genetic algorithm and the JEGA library which implements it, and finally chapter 7 contains some basic fluid dynamics explanations that are integrated in the CFD tool.
After this mostly theoretical background, chapter 8 describes the procedure to setup and perform an optimization, then in chapters 9, 10 and 11 are performed several optimizations on a turbine stator blade; among them we have tested the size of the DOE, global and local SBO comparison, different selection of control points, randomness analysis and 3D optimizations neglecting the secondary flows.
Finally chapter 12 combines together reference flow and fully 3D flow with secondary effects in two simulations, extracting the relevant data from the proper one.

Chapter 2

State of art in fluid-dynamics shape optimization

Many applications of fluid-dynamic optimization are available in literature. Actually with the increase in computational available power and the improvement of the algorithm, the CFD is widespread applied not only as a tool to validate the design performed by experienced engineers, but nowadays is also used as design instrument in combination with optimization algorithms.

Among the others, development was made in the fields of adjoint optimization, genetic algorithm optimization, surrogate models, constrained, multi-point, multi-objective, multidisciplinary and robust optimization.

Some of the automatic tools available in literature are briefly reported in the current character.

2.1 Multidisciplinary and multiple operating points shape optimization of three-dimensional compressor blades

Pierret et al. [3] develop and uses a tool that combine an high-fidelity fluid-dynamic simulation with a FEM structural mechanical simulation.

Their aim was to present the method optimizing the well known NASA rotor 67 representative of a compressor blades.

The objective functions comes from a multi-point simulation in three different operating points weighted to obtain a single objective. The constraints come from both fluid-dynamic and structural analysis and are included inside the objective function through a penalty strategy. They considered for the fluid-dynamic part both the pressure ratio and the mass flow rate, and they included both static and dynamic quantities, in particular they limit the value of the Von Mises stress on the blade for the former and they impose the first and second vibration modes to be outside critical ranges.

At first a design of experiments algorithm generates a set of individuals to be evaluated with the high-fidelity tools. This procedure is really heavy but has the great

advantage that can be easily parallelized with the MPI linux tool. Then a radial basis function interpolation technique is used to construct an accurate approximate model that provides an analytical relation between the design variable and the responses (objectives and constraints). The genetic algorithm applied to this analytical function is able to find the optimum of the meta-model. The CFD and the FEM high fidelity simulations are executed to evaluate and verify objectives and constraint. The new value is appended to the database and the surrogate model is updated, and the GA optimization is repeated in a loop until the maximum number of iterations is reached.

The advantage of introducing a complete structural analysis is that the accuracy is much better than other fast and approximated methods while the computational cost remains limited (5 min vs 1 hour of the CFD simulation).

2.2 Turbomachinery Design Optimization using Adjoint Method and Accurate Equations of State

Professor Pini developed a discrete adjoint based shape optimization tool at Politecnico di Milano [4].

The discrete adjoint method is required to compute the derivative of the objective function respect to the design parameters. In particular differently from the continuous method, the differential equation that model the system is before discretized and than the adjoint operator is applied to the algebraic equation. In this work, the design variables are the geometrical parameters of the problem. A NURBS interpolation technique is applied and the set of control points are considered as design variable.

The solver they used is a custom in-house implementation of the the integration of the discrete form of the Euler equations by applying finite element or volume with Runge-Kutta forth order implicit time integration. An improvement of this thesis is the management of real gas fluid. Equations of state (EoS) implementation has usually a too high computational cost when paired with CFD simulations. Look-up Tables on the other hand have consistency error that can even prevent solver convergence. The trade-off between accuracy and computational cost was found using LoT with $(\log(v), s)$ as thermodynamic variables.

They use a preconditioned steepest descend optimizer to reach robustly the optimal point. The algorithm has also been extended to threat off-design conditions with a multi-point approach where the weights are assessed with uncertainty quantification.

The proposed method was tested firstly with the redesign of a 2D wind tunnel nozzle with both ideal and real gases. Then two turbomachinery cases are shown, the former maximizing performances of a transonic cascade and the latter to achieve a more uniform flow on a converging-diverging transonic turbine cascade.

The work showed that in this case adjoint methods are more efficient than GA algorithms.

2.3 Uncertainty Quantification of an ORC turbine blade under a low quantile constrain

Razaaly et Al. [5] faces the problem of solving deterministic CFD equations in problems where uncertainty quantities are really significant. In organic Rankine cycle (ORC) turbines the working fluids are in general characterized by complex molecules and moderate to large molecular weights. This makes the development of an accurate equation of state a quite difficult process. Moreover this machines are inside biomass, geothermal or solar energy plants which typically feature variable load and inlet condition. Furthermore to reduce the manufacturing cost many companies applies low quality machining ans assemblies technologies that increases the production tolerances.

In such a complex and uncertain field the work investigate accuracy and efficiency of a sparse polynomial dimensional decomposition for the analysis of an ORC turbine under multiple uncertainties.

The boundary conditions set to the CFD analysis are inlet total pressure p_{in}^{T} and and temperature T_{in}^{T} and outlet static pressure p_{out}. Then they study three different cases, keeping the same inlet conditions with uncertainty and considering:

- Case 1: outlet pressure with large uncertainty;
- Case 2: lower outlet pressure with small uncertainty;
- Case 3: higher outlet pressure with small uncertainty;

The results of the ANOVA analysis show that in case 1 the inlet conditions has almost no influence of the simulations, in case 2 the total inlet temperature effect is negligible while the effect of total inlet pressure is small but non negligible (8 % vs 92 %). Finally in case 3 the temperature influence is still negligible but the Sobol index for p_{in}^{T} significantly increases (25 % vs 75 %). However we can summarize that the variable that has the greater influence on the simulation result is the outlet pressure.

The conclusion of their work is that since the variability on the objective function is really high a robust optimization is expected to have a large impact on the geometry respect to the deterministic optimal one.

2.4 Efficient Global Optimization (EGO) for Multi-Objective Problem and Data Mining

Jeong et Al. [6] applies a surrogate model to multi-objective aerodynamic optimization design, in particular they show the application of the method to a transonic airfoil design. Differently from out implementation, the surrogate model is not used to simply take its optimum with a genetic algorithm, but a more sophisticate approach is applied. They use the efficient global optimization (EGO) algorithm which chooses the optimum of the surrogate model based on the concept of *expected improvement* EI. Actually a surrogate model like the Kriging used in our and in their works is able to provide both

the approximated function value and its uncertainty. And the EGO combines both information, meaning that a point which has a high fitness value (in a minimization case) but with an high uncertainty may be better that the optimum value which has a small uncertainty. In EGO, the exploration is based on the potential of being optimum instead of the function value itself.

The author was able to successfully apply this approach in both single objective [7] and multi-objective aerodynamic optimization [6].

For the multi-objective optimization the application of the ParEGO [1] algorithm is possible, but its ability to find the correct non-dominated solution highly depends on the choice of the weighting vector. For this reason they decided to compute the expected improvement of each objective functions and directly apply a multi-objective optimization algorithm (like the MOGA 6.2.2) which are capable of obtaining the correct non-dominated solutions.

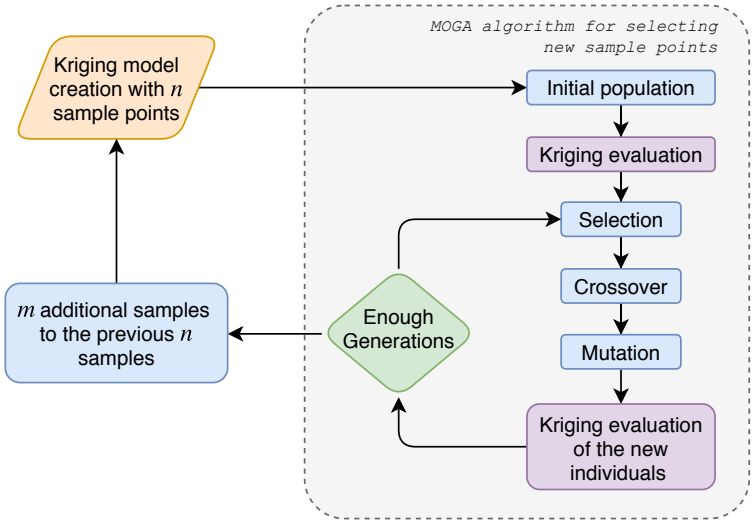


Figure 2.1: Schematization of the algorithm in the article.

Furthermore they simplify the design space removing the variables that has negligible effect on the solution by applying data mining algorithm. To kind of strategies are applied, one quantitative and the other qualitative. The former is the analysis of variance or ANOVA, which uses the variance of the function due to the design variable on the surrogate model. The contribution of each design variable on the objective function is identified decomposing the total variance into the variance due to a single variable. The latter is the self-organizing map or SOM, which is an unsupervised learning, non linear projection algorithm, based on an array of neurons.

The results of the ANOVA in terms of the parametrization with NURBS curves are not really clear. A transformation of the variables in terms of geometrical properties of the blade has revealed that the results are consistent with the aerodynamic knowledge.

The SOM analysis shows the same qualitative results of the ANOVA.

[1] The objectives are converted to a single one by using a parameterized weighting vector.

Chapter 3

Geometrical parametrization

In blade shape optimization the parametrization of the base shape is a critical factor.

An arbitrary shape requires an infinite number of coordinates for its representation, and a direct calculation of that is clearly impossible. Blade shapes are usually represented by hundreds or thousands of points, which is clearly optimal from the point of view of accuracy but will, on the other hand, bring to an unworkable problem. The parametrization in terms of a limited number of variables is a fundamental step in the setup of an optimization. The use of Bezier curves or B-spline is recommended [8], and actually we have chosen to use B-splines. It is clear that the profile interpolation has to be executed with fewer variables possible, but meanwhile we want to be able to modify the curve locally to exploit the best profile during the optimization.

In this chapter we will showcase the main techniques in blade parametrization, focusing in particular on our implementation of the B-spline interpolation.

3.1 Ferguson's curves

The Ferguson's curve [9] is a parametric cubic curve $\mathbf{r}(u)$ with $u \in [0,\ 1]$ that connects two points A and B, at which we impose the tangents in that points.

$$\mathbf{r}(0) = \mathbf{A} \qquad \mathbf{r}(1) = \mathbf{B} \qquad \left.\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}u}\right|_{u=0} = \mathbf{T}_A \qquad \left.\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}u}\right|_{u=1} = \mathbf{T}_B \tag{3.1}$$

Since the curve is a cubic it can be written in the polynomial form as:

$$\mathbf{r}(u) = \sum_{n=0}^{3} \mathbf{a}_n\, u^n \quad \text{for } u \in [0,\ 1] = \mathbf{a}_0 + \mathbf{a}_1\, u + \mathbf{a}_2\, u^2 + \mathbf{a}_3\, u^3 \tag{3.2}$$

The derivative of the curve respect to u can be easily computed as $\mathrm{d}\mathbf{r}/\mathrm{d}u = \mathbf{a}_1 + 2\mathbf{a}_2\, u +$

$3\mathbf{a}_3\, u^2$, and with that it is possible to set up the four boundary conditions.

$$\begin{cases} \mathbf{r}(0) = \mathbf{a}_0 = A \\ \mathbf{r}(1) = \mathbf{a}_0 + \mathbf{a}_1 + \mathbf{a}_2 + \mathbf{a}_3 = B \\ \frac{\mathrm{d}\mathbf{r}}{\mathrm{d}u}\big|_{u=0} = \mathbf{a}_1 = \mathbf{T}_A \\ \frac{\mathrm{d}\mathbf{r}}{\mathrm{d}u}\big|_{u=1} = \mathbf{a}_1 + 2\,\mathbf{a}_2 + 3\,\mathbf{a}_3 = \mathbf{T}_B \end{cases} \tag{3.3}$$

The solution of the linear system 3.3 can be easily computed by hand, and it is reported in equation 3.4.

$$\begin{cases} \mathbf{a}_0 = A \\ \mathbf{a}_1 = \mathbf{T}_A \\ \mathbf{a}_2 = 3\mathbf{B} - 3\mathbf{A} - 2\mathbf{T}_A - \mathbf{T}_B \\ \mathbf{a}_3 = 2\mathbf{A} - 2\mathbf{B} + \mathbf{T}_A + \mathbf{T}_B \end{cases} \tag{3.4}$$

Replacing the terms inside the general Ferguson equation we obtain the explicit formulation for the parametric curve. This can be then rewritten in terms of matrices as in equation 3.5.

$$\mathbf{r}(u) = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{A} \\ \mathbf{B} \\ \mathbf{T}_A \\ \mathbf{T}_B \end{bmatrix} \tag{3.5}$$

The resultant expression isolates the contribution of the input parameters to the curve, the so-called Hermitian basis functions, sketched in figure 3.1.

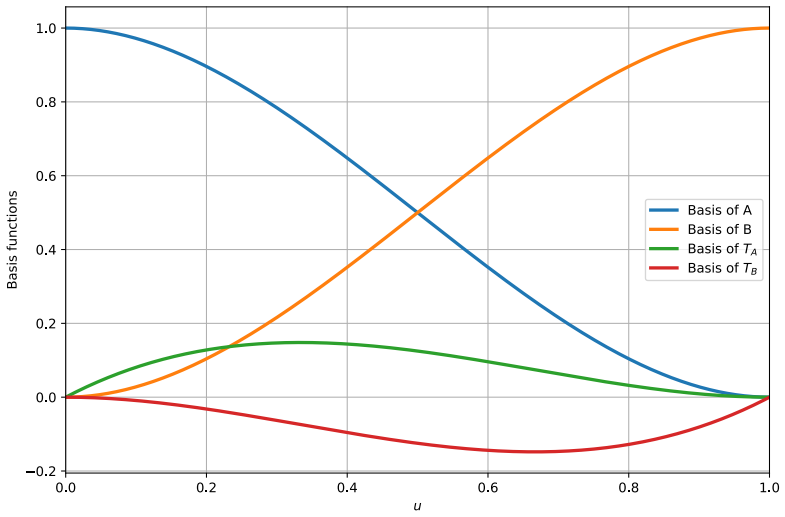


Figure 3.1: Representation of the basis of the Ferguson curves.

A single parametric curve of this kind is not able to represent a full blade profile, but we can combine two of them, one for the upper and one for the lower part of the

profile. For continuity constraint the extreme points of the upper and the lower parts have to be the same. This reduced the number of parameters necessary to describe a blade with the Ferguson's curve to 6.

An example of the parametrization of the NACA 5410 blade is shown in figure 3.3

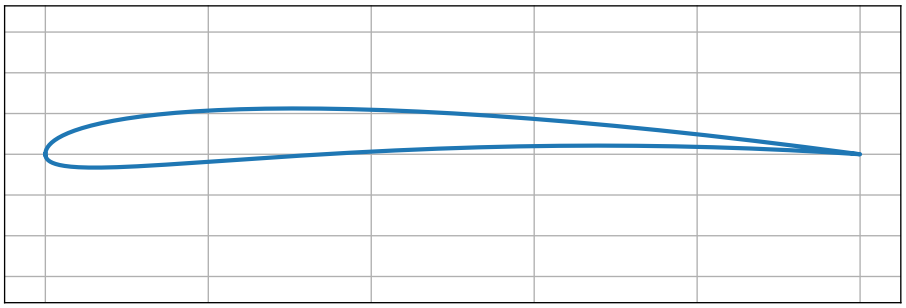


Figure 3.2: Example of a Ferguson curves.

This kind of representation is really basic and it is not suitable for blade optimization.

3.2 Class Shape Function Transformations (CST)

CSTs were designed for approximating a wide range of aerofoils with relatively few design variables [10]. It is defined by separating the upper and the lower part of the blade in two different expression:

$$z_{\text{upper}} = C_{N_1}^{N_2}(u) \cdot S_{\text{upper}}(u) + u \cdot \Delta z_{\text{upper}} \tag{3.6}$$

$$z_{\text{lower}} = C_{N_1}^{N_2}(u) \cdot S_{\text{lower}}(u) + u \cdot \Delta z_{\text{lower}} \tag{3.7}$$

where $u \in [0,\ 1]$, Δz describes the trailing edge thickness and the function $C_{N_1}^{N_2}(u)$ is equal to $u^{N_1} \cdot (1-u)^{N_2}$.

Even if it is developed for blades, the CST method has a great flexibility thanks to the parameters N_1 and N_2; the choice of $N_1 = 0.5$ and $N_2 = 1$ leads to the aerofoil class, while we can generate rectangles or ellipsoids simply taking $N_1 = N_2 = 0.01$ or $N_1 = N_2 = 0.5$.

For the definition of the function $S(u)$ Kulfan [11] proposed a linear combination of the Bernstein polynomial:

$$S(u) = \sum_{i=0}^{n} a_i \cdot \binom{n}{i} u^i \ (1-u)^{n-i} \tag{3.8}$$

The relationship between the aerofoils parameters and the boundaries of $S(x)$ was highlighted by Kulfan and it is reported in equation 3.9.

$$\begin{aligned} S(0) &= \sqrt{2\, r_{\text{le}}} \\ S(1) &= \tan\beta + \Delta z \end{aligned} \tag{3.9}$$

where: r_{le} is the leading edge radius
β is the boat-tail angle
Δz is the trailing edge thickness

The advantage of the use of Bernstein polynomial as basis function is that it implies the existence of an interpolating curve for any aerofoil to any tolerance needed, the drawback however is that there is no guarantee that the number of the design variables required is suitable.

3.3 Hicks-Henne Bump functions

Hicks-Henne Bump functions belong to the so-called *deformative methods*, which describe the shape of interest perturbing a baseline geometry. Consequently the final shape is defined as the sum of two components:

- a base aerofoil definition;
- a linear combination of a set of n basis functions.

The analytical expression is reported in equation 3.10.

$$z(u) = z_{\text{baseline}}(u) + \sum_{i=0}^{n} a_i \, \phi_i(u) \tag{3.10}$$

where $\boldsymbol{\phi}_i(u)$ are the basis functions. Hicks and Henne [12] proposed to use as basis, the sin function.

$$\phi_i(u) = \sin^{t_i}(\pi u^{m_i}) \tag{3.11}$$

$$m_i = \frac{\log 0.5}{\log(u_{\max_i})} \tag{3.12}$$

where: $u_{\max_i}$ is the set of maxima of the basis function $\phi_i(u)$;
t_i is the parameter that controls the width of the function;

3.4 Parametric Section (PARSEC)

The PARSEC approach was developed by Sobieczky [13] and it consists in representing an aerofoil through a set of meaningful geometrical properties. The upper and lower surfaces are defined by the following 6^{th} order polynomials centred in the origin:

$$z_{\text{upper}}(u) = \sum_{i=1}^{6} a_i \, u^{i-0.5} \tag{3.13}$$

$$z_{\text{lower}}(u) = \sum_{i=1}^{6} b_i \, u^{i-0.5} \tag{3.14}$$

$$\tag{3.15}$$

The 12 parameters $a_i,\ b_i$ with $i = 1, \ldots, 6$ are brought back to the geometrical parameters in table 3.1.

Symbol	Parameter	Definition
$r_{\mathrm{le_u}}$	Upper leading edge radius	$r_{\mathrm{le_u}} = a_1$
$r_{\mathrm{le_l}}$	Lower leading edge radius	$r_{\mathrm{le_l}} = b_1$
u_{u}	Upper crest position	$z'_{\mathrm{u}}(u_{\mathrm{u}}) = 0$
u_{l}	Lower crest position	$z'_{\mathrm{l}}(u_{\mathrm{l}}) = 0$
z_{u}	Upper crest point	$z_{\mathrm{u}} = z_{\mathrm{u}}(u_{\mathrm{u}})$
z_{l}	Lower crest point	$z_{\mathrm{l}} = z_{\mathrm{l}}(u_{\mathrm{l}})$
c_{u}	Upper crest curvature	$c_{\mathrm{u}} = z''_{\mathrm{u}}(u_{\mathrm{u}})$
c_{l}	Lower crest curvature	$c_{\mathrm{l}} = z''_{\mathrm{l}}(u_{\mathrm{l}})$
z_{te}	Trailing edge offset	$z_{\mathrm{te}} = z_{\mathrm{l}}(1)$
Δz_{te}	Trailing edge thickness	$\Delta z_{\mathrm{te}} = z_{\mathrm{u}}(1) - z_{\mathrm{te}}$
α_{te}	Trailing edge angle	$z'_{\mathrm{u}}(1) = -\tan(\alpha_{\mathrm{te}} + 0.5\beta_{\mathrm{te}})$
β_{te}	Boat-tail angle	$z'_{\mathrm{l}}(1) = -\tan(\alpha_{\mathrm{te}} - 0.5\beta_{\mathrm{te}})$

Table 3.1: PARSEC list of geometrical parameters

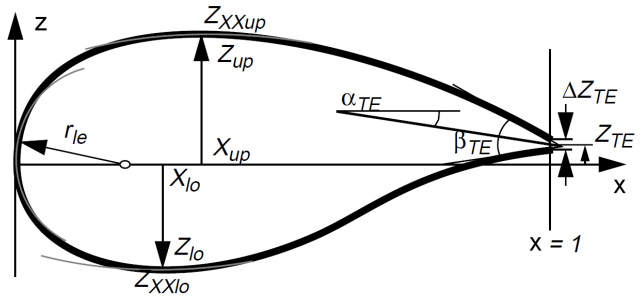


Figure 3.3: Example of a PARSEC blade.

3.5 Bezier Curves

Bezier curves are commonly applied in many fields where the model of smooth curves is required. They are build as a linear combination of a set of $n + 1$ control points $\mathbf{CP}$, and the parameter n is the degree of the curve. The most common curves are quadratic

or cubic, since higher orders increase the computational cost.

$$\mathbf{x}(t) = \sum_{i=0}^{n} B_i^n(t) \cdot \mathbf{CP}_i = [x(t),\ y(t)] \qquad \text{with } t \in [0,\ 1] \tag{3.16}$$

where: B_i^n is a Bernstein polynomial.

One of most important property of Bernstein polynomials is that they satisfy recursion in equation 3.17.

$$B_i^n(t) = (1-t)B_i^{n-1}(t) + t\, B_{i+1}^{n-1}(u) \qquad \text{with } B_0^0(t) = 0 \tag{3.17}$$

An explicit expression for the generic Bernstein polynomial can be easily derived and is reported in 3.18.

$$B_i^n(t) = \binom{n}{i} t^i\, (1-t)^{n-i} \tag{3.18}$$

Bezier Curves of first degree
Bezier curves of degree equal to 1 are simply a linear interpolation between two points.

$$\mathbf{P}(t) = (1-t)\, \mathbf{CP}_0 + t\, \mathbf{CP}_1 \tag{3.19}$$

Bezier Curves of second degree
Bezier curves of degree equal to 2 describe a parabola, which is uniquely identified given three conditions, the three control points $\mathbf{CP}_0$, $\mathbf{CP}_1$ and $\mathbf{CP}_2$.

$$\mathbf{P}_0^2(t) = (1-t)\, \mathbf{P}_0^1 + t\, \mathbf{P}_1^1 \tag{3.20}$$
$$\mathbf{P}_0^1(t) = (1-t)\, \mathbf{P}_0^0 + t\, \mathbf{P}_1^0 = (1-t)\, \mathbf{P}_0 + t\, \mathbf{P}_1 \tag{3.21}$$
$$\mathbf{P}_1^1(t) = (1-t)\, \mathbf{P}_1^0 + t\, \mathbf{P}_2^0 = (1-t)\, \mathbf{P}_1 + t\, \mathbf{P}_2 \tag{3.22}$$
$$\tag{3.23}$$

Substituting equations 3.22 and 3.23 into equation 3.21, we obtain the explicit expression for the points of the curve.

$$\mathbf{P}(t) = (1-t)^2\, \mathbf{P}_0 + 2\, t(1-t)\, \mathbf{P}_1 + t^2\, \mathbf{P}_2 \tag{3.24}$$

Bezier Curves of third degree
Similar procedure can be applied for a Bezier curve with $n = 3$, and the result is reported in equation 3.25.

$$\mathbf{P}(t) = (1-t)^3\, \mathbf{P}_0 + 3\, t(1-t)^2\, \mathbf{P}_1 + 3\, t^2(1-t)\, \mathbf{P}_2 + t^3\, \mathbf{P}_3 \tag{3.25}$$

3.5.1 De Casteljau Algorithm

By generalizing expression 3.25 we can write the generic term $\mathbf{P}_i^n(t)$ as function of two previous terms, which is an universal recursive expression for building Bezier curves.

$$\mathbf{P}_i^n(t) = (1-t)\mathbf{P}_i^{n-1}(t) + t\,\mathbf{P}_{i+1}^{n-1}(u) \qquad \text{with } \mathbf{P}_0^i(t) = \mathbf{P}_i \tag{3.26}$$

The expression is analogous to that of Bernstein polynomial, and it is useful to build Bezier curves of any degree. An alternative representation of the De Casteljau Algorithm is made by a triangular array, and below it is sketched for the cubic case.

$$\begin{array}{cccc} \mathbf{P}_0 & & & \\ \mathbf{P}_1 & \mathbf{P}_0^1 & & \\ \mathbf{P}_2 & \mathbf{P}_1^1 & \mathbf{P}_0^2 & \\ \mathbf{P}_3 & \mathbf{P}_2^1 & \mathbf{P}_1^2 & \mathbf{P}_0^3 \end{array}$$

3.5.2 Bezier Curves Properties

The key features of the Bezier curves are [14]:

Geometry invariance It means that rotating or translating the control polygon does not affect the curve other that the rotation or the translation.

Endpoint Interpolation The first and the last point of the control polygon are also the extremant of the Bezier curve. Furthermore in the endpoints the curve is tangent to the control polygon.

Symmetry The labelling of the control point does not matter for the generation of the curve, and $\mathbf{CP}_0,\ \mathbf{CP}_1,\ \ldots\ ,\mathbf{CP}_n$ is the same as $\mathbf{CP}_n,\ \mathbf{CP}_{n-1},\ \ldots\ ,\mathbf{CP}_0$. This can be written in formula as:

$$\sum_{i=0}^{n} B_i^n(t)\cdot\mathbf{CP}_i = \sum_{i=0}^{n} B_i^n(1-t)\cdot\mathbf{CP}_{n-i} \tag{3.27}$$

Invariance under barycentric combinations The weighted average of two curve can be taken directly from the points of the curve or just take the weighted average of the control polygon and then computing the curve. In formula this is equivalent to the linearity property, meaning that given α and β so that $\alpha+\beta=1$ the following equation holds:

$$\sum_{i=0}^{n} B_i^n(t)\cdot(\alpha\mathbf{P}_i + \beta\mathbf{Q}_i) = \alpha\sum_{i=0}^{n} B_i^n(t)\cdot\mathbf{P}_i + \beta\sum_{i=0}^{n} B_i^n(t)\cdot\mathbf{Q}_i \tag{3.28}$$

Linear precision The Bezier curve is able to correctly reproduce a straigh line between two points. This follows from the identity in 3.29.

$$\sum_{i=0}^{n} \frac{i}{n} B_i^n(t) = t \tag{3.29}$$

Convex hull property The entire curve is contained inside the convex hull of the set of the control points.

Variation diminishing property For a 2D Bezier curve, the number of intersections of a straight line with the curve is no greater than the number of intersections of the line with the control polygon.

Pseudolocal control The curve is mainly affected by change in control points close to its region. This is due to the fact that the Bernstein polynomials $B_i^n(t)$ just have a single maximum at $t = i/n$. The effect of the change in a control point are in general reasonably predictable. The not fully local controllability have driven us to the choice of B-splines as parametrization technique for the blades in the current work.

3.6 B-spline Curves

The features of reduced number of variables and local controllability besides relative implementation simplicity can be obtained with B-spline curve.

B-splines are defined as sequence of polynomial curve segments of degree n that provide local support; they are linear combination of polynomial functions weighted by coefficients represented by control points [14].

To characterize a B-spline we need several elements:

- an interval I = $[a,\ b]$ over the B-spline is defined;
- the degree n of the polynomials;
- a set of $N + 1$ control points $\mathbf{CP}_i$ with $i \in [0, N]$;
- a knot sequence $\mathbf{u} = [u_0 = a, \ldots, u_i, \ldots, u_m = b]$ with $m = (N + 1) + n$

A great advantage of B-splinesis that, given a polynomial degree of n, if a knot has multiplicity r the curve in the knot belongs to C^{n-r} [14], so we are able to control both continuity and primarily smoothness.

In general, if knots multiplicity r is smaller than the degree n the B-splinedoes not pass through the control points. We can force the passage by increasing the multiplicity of a knot to n.

In this work we will just fix the first and the last points, leaving all the others knots with singular multiplicity.

To clarify the effect of some of the characteristic parameters of the B-splines we will show examples of how that parameters affect the curve.

Effect of the degree

To obtain the figure 3.4 we have used 7 control points and we have generated five different curves changing the degree of the B-spline.

As we can appreciate in figure 3.4, a B-spline whose bases are polynomials of order 1 simply means to perform a piecewise linear interpolation between the control points. When we increase the order n to 2, the curve does not pass anymore through the control points, but it is tangent to the segments that links the control points together.
Increasing n we can notice that the 'smoothness' of the curve progressively increases. In fact we have previously stated that a B-spline of degree n belongs to C^{n-1}, assuming that the control points have singular multiplicity, which is the case of the current example.

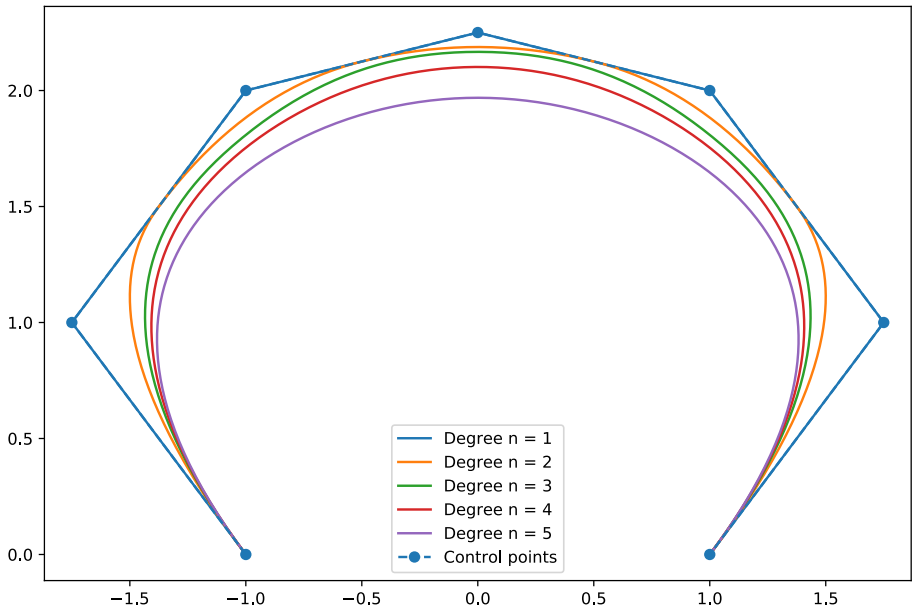


Figure 3.4: B-spline varying the degree n.

Effect of control points
As already stated, one great advantage of B-splines is the local controllability. This property is highlighted in figure 3.5 where changing a single control point, will result in a local variation of the curve with no effect on points far from the moved **CP**.

This property is particularly useful in this work, since let us to interpolate in a proper way both almost straight parts of the blade, like the center and close to the trailing edge, and regions with an higher gradient close to the leading edge.

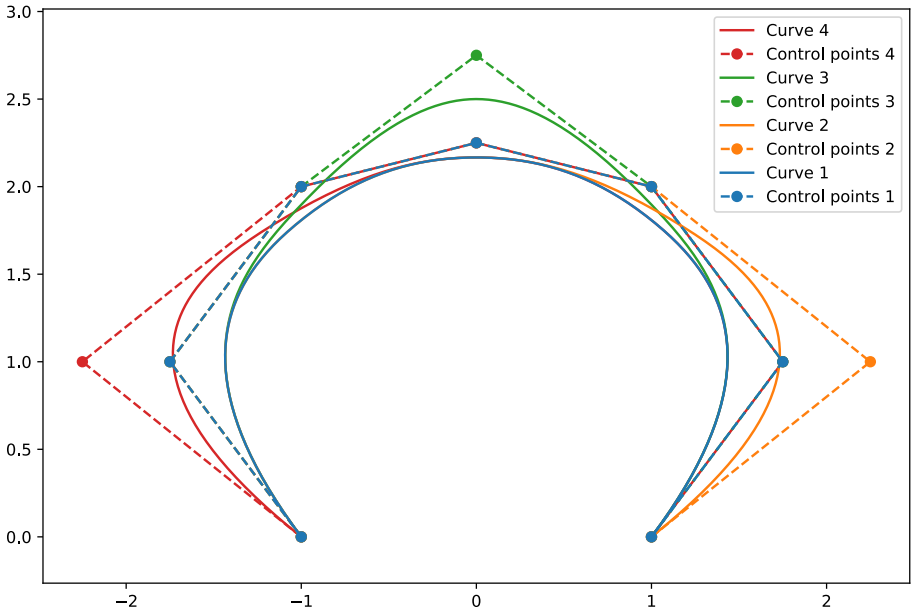


Figure 3.5: B-spline moving a control point.

Effect of knots array

Up to know we have just made use of uniformly spaced knots, except for the firsts and the lasts which have multiplicity greater than 1 to make the curve passing through first and last control points. The knots can manipulate the curve to make it passes through a control point (curve 2 in figure 3.6) or make it tangent to a segment of the control polygon (curves 3 and 4 in figure 3.6).
Even without increasing knot multiplicity the curve can be a lot manipulated to reproduce the required design, shifting it closer to control point and further to another.

The knots arrays of figure 3.6 are shown in table 3.2.

Curve 1	[0, 0, 0, 0, 0.25, 0.50, 0.75, 1, 1, 1, 1]
Curve 2	[0, 0, 0, 0, 0.50, 0.50, 0.50, 1, 1, 1, 1]
Curve 3	[0, 0, 0, 0, 0.20, 0.20, 0.80, 1, 1, 1, 1]
Curve 4	[0, 0, 0, 0, 0.20, 0.80, 0.80, 1, 1, 1, 1]

Table 3.2: Testing cases for bsplines in figure 3.6.

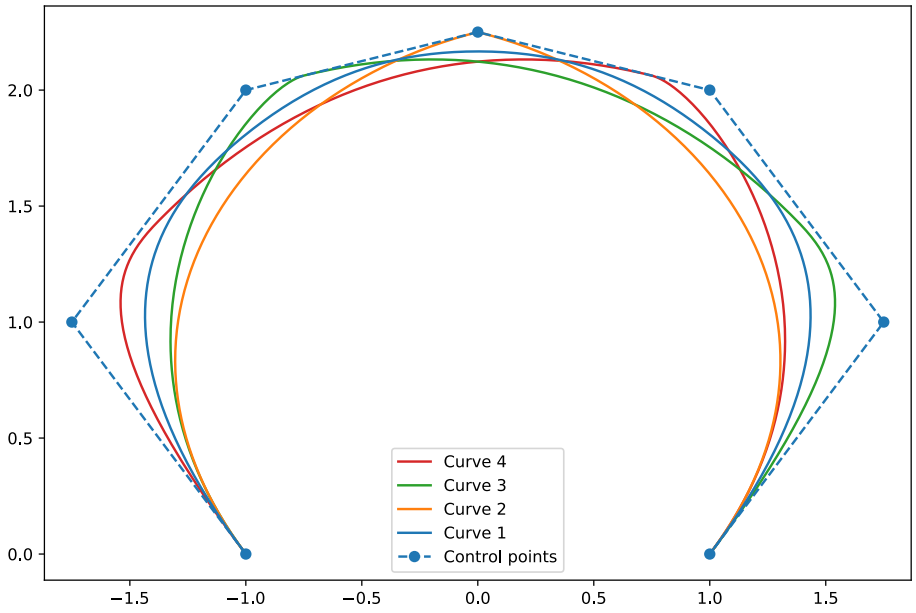


Figure 3.6: B-spline with different knots arrays.

3.6.1 B-splines Formulation

Given the elements previously defined, a B-splinecan be described as the weighted average between all their basis:

$$\mathbf{x}(u) = \sum_{i=0}^{N} N_i^n(u) \cdot \mathbf{CP}_i = [x(u),\ y(u)] \tag{3.30}$$

where the basis $N_i^n(u)$ are polynomials of degree n (Bezier curves), and the weighting elements are the control points $\mathbf{CP}_i$.

With the De Casteljau algorithm previously explained in chapter 3.5.1 the basis function can be recursively written as:

$$N_i^n(u) = \frac{u - u_{i-1}}{u_{n+i-1} - u_{i-1}} N_i^{n-1}(u) + \frac{u_{n+1} - u}{u_{n+i} - u_i} N_{i+1}^{n-1}(u) \tag{3.31}$$

This is valid while n reaches 0; in that case actually the term $N_i^{-1}(u)$ would have been required but it is not available so we have to define $N_i^0(u)$, the so-called *anchor* of the recursion, as equation 7.70.

$$N_i^0(u) = \begin{cases} 1 & \text{if } u_{i-1} \leq u < u_i \\ 0 & \text{if } u < u_{i-1} \wedge u \geq u_i \end{cases} \tag{3.32}$$

As we can notice in equations 3.31 and 7.70, we will compute for the whole interval I, all the basis but for most of the values, when $u < u_{i-1} \wedge u \geq u_i$ for each interval we perform useless calculation that will be multiplied by 0.

De Boor Algorithm

The De Boor algorithm is the generalization of the Casteljau for bezier curves for the B-splines.

Instead of computing directly $N_i^n(u)$ we evaluate each base through a simpler recursion formula.

$$\mathbf{d}_i^r = (1 - \alpha_i^r) \cdot \mathbf{d}_{i-1}^{r-1} + \alpha_i^r \cdot \mathbf{d}_i^{r-1} \tag{3.33}$$

$$\alpha_i^r = \frac{u - u_i}{u_{i+1-r+n} - u_i} \tag{3.34}$$

where: $k =$ index of the interval $[u_k,\, u_{k+1})$ which contains u;

$r = 1,\ \ldots,\ n$;

$i = k - n + r,\ \ldots,\ k$.

As before we also have to define the first item of the iteration, that in this case are $\mathbf{d}_i^0$.

$$\mathbf{d}_i^0 = \mathbf{CP}_i \tag{3.35}$$

Finally the result we are interested in is the last term of the iterations.

$$\mathbf{x}(u) = \mathbf{d}_k^n \tag{3.36}$$

In terms of performances the De Boor algorithm has a great advantage respect to equation 3.31 since it avoids to compute all the terms that does not compete to the results since they are multiplied by zero. We can also improve the memory consumption iterating backward, obtaining the final version of the De Boor algorithm.

$$\mathbf{d}_j = (1 - \alpha_j) \cdot \mathbf{d}_{j-1} + \alpha_j \cdot \mathbf{d}_j \tag{3.37}$$

$$\alpha_j = \frac{u - u_{j+k-n}}{u_{j+k-r+1} - u_{j+k-n}} \tag{3.38}$$

where: $k =$ index of the interval $[u_k,\, u_{k+1})$ which contains u;

$r = 1,\ \ldots,\ n$;

$j = n,\ \ldots,\ r$ which is in backward direction.

Finally $\mathbf{x}(u) = \mathbf{d}_n$.

So given a knot array $\mathbf{u}$, a list of control points $\mathbf{CP}$, the degree of the B-spline n and a list of values $u \in [u_0,\ u_m]$ in which evaluate the B-spline the procedure is shown in figure 3.7.

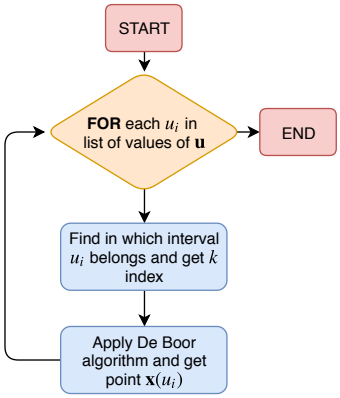


Figure 3.7: De Boor procedure.

3.6.2 B-splines Interpolation

As we have seen in previous sections, once we have the control points, the order and the degree, the process of building a B-spline is not trivial, but it is relatively straightforward.

An interpolation is required to obtain the B-spline curve that corresponds to the reference blade that we are going to modify in the optimization phase.

As we have notice in figure 3.4 for the degree higher than 2, the controls points are not directly related to the control points in the terms that the points of the curve nor passes through the control points or are tangent to the segments. So the control points that we need to define the B-spline are not the points that we want to interpolate.

The parameters of the problem (both known and unknown) are:

- $P+1$ points $\mathbf{Y}_i$ that we want to interpolate with $i = [0,\ P]$;
- the degree n of the polynomials;
- a set of $N+1$ control points $\mathbf{CP}_i$ with $i \in [0, N]$;
- a knot sequence $\mathbf{w} = [w_0,\ \ldots,\ w_m]$ with $m = (N+1) + n$

In general the number of points to interpolate are much bigger than the number of control points that we decide to use.
The problem in this way does not have an exact solution but we will minimize the error between the points and the curve.

We can write this error as the norm two of the difference of each point.

$$\varepsilon_j = \|\mathbf{Y}_j - \mathbf{x}(w_j)\| \tag{3.39}$$

$$\varepsilon = \sum_{j=0}^{P} \|\mathbf{Y}_j - \mathbf{x}(w_j)\| \tag{3.40}$$

Remembering equation 3.30 the error ε becomes:

$$\varepsilon = \sum_{j=0}^{P} \left\| \mathbf{Y}_j - \sum_{i=0}^{N} N_i^n(w_j) \cdot \mathbf{CP}_i \right\| \tag{3.41}$$

Minimizing ε can be solved with a least square approach that brings to the solution of a linear system of equations, where the unknowns are the coordinates of the control points.

$$\min \varepsilon \quad \Rightarrow \quad \sum_{i=0}^{N} \mathbf{CP}_i \cdot \sum_{j=0}^{P} N_i^n(w_j) N_k^n(w_j) = \sum_{j=0}^{P} \mathbf{Y}_j \cdot N_k^n(w_j) \qquad \forall k \tag{3.42}$$

This set of equations can be written in matricial form as:

$$\begin{bmatrix} m_{0,0} & m_{0,1} & \dots & m_{0,N} \\ m_{1,0} & m_{1,1} & \dots & m_{1,N} \\ \vdots & \vdots & \ddots & \vdots \\ m_{N,0} & m_{N,1} & \dots & m_{N,N} \end{bmatrix} \begin{bmatrix} \mathbf{CP}_0 \\ \mathbf{CP}_1 \\ \vdots \\ \mathbf{CP}_N \end{bmatrix} = \begin{bmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_N \end{bmatrix} \tag{3.43}$$

This equation can be rewritten in a very compact form as:

$$\boldsymbol{M} \cdot \mathbf{CP} = \mathbf{b} \tag{3.44}$$

The generic term of matrix in equation 3.43 is:

$$m_{j,k} = \sum_{j=0}^{P} N_i^n(w_j) N_k^n(w_j) \tag{3.45}$$

The matrix $\boldsymbol{M}$ is in general symmetric and ill conditioned, so it is necessary to apply specific solver such as the Cholesky decomposition [14].

The Cholesky decomposition is an algorithm that let to write a symmetric positive definite matrix $\boldsymbol{M}$ as function of a lower triangular matrix $\boldsymbol{L}$. This kind of decomposition for such a matrix $\boldsymbol{M}$ is unique.

$$\boldsymbol{M} = \boldsymbol{L}\boldsymbol{L}^* \tag{3.46}$$

where: $\boldsymbol{L}$ = is a lower triangular matrix
$\boldsymbol{L}^*$ = is the conjugate transpose of $\boldsymbol{L}$

Linear system 3.44 can now be rewritten as:

$$\boldsymbol{M} \cdot \mathbf{CP} = \mathbf{b} \quad \Rightarrow \quad \boldsymbol{L}\boldsymbol{L}^* \cdot \mathbf{CP} = \mathbf{b} \tag{3.47}$$

This procedure makes the solution of the system quite trivial since we reduced the

problem to just solve two triangular system of equations; actually calling $\boldsymbol{L}^* \cdot \mathbf{CP} = \mathbf{z}$ the system becomes:

$$\boldsymbol{L}\mathbf{z} = \mathbf{b} \quad \Rightarrow \quad \mathbf{z} \tag{3.48}$$

$$\boldsymbol{L}^*\mathbf{CP} = \mathbf{z} \quad \Rightarrow \quad \mathbf{CP} \tag{3.49}$$

Before really solving the two systems it is necessary to threat the first and the last points. We have decided in fact to make the curve passing through both first and last points, so we already know two of the unknowns. We have to correct the system to include this consideration.

$$\widetilde{\mathbf{b}}_\mathbf{i} = \mathbf{b}_\mathbf{i} - m_{i,0}\,\mathbf{CP}_0 - m_{i,N}\,\mathbf{CP}_N \tag{3.50}$$

Then all the terms in first and last columns and rows are 0 except that $m_{0,\,0}$ and $m_{N,\,N}$ that are equal to 1. Finally the system of equations becomes:

$$\begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & m_{1,1} & \dots & m_{1,N-1} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & m_{N-1,1} & \dots & m_{N-1,N-1} & 0 \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{CP}_0 \\ \mathbf{CP}_1 \\ \vdots \\ \mathbf{CP}_{N-1} \\ \mathbf{CP}_N \end{bmatrix} = \begin{bmatrix} \mathbf{CP}_0 \\ \widetilde{\mathbf{b}}_1 \\ \vdots \\ \widetilde{\mathbf{b}}_{N-1} \\ \mathbf{CP}_N \end{bmatrix} \tag{3.51}$$

After solving equation 3.51 we finally obtain the coordinates of the control points.

3.6.3 Knot sequence

Up to now we have assumed to already have a sequence of knots w_i but we have not specified how to generate it.

And the most critical aspect in the interpolation is to find the best possible list of knots so that the B-spline curve would fit the data points. In figure 3.6 we have appreciated how the resulting B-spline changes just modifying the knots.

An completely analytical formulation for the optimization of the knots it is in general not possible unlike the choice of control points. This kind of problem is in fact *non-linear* and with several constraints.

An heuristic solution is theoretically possible (at least find a local optimum) but it would be quite expensive since we have $N - n - 1$ unknowns given $N + 1$ control points and fixing the first and last points.

So in literature have been found different ways of generating the knots or a priori or depending on the available points.

The centripetal parametrization proposed by Lee 1989 [15] is in general suitable for cases where sharp turns are present.

According to this approach we compute w_i with equation 3.52 after defining $w_0 = 0$.

$$w_i = w_{i-1} + \frac{\sqrt{\|\mathbf{Y}_i - \mathbf{Y}_{i-1}\|}}{\sum_{i=0}^{P} \sqrt{\|\mathbf{Y}_i - \mathbf{Y}_{i-1}\|}} \tag{3.52}$$

For the current profile a different approach has been applied. The knot sequence has been constructed relying on the rational parametrization. The interval $[0,\ 1]$ has been split in two sections in a center point c obtaining $[0,\ c)$ and $(c,\ 1]$, then the points are distributed inside the first interval with the rule:

$$w_i = w_{i-1} + w_1\, q^{i-1} \tag{3.53}$$

So for example w_3 is:

$$w_3 = w_1 + w_1\, q + w_1\, q^2 = w_1 \cdot \left(1 + q + q^2\right) \tag{3.54}$$

If the number of knot points is odd we have a point exactly in c otherwise we split the mid interval in two. For the odd case, we have to distribute $(m + 1 - 1)/2$ points in each sub-interval. But each interval contains $n + 1$ knots which are fixed to 0 or to 1 due to fixed first and last points.

Remembering that the summation of a geometric series of ratio q is:

$$1 + q + q^2 + q^3 + \ \ldots\ + q^n = \frac{1 - q^{n+1}}{1 - q} \tag{3.55}$$

We can now compute the first term w_1 for the interval 1 and for interval 2.

$$w_1^{(1)} \cdot \left(1 + q + q^2 + q^3 + \ \ldots\ + q^n\right) = w_1^{(1)}\, \frac{1 - q^{n+1}}{1 - q} = c \tag{3.56}$$

$$w_1^{(2)} \cdot \left(1 + q + q^2 + q^3 + \ \ldots\ + q^n\right) = w_1^{(2)}\, \frac{1 - q^{n+1}}{1 - q} = 1 - c \tag{3.57}$$

And from w_1, with equation 3.53 we compute all w_i terms.

An advantage of this approach is that we can concentrate more control points close to the blade leading edge, where the curvature is greater and the reconstruction more critical.

Moreover we have reduced the number of unknowns from tens to 2.
With just two variables it is also possible to perform an optimization to find best knots as possible. The optimization is performed minimizing the error in the reconstruction of the profile with the tools integrate in the optimization tool in the Scipy library of Python. We use the fmin function that applies the *downhill simplex algorithm* which does not requires informations on derivatives which are not available.

To show why we have decided to apply in this case the rational parametrization, we will apply both techniques to a profile of our blade and we will compare the results.

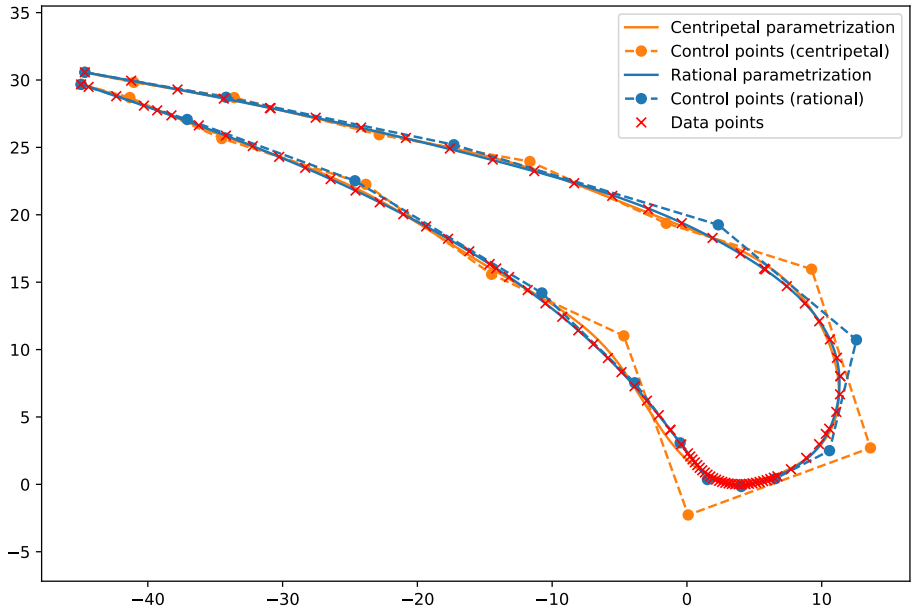


Figure 3.8: Comparison of the final interpolation between centripetal and radial parametrization.

Just comparing the results in figure 3.8 the two interpolated curve seem pretty close together and also close to the available data.

But if we also compare the distances between the interpolated curve and the data points, point by point, in figure 3.9 it is clear the the rational parametrization guarantees a much more accurate result.

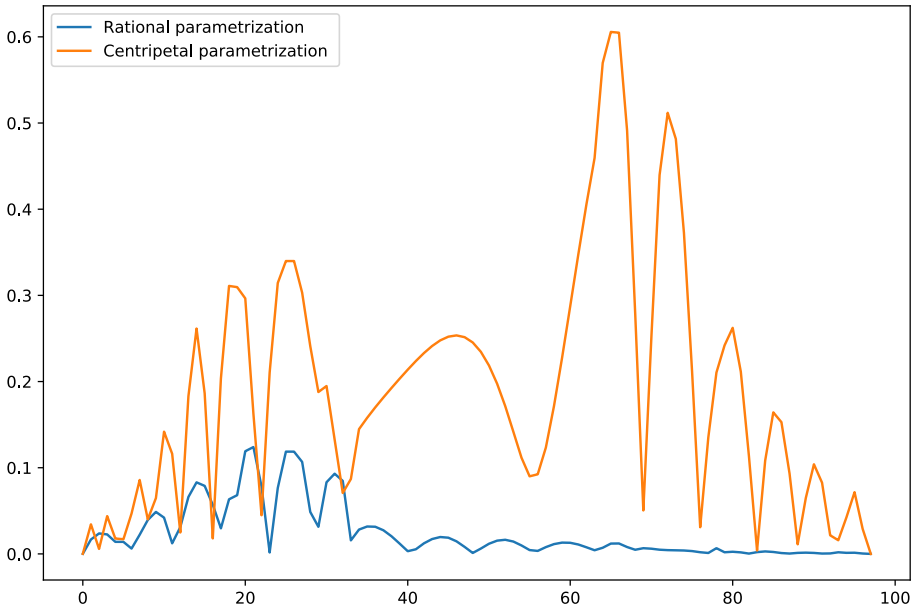


Figure 3.9: Point by point error comparison between centripetal and radial parametrization.

3.7 B-spline modification

In the previous section we have explained how we have interpolated the baseline profile, which is the starting point of the optimization process. To explore the design space of the profile however it is necessary to slightly modify the profile and check if it performs better or worst.

In the process of the generation of the B-spline we will obtain 1000 points for describing the curve. It is clearly impossible to manage such a number of points, so we have decided to run the optimization moving the control points of the baseline and then recompute the 1000 interpolating points considering a new B-spline with same knot array and degree of the original but with the moved control points.

Since we are dealing with a plane parametrization, then, the modification of the control points have to be done both in the x and in the y directions. To reduce the complexity, instead of duplicating the number of design variables, we have decided to move the points along the direction normal to the curve. So, given the tangent m of the curve in a control points and the amount of which to move the point d, the new control point $\mathbf{Q}$ will be:

$$\begin{aligned} x_{\mathbf{Q}} &= x_{\mathbf{CP}} + d \cdot \cos\left(\operatorname{atan}\frac{1}{m}\right) \\ y_{\mathbf{Q}} &= y_{\mathbf{CP}} - d \cdot \sin\left(\operatorname{atan}\frac{1}{m}\right) \end{aligned} \tag{3.58}$$

The calculation of the tangent requires to find the derivative of a B-spline; being linear combination of polynomial basis, one advantage of the use of B-spline curves is the easiness of calculating its derivatives. The procedure is explained in section 3.7.1.
In figure 3.10 an example of the resultant design space of a blade is represented.

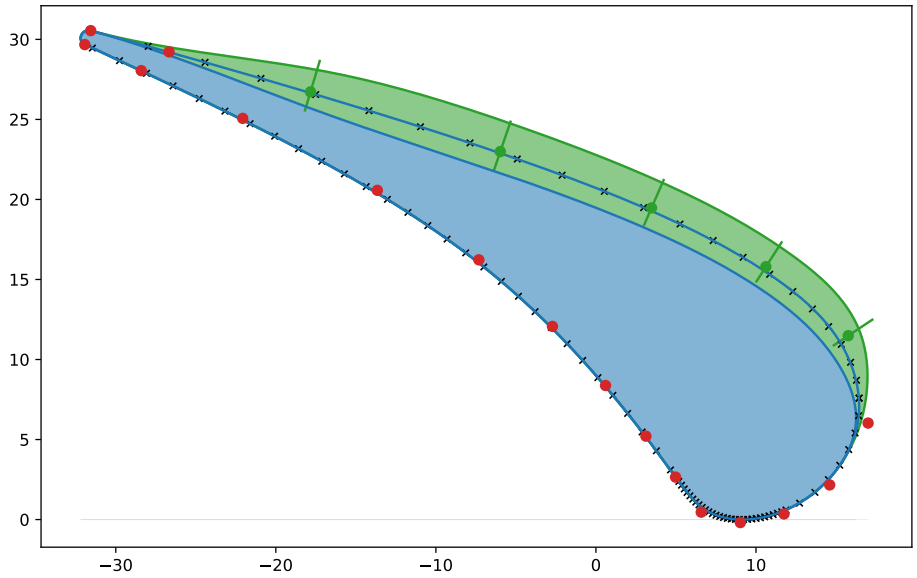


Figure 3.10: Example of a blade with design space.

3.7.1 Derivative of a B-spline

As previously stated a B-spline is a piecewise polynomial curve.
Since each section is a polynomial of degree n we are able to compute its derivative.

$$\mathbf{x}(u) = \sum_{i=0}^{N} N_i^n(u) \cdot \mathbf{CP}_i \quad \Rightarrow \quad \frac{\mathrm{d}}{\mathrm{d}x}\mathbf{x}(u) = \sum_{i=0}^{N} \frac{\mathrm{d}}{\mathrm{d}x} N_i^n(u) \cdot \mathbf{CP}_i \tag{3.59}$$

So it is required to evaluate the derivative of the term $N_i^n(u)$.

$$\frac{\mathrm{d}}{\mathrm{d}x} N_i^n(u) = \frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{u - u_{i-1}}{u_{n+i-1} - u_{i-1}} N_i^{n-1}(u) + \frac{u_{n+1} - u}{u_{n+i} - u_i} N_{i+1}^{n-1}(u)\right) \tag{3.60}$$

$$= \frac{n}{u_{n+i-1} - u_{i-1}} N_i^{n-1}(u) - \frac{n}{u_{n+i} - u_i} N_{i+1}^{n-1}(u) \tag{3.61}$$

Plugging equation 3.61 into equation 3.59 we will obtain:

$$\frac{\mathrm{d}}{\mathrm{d}x}\mathbf{x}(u) = \sum_{i=0}^{N}\left(\frac{n}{u_{n+i-1} - u_{i-1}} N_i^{n-1}(u) - \frac{n}{u_{n+i} - u_i} N_{i+1}^{n-1}(u)\right) \cdot \mathbf{CP}_i \tag{3.62}$$

$$= \sum_{i=0}^{N-1} N_i^{n-1}(u) \cdot \mathbf{Q}_i \tag{3.63}$$

The really interesting results of equation 3.63 is that computing the derivative of a B-spline will result in a new B-spline of degree $n-1$ with the same knot array but different control points $\mathbf{Q}_i$, where $\mathbf{Q}_i$ can be found in equation 3.64.

$$\mathbf{Q}_i = \frac{n}{u_{n+i} - u_i}\left(\mathbf{CP}_{i+1} - \mathbf{CP}_i\right) \tag{3.64}$$

With the tools developed in the previous section we are able to manage the derivative of a B-spline in each point, where the new curve has:

- a degree $n-1$;
- the same knots sequence except the first and the last;
- a set of N control point $\mathbf{Q}_i$ with $i = 0, \ldots, N-1$.

When we have evaluated the derivative in a point, we are able to compute the slope of the tangent.

$$m = -\frac{1}{\mathbf{x}'(u)} \qquad q = y_P - m\, x_P \tag{3.65}$$

$$y = m\, x + q = y_P - \frac{x - xp}{\mathbf{x}'(u)} \qquad \text{(normal line equation)} \tag{3.66}$$

In this way, each optimization variable represents how much the correspondent control point is moving along the direction normal to the curve.
If the value is positive, the direction is considered to go outside of the blade, while if negative inside.

3.8 Trailing edge management

Interpolating also the trailing edge with the B-spline tool will probably require much more control points and a proper design of the knot sequence that has to manage with a higher density of control points both the leading and the trailing edge. To simplify an already really massive optimization problem has been decided to model the trailing edge regardless of the B-spline, with a specific and more simple curve. In the following paragraphs two alternatives are presented:

- a section of a *circle*;
- a section of an *ellipse*.

3.8.1 Circular trailing edge

A circle can be represented by 3 parameters:

- the center $(c_x,\ c_y)$;
- the radius r.

The problem rises since we have 4 constraints:

- the passage through the first point of the B-spline;
- the passage through the last point of the B-spline;
- tangent to the first point of the B-spline;
- tangent to the last point of the B-spline;

Since an over-constrained problem like this is in general impossible to solve, we have decided to keep the passage through both point, but with just one tangent.

To find all the three unknown we have to find three equations.
First at all we remember the expression of a generic circumference in equation 3.67

$$(x - x_c)^2 + (y - y_c)^2 = r^2 \quad \text{or} \quad x^2 + y^2 + a\,x + b\,y + c = 0 \tag{3.67}$$

Given equation 3.67 the passage constraints become:

$$x_1^2 + y_1^2 + a\,x_1 + b\,y_1 + c = 0 \tag{3.68}$$
$$x_1^2 + y_1^2 + a\,x_1 + b\,y_1 + c = 0 \tag{3.69}$$

The third condition is a little bit trickier and require the derivative of an implicit curve $C(x, y) = 0$.

$$\left.\frac{\mathrm{d}y}{\mathrm{d}x}\right|_{(x_p,y_p)} = -\frac{\partial C(x,y)/\partial x|_{(x_p,y_p)}}{\partial C(x,y)/\partial y|_{(x_p,y_p)}} = -\frac{2\,x_p + a}{2\,y_p + b} \tag{3.70}$$

Then, depending on the choice of from which point we take the tangent, the third constraint becomes:

$$-\frac{2\,x_1 + a}{2\,y_1 + b} = m_1 \quad \text{or} \quad -\frac{2\,x_2 + a}{2\,y_2 + b} = m_2 \tag{3.71}$$

where: m_1 = derivative of the B-spline in first point;
m_2 = derivative of the B-spline in last point.

Alternatively we can solve the circumference problem with a slightly different approach, applying the rules of analytic geometry.
The center of the circle is actually the intersection between:

- the axis of the segment $\overline{P_1 P_2}$
- the normal to the tangent (m_1 or m_2) passing through the corresponding point (P_1 or P_2)

$$\begin{cases} y = & \dfrac{y_1 + y_2}{2} + \dfrac{1}{2}\,\dfrac{x_2 - x_1}{y_2 - y_1}\,(x_1 + x_2 - 2\,x) \\ y = & y_1 - \dfrac{x - x_1}{m_1} \qquad\qquad (\text{or } x_2,\ y_2,\ m_2) \end{cases} \tag{3.72}$$

Solving equation 3.72 we find the coordinates of the center x_c, y_c. Given the center and a point, the distance between the two is simply the radius r.

$$r = \sqrt{(x_c - x_1)^2 + (y_c - y_1)^2} \tag{3.73}$$

The geometrical representation is sketched in figure

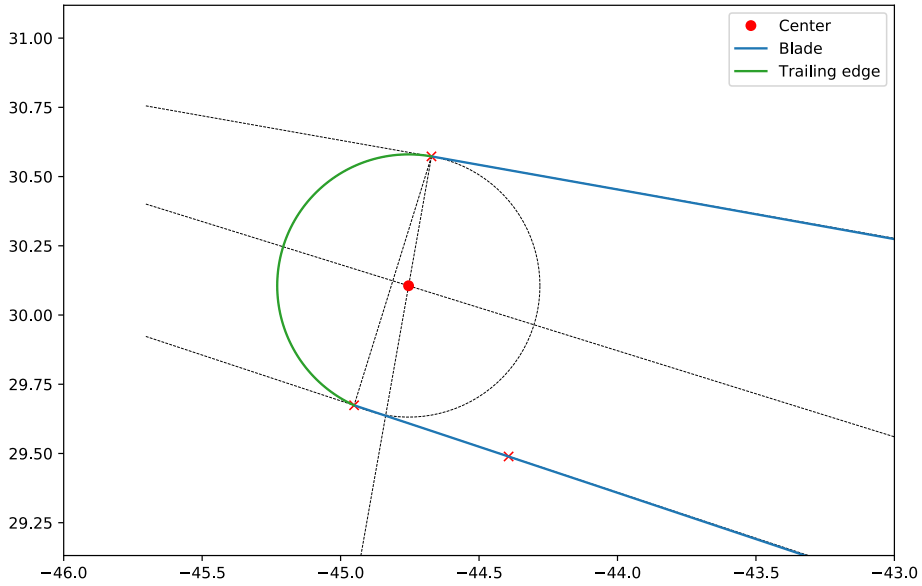


Figure 3.11: Construction of the circumpherence with a geometrical approach.

3.8.2 Ellipse trailing edge

An ellipse as trailing edge is able to overcome the problem that arises with the circle. Actually a generic ellipse in a 2D plane can be represented by equation 3.74

$$x^2 + Bxy + Cy^2 + Dx + Ey + F = 0 \tag{3.74}$$

The 5 parameters can be rewritten in terms of the geometrical interpretation as:

- the center $(c_x,\ c_y)$;
- the major semi-axis a;
- the minor semi-axis b;
- the rotation angle θ respect to the x axis.

As previously stated the intersection between the trailing edge and the B-splineshape has 4 constraints, the passage through the two points and the tangent to that points. The situation is the opposite respect to the circle case, indeed we now have one free variable to set to fulfil the parameters that define the ellipse.
The solutions can be really different respect to that free variable, like sketched in figure 3.12.

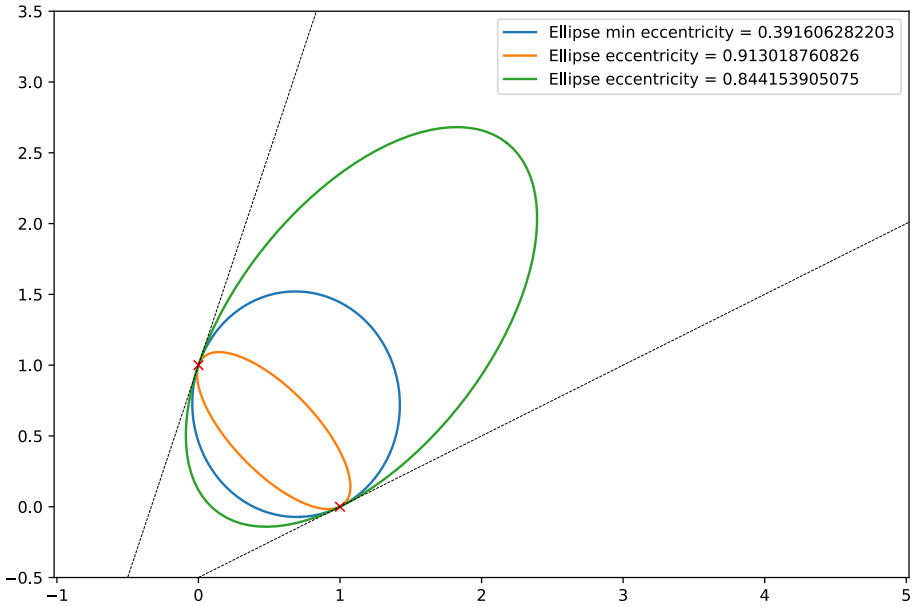


Figure 3.12: Ellipses of different eccentricity.

The strategy that we decided to apply to solve this problem is to find the ellipse which is as round as possible, meaning that from the infinite space of ellipses that satisfy our constraints we will choose the one that is closest to a circle.
The mathematical meaning of this statement is that we take the ellipse with minimum eccentricity, where the eccentricity of an ellipse represented in canonical form [1] is defined as:

$$e = \sqrt{1 - \frac{b^2}{a^2}} \tag{3.75}$$

Having defined our aim now we need to find an expression for such an ellipse.

The general procedure is quite cumbersome from the point of view of the mathematics, but this can be a lot simplified with a smart choice of the frame of reference.
We assume that the 2 points are in the form $P_1 = (\alpha, 0)$ and $P_2 = (-\alpha, 0)$, and the corresponding tangents m_1 and m_2 are in this frame of reference. Applying the 4 constraints we obtain the following system of equations:

$$\begin{cases} \alpha^2 + D\alpha + F = 0 \\ \alpha^2 - D\alpha + F = 0 \\ +B\alpha + E = -\frac{1}{m_1}\,(2\,\alpha + D) \\ -B\alpha + E = -\frac{1}{m_2}\,(-2\,\alpha + D) \end{cases} \tag{3.76}$$

It is interesting to notice that in this frame of reference the system has 4 unknowns

[1] The canonical form of an ellipse is when its principal axis are oriented as x and y axis, and its center is the origin of axis: $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$

B, D, E and F in 4 equations, which brings to the solution:

$$B = -\frac{m_1 + m_2}{m_1\, m_2} \qquad D = 0 \qquad E = \alpha\, \frac{m_1 - m_2}{m_1\, m_2} \qquad F = -\alpha^2 \tag{3.77}$$

In this way we have isolated the effect of the eccentricity just into the variable C, which has remained as the only unknown.
We can determine C minimizing the eccentricity. First at all is required to write the eccentricity of a conic section as a function of A, B, C, D, E and F; the expression is reported in equation 3.78.

$$e^2 = \frac{2\sqrt{B^2 + (A-C)^2}}{A + C + \sqrt{B^2 + (A-C)^2}} \tag{3.78}$$

Since minimizing a positive function is the same as minimizing its square, we will derive e^2 instead of e.
The derivative of e^2 respect to the unknown C values:

$$\frac{\partial e^2}{\partial C} = \frac{2\left(-(A-C)\left(A + C + \sqrt{K}\right) + \sqrt{K}\left(A - C - \sqrt{K}\right)\right)}{\sqrt{K}\left(A + C + \sqrt{K}\right)^2} \tag{3.79}$$

where: $K = B^2 + (A-C)^2$.

The derivative evaluated in equation 3.79 is zero for the value of C equal to:

$$C = A + \frac{B^2}{2\,A} = 1 + \frac{(m_1 + m_2)^2}{2\, m_1^2\, m_2^2} \tag{3.80}$$

The final result is obtained replacing equations 3.77 into C.

Finally the equation of the desired ellipse is quite astonishing for its simplicity and it is written in equation 3.81.

$$x^2 - \frac{m_1 + m_2}{m_1\, m_2}\, xy + \left(1 + \frac{(m_1 + m_2)^2}{2\, m_1^2\, m_2^2}\right) y^2 + \alpha\, \frac{m_1 - m_2}{m_1\, m_2}\, y - \alpha^2 = 0 \tag{3.81}$$

Up to now we have always considered to have the points in a specific frame of reference, which in general is not the case. So we have to expect to bring the points and the tangents in that frame, compute the ellipse of minimum eccentricity and than translate and rotate back to the original reference.

$$\alpha = \sqrt{(x_{P_1} - x_{P_2})^2 + (y_{P_1} - y_{P_2})^2} \tag{3.82}$$

$$m_1 = \frac{m_1' \cos\theta + \sin\theta}{\cos\theta - m_1' \sin\theta} \tag{3.83}$$

$$m_2 = \frac{m_2' \cos\theta + \sin\theta}{\cos\theta - m_2' \sin\theta} \tag{3.84}$$

where: θ is the angle of the two points P_1 and P_2;
m_1' is is the slope of the tangent in P_1 in the original frame;
m_2' is is the slope of the tangent in P_2 in the original frame;

Once we compute the ellipse we have to rotate it of an angle $-\theta$ and translate it back of a vector which is the mid point of the segment $\overline{P_1P_2}$.

Translating a conic section of a vector (u, v) mean modifying the terms D, E and F as:

$$D \to D - 2Au - Bv \tag{3.85}$$
$$E \to E - 2Cv - Bu \tag{3.86}$$
$$F \to Au^2 + Buv + Cv^2 - Du - Ev + F \tag{3.87}$$

Rotating a conic section of an angle α can be expressed as products between the rotation matrix and the matrix of the conic.

$$\boldsymbol{M'} = \boldsymbol{R}^{\mathbf{T}}(\alpha)\boldsymbol{M}\boldsymbol{R}(\alpha) \tag{3.88}$$

Where the matrices $\boldsymbol{M}$ and $\boldsymbol{R}$ are:

$$\boldsymbol{M} = \begin{bmatrix} A & B/2 & D/2 \\ B/2 & C & E/2 \\ D/2 & E/2 & F \end{bmatrix} \qquad \boldsymbol{R} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.89}$$

To draw the ellipse we will apply the parametric equations for x and y as function of an angle $\alpha \in [0, 2\,\pi)$.

$$x(\alpha) = a\cos\theta\cos\alpha - b\sin\theta\sin\alpha + c_x \tag{3.90}$$
$$y(\alpha) = a\sin\theta\cos\alpha + b\cos\theta\sin\alpha + c_y \tag{3.91}$$

From equations 3.91 it is clear that we need to find the 5 geometric parameters that define the ellipse.

$$c_x = \frac{2CD - BE}{B^2 - 4AC} \tag{3.92}$$

$$c_y = \frac{2AE - BD}{B^2 - 4AC} \tag{3.93}$$

$$\theta = \tan^{-1}\left(\frac{-A + C - \sqrt{B^2 + (A - C)^2}}{B}\right) \tag{3.94}$$

We just miss the major and minor semi-axis a and b. The explicit formula to compute

them from A, B, C, D, E and F exists but it is really complex, so we compute the canonical form of the ellipse translating to the origin and rotating it of an angle θ. After the roto-translation we get expression 3.95 and from that it is straightforward to compute the semi-axis with equations 3.96.

$$Ax^2 + Cy^2 + F = 0 \tag{3.95}$$

$$a = \sqrt{\frac{-F}{A}} \qquad b = \sqrt{\frac{-F}{C}} \tag{3.96}$$

The real points of the ellipse in which we are interested in are just that between the two given points. Since we draw the ellipse from the parameter α we need α_1 and α_2 corresponding to P_1 and P_2. At first we need to bring the points in the ellipse frame of reference and then it is easy to compute the desired angle.

$$\mathbf{P_1}^{(\text{ellipse center})} = \mathbf{P_1'} = \mathbf{P_1} + \mathbf{c} \tag{3.97}$$

$$\mathbf{P_2}^{(\text{ellipse center})} = \mathbf{P_2'} = \mathbf{P_2} + \mathbf{c} \tag{3.98}$$

Then we compute $\mathbf{P_1''}$ and $\mathbf{P_2''}$ rotating $\mathbf{P_1'}$ and $\mathbf{P_2'}$ of the angle θ. Finally:

$$\alpha_1 = \tan^{-1}\left(\frac{y_{P_1''}}{b} \cdot \frac{a}{x_{P_1''}}\right) \tag{3.99}$$

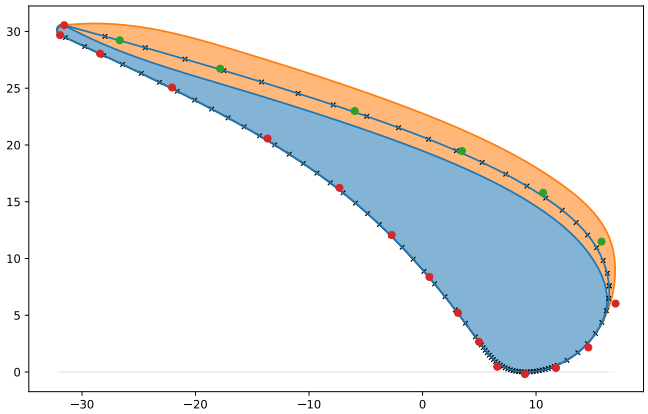
$$\alpha_2 = \tan^{-1}\left(\frac{y_{P_2''}}{b} \cdot \frac{a}{x_{P_2''}}\right) \tag{3.100}$$

3.8.3 Rigid Motion

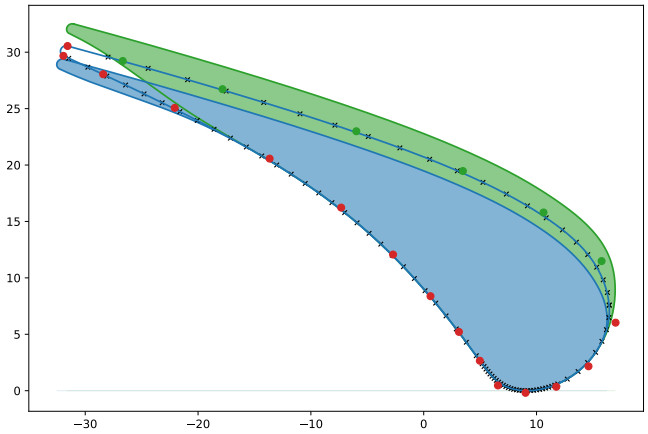
With the current implementation, each control point can be or rigidly fixed or moved independently respect to the others. This flexibility is in general enough for most of the points, but can be useful to move some regions, like the trailing edge, rigidly. In this way we keep the exact shape of the trailing edge that we design, but we provide to the optimizer the possibility to adjust its position respect to the remaining part of the blade.

In our B-spline implementation, this effect is reached giving the possibility to add to the design variables just one control point of the four required to lock the trailing edge and then translate all the other three points of the same quantity as the first.

In figure 3.13 it is shown the difference between two blades, the former parametrized with the previous implementation and the latter with the rigid motion of the trailing edge. The same control points are fixed and movable.



(a) Default implementation



(b) Rigid trailing edge

Figure 3.13: Example of the implementation of a rigid trailing edge.

Chapter 4

Design of experiments

The first fundamental step in the application of the surrogate modeling techniques is the sampling of the design variable space. This sampling procedure is called *design of experiments* or *DOE*, and its aim is to maximize the amount of information acquired to make the interpolation with the surrogate models as reliable as possible.

In most applications, where the assumed model is in doubt, and the data is acquired by deterministic computer simulations, which is the case of this thesis, the primary interest is minimizing the bias error.

Two kind of errors can be identified:

- the *bias* quantifies the difference between the surrogate model prediction and the true values for all possible data sets;
- the *variance* measures how much the surrogate model is sensitive to a particular data set.

Usually bias and variance are in trade-off one to each other; smoothing the surrogate model would reduce variance but would increase the bias.
We can theoretically decrease both of them but we need to increase the number of samples; however this is in general severely limited since we are dealing with problems with an high number of variables, and each additional sample would require a high fidelity simulation.

In the literature many DOE strategies exist, but we can roughly distinguish between *classic methods* and *modern methods*

4.1 Classical design of experiments

Many classic approaches are based on *Factorial Design*

The first and most common method is to perform a full factorial design.
It consists in performing tests considering for each design variable two values, one low and one high.
Then we will test each combination of low and high values for each variable. In table 4.1 it is shown a sequence of run needed for a 3 variables problem.

Run number	Variable A	Variable B	Variable C
1	low	low	low
2	low	low	high
3	low	high	low
4	low	high	high
5	high	low	low
6	high	low	high
7	high	high	low
8	high	high	high

Table 4.1: List of runs required for full factorial design.

The fractional factorial design is very similar to the full case, with the difference that it takes just a subset of the full run sequence. A 1/2 fractional factorial design of a 3 variable problem is shown in table 4.2.

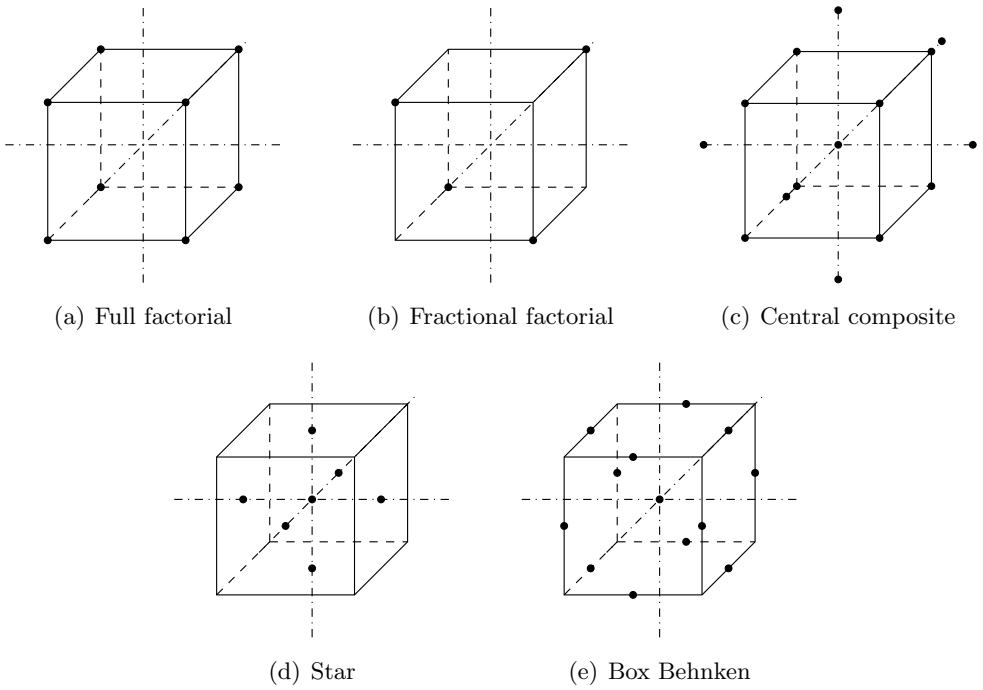
Run number	Variable A	Variable B	Variable C
1	low	low	high
2	low	high	low
3	high	low	low
4	high	high	high

Table 4.2: List of runs required for $1/2$ *fractional factorial design.*

In figure 4.1 we show in a graphical manner the comparison between various classical techniques. Different technique leads to different number of sample points but all these classical methods requires a huge number of samples when the number of design variable increases (i.e the full factorial design scales exponentially, requiring $2^{\text{number of variables}}$ samples)

4.2 Latin hypercube sampling (LHS)

An alternative to a full design approach is to maximize the minimum distance between all the samples, or minimizing the correlation measures among the sample data.
Implementations of these strategies are for example the latin hypercube sample (LHS) or the orthogonal array.
We will focus on the LHS since it is the design of experiments algorithm chosen in this thesis work.



(a) Full factorial (b) Fractional factorial (c) Central composite

(d) Star (e) Box Behnken

Figure 4.1: Comparison between different DOE techniques for 3 variables case.

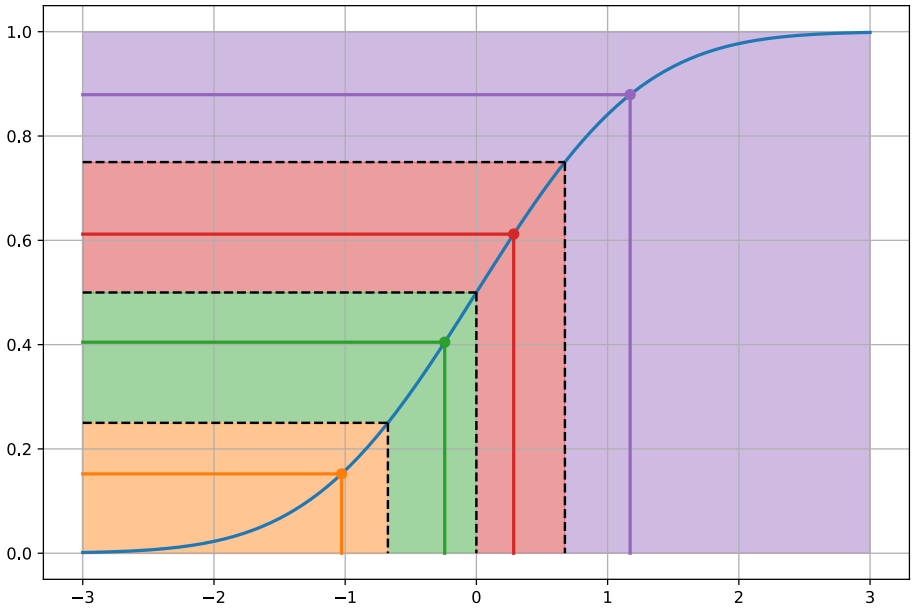


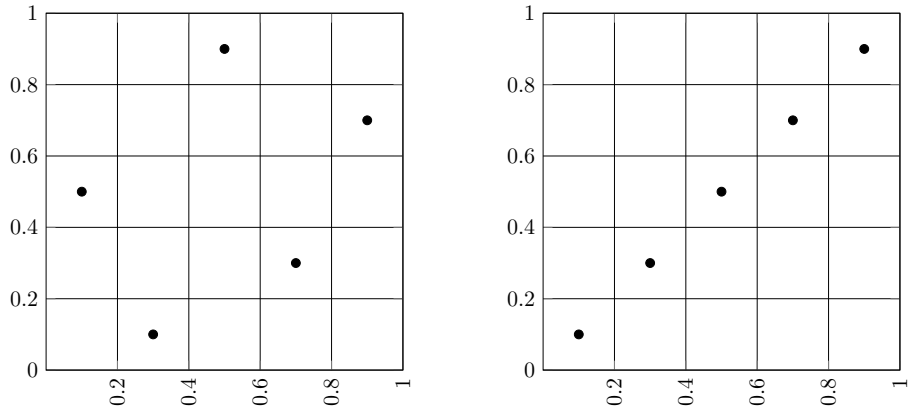
Figure 4.2: Example of LHS for a 1D interval

The main idea of this approach is the *stratification* of the input probability distribution; we divide the cumulative curve into intervals of equal probability and than we randomly take *one and only one* sample from each interval.

A one-dimensional example is represented in figure 4.2.

Extending this concept to a two-dimensional case would generate a latin square. The latin adjective means that we have to take just one sample from each row and column.
Two different cases are sketched in figure 4.3.

Uniformity performances between different sampling can be measured by the maximum minimum-distance between points. With this definition it is clear that the case 4.3(a) is much better that the case 4.3(b)



(a) Space filling LHS (b) No space filling LHS

Figure 4.3: Comparison between different DOE techniques for 3 variables case.

Two of the features that makes the latin hypercube sampling particularly effective are the fact that we can easily control the number of sampling point required with equation 4.1, and that it is able to deal with wide design space and it is able to generate set of points characterized by uniform projections on axis variables.

$$\left(\prod_{n=0}^{M-1}(M-n)\right) = (M!)^{N-1} \tag{4.1}$$

where: N is is the number of variables;
M is is the number of intervals of the ranges of the variables;

Chapter 5

Surrogate Models

Surrogate modeling is defined as a non-linear inverse procedure which aims to determine a continuous function of a set of design variables from a limited amount of available data [16].

In optimization problems where the function that we need to optimize is complex and computationally intensive, it is often impossible to run the optimization procedure directly on the high-fidelity function, since heuristics methods we are going to use in this work require, in general, a large number of function evaluations to provide reliable results, so surrogate modeling is required.

The preliminary requirements of each surrogate model is the initial data, which is obtained through the sampling process called design of experiments or DOE, explained in chapter 4.

The formulation of generic surrogate models requires a set of sampling points $\mathbf{S} = [\mathbf{x}^{(1)}, \cdot, \mathbf{x}^{(n)}]$ to which corresponds the response vector $\mathbf{y}_S = [y^{(1)}, \cdot, y^{(n)}]$. Then with the model, we compute the output as:

$$y(\mathbf{x}) = \hat{y}(\mathbf{x}) + \varepsilon(\mathbf{x}) \tag{5.1}$$

where: $\hat{y}(\mathbf{x})$ = the interpolation of the data;
$\varepsilon(\mathbf{x})$ = the random error.

An all way around best model does not exist, so many different ones are available, and each one performs better in specific problems [16].

In the following sections we will explain some of the meta-models among the many available in the literature, exploiting the formulations of $\hat{y}(\mathbf{x})$and $\varepsilon(\mathbf{x})$

5.1 Polynomial regression model (PRG)

The polynomial regression model applies a polynomial interpolation for the term $\hat{y}(\mathbf{x})$, meaning that the function of interest is a linear combination of K polynomial basis. The error terms $\varepsilon(\mathbf{x})$ instead are considered to be independent one to each other, with

zero expected value and with variance σ^2.

$$y_i(\mathbf{z}) = \sum_{j=1}^{K} \beta_j \, z_j^{(i)} + \varepsilon_i \tag{5.2}$$

For example if the polynomial degree is set to 2 the interpolation becomes:

$$\hat{y}(\mathbf{x}) = \beta_0 + \sum_{i=1}^{2} \beta_i \, x_i + \sum_{i=1}^{2} \sum_{j \leq i}^{2} \beta_{ij} \, x_i \, x_j \tag{5.3}$$

The aim of the regression procedure is to compute the vector $\boldsymbol{\beta}$. We can rewrite equation 5.2 in matrix form as:

$$\mathbf{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \qquad \text{with} \quad E(\varepsilon) = 0 \quad \text{and} \quad V(\varepsilon) = \sigma^2 \, \boldsymbol{I} \tag{5.4}$$

Where $\boldsymbol{X}$ is the matrix of the basis functions with design variables evaluated at sample points. With least square minimization algorithm we can compute the unknown vector.

$$\boldsymbol{\beta} = (\boldsymbol{X}^{\mathrm{T}} \, \boldsymbol{X})^{-1} \boldsymbol{X}^{\mathrm{T}} \mathbf{y} \tag{5.5}$$

5.2 Radial basis functions (RBF)

Radial basis function metamodel has been developed for interpolation of scattered multivariate data [16]. The method works with a linear combination of K radially symmetric functions $h_i(\mathbf{x})$ which are dependent on Euclidean distance. The mathematical formulation is written in equation 5.6.

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^{K} \omega_i \, h_i(\|\mathbf{x_i} - \mathbf{x}\|) \tag{5.6}$$

where: ω_i is the i-th unknown weight coefficient;
h_i is the radially symmetric base.

The flexibility of this model comes from the different choices of the weights.
To compute the weights the same procedure as the polynomial regression model can be applied, and we will get the best weights in term of least square error minimization.

$$\boldsymbol{\omega} = (\boldsymbol{H}^{\mathrm{T}} \, \boldsymbol{H})^{-1} \boldsymbol{H}^{\mathrm{T}} \hat{\mathbf{y}} \tag{5.7}$$

where: $\boldsymbol{H}$ is the matrix that contains the basis evaluated in the data.

A typical basis is the Gaussian one that can be expressed as:

$$h_i(\mathbf{x}) = \exp\left(-\frac{(\mathbf{x} - \mathbf{c})^2}{\delta^2} \right) \tag{5.8}$$

where: $\mathbf{c}$ is the center of the model;
δ is the radius of the model.

Both $\mathbf{c}$ and δ are parameters of the regression model.
A list of the most common basis can be found in table 5.1.

Basis function	Formula
Gaussian	$\exp\left(-\frac{r^2}{2\,\sigma^2}\right)$
Power function	r^{β}
Thin plate spline	$r^2\,\log(r)$
Hardy's multiquadric	$\sqrt{1+r^2}$
Hardy's inverse multiquadric	$\frac{1}{\sqrt{1+r^2}}$

Table 5.1: List of basis function of RBF

5.3 Neural networks

Neural networks are one of the most interesting techniques being developed that can fit in almost any industrial and technological field, like image and speech recognition, pattern classification, telecommunication, control of sound, vibrations, particle beam accelerator, quality control, medical applications, marketing analysis and a tons of other use-cases [17].
They try to replicate the structure of the human brain to mimic the way in which we learn.

The network is a collection of very simple computational unit, called perceptron, and their power is related to the interlinks between these units. The number of units can be very large and the lattice really complicating, with several intermediate layers.

The task of the perceptron is just to make the weighted summation of all the inputs and then to pass the result through an activation function.
At beginning we compute the sum.

$$\eta = \omega_1\,x_1 + \omega_2\,x_2 + \cdots + \omega_n\,x_n + \gamma = \gamma + \sum_{i=1}^{n} \omega_i\,x_i \tag{5.9}$$

Then we pass this summation through the activation function and we get the output of the node.

$$y = \frac{1}{1+\exp\left(\frac{\eta}{T}\right)} \tag{5.10}$$

where: T is a predefined slope parameter.

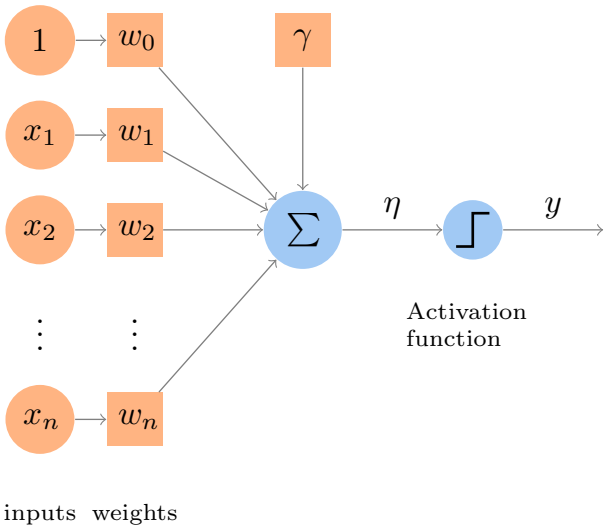


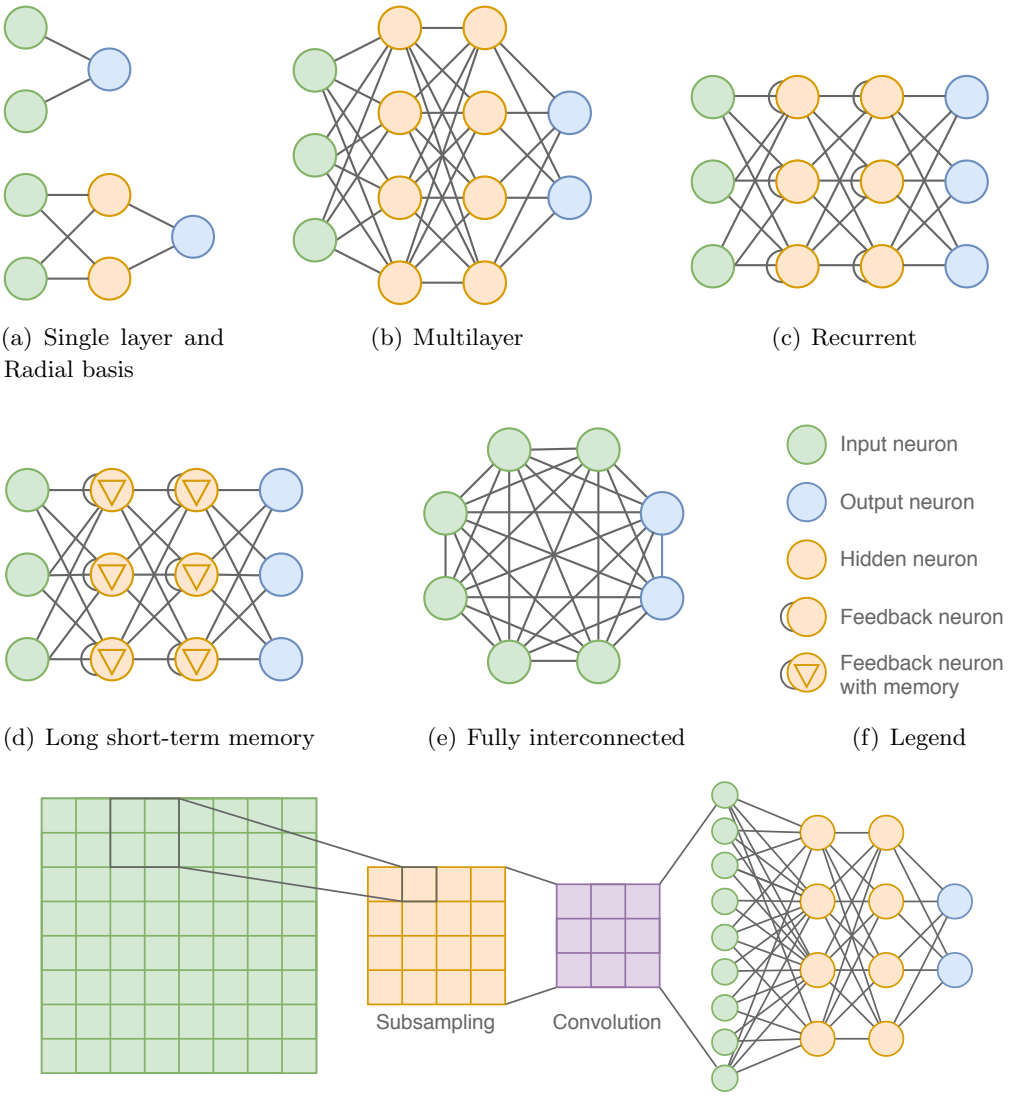
Figure 5.1: Representation of a perceptron

The neural network contains many perceptron belonging to different layers arranged in infinitely many ways (Figure 5.2). The power of the network is into the choice of the architecture and the ability to train it with a proper number of examples.

A large number of different architectures exists, each one with different characteristics. Among the many available, we will report the most common:

- single layer preceptron: the most simple structure (Figure 5.2(a) at top);
- radial basis network, which uses radial basis function as activation function (Figure 5.2(a) at bottom);
- multi-layer network, which links the input and the output layers through the introduction of additional hidden layers (Figure 5.2(b));
- recurrent neural network, which introduce a self-connection for the hidden preceptron, giving to them a memory capability (Figure 5.2(c));
- long short-term memory neural network, incorporate the memory cell into the neurons of the hidden layer (Figure 5.2(d));
- fully interconnected network, in example the Hopfield or the Boltzmann machine network (Figure 5.2(e));
- convolutional neural network, particularly suitable where there the link between data depends on the distance between them, and introduces some pre-processing steps between the input data of the problem and the input of the actual neural network and it has revealed particularly suitable for image recognition (Figure 5.2(g));

- modular neural network, which combines the previous architectures in a single structure.



(a) Single layer and Radial basis

(b) Multilayer

(c) Recurrent

(d) Long short-term memory

(e) Fully interconnected

(f) Legend

(g) Convolutional

Figure 5.2: Representation of some of the neural networks architectures.

The training procedure is performed with recursive procedures, like the gradient descent or the back propagation algorithms.

The training requires many input-output couples; in the case of fluid dynamic shape optimization the input are the parameters that generate the shape, while the output is the result of an high fidelity CFD simulation.

5.4 Kriging modeling (KRG)

The Kriging modeling, also called Gaussian process, were originally developed for geostatics problems, but nowadays is applied in many fields, from statistics to surrogate modeling. This model is able to predict the value of an unknown point using stockatstic process [7].
In this thesis the Kriging model is applied as surrogate to interpolate the output values (blade performances) and the eventual constraints over the set of input optimization variables (geometrical shape). The relation that the model generates has no physical meaning, it is just a mathematical expedient to predict the output that corresponds to an input, given an initial set of samples through which we build the model. This has sufficient flexibility to represent either *non-linear* or *multimodal* functions.

The unknown function $y(\mathbf{x})$ is expressed as:

$$y(\mathbf{x}) = \beta + \varepsilon(\mathbf{x}) \tag{5.11}$$

where: β is a constant global model;
$\varepsilon(\mathbf{x})$ is a local deviation from the global model.

Considering the β term as constant, the responsibility of a proper interpolation moves to the error term $\varepsilon(\mathbf{x})$.
The assumption of the model is that the correlation between $\varepsilon(\mathbf{x}^i)$ and $\varepsilon(\mathbf{x}^j)$ is strongly related to the distance between $\mathbf{x}^i$ and $\mathbf{x}^j$. The distance is not Euclidean and it is computed with equation 5.12.

$$d(\mathbf{x}_i,\ \mathbf{x}_j) = \sum_{k=1}^{m} \theta_k \left| x_k^i - x_k^j \right|^2 \tag{5.12}$$

where: θ_k is the k-th item of the correlation vector
m is the number of design variables

In this case the correlation between $\varepsilon(\mathbf{x}^i)$ and $\varepsilon(\mathbf{x}^i)$ can be computed as:

$$\mathrm{corr}[\varepsilon(\mathbf{x}^i,\ \varepsilon(\mathbf{x}^j)] = \exp(-d(\mathbf{x}_i,\ \mathbf{x}_j)) = -\exp\left(\sum_{k=1}^{m} \theta_k \left| x_k^i - x_k^j \right|^2 \right) \tag{5.13}$$

Then the Kriging predictor becomes:

$$\hat{y}(\mathbf{x}) = \hat{\beta} + \mathbf{r}^{\mathrm{T}}\, \boldsymbol{R} \left(\mathbf{y} - \mathbf{1}\hat{\beta} \right) \tag{5.14}$$

where: $\hat{\beta}$ is the estimated value of the constant global model β;
$\boldsymbol{R}$ is the $m \times m$ correlation matrix of the distances;
$\mathbf{r}$ is a $m \times 1$ vector where $r_i = \mathrm{corr}[\varepsilon(\mathbf{x}),\ \varepsilon(\mathbf{x}^i)]$;
$\mathbf{1}$ is a $m \times 1$ vector with all elements equal to 1.

The solution of the Kriging model consists in finding the parameters θ_k. The rigorous definition of $\boldsymbol{\theta}$ requires the maximization of the likelihood function [1] in equation 5.15.

$$\mathcal{L}\left(\hat{\beta},\ \hat{\sigma}^2,\ \boldsymbol{\theta}\right) = -\frac{m}{2}\mathcal{L}\left(2\,\pi\right) - \frac{1}{2}\mathcal{L}\left(|\boldsymbol{R}|\right) - \frac{m}{2}\mathcal{L}\left(\hat{\sigma}^2\right) - \frac{1}{2\,\hat{\sigma}^2}\left(\mathbf{y} - \mathbf{1}\hat{\beta}\right)^{\mathrm{T}}\boldsymbol{R}^{-1}\left(\mathbf{y} - \mathbf{1}\hat{\beta}\right) \tag{5.15}$$

Through an iterative procedure we evaluate $\boldsymbol{\theta}$ that maximize the likelihood function $\mathcal{L}$.

The accuracy of the prediction depends on the distance from the sample points. The mean square error of the prediction can be computed as:

$$s^2(\mathbf{x}) = \hat{\sigma}^2\left[1 - \mathbf{r}^{\mathrm{T}}\boldsymbol{R}\mathbf{r} + \frac{1 - \mathbf{1}\,\boldsymbol{R}^{-1}\,\mathbf{r}}{\mathbf{1}\,\boldsymbol{R}^{-1}\,\mathbf{1}}\right] \tag{5.16}$$

5.5 Low fedelity CFD model

All the surrogate model previously explained does not have any physical meaning; they are just a mathematical interpolation of the high fidelity data that try to predict best value for unknown points.
In literature an alternative approach exists: instead of performing a purely mathematical model of the phenomenon of study, the approach consists in performing a simulation with lower fidelity .
Low fidelity models can be the same as the high-fidelity ones but with higher tolerance of convergency [18], with a coarser mesh [19] or with a simplified physical model [20].

If properly tuned, this can guarantee to get reasonable results with much less computational cost, reducing the role of the high fidelity simulation just to validate the optimal solution computed over the low fidelity ones.

The concept on which this kind of surrogates are based is that even if a low fidelity simulation cannot properly compute the value of a quantity of interest (like drag and lift coefficients), it is able to keep the ranking between different simulations, meaning that if a blade is better respect to another after an high fidelity simulation, it will be the same even with a low fidelity one.

Koziel and al. [21] proposed an automatic way to to build a set of physics-based surrogate model of variable fidelity for constrained non-linear optimization problems. They stated that the main benefit is that starting from a fast less accurate model allows to quickly find an approximate region for the optimum that is then accurately explored with progressively more accurate simulations, without an excessive amount of computational cost. Furthermore in contrast with other physics-based low-fidelity SBO methods, no improvement on the low fidelity model is required, and then, multi-level approach is less dependent on low-fidelity model quality. The main parameters that change in variable fidelity simulations are the number of iterations and the number of cells. These parameters are in general obtained after a proper convergence and grid-dependence analysis.

[1] Given a probability distribution and the corresponding parameters, the likelihood function is a measure of the probability of the sample data being drawn from it [16].

5.6 Surrogate based optimization

By resorting to a design of experiments technique to adequately sample the whole design space, the initial database of high fidelity simulations is built. Then, given the data, we build on it the surrogate model interpolation. Since the initial data set is limited we want to improve it when the iterations of the optimization algorithm progress with the new high-fidelity simulations available.

Two different strategies to manage the improvement in reliability are available, each one with pros and cons:

- Global optimization;
- Local optimization.

The main difference between the two is that once the DOE is performed, the global version advances in a serial manner, iteration by iteration, while the local one recreate a new DOE after each iteration. Even if it seems heavy and useless to recreate the doe from scratch after each iteration, the big advantage of the SBLO is that it is highly scalable when a lot of computational power is available, and from a theoretical point of view it guarantees to find the global optimum

5.6.1 Surrogate-Based Local Optimization (SBLO)

The SBLO method is based on the idea of local reliability of the surrogate model, meaning that it assumes that the approximation model is sufficiently accurate locally to find the optimum. The algorithm progressively builds the surrogate function over the local subset of the original design space. The subsets are called *trust regions*.
The size of the trust region is adopted and modified during the optimization process, trying to match two concurrent aims:

- find the region which contains the optimum;
- maximize the local reliability of the surrogate model.

At this point it is clear that a metric to evaluate the reliability of a given thrust region is required.
We define the *reliability index* or *thrust region ratio* $r^{(k)}$ as:

$$r^{(k)} = \frac{f\left(\left(x_c^{(k)}\right) - f\left(x_{\text{opt}}^{(k)}\right)\right.}{\hat{f}\left(x_c^{(k)}\right) - \hat{f}\left(x_{\text{opt}}^{(k)}\right)} \tag{5.17}$$

where: f is the high-fidelity objective function;
$\hat{f}$ is the surrogate function;
k is the index of the iteration;
x_c is the center of the thrust region;
x_{opt} is the optimum inside the thrust region.

Advancing in iterations we set the center of the new iteration as the optimum of the previous one, $x_c^{k+1} = x_{\text{opt}}^k$. For what that concern the optimum x_{opt} instead, it is the point that results from the application of the genetic algorithm applied to the surrogate function $\hat{f}$. After that the genetic algorithm finds the optimum at a given iteration, we run an high fidelity simulation in that optimum. Then with equation 5.17 we can finally compute the reliability index, which is related to the extension of the thrust region of the next iteration.
The scheme of the algorithm is represented in figure 5.3.

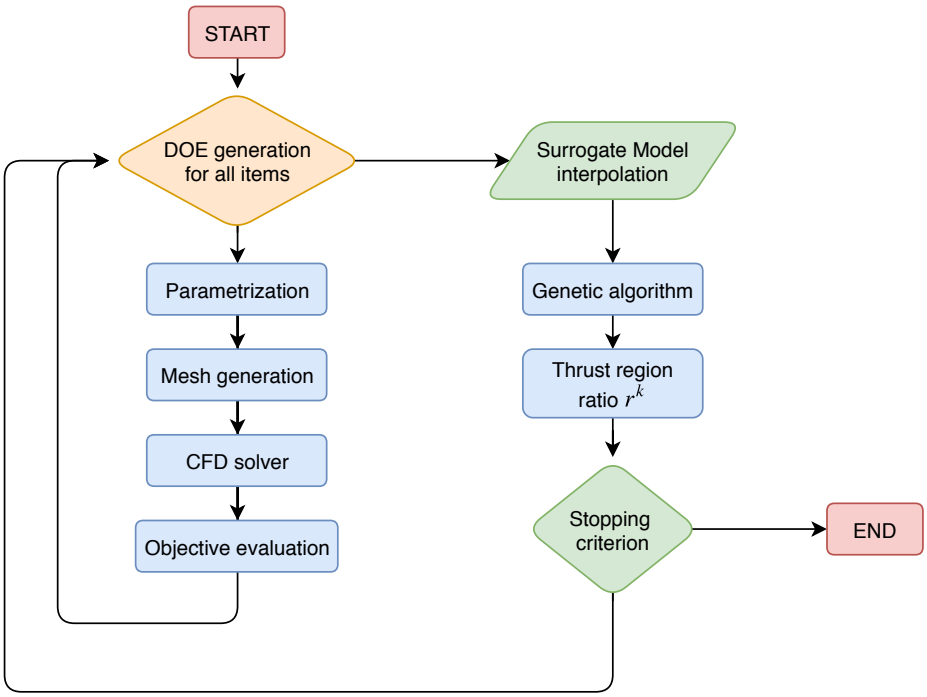


Figure 5.3: Surrogate-based local optimization algorithm

Depending on the value of $r^{(k)}$ a different behavior is applied.
If $r^{(k)} < 0$ is a symptom of a poor accuracy of the surrogate since an apparent improvement in the surrogate optimum leads to a worsening of the real function. The thrush region is reduced trying to improve the surrogate model accuracy.
If $0 < r^{(k)} < 0.25$ the accuracy is however low, so the thrust region is reduced even in this case.
If $0.25 < r^{(k)} < 0.75$ the precision is better, and we keep the same region also for the next iteration.
If $0.75 < r^{(k)} < 1.25$ the accuracy is good, we can, in this case, expand the region.
If $r^{(k)} > 1.25$ the accuracy is fine, but not great, so we continue with the same region for the next iteration.

To ensure convergence the *Pareto optimality filter* is applied, according to which an optimum point is accepted only if it is better than any previous optimum evaluated.

We stop iterating when a stopping criterion is met. In general it is the combination of two concurrent criteria, one related to the maximum number of iterations allowed, and the other that checks the maximum number of iteration with an improvement smaller that the convergence tolerance. Once one of the two criteria is satisfied we stop the algorithm.

5.6.2 Surrogate-Based Global Optimization (SBGO)

Differently from the local approach, the surrogate-based global optimization, the database of the high-fidelity simulations is generated just once with a design of experiments approach at the beginning of the optimization and then it is improved after every iteration. The surrogate model is not supported by a thrust region, and this needs a careful application since there is no guarantee of convergence.

An important design parameter of the SBGO is the size of the initial population. A too small number of items in the database is no longer able to reconstruct the input-output relation between the many variables, a too high number instead would waste a lot of time to generate the data-set for the surrogate model.
The rule applied in this work is to take 10 samples for each design variable to build the surrogate model.

After the DOE generation, a genetic algorithm is applied to find the global optimum of the surrogate function. Then an high-fidelity simulation of the expected optimum is performed, and the new point is used to build a better interpolation model of the physical phenomenon.

The insertion of the latest item into the database can happen in two modes: or we replace it with an old item in the pool keeping the same database size, or we append it, increasing progressively the size of the samples.
We will apply the second choice due to both increasing in accuracy of the surrogate model and robustness to simulation that can fail because of impossible meshing or errors in the solving process.

As before for the SBLO the process stops when the stopping condition in matched, and in general it is one of the maximum number of iterations allowed or the convergence tolerance is met.

A sketch of the algorithm is drawn in figure 5.4.

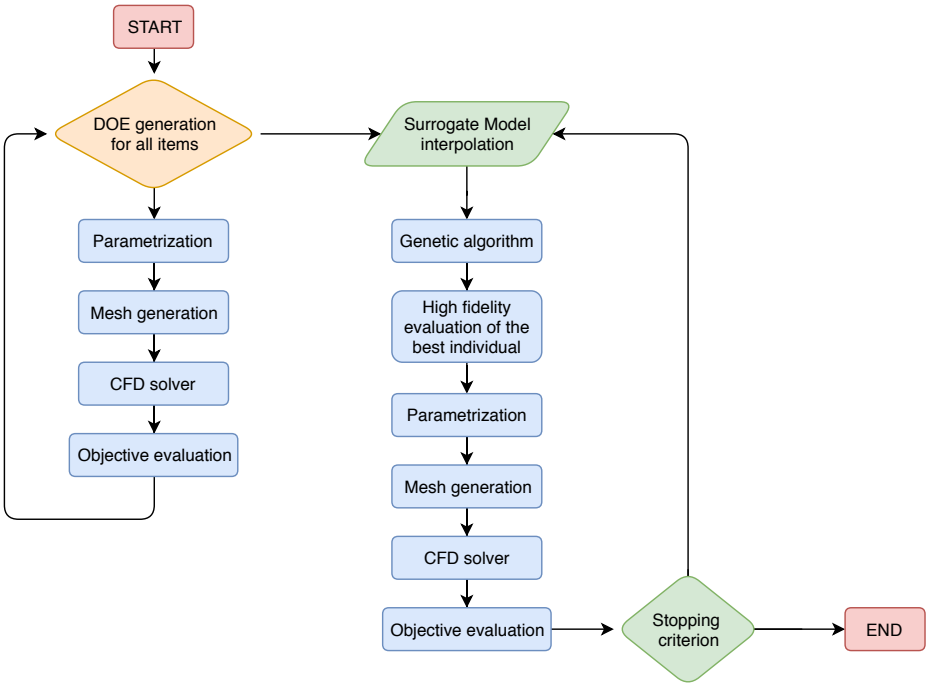


Figure 5.4: Surrogate-based global optimization algorithm

Chapter 6

Genetic Algorithms

Evolutionary algorithms are a branch of heuristic optimization algorithms that are able to find a global optimum without requiring any information on the derivatives. Their versatility is particularly useful in engineering application where we are dealing with a mix of continuous, discrete and integer design variables and non-convex or even disjointed design space.

Among these, genetic algorithms received a considerable grown of interest [22].

GA are based on the process of natural selection; from a populations only the best individuals advance to the next iteration. In this process of evolution, the genetic heritage of the parents is transmitted to the children undergoing to crossover and mutation.

What happens spontaneously in nature is replicated with the set of points of the optimization problem. The phases of the algorithm are explained in the following paragraphs.

The generation of the population. The process begins with the generation of a set of individuals called *population*.
Each individual represents a set of design variable, which is the input data of the function that we want to optimize. The variable in the genetic similitude are called *chromosome*.

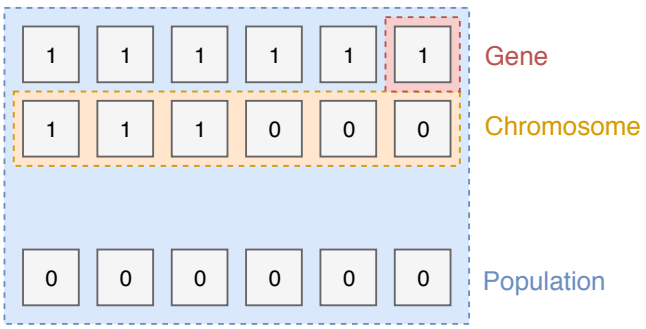
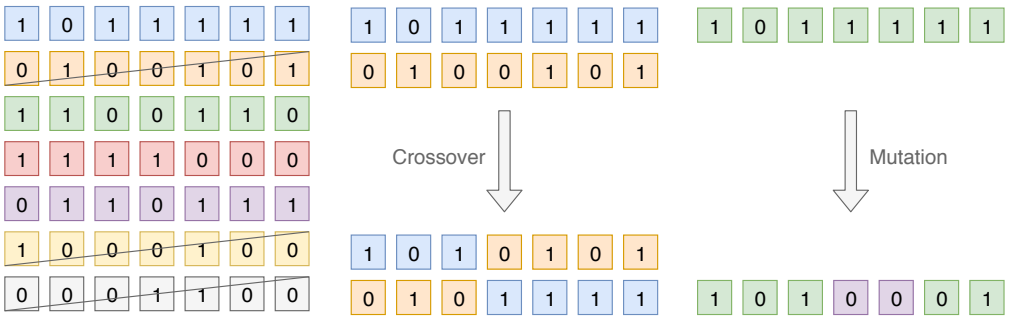


Figure 6.1: Genetics algorithm units

Each chromosome is modeled as the combination of many *genes* which are the simpler unit of the genetic evolution. The characteristic of the individuals are encoded or as binary or as real number.

Fitness function After having generated the population, we have to compute the fitness function for each of the individual of it. The value is called *fitness score*; the provability than an individual will be selected for reproduction depends on its fitness score. For maximization problem, higher fitness score is better, for minimization problem is the opposite. For surrogate-based optimization, the fitness function is the interpolated model.

Selection The selection process is the phase that makes the algorithm progressively going toward the optimal solution. The idea is to let fittest individuals to pass their genes to the next generation.
Thanks to implementation of the *elitism* GAs are able to have success. On average we take the best individuals and we discard the worst, depending on their fitness score. Selection is *not* fully deterministic, best individuals are kept with an high probability, and worst are rapidly discarded.



(a) Selection (b) Crossover (c) Mutation

Crossover After selecting each pair of parents, crossover is applied. The crossover point is randomly chosen; we build the offspring individuals picking the genes of the first parent from the begin to the crossover point and the genes of the second parent from the crossover point to the end for the first child, and the opposite for the second child. Finally the offspring are added to the population.

Mutation With low probability some of the new individuals can be subjected to mutations. Some of the genes are randomly flipped. Mutation is an import strategy that prevents premature convergence to local minima and improve exploration capability of the genetic algorithm.

After selection, crossover and mutation the process restart with the evaluation of the fitness function for the whole population, until or the maximum number of iterations (or function evaluations) is reached or the algorithm converges to the optimum.

The explanation of the algorithm is drawn in figure 6.2.

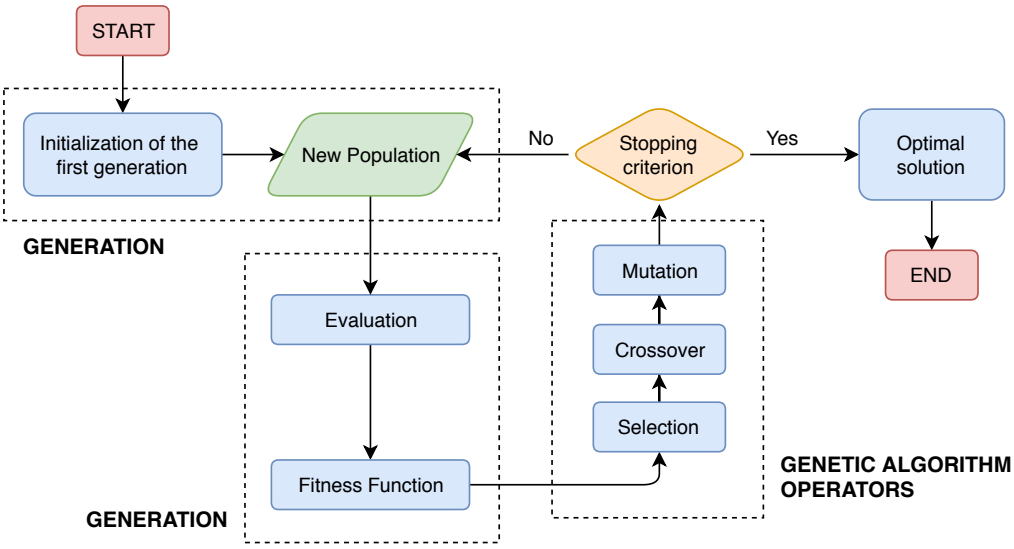


Figure 6.2: Genetics algorithm

6.1 Constraint handling

Standard genetic algorithm does not include a tool to manage constraint optimization problems. In engineering applications, it is really common to have one or even more constraints, and we can easily identify in aerodynamic optimization, many of that like structural, mass flow rate, outlet flow angle, lift coefficient, blade thickness and more.

Usually the optimal solution is located on an intersection of hypersurfaces of of different dimensions, generated by linear and non-linear constraints not even known in advance and with irregular topology.

The approach proposed by Peigin and Epstein [23] was to employ a search path that passes through both feasible and unfeasible solution, instead of traveling only through the valid space. The basic idea is that passing from unfeasible sub-domains implies a shorter path to the optimum.

We reach this keeping within the population some individual outside the domain but close to the constraint boundaries. We expect that the crossover between feasible and unfeasible individuals would produce high-fitness children on the boundaries.

We call fitness function $f(\mathbf{x})$ defined over the feasible region R_f, subjected to the non-linear constraint $g(\mathbf{x}) \leq 0$. The boundary satisfies the equation $g(\mathbf{x}) = 0$.

First at all we need to estimate the value of the fitness function inside the feasible region R_f.

$$f(\mathbf{x}) \approx A \quad \text{with} \quad x \in \mathrm{R}_f \tag{6.1}$$

Then we define the first approximation $f^*(\mathbf{x})$ as:

$$f^*(\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{if } g(\mathbf{x}) \leq 0 \\ B & \text{if } g(\mathbf{x}) > 0 \end{cases} \tag{6.2}$$

where $B \gg A$.

Considering the function $f^*(\mathbf{x})$ as the function that we want to optimize, the problem is no more constrained, and we can solve it with a GA, and we will find that the optimum in the feasible region of $f^*(\mathbf{x})$ we estimate in the neighborhood of the boundaries as $f(\mathbf{x}) \approx C$.
Finally the modified objective function $f^{**}(\mathbf{x})$ in the total search region R is:

$$f^{**}(\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{if } g(\mathbf{x}) \leq 0 \\ \alpha_1 C + \alpha_2 B \cdot g(\mathbf{x}) & \text{if } g(\mathbf{x}) > 0 \end{cases} \tag{6.3}$$

where α_1 and α_2 are problem-dependent parameters.

6.2 The JEGA library

The Dakota framework interfaces with the JEGA library [24] to run genetic algorithm. In the library two main algorithm are available:

- single-objective genetic algorithm or SOGA;
- multi-objective genetic algorithm or MOGA.

Both the algorithms support constraints and a any combination of real and discrete variables.

Each design variable is represented as a single 32 bits signed integer, but working with this representation reduces the precision to 6 significant digits.

After the creation of each new individual, a duplicity test is performed to assure uniqueness among the population to avoid evaluate more times the same values.

The library support many choices for the all characteristic phases of the genetic algorithm.

Population initialization

- Simple Random;
- Unique Random;
- Specified.

Crossover type

- Multipoint Binary;
- Multipoint Parametrized Binary;
- Multipoint Real;
- Shuffle Random.

Mutation Type

- Replace Uniform;
- Bit Random;
- Offset Normal;
- Offset Cauchy;
- Offset Uniform.

Replacement Type

- Roulette-Wheel;
- Below-Limit;
- Elitist;
- Favor Feasible.

Convergence Type

- Average Fitness Tracker;
- Best Fitness Tracker.

6.2.1 SOGA

The main characteristics of the single objective genetic algorithms are explained in previous paragraphs. SOGA is the implementation of the algorithm specifically for problem with a single fitness function or for many objectives at which a constraint weights average coefficient is applied.

$$f(\mathbf{x}) = w_1\, f_1(\mathbf{x}) + w_2\, f_2(\mathbf{x}) + \cdots + w_n\, f_n(\mathbf{x}) \tag{6.4}$$

where the weight coefficients w_i are constant and chosen by the designer before of the run of the simulation.

6.2.2 MOGA

Many engineering problems does not have a single performance parameter that can be optimized, but very often there are trade-off between opposite aims.

These class of problems can be solved building a single objective fitness function through the use of weights, as anticipated in the previous paragraph. This approach is called in our routine *multipoint optimization*. We perform a weighted average between multiple objectives.

$$f(\mathbf{x}) = w_1\, f_1(\mathbf{x}) + w_2\, f_2(\mathbf{x}) + \cdots + w_n\, f_n(\mathbf{x}) \tag{6.5}$$

Once we have chosen the constant weights, we optimize the function $f(\mathbf{x})$ with the standard single objective genetic algorithm.

The multi-point optimization requires to guess in advance the weights. Sometimes they are known, for example if we need to optimize a blade at the hub, mid and tip, but in many other cases, this is not possible, or at least inaccurate.

The real and more general alternative is to perform a full multiple objective optimization, that requires an adaptation of the genetic algorithm. This algorithm is called MOGA and it was proposed by Murata and Ishibuchi [25]. The idea is that if we fix the weights in equation 6.5 we are fixing the search direction in the genetic algorithm, so the weights w_i are taken randomly, not to consider just a single fixed direction.

$$w_i = \frac{\mathrm{rand}_i()}{\sum_{j=1}^{n} \mathrm{rand}_j()} \qquad \text{for } i = 1, 2, \cdots, n \tag{6.6}$$

From equation 6.6 it can be seen that the weights w_i belong to the interval $[0, 1]$.

The MOGA algorithm follow 8 steps, like drawn in figure 6.3.

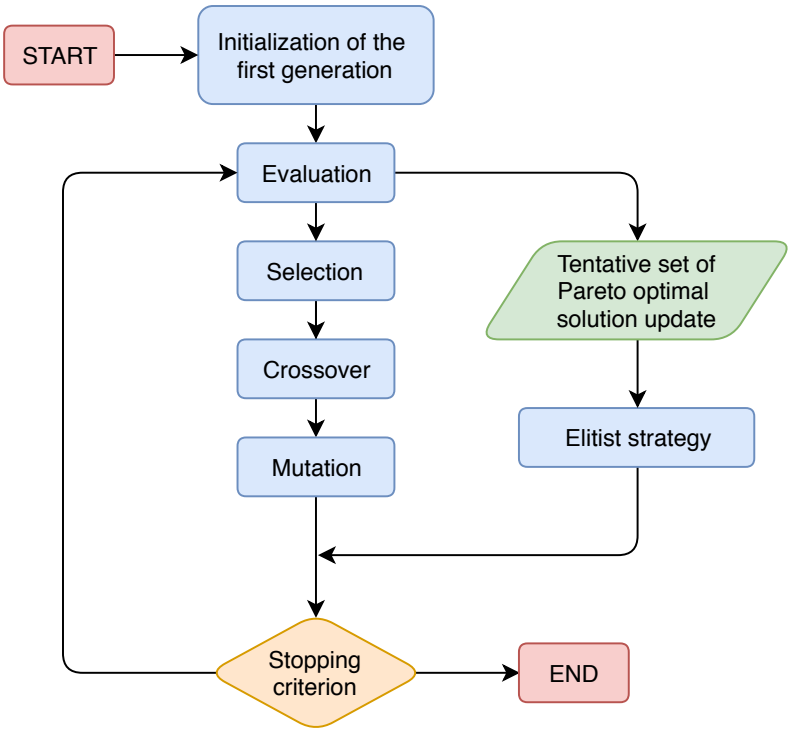


Figure 6.3: Multiple Objective Genetics algorithm (MOGA)

Step 0 - Initialization Generate an initial population of N individuals.

Step 1 - Evaluation The objective functions are evaluate for each individual and the tentative set of Pareto optimal solution is updated.

Step 2 - Selection The fitness value of each individual is evaluated with equation 6.5 where w_i are randomly generated with equation 6.6. then a pair of parents is selected based on the selection probability $P(\mathbf{x})$ of an individual $\mathbf{x}$ within the population Λ is computed with equation

$$P(\mathbf{x}) = \frac{f(\mathbf{x}) - f_{\min}|_\Lambda}{\sum_{\mathbf{x}\in\Lambda} (f(\mathbf{x}) - f_{\min}|_\Lambda)} \tag{6.7}$$

where: $f_{\min}|_\Lambda = \min f(\mathbf{x}) \mid \mathbf{x} \in \Lambda$.

The process selects $N/2$ pairs of parents.

Step 3 - Crossover Two new children are generated from each pair by means of the crossover.

Step 4 - Mutation Some of the bits are mutated according to a predefined probability.

Step 5 - Elitist Strategy The main different with the standard genetic algorithm is the elitism procedure. Actually in the MOGA M individuals from the population are replaced with the same number of items randomly extracted from the tentative set of Pareto optimal solution.

Step 6 - Stopping criteria If no one of the stopping criteria is satisfied the loop start again from step 1.

Step 7 - User decision Once a stopping criterion is satisfy the algorithm stops returning a set of Pareto optimal solutions. Depending on the problem the user has the final choice of which one of the solution is effectively taken.

Chapter 7

Computational Fluid Dynamic

Fluid dynamic phenomena are of interest in many practical and engineering applications. Their study is based on the Navier-Stockes equations, a set of six partial derivative non-linear unsteady coupled differential equations whose analytic solution is available only in a few of simplified cases, far from any engineering application.

The solution of Navier-Stockes equations is a broad field of research even today, more that 150 years after their formulation, the existence of a solution and its smoothness is not proven yet. Furthermore the Clay Mathematics Institute considers it one of the most important open problem in mathematics (it actually belongs to the Millennium Prize Problems) and who will solve it will receive the one million dollar prize.

In the last decades, thanks to the improvement in computer performance, the application of numerical algorithms is widely used in engineering design. The numerical strategies are grouped under the acronym of CFD or Computational Fluid Dynamics. CFD have advantages respect to experiments, like the simpleness of simulating different configurations once the first setup is done, or the fact that investigating the value of the variable of interest inside the flow in any point does not affect the simulation.

However CFD simulations have to be carefully set-up and post-processed since they can be source of macroscopic errors. Any simulation does not exactly represent the real phenomenon to to approximation errors like:

- Modeling errors: the differential equation that we want to solve are a mathematical representation of the real problem, not the problem itself. Furthermore, it is almost always impossible to run fully 3D unsteady DNS simulations; that means that we further simplify the equations that describe our physical phenomenon, running steady state or bi-dimensional simulations, or modeling the turbulence.

- Discretization errors: since in engineering problem no analytic solution exists, a discretization of the differential equations in space and time is required, but this procedure obviously introduces an error.

- Convergence errors: the algorithm that solves the discretized equations are often iterative. Limiting the number of iterations means introducing an approximation.

The main aspects of CFD theory are summarized in this chapter; for more informations please refer to fluid-dynamic books such as [26].

7.1 Analysis of the numeric

Given a differential equation and a numeric discretization we want to be sure that the output of the simulation is a correct representation of the model we solve, and if we reduce the size of the time and space steps Δt and Δx the numerical solution have to tend to the real solution, and for Δt and Δx approaches zero, the two solutions have to be the same. This intuition has the name of *convergence*. To ensure convergence we need to apply the Equivalence Theorem of Lax [27] which statement is written in paragraph 7.1.

Theorem 7.1 (Equivalence Theorem of Lax) *For a well-posed initial value problem and a consistent discretization scheme, stability is the necessary and sufficient condition for convergence.*

To understand theorem 7.1 it is necessary to define the concepts of *consistency*, *stability* and *convergence*.

Definition 7.2 (Consistency) *is a condition on the numerical scheme, namely that the numerical scheme must tend to the differential equation, when time and space steps tend to zero. Accuracy measures how quickly the numerical scheme tends to the differential equation.*

Definition 7.3 (Stability) *is a condition on the numerical solution, namely that all errors, such as round-off errors (due to the finite arithmetic of the computer) must remain bounded when the iteration process advances. That is, for finite values of Δt and Δx, the error (defined as the difference between the numerical solution and the exact solution of the numerical scheme) has to remain bounded, when the number of time steps n tends to infinity.*

Definition 7.4 (Convergence) *is a condition on the numerical solution: we have to be sure that the output of the simulation is a correct representation of the model we solve, i.e. the numerical solution must tend to the exact solution of the mathematical model, when time and space steps tend to zero.*

In a mathematical way we have:

- the differential equation $D(u) = 0$ and its exact solution $\tilde{u}(x, t)$;
- the discretized numerical equation $N(u) = 0$ and its exact solution $\tilde{u}_i^n$.

Calling *truncation error* ε_{T} the difference between the numerical and the differential equation evaluated in u_i^n:

$$\varepsilon_{\mathrm{T}} = N(u_i^n) - D(u_i^n) \tag{7.1}$$

Truncation error can in general be written as function of Δt^q and Δx^p, where p and q are the order of accuracy.

$$\varepsilon_{\mathrm{T}} = N(u_i^n) - D(u_i^n) = O(\Delta t^q,\ \Delta x^p) \tag{7.2}$$

Consistency formulation can be rewritten as equation 7.3.

$$\lim_{\substack{\Delta t \to 0 \\ \Delta x \to 0}} |\varepsilon_{\mathrm{T}}| = 0 \tag{7.3}$$

Stability instead is formulated in equation 7.4.

$$\lim_{n \to \infty} |u_i^n - \tilde{u}_i^n| \leq K \qquad \text{at fixed } \Delta t \tag{7.4}$$

where: n is the time step number;
u_i^n is the computed solutions;
$\tilde{u}_i^n$ is the exact solutions of the numerical equation;
K is an arbitrarily number independent of n.

Finally convergence is formulated in equation 7.5.

$$\lim_{\substack{\Delta t \to 0 \\ \Delta x \to 0}} |u_i^n - \tilde{u}(i\Delta x, n\Delta t)| = 0 \tag{7.5}$$

where: n is the time step number of the computed solution;
i is the space step number of the computed solution;
u_i^n is the computed solutions;
$\tilde{u}$ is the exact solutions of the differential equation;

7.2 Discretization

Different approaches exist for discretizing numerically differential equations:

- Finite element method (FEM);
- Finite difference method (FD);
- Finite volume method (FVM).

The first method was developed and mainly applied for solving structural analysis, while both the second and the third can be applied in fluid dynamics. The finite difference method can be highly optimized but don't let manage too complex geometries, its main application is for DNS simulations. Most of the CFD codes apply the finite volume method for discretizing Navier-Stokes equations, and also in this study.

In FVM the domain is divided into small control volumes or cells. For each cell the value of the fields are referred to the centroid, and then interpolated over the volume to

compute fields values onto the face center and on other points of interest. Centroid and face center are defined in equation 7.6.

$$\mathbf{x}_C = \mathbf{x}_P \mid \int_\Omega (\mathbf{x} - \mathbf{x}_P)\,\mathrm{d}\mathbf{x} = 0 \qquad \mathbf{x}_f = \mathbf{x}_P \mid \int_S (\mathbf{x} - \mathbf{x}_P)\,\mathrm{d}\mathbf{x} = 0 \tag{7.6}$$

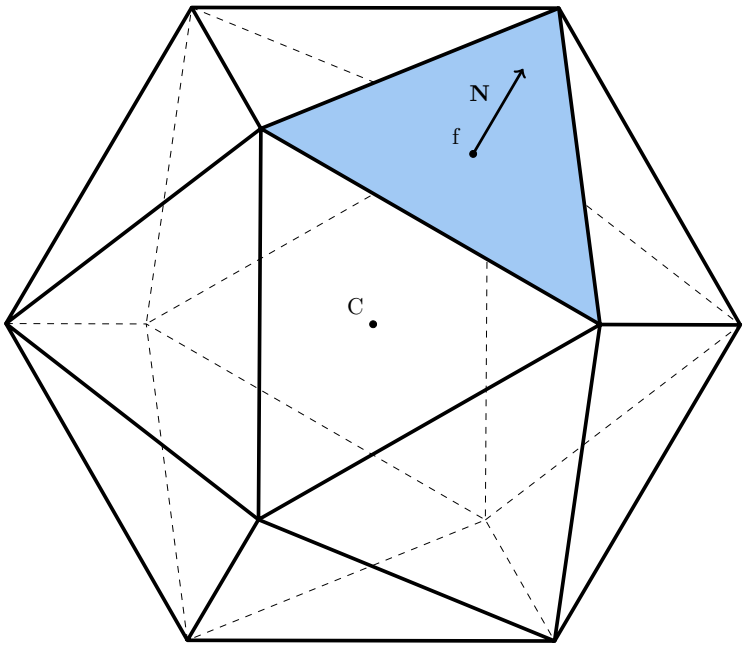


Figure 7.1: Example of a cell for the finite volume method.

7.2.1 Volume integral discretization

Assuming linear variation of the fields inside the cell a scalar field can be written as:

$$\phi = \phi_C + \nabla\phi|_C \cdot (\mathbf{x} - \mathbf{x}_C) + O\left((\mathbf{x} - \mathbf{x}_C)^2\right) \tag{7.7}$$

$$\begin{aligned} \int_\Omega \phi\,\mathrm{d}V &= \int_\Omega (\phi_C + \nabla\phi|_C \cdot (\mathbf{x} - \mathbf{x}_C))\,\mathrm{d}V \\ &= \int_\Omega \phi_C\,\mathrm{d}V + \nabla\phi|_C \cdot \int_\Omega (\mathbf{x} - \mathbf{x}_C)\,\mathrm{d}V = \Omega\phi_C \end{aligned} \tag{7.8}$$

The volume integral reduces to the value of the field in the centroid multiplied times the volume of the cell as shown in equation 7.8.

7.2.2 Surface integral discretization

$$\int_S \boldsymbol{\phi} \cdot \hat{\mathbf{n}}\,\mathrm{d}S = \sum_k^{\text{faces}} \int_{S_k} \boldsymbol{\phi} \cdot \hat{\mathbf{n}}\,\mathrm{d}S \tag{7.9}$$

The solution of the integral of each face requires two levels of approximation; first at all we have to approximate the surface integral as function of the value in the face

center and of other face points, and that we need to interpolate the field to approximate the value of the field in that points.
Some example of the first level of the approximation are written in equation 7.10.

$$\int_S \phi \, \mathrm{d}S \approx \phi_f S \quad \textbf{or} \quad \int_{S_E} \phi \, \mathrm{d}S \approx \frac{S}{2}\left(\phi_{ne} + \phi_{se}\right) \quad \textbf{or} \quad \int_{S_E} \phi \, \mathrm{d}S \approx \frac{S}{6}\left(\phi_{ne} + 4\phi_e + \phi_{se}\right) \tag{7.10}$$

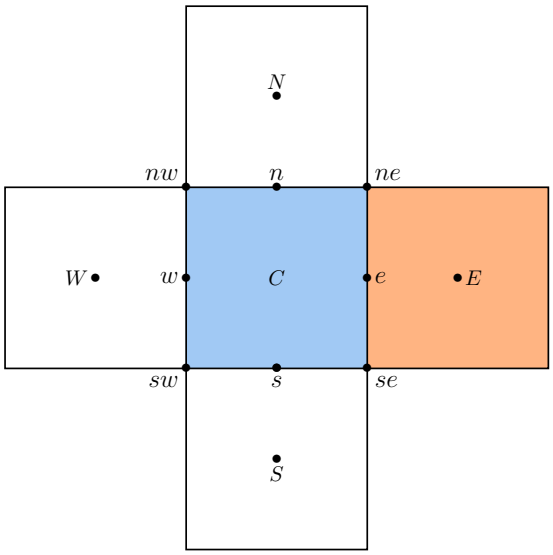


Figure 7.2: 2D Example of the points for surface integral calculations.

Up to now we do not know the values of all the points that we need to approximate the surface integral, and we obtain that by interpolating between the cells. The easiest second level of approximation is called *upwind* and it is written in equation 7.11 for the east surface point e.

$$\phi_e = \begin{cases} \phi_C & \text{if } \mathbf{U} \cdot \hat{\mathbf{n}} > 0 \\ \phi_E & \text{if } \mathbf{U} \cdot \hat{\mathbf{n}} < 0 \end{cases} \tag{7.11}$$

Even if the upwind interpolation seems really inaccurate has the great advantage to never yield to oscillatory behavior and instability. However, if possible, it is always better to choose higher order methods like the linear interpolation.

$$\phi_e = \phi_E \, k_x + (1 - k_x)\phi_C \tag{7.12}$$

where: $k_x = \dfrac{\mathbf{x}_e - \mathbf{x}_C}{\mathbf{x}_E - \mathbf{x}_C}$

7.2.3 Convective discretization

Applying and combining the rules previously explained it is possible to discretize the term that are present in the Navier-Stockes equations. The convection term is the

divergence of the velocity flux:

$$\nabla \cdot (\mathbf{U}\phi) \qquad \Rightarrow \qquad \int_\Omega \nabla \cdot (\mathbf{U}\phi)\, \mathrm{d}V \tag{7.13}$$

Applying the Gauss theorem we can pass from volume to surface integral:

$$\begin{aligned} \int_\Omega \nabla \cdot (\mathbf{U}\phi)\, \mathrm{d}V &= \int_S (\mathbf{U}\phi) \cdot \hat{\mathbf{n}}\, \mathrm{d}S \\ &= \sum_k^{\text{faces}} \int_{S_k} (\mathbf{U}\phi) \cdot \hat{\mathbf{n}}\, \mathrm{d}S = \sum_k^{\text{faces}} S_k \left(\mathbf{U}\phi\right)|_{S_k} \cdot \hat{\mathbf{n}} \end{aligned} \tag{7.14}$$

Defining the fluid flux F as the product between the normal veocity $U_n = \mathbf{U} \cdot \hat{\mathbf{n}}$ and the surface, $F = U_n\, S$, the summation can be rewritten as:

$$\int_\Omega \nabla \cdot (\mathbf{U}\phi)\, \mathrm{d}V = \sum_k^{\text{faces}} \phi_{f_k}\, F_k \tag{7.15}$$

The field on the face center can be interpolated with various levels of accuracy for example with the upwind or the linear interpolation made explicit in equation 7.11 and 7.12 or any other more advanced method.

Skewness In the discretization of the convection term an error is committed when the skewness is greater than zero where the skewness is defined as:

$$\Psi = \frac{|\mathbf{m}|}{|\overline{CE}|} \tag{7.16}$$

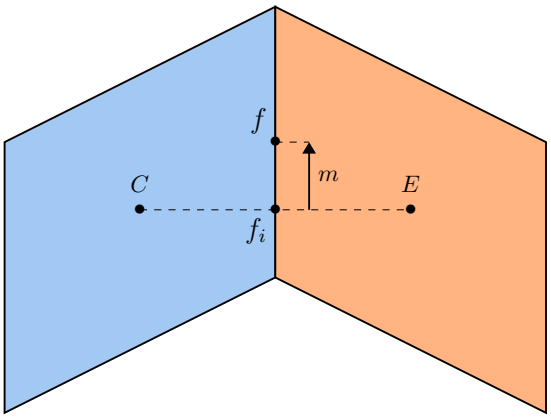


Figure 7.3: Skewness representation.

If we in example apply the linear interpolation to compute the field in the face center we normally write:

$$\phi_f = \phi_E\, k_x + (1 - k_x)\phi_C \tag{7.17}$$

In reality the previous expression is only partially correct if the skewness is positive; actually we are calculating ϕ_{f_i} instead of ϕ_f; so the more appropriate equation for ϕ_f is reported in 7.18.

$$\phi_f = \phi_{f_i} + \mathbf{m} \cdot \nabla\phi = \phi_E \, k_x + (1 - k_x)\phi_C + \mathbf{m} \cdot \nabla\phi \tag{7.18}$$

The error that we commit is called truncation error and it is proportional to the value of the skewness.

$$\varepsilon_{\mathbf{T}} = \Psi(\Delta x)^m \frac{\partial^{m+n}\phi}{\partial x^{m+n}} \tag{7.19}$$

A mesh with high skewness introduces an error when discretizing every convective flux.

7.2.4 Gradient discretization

Applying and combining the rules previously explained it is possible to discretize the term that are present in the Navier-Stockes equations. The gradient term appers as gradient of the pressure in the momentum equation.

$$\nabla\phi \qquad \Rightarrow \qquad \int_\Omega \nabla\phi \, \mathrm{d}V \tag{7.20}$$

Applying the Gauss theorem we can pass from volume to surface integral:

$$\begin{aligned} \int_\Omega \nabla\phi \, \mathrm{d}V &= \int_{\delta\Omega} \phi \, \mathrm{d}\mathbf{S} = \int_S \phi \, \mathrm{d}\mathbf{S} \\ &= \sum_k^{\text{faces}} \int_{S_k} \phi \, \mathrm{d}\mathbf{S} = \sum_k^{\text{faces}} \phi_{f_k} \mathbf{S}_k \end{aligned} \tag{7.21}$$

And for each face S_k we interpolate the field ϕ_{f_k} with the same interpolation schemes explained before.

7.2.5 Diffusion discretization

The diffusion term is the divergence of the gradient of a field.

$$\nabla \cdot (\Lambda\nabla\phi) \qquad \Rightarrow \qquad \int_\Omega \nabla \cdot (\Lambda\nabla\phi) \, \mathrm{d}V \tag{7.22}$$

Also in this case the Gauss Theorem let us to convert the volume integral into a surface integral, that we can manage more easily.

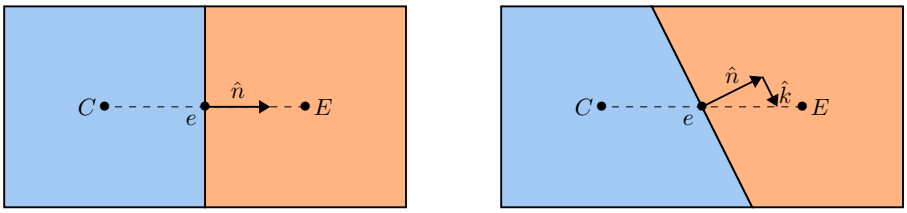
$$\begin{aligned} \int_\Omega \nabla \cdot (\Lambda\nabla\phi) \, \mathrm{d}V &= \int_S \Lambda\nabla\phi \cdot \hat{\mathbf{n}} \, \mathrm{d}S = \sum_k^{\text{faces}} \int_{S_k} \Lambda\nabla\phi \cdot \hat{\mathbf{n}} \, \mathrm{d}S \\ &= \sum_k^{\text{faces}} \Lambda_{f_k} \left(\nabla\phi_{f_k} \cdot \mathbf{S}_k\right) = \sum_k^{\text{faces}} \Lambda_{f_k} \, S_k \left(\nabla\phi_{f_k} \cdot \hat{\mathbf{n}}\right) \end{aligned} \tag{7.23}$$

If the surface is orthogonal to the direction of line linking the center cell and its neighbor equation 7.23 can be simplified as:

$$\int_\Omega \nabla \cdot (\Lambda\nabla\phi) \, \mathrm{d}V = \sum_k^{\text{faces}} \Lambda_{f_k} \left(\nabla\phi_{f_k} \cdot \mathbf{S}_k\right) = \sum_k^{\text{faces}} \Lambda_{f_k} \, S_k \frac{\phi_E - \phi_C}{\Delta x} \tag{7.24}$$

Non-orthogonality For an orthogonal mesh like in figure 7.4(a), expression 7.24 is perfectly fine, but in general this is not the case since to simulate complex geometry, non-orthogonality is always present.
To reduce the approximation error of equation 7.24, two strategies can be applied:

- build a mesh with as low as possible non-orthogonality;
- when it is not possible, a non-orthogonal correction term can be added into the equation.



(a) 2D Orthogonal mesh (b) 2D Non-orthogonal mesh

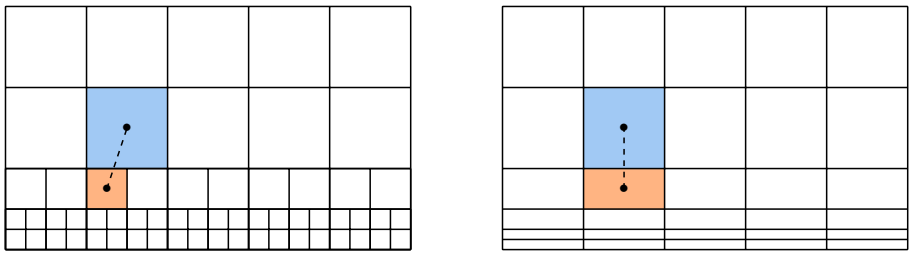
Figure 7.4: Comparison between an orthogonal and a non-orthogonal mesh.

In case of non-orthogonal correction the scalar product between the gradient of the field and the surface becomes:

$$\nabla\phi_{f_k} \cdot \mathbf{S}_k = S_k \frac{\phi_E - \phi_C}{\Delta x} + \hat{\mathbf{k}} \cdot \nabla\phi_{f_k} \tag{7.25}$$

Even if equation 7.25 seems similar to equation 7.24, there is a significant difference between the two, since the first is implicit and requires iterating to get a solution.

The mesh refinement process near the walls can introduce unexpected non-orthogonality even in an apparently orthogonal mesh grid.



(a) Mesh Refinement (b) Better Mesh Refinement

Figure 7.5: Comparison between two different ways to refine the mesh.

7.3 Navier-Stockes equations

The dynamic of a fluid is described by a set of partial derivative differential equation. The core of the problem are three equations of conservation, of mass, momentum and

energy, at which we have to add the state equation of the fluid.

In general the conservation equation of a scalar quantity ϕ can be written as:

$$\frac{\partial}{\partial t}\int_{\Omega}\phi\,\mathrm{d}V = \text{Fluxes} + \text{Sources} \tag{7.26}$$

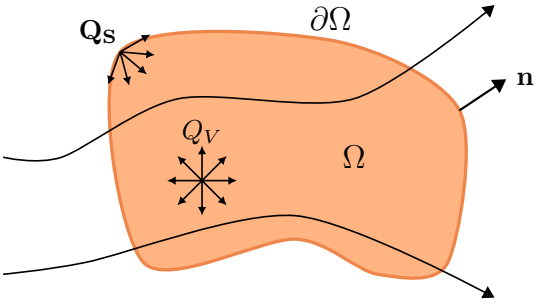


Figure 7.6: Control volume.

Conservation of a scalar field Considering the fluxes positive when entering into the control volume and distinguishing between surface and volume source we can rewrite equation 7.26 as:

$$\frac{\partial}{\partial t}\int_{\Omega}\phi\,\mathrm{d}V + \oint_{S}\mathbf{F}\cdot\hat{\mathbf{n}}\,\mathrm{d}S = \int_{\Omega}Q_v\,\mathrm{d}V + \oint_{S}\mathbf{Q}_s\cdot\hat{\mathbf{n}}\,\mathrm{d}S \tag{7.27}$$

Accordingly to the Gauss theorem the surface integrals can be rewritten in terms of volume:

$$\oint_{S}\mathbf{F}\cdot\hat{\mathbf{n}}\,\mathrm{d}S = \int_{\Omega}\nabla\cdot\mathbf{F}\,\mathrm{d}V \quad \text{and} \quad \oint_{S}\mathbf{Q}_s\cdot\hat{\mathbf{n}}\,\mathrm{d}S = \int_{\Omega}\nabla\cdot\mathbf{Q}_s\,\mathrm{d}V \tag{7.28}$$

And substituting back into the original equation and collecting all the terms under the same volume integral we obtain the differential conservative form of the equation.

$$\frac{\partial\phi}{\partial t} + \nabla\cdot\mathbf{F} = Q_v + \nabla\cdot\mathbf{Q}_s \tag{7.29}$$

Conservation of a vector field When we are dealing with a vector field, from the conceptual point of view equation 7.26 still holds, just replacing the scalar with a vector. The only other difference is that the field that before are vector like the flux F now becomes tendors. The differential conservative form of the conservation of a vector field is reported in equation 7.30.

$$\frac{\partial\boldsymbol{\phi}}{\partial t} + \nabla\cdot\boldsymbol{F} = \mathbf{Q}_v + \nabla\cdot\boldsymbol{Q}_s \tag{7.30}$$

Mass equation The mass equation is the conservation of the density field.

$$\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\,\mathbf{U}) = 0 \tag{7.31}$$

As we can notice in equation 7.31 there is no diffusion term, and actually the mass does *not* diffuse.

Momentum equation The momentum equation is the conservation of the flux $\phi\,\mathbf{U}$.

$$\frac{\partial(\rho\mathbf{U})}{\partial t} + \nabla\cdot(\rho\mathbf{U}\otimes\mathbf{U}) + \nabla\cdot(p\boldsymbol{I} - \boldsymbol{\tau}) = \rho\,\mathbf{f}_{\text{ext}} \tag{7.32}$$

$$\frac{\partial(\rho\mathbf{U})}{\partial t} + \nabla\cdot(\rho\mathbf{U}\otimes\mathbf{U}) = -\nabla p - \nabla\cdot\boldsymbol{\tau} + \rho\,\mathbf{g} \tag{7.33}$$

In most of the cases the only internal force is the gravitational force. As we can notice in equation 7.32 there is no diffusion term, and actually the momentum does *not* diffuse. The viscous stress tensor $\boldsymbol{\tau}$ can be evaluated for Newtonian fluids as:

$$\boldsymbol{\tau} = \mu\left(\nabla\mathbf{U} + (\nabla\mathbf{U})^{\mathrm{T}} - \frac{2}{3}\boldsymbol{I}\nabla\cdot\mathbf{U}\right) \tag{7.34}$$

The difficulties in solving the momentum equations mainly comes from the inertia term $\nabla\cdot(\rho\mathbf{U}\otimes\mathbf{U})$ which introduces non-linearity into the problem.

Energy equation Two equivalent versions of the energy equation exists, the first one is the conservation of the internal energy ρe and the second one that of the enthalpy ρh.

$$\frac{\partial(\rho e)}{\partial t} + \nabla\cdot(\rho e\mathbf{U}) = \nabla\cdot(k\nabla T) + \nabla\cdot(\boldsymbol{\sigma}\cdot\nabla\mathbf{U}) + \rho\mathbf{f}_{\text{ext}}\cdot\mathbf{U} + q_{\text{heat}} \tag{7.35}$$

$$\frac{\partial(\rho h)}{\partial t} + \nabla\cdot(\rho h\mathbf{U}) = \nabla\cdot(k\nabla T) + \nabla\cdot(\boldsymbol{\tau}\cdot\nabla\mathbf{U}) + \rho\mathbf{f}_{\text{ext}}\cdot\mathbf{U} + q_{\text{heat}} \tag{7.36}$$

In this case we have that in the energy equation the temperature diffuse.

7.4 Turbulence

Most of the flows in nature and in engineering interest are turbulent. A proper understand of the physics and the numeric is necessary to deal with practical problem and obtain significant results.

Definition 7.5 (Turbulence) *is a three-dimensional, unsteady, rotational fluid motion with broad-banded fluctuations of flow quantities (velocity, pressure, temperature, etc...) occurring in both time and space.*

Kundu and Cohen identifies in the following list the main physical characteristics of turbulence [28].

- randomness;
- non-linearity;
- diffusivity;
- vorticity;

- dissipation.

Theoretically the equations may be solved neglecting the turbulence behavior to obtain an approximated solution much easily. The problem is that the turbulence is highly enhancing mass, momentum and energy diffusion which are some of the characteristic of interest in when we run CFD simulations.

If we want to fully simulate all the turbulence scales, we need to know how much small the cells and the time step have to be. The answer to this question was provided by Kolmogorov in its energy cascade theory. In turbulence the energy is transferred from the larger eddies to the smaller, and how much energy is dissipated depending on the vortex size. Large eddies are unstable and easily break-up, transferring their energy to smaller ones. Even this smaller eddies undergo the same break up process transferring energy to yet smaller eddies. This continues until the Reynolds number of the eddy is small enough that it is stable, and finally molecular viscosity effectively dissipates the kinetic energy.

Kologorov [29] identified three different regions in the energy spectrum $E(\kappa)$, represented in figure 7.7:

- the energy containing range;
- the inertial subrange;
- the dissipation range.

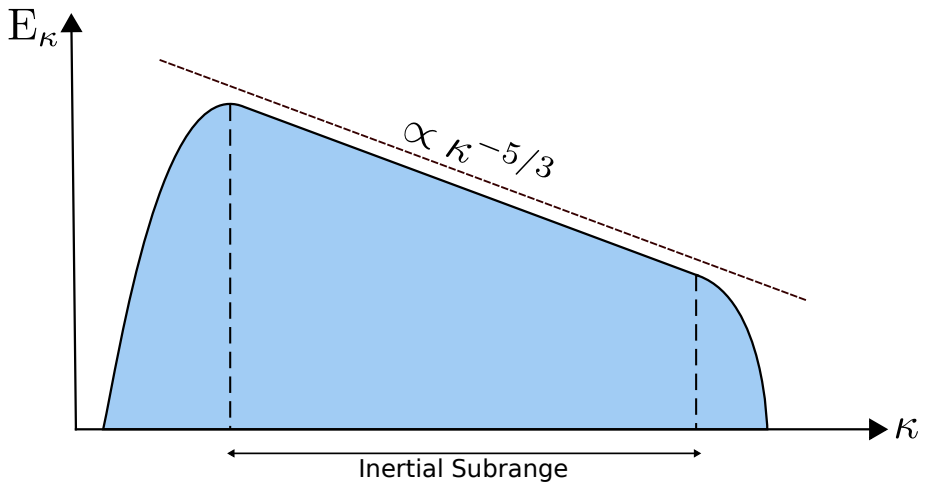


Figure 7.7: Kolmogorov energy cascade in a log-log plot.

To fully simulate the turbulence we must have cells smaller than the smallest length scale and time steps smaller that the smallest time scale.

The results shown in equation 7.37 are chilling since for a medium scale turbulence with Reynolds number of a fully turbulent case the computational power required to get

a solution in a reasonable time cannot be managed even by the best supercomputers available.

$$\eta = \left(\frac{\nu^3}{\varepsilon}\right)^{1/4} = l\, Re^{-3/4} \quad \text{and} \quad \tau_\eta = \left(\frac{\nu}{\varepsilon}\right)^{1/2} = t\, Re^{-1/2} \tag{7.37}$$

where: η is the Kolmogorov length scale;
τ_η is the Kolmogorov time scale;
ε is the average rate of dissipation of turbulence kinetic energy per unit mass;
ν is is the kinematic viscosity of the fluid.

Without a prohibitively refined mesh and time step the turbulence cannot be simulated, and a modeling effort is necessary. This is one of the biggest issues in computational fluid dynamic field. Actually simulating it, is much easier from the point of view of the implementation and validation effort, but it is unfeasible.

Several ways to handle the turbulence modeling has been developed, with different levels of accuracy and computational cost.

- Direct Navier Simulations (DNS): simulation of all the turbulence scales;
- Large Eddies Simulations (LES): simulation up to the viscous sub-range;
- Reynolds-averaged Navier-Stokes equations (RANS): simulation of the largest scales, and modeling of all the others.

The DNS approach is almost never applied, the LES are implemented in some advanced research environment and the RANS are common into industrial field.

7.5 Reynolds-averaged Navier-Stokes equations (RANS)

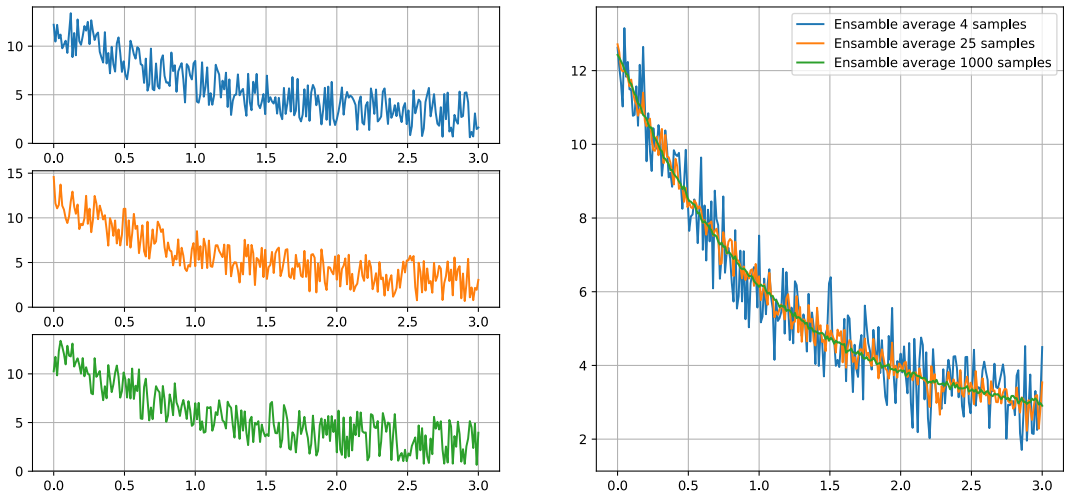
Since we cannot manage the whole complexity of the turbulence we only consider its gross characteristic with a statistical approach. The basic principle beyond the RANS equations is to solve the Navier-Stokes equation after performing the ensemble average of all the variables.

Definition 7.6 (Ensemble average) *is defined as the collection of an infinite number of experiments performed with the same conditions, at which we apply the average point by point and time by time along the different experiments.*

$$\langle u(x,t)\rangle = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} u(x, t:n) \tag{7.38}$$

Reducing the ensemble average to a mere time or space average of a single experiments is a *crucial simplification.*

The great advantage of this kind of average to the simple time average is that the former can manage both steady-state and unsteady phenomena, while the latter is effective just for steady-state flows.



(a) Different measurements

(b) Ensamble average

Figure 7.8: Example of the application of the ensemble average.

Definition 7.7 (Time average) *is defined as the average of a quantity of a single simulation in a specific point, time by time.*

$$\overline{u(x,t)} = \frac{1}{\Delta t}\int_{t-\Delta t/2}^{t+\Delta t/2} u(x,t)\,\mathrm{d}t \tag{7.39}$$

The identification between ensemble and time averages is particularly feasible for statistically stationary processes, whose statistics are invariant to a time shift.

$$\langle u(x,t)\rangle = \langle u(x,t+\Delta t)\rangle = \overline{u(x,t)} = \overline{u(x)} \tag{7.40}$$

For sake of simplicity we will show how to obtain the RANS equations for an incompressible fluid with the time average. From the syntactic point of view we can replace the time average symbol with the ensamble average one. Any property ϕ can be seen as the sum of a mean and a fluctuating component.

$$\phi = \overline{\phi} + \phi' \tag{7.41}$$

where: $\overline{\phi} = \dfrac{1}{t}\displaystyle\int_t \phi(t)\,\mathrm{d}t$

In the compressible case the procedure is really similar from the theoretical point of view, the difference is to consider the Favre mass-weighted quantities:

$$\phi = \frac{\overline{\rho\phi}}{\overline{\rho}} \tag{7.42}$$

We want to take the average of the Navier-Stokes equations, but before to start it is convenient to remember some properties of the average:

$$\overline{\phi+\psi}=\overline{\phi}+\overline{\psi} \qquad \overline{\lambda\psi}=\lambda\overline{\psi} \qquad \overline{\int \phi\,\mathrm{d}x}=\int \overline{\phi}\,\mathrm{d}x \qquad \overline{\frac{\partial\phi}{\partial t}}=\frac{\partial\overline{\phi}}{\partial t}$$

$$\overline{\phi'}=0 \qquad \overline{\phi'\cdot\phi}=\overline{\phi\cdot\phi'}=0 \qquad \overline{\phi'\cdot\phi'}\neq 0 \qquad \overline{\phi'^{n}}\neq 0 \qquad \overline{\overline{\phi}}=\overline{\phi}$$

Since the mass equation is linear and we are considering an incompressible fluid, the averaging operation is straightforward.

$$\overline{\frac{\partial\rho}{\partial t}+\nabla\cdot(\rho\,\mathbf{U})}=\frac{\partial\rho}{\partial t}+\nabla\cdot\left(\rho\,\overline{\mathbf{U}}\right)=0 \tag{7.43}$$

The application of the average to the momentum equation requires instead a particular attention to the non-linear term.

$$\overline{\frac{\partial\left(\rho\mathbf{U}\right)}{\partial t}+\nabla\cdot(\rho\mathbf{U}\otimes\mathbf{U})}=\overline{-\nabla p-\nabla\cdot\boldsymbol{\tau}+\rho\,\mathbf{g}} \tag{7.44}$$

$$\frac{\partial\left(\rho\overline{\mathbf{U}}\right)}{\partial t}+\overline{\nabla\cdot(\rho\mathbf{U}\otimes\mathbf{U})}=-\nabla\overline{p}-\nabla\cdot\overline{\boldsymbol{\tau}}+\rho\,\mathbf{g} \tag{7.45}$$

The average of the inertia term is more complex, so we analyze it separately. The velocity $\mathbf{U}$ can be seen as the sum of two components, one mean and one fluctuating $\mathbf{U}=\overline{\mathbf{U}}+\mathbf{U}'$. Substituting the previous expression into the inertia term we obtain:

$$\overline{\nabla\cdot(\rho\mathbf{U}\otimes\mathbf{U})}=\overline{\nabla\cdot\left(\rho(\overline{\mathbf{U}}+\mathbf{U}')\otimes(\overline{\mathbf{U}}+\mathbf{U}')\right)} \tag{7.46}$$

$$=\overline{\nabla\cdot\left(\rho(\overline{\mathbf{U}}\otimes\overline{\mathbf{U}}+2\overline{\mathbf{U}}\otimes\mathbf{U}'+\mathbf{U}'\otimes\mathbf{U}')\right)} \tag{7.47}$$

$$=\overline{\nabla\cdot\left(\rho\overline{\mathbf{U}}\otimes\overline{\mathbf{U}}\right)}+\overline{\nabla\cdot\left(2\rho\overline{\mathbf{U}}\otimes\mathbf{U}'\right)}+\overline{\nabla\cdot\left(\rho\mathbf{U}'\otimes\mathbf{U}'\right)} \tag{7.48}$$

But remembering the previous mean relationship we obtain:

$$\overline{\rho\overline{\mathbf{U}}\otimes\overline{\mathbf{U}}}=\rho\overline{\mathbf{U}}\otimes\overline{\mathbf{U}} \tag{7.49}$$

$$\overline{\rho\overline{\mathbf{U}}\otimes\mathbf{U}'}=0 \tag{7.50}$$

$$\overline{\rho\mathbf{U}'\otimes\mathbf{U}'}=\rho\overline{\mathbf{U}'\otimes\mathbf{U}'}\neq 0 \tag{7.51}$$

From the non linear inertia term we obtain two terms, one related to the mean value of the velocity, and one only related to the oscillating component of $\mathbf{U}$. We don't know anything about the fluctuating components, and actually this term will require a specific modeling effort.

The average momentum equation of the RANS is explicated in equation 7.52.

$$\frac{\partial\left(\rho\overline{\mathbf{U}}\right)}{\partial t}+\nabla\cdot\left(\rho\overline{\mathbf{U}}\otimes\overline{\mathbf{U}}\right)+\nabla\cdot\left(\rho\overline{\mathbf{U}'\otimes\mathbf{U}'}\right)=\rho\,\mathbf{g}-\nabla\overline{p}+\mu\nabla^{2}\overline{\mathbf{U}}+\frac{1}{3}\mu\nabla\left(\nabla\cdot\overline{\mathbf{U}}\right) \tag{7.52}$$

The fluctuating term is called Reynolds Stress Tensor and it is a symmetric tensor which contains the fluctuating component of the velocity field times the density. Defining the fluctuating velocity vector $\mathbf{U'} = u'\,\hat{\mathbf{i}} + v'\,\hat{\mathbf{j}} + w'\,\hat{\mathbf{k}}$ the Reynolds stress tensor is:

$$\boldsymbol{r} = -\rho\overline{\mathbf{U'} \otimes \mathbf{U'}} = -\rho \begin{bmatrix} \overline{u'^2} & \overline{u'v'} & \overline{u'w'} \\ \overline{u'v'} & \overline{v'^2} & \overline{v'w'} \\ \overline{v'w'} & \overline{v'w'} & \overline{w'^2} \end{bmatrix} \tag{7.53}$$

where: $\mathrm{tr}(\boldsymbol{r}) = -\rho\left(\overline{u'^2} + \overline{v'^2} + \overline{w'^2}\right) = -2\rho\kappa$;
κ = the turbulent kinetic energy.

The averaging of the energy equation is similar to the equation since it contains non linear terms.

$$\overline{\frac{\partial(\rho e)}{\partial t} + \nabla\cdot(\rho e\mathbf{U})} = \overline{\nabla\cdot(k\nabla T) + \rho\,\mathbf{g}\cdot\mathbf{U} - p\,\nabla\cdot\mathbf{U} + \varepsilon + q_{\text{heat}}} \tag{7.54}$$

where: $\varepsilon = \boldsymbol{\tau} : \nabla\mathbf{U}$ is the dissipation function;

We will neglect the deformation work $\varepsilon - p\,\nabla\cdot\mathbf{U}$, the external heat and the gravitational force field. Even though this simplify the equation, the problematic non-linear term $\nabla\cdot(\rho e\mathbf{U})$ still survive. We treat it in the same way as before and the resultant energy averaged equation is written in 7.55 or replacing the internal energy e with the thermodynamic relation $e = c\,T$ in 7.56.

$$\frac{\partial(\rho\overline{e})}{\partial t} + \nabla\cdot\left(\rho\overline{e}\,\overline{\mathbf{U}}\right) = \nabla\cdot\left(k\nabla\overline{T}\right) - \nabla\cdot\left(\rho\overline{e'\mathbf{U'}}\right) \tag{7.55}$$

$$\frac{\partial\left(\rho c\overline{T}\right)}{\partial t} + \nabla\cdot\left(\rho c\overline{T}\,\overline{\mathbf{U}}\right) = \nabla\cdot\left(k\nabla\overline{T}\right) - \nabla\cdot\left(\rho c\overline{T'\mathbf{U'}}\right) \tag{7.56}$$

In analogy with the Reynolds stress tensor the term $\rho c\overline{T'\mathbf{U'}}$ is called *turbulent heat flux*, which represents how the turbulent transport enhance the diffusivity respect to the molecular one.

Both the Reynolds stress tensor and the turbulent heat flux lead a closure problem, since no information on how these terms can be expressed derives from the averaging procedure.

7.5.1 Closure solutions

In the literature a lot of different solutions exist, many of that performing well in some specific kinds of problems. The explanation on this thesis do not want to be extensive or complete, and it is just a summary of the main available methods.

The aim of the turbulence models is to find a way to evaluate the 6 unknowns of the symmetric Reynolds stress tensor $\boldsymbol{r}$.

We can generally classify the methods in three categories of increasing complexity:

- algebraic turbulence models;
- one-equation turbulence models;
- two-equations turbulence models.

Algebraic Turbulence Models

Their main feature is that they do not require the solution of any additional differential equations, and for this reason they are pretty basic. These models usually use the Boussinesq eddy viscosity approximation to compute the Reynolds stress tensor $\boldsymbol{r}$, which with some simplified hypothesis reduces to:

$$-\rho\overline{u'v'} = \mu_{\mathrm{T}}\frac{\partial U}{\partial y} \tag{7.57}$$

From the dimensional analysis the turbulent viscosity μ_{T} is proportional to the product between a characteristic turbulence length l_{T} and a turbulence velocity $\mathrm{U_T}$.

$$\mu_{\mathrm{T}} \propto l_{\mathrm{T}} \cdot \mathrm{U_T} \tag{7.58}$$

Most of them rely on the Prandtl's mixing length hypothesis which states that an eddy of dimension l_{T} driven by the mean shear stress causes a velocity fluctuation proportional to the velocity gradient and the turbulence length l_{T}.

$$\mathrm{U_T} \propto l_{\mathrm{T}}\frac{\partial U}{\partial y} \quad \Rightarrow \quad \mu_{\mathrm{T}} \propto l_{\mathrm{T}} \cdot \mathrm{U_T} \propto l_{\mathrm{T}}^2\frac{\partial U}{\partial y} \tag{7.59}$$

The Reynolds stress tensor component can be written as:

$$-\rho\overline{u'v'} = \mu_{\mathrm{T}}\frac{\partial U}{\partial y} \propto \left(l_{\mathrm{T}}\frac{\partial U}{\partial y}\right)^2 \tag{7.60}$$

We report as an example the Baldwin and Lomax [30] algebraic turbulent model in equation 7.61.

$$\mu_{\mathrm{T}} = \rho\, l_{\mathrm{T}}^2\sqrt{2\boldsymbol{R}:\boldsymbol{R}} \tag{7.61}$$

where: $\boldsymbol{R} = \frac{1}{2}\left(\nabla\mathbf{U} - \nabla\mathbf{U}^{\mathrm{T}}\right)$

One-equation Turbulence Models

One equation turbulence models introduce the solution of one turbulent transport equation, usually the turbulent kinetic energy.

An improvement in this field comes from Spalart and Allmaras [31] which overcome the necessity of expressing the turbulent length l_{T} introducing a transport equation

directly to the quantity that they need, the eddy viscosity. The so called Spalart-Allmaras model, or SA model equation is reported in 7.62.

$$\rho \frac{\partial \mu_{\mathrm{T}}}{\partial t} + \rho \overline{\mathbf{U}} \cdot \nabla \mu_{\mathrm{T}} = \nabla \cdot (\mu_{\mathrm{T}} \nabla \mu_{\mathrm{T}}) + \text{sources} \tag{7.62}$$

The model was designed for solving aerodynamic problems in transonic and supersonic regime, and behaves reasonably well in turbo-machinery applications, but it has poor performances with strong adverse pressure gradients.

Two-equation Turbulence Models

In this category the two main approaches are:

- $\kappa - \varepsilon$ turbulent model;
- $\kappa - \omega$ turbulent model.

The variables in the $\kappa - \varepsilon$ are the turbulent kinetic energy κ and the dissipation rate ε previously defined. The ω is instead called *turbulent frequency* and is the reciprocal of the turbulence time scale.

Both the models evaluate the κ balance equation as in many one-equation case, but introduces one additional differential equation for or the dissipation rate ε or the turbulent frequency ω.

All the equation for κ, ε and ω can be generically written as:

$$\rho \frac{\partial \phi}{\partial t} + \rho \overline{\mathbf{U}} \cdot \nabla \phi = \text{Production} + \text{Dissipation} + \text{Transport} \tag{7.63}$$

For sake of completeness the three differential equations are reported in 7.64, 7.65 and 7.66.

$$\rho \frac{\partial \kappa}{\partial t} + \rho \overline{\mathbf{U}} \cdot \nabla \kappa = \boldsymbol{r} : \nabla \mathbf{U} - C_D \rho \frac{\kappa \sqrt{\kappa}}{l_{\mathrm{T}}} + \nabla \cdot \left(\left(\mu + \frac{\mu_{\mathrm{T}}}{\sigma_k} \right) \nabla \kappa \right) \tag{7.64}$$

$$\rho \frac{\partial \varepsilon}{\partial t} + \rho \overline{\mathbf{U}} \cdot \nabla \varepsilon = C_{\varepsilon_1} \frac{\varepsilon}{\kappa} \boldsymbol{r} : \nabla \mathbf{U} - C_{\varepsilon_2} \rho \frac{\varepsilon^2}{\kappa} + \nabla \cdot \left(\left(\frac{\mu}{\rho} + \frac{\mu_{\mathrm{T}}}{\rho \sigma_\varepsilon} \right) \nabla \varepsilon \right) \tag{7.65}$$

$$\rho \frac{\partial \omega}{\partial t} + \rho \overline{\mathbf{U}} \cdot \nabla \omega = C_{\omega_1} \frac{\omega}{\kappa} \boldsymbol{r} : \nabla \mathbf{U} - C_{\omega_2} \rho \omega^2 + \nabla \cdot \left(\left(\frac{\mu}{\rho} + \frac{\mu_{\mathrm{T}}}{\rho \sigma_{\omega_3}} \right) \nabla \omega \right) \tag{7.66}$$

Although equations 7.65 and 7.66 seems really similar one to each other, there is actually a big difference between the two turbulence models. The former goes to infinity when approaching the wall, so wall functions are required to obtain a reasonable solution, the latter instead is able to solve even flow detachment, but has the opposite problem, going to infinity far from the wall, being unable to correctly model free-stream turbulence.

From advantages and limitations of the two models, in 1994 Menter [32] proposed a modified version called *Shear Stress Transport* or $\kappa - \omega$ SST which combines both the κ and ω equations in to a single one that contains a blending factor, and depending on the wall distance is able to pass between the two. He replaced ω into the ε equation, obtaining an equation very similar to the original but with an additional term. That term is really important since modulating it with a proper blending factor let to pass between the two models depending on the needs. In equation 7.67 is reported the expression for the second equation for the $\kappa - \omega$ SST model.

$$\rho \frac{\partial \omega}{\partial t} + \rho \overline{\mathbf{U}} \cdot \nabla \omega = C_{\omega_1} \frac{\omega}{\kappa} \boldsymbol{r} : \nabla \mathbf{U} - C_{\omega_2} \rho \omega^2 + \nabla \cdot \left(\left(\frac{\mu}{\rho} + \frac{\mu_{\mathrm{T}}}{\rho \sigma_{\omega_3}} \right) \nabla \omega \right) + (1 - F) \frac{2 \nabla \kappa \nabla \omega}{\sigma_{\omega_2} \omega} \tag{7.67}$$

$\kappa - \omega$ SST is currently the standard turbulence model in industrial turbo-machinery RANS simulations, and it is the one chosen also in this work.

To get the most out of the model a regular structure *o-grid* is required along the wall that let to easily compute the distance to the wall of the cells inside the boundary layer. A further advantage of the o-grid mesh, is that it let to properly distribute the cells near a wall or a blade to get reliable results from the simulation of the boundary layer.

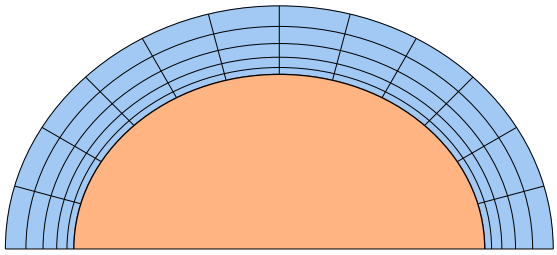


Figure 7.9: Example of an o-grid mesh.

7.6 Boundary Layers

In not hypersonic flows, which is always the case in turbo-machinery the *no slip condition* holds. It ensures that the fluid at the wall has its same velocity; in the case in which the wall is stationary, the velocity of the fluid in that position is null.

After this statement is clear that there must be a region where the velocity passes from the free stream value to zero; the region close to the wall, where there are strong velocity gradients, is called *boundary layer*, and the velocity profile there it is in general independent from the free stream condition. The fluid there cannot be described with the classical Reynolds number theory.

Approximating the fluid in the boundary layer with the wall bounded shear-flow model, which is almost the case when the flow is attached to the wall, we obtain a

simplified equation for the velocity:

$$\left(\frac{\mu}{\rho} + l_{\mathrm{T}}^2 \frac{\partial U}{\partial y}\right) \frac{\partial U}{\partial y} = \tau_{\mathrm{wall}} \tag{7.68}$$

where: l_{T} is the turbulent mixing length;
y is the coordinate normal to the wall;
U is velocity component parallel to the wall;
τ_{wall} is shear stress at the wall;

With the introduction of dimensionless quantities below, we can non-dimensionalize equation 7.68 to obtain a simpler equation that we can solve in special regions.

$$U_\tau = \sqrt{\frac{\tau_{\mathrm{wall}}}{\rho}} \qquad \text{(Friction velocity)}$$

$$\delta_v = \frac{\mu}{\rho U_\tau} \qquad \text{(Viscous lenght scale)}$$

The dimensionless quantities we obtain are the dimensionless wall distance $y^+ = y/\delta_v$ and the dimensionless velocity $U^+ = U/U_\tau$; with these, equation 7.68 becomes:

$$\left(1 + \left(l_{\mathrm{T}}^+\right)^2 \frac{\partial U^+}{\partial y^+}\right) \frac{\partial U^+}{\partial y^+} = 1 \tag{7.69}$$

Equation 7.69 can be easily solved in two different conditions, really close to the wall where $y^+ \to 0$ and far from the wall where $\left(l_{\mathrm{T}}^+\right)^2 \frac{\partial U^+}{\partial y^+} \gg 1$. The solution in this two extreme conditions are:

$$U^+ = \begin{cases} y^+ & \text{if} \quad 0 < y^+ < 5 \\ \dfrac{1}{k} \log y^+ + C & \text{if} \quad 30 < y^+ < 1000 \end{cases} \tag{7.70}$$

The region in between the two is called *buffer layer*. The analytical expression there is in general obtained interpolating between the *linear viscous sub-layer*, close to the wall, and the *logarithmic layer* further, or with a numerical method for solving implicit non-linear differential equations.

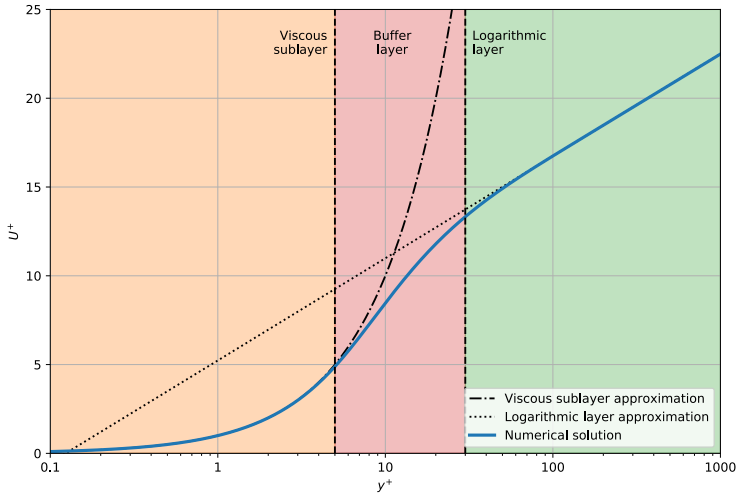


Figure 7.10: Boundary layer - dimensionless velocity profile.

Finding an analytical dimensionless expression for the velocity near the wall has a great importance in computational fluid dynamic, in particular for the $\kappa - \varepsilon$ turbulence model since we cannot solve the equations close to the wall and we need an alternative way to manage it, and this way can be a wall function. And even if the $\kappa - \omega$ turbulence model is particularly effective there, in regions where we do not expect separation and recirculation we may want to save computational time applying a wall function where it is possible.

7.7 The solution of the equations

The finite volume method can be applied in two different ways to discretize the Navier-Stokes equations:

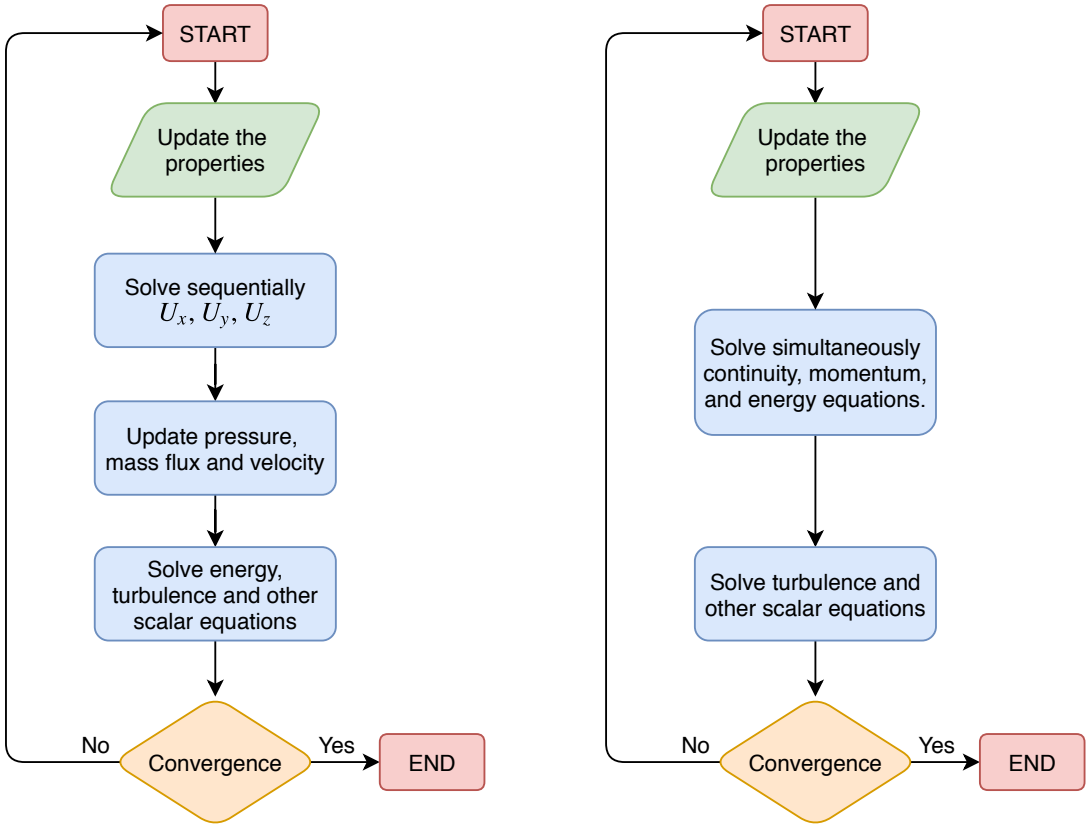
- segregated;
- coupled.

The former consist in solving the system of equations one by one, but since the equations are coupled, in particular the mass and the momentum ones, particular strategies have to be exploited to obtain a solution. Between them we cite the so called SIMPLE algorithm which is designed for steady-state incompressible flows, the PISO for transient incompressible flows and analogous solvers for compressible flows.

The latter instead solves all the equations together, considering as unknown the vector $\mathbf{U} = [\rho,\ \rho\mathbf{U},\ \rho e]^{\mathrm{T}}$ and directly discretizing equation 7.71 to obtain a single big matrix which contains all the unknowns in each cell.

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{F} - \nabla \cdot \mathbf{V} = \mathbf{R} \tag{7.71}$$

The coupled solver is particularly indicated for transonic and supersonic flows, exhibiting better performances respect to the segregated, while it has problem when the Mach number tends to zero, since in that case the density is no more function of the pressure.



(a) Segregated algorithm

(b) Coupled algorithm

Figure 7.11: Comparison of solution algorithms.

The Ansys commercial code implements a coupled solver and it has good performance for the case of study.

After in one way or in the other we build up the matrix of the system, then we have to solve a linear system in the form $\boldsymbol{A}\mathbf{x} = \mathbf{b}$, which in general has hundreds of thousands or even several millions of unknowns. Such big systems cannot be solved directly, because, even if direct methods exist, they are too computational expensive; so the only available approach that we can apply consists in iterative algorithms.

Between iterative methods we cite:

- *conjugate gradient methods*: they start from an initial guess and progressively search the solution in the best possible direction which is normal to the gradient; suffering from bad conditioning a matrix preconditioning method is required to obtain a solution. The most common techniques are the incomplete Cholesky

conjugate gradient (ICCG), the bi-conjugate gradient (BCG) and the conjugate gradient squared (CGS);

- *multigrid methods*: they are based on the fact that solving a bigger linear system requires not only more time for each iterations, which is trivial, but also more iterations. The idea is to accelerate the solution starting with the problem with a coarser mesh and only when we are closer to the solution, to solve the fine mesh, obtaining both a fast convergence and a good accuracy. The coarser mesh are in general extrapolated from the original one to avoid multiple meshing procedure.

Multi-grid methods are in general combined with iterative Gauss-Siedel and incomplete LU decomposition, and they are often used due to their advantage in required time.

Chapter 8

Blade Optimizazion Algorithm

In chapters 3, 4, 5, 6 and 7 we have introduced the theoretical explanation behind the optimization procedure. Many different mathematical and technical tools are necessary to drive an optimization application. In this chapter we will put together the knowledge necessary to highlight the steps that will build up the developed tool.

The software used for the whole procedure are essentially three:

- *Ansys cfx*: it is a commercial tool made available by Politecnico Di Milano. It is used to build the mesh from the geometry and to perform CFD simulations.
- *Dakota*: it is an open source framework above GPL (General Public License), defined by the authors Adams et Al. [33] as:

  > *A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis*

  In this work the Dakota framework manages the surrogate and the optimization algorithms.
- *Python*: the code written with open source Python language is the main part developed in this work. This is responsible for the parametrization of the blade and for it works as interface between the Dakota software and the CFX high-fidelity CFD code.

The software requires many input data provided or directly by hand by the user or generated through the Python script. In both cases however the user has to decide many parameters for the simulation, and the script can just reduce the effort of formatting correctly that data.

The inputs of the algorithms are:

- the baseline profile;
- parameters of the B-spline interpolation (number of control points, degree, number of points, ...);

- trailing edge shape;
- fixed and adjustable control points;
- bounds for the adjustable points;
- number of samples of the DOE for Kriging surrogate model;
- global or local surrogate approach;
- number of iteration of the optimization algorithm;
- genetic algorithm parameters (population size, mutation rate, ...);
- definition of the objective functions;
- definition of constraints and the bounds;
- definition of other functions that are useful for the designer but that are nor objectives or constraints;
- failure management strategy;
- State and Session files for the mesh generation;
- State and Session files for the pre-processing;
- Session files for the post-processing;
- number of cells and iterations of the solver, or eventually the maximum running time in case of solver based on time;
- kind of optimization (3D, both stator and rotor, multi-point, of the reference flow, ...);
- the number of cores and of concurrent number of simulations;
- other minor parameters for post-processing (how to save results and images, ...).

8.1 Pre-processing setup

As we have said before, to run the optimization algorithm we need to set-up the files required by the B-spline parametrization, by the solver and by Dakota. In this section we will explain the way to prepare the case to run the optimization.

8.1.1 Blade Interpolation

Even before to run the interpolation code to generate the data required by the optimization, the blade has to be carefully prepared. First at all the points has to be sorted, in the meaning that the points are ordered like they are linked together. In the current project this was unfortunately not the case. We will explain in chapter 8.1.1 the custom algorithm that has been developed to perform the sorting. This has been a problematic step due to the shape of the blade which is not fully convex, and deterministic algorithms only exist for a convex set of points; the relatively high number of sections makes the hand sorting process hard and time consuming.

Profiles order fix

In the extraction of the available profiles, some of that present partially unsorted points (Example in figure 8.1).

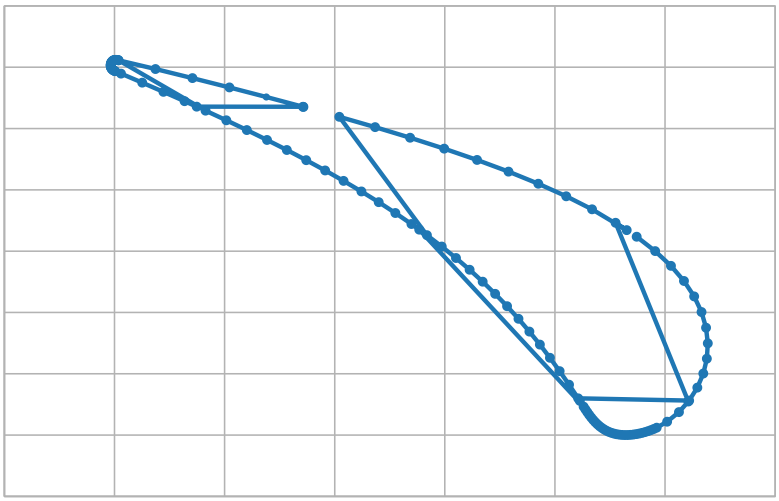


Figure 8.1: Example of unordered profile

Even if this seems a minor problem, no straightforward way of ordering such points exists, since the blade represents a *concave* polygon, in the pressure side region, close to the trailing edge.
For *convex* ones, points can be sorted by their angle to respect to the center of the figure, or alternatively the traveling salesman problem algorithm can be applied, even if it is quite more complex.

A simple approach for concave polygons, that works when there are not this regions, is to pick up a starting point and then start sorting the others taking every as new point, the closest to the previous one. Unfortunately even this approach fails since in the thinnest region of the blade, close to the trailing edge, some points are closer to the opposite side one respect to the right neighbor.

So a mixed approach has been developed, which starts from the points with the maximum x coordinate (approximately the leading edge) as first point and then continues adding points based on the distance with the previous one. The difference with the standard closest point method is that in case one point is far closer that the others that is added in the list, but if it is not the case, so more that one have similar distances to the previous one, the choice of the points depends on both the distance and the alignment with the latest direction.

Even if it does not seem a theoretically robust method it works properly with all the 21 profiles, and we managed to properly sort all the profiles.
The logical flow chart is displayed in figure 8.2

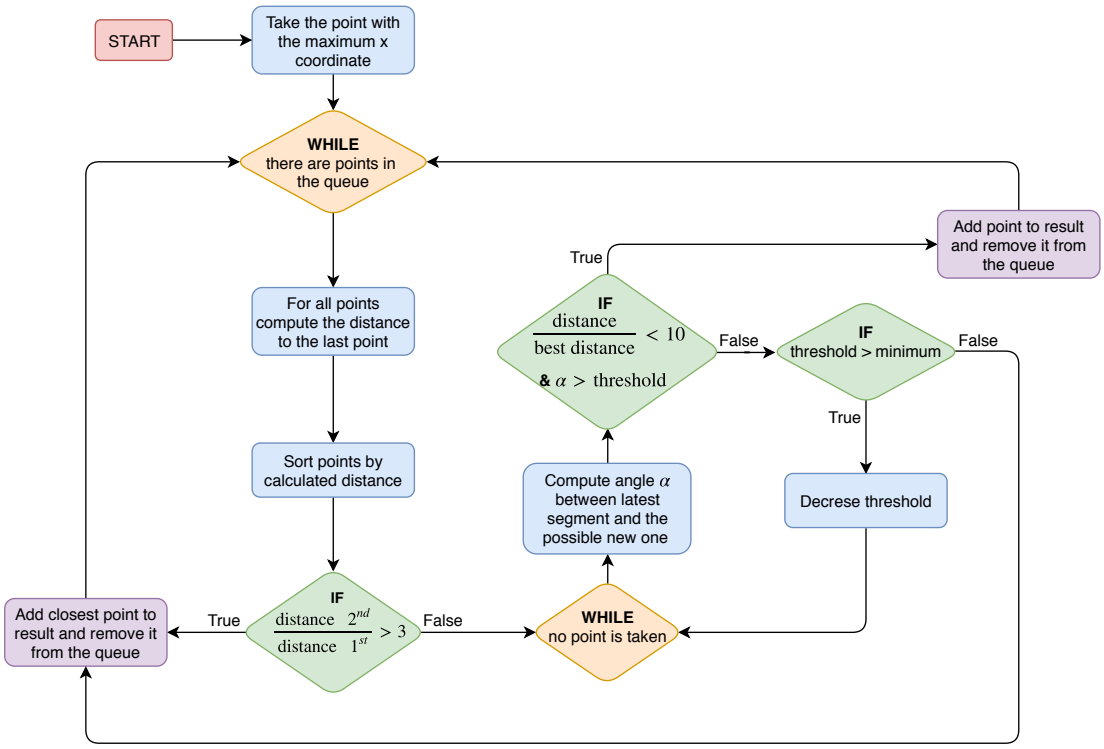


Figure 8.2: Schematic representation of the algorithm used to sort points of the blades.

Trailing edge management

For the trailing edge, basically two options exist: leaving trailing edge points directly in the input data and interpolate them with B-splines or removing that points and replace them with a circle or an ellipse shape. The latter is highly recommended since it makes the B-spline interpolation easier and much more accurate given the number of

control points; however a specific processing on the blade is often necessary to remove the trailing edge.

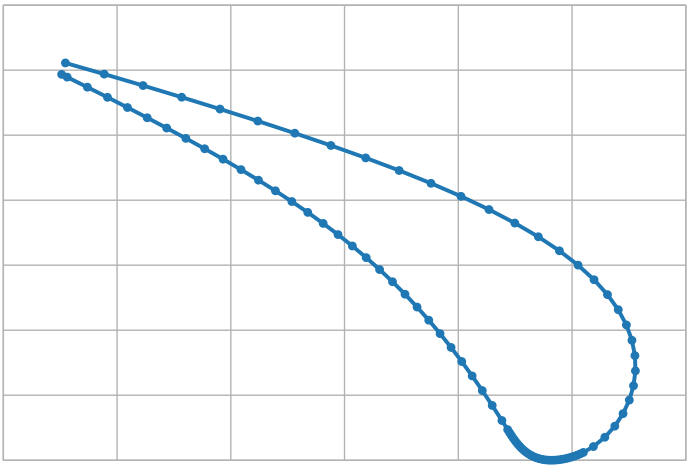


Figure 8.3: Example of a blade ready for the interpolation

Actually not only the points that belong to the trailing edge have to be removed, but a second sorting procedure is required; the first point has to be one of the linking points with the trailing edge, and the last one, the other.

Coordinate conversion

Depending on the kind of sections available a further step may be necessary; if the sections are cut with a Cartesian flat plane, and the points are provided in Cartesian coordinates, nothing more has to be done, otherwise we have to convert the points in the proper frame of reference. The parametrization algorithm works with a bi-dimensional profile, and in the current thesis the available sections are cylindrical cuts at constant radius of the original three-dimensional blade represented in Cartesian x, y, z coordinates. To bring them on a 2D plane with have to pass from the x, y, z to the $r\theta, z, R$ frame of reference with equations 8.1. Then after computing the 1100 points that describes the blade, we need to apply the opposite operation and get the points in x, y, z frame of reference required by TurboGrid, the CFX meshing program.

$$\begin{cases} R = \max(y) \\ r\theta = R \operatorname{atan} \dfrac{x}{y} \\ z = z \end{cases} \qquad \begin{cases} x = R \sin\theta \\ y = R \cos\theta \\ z = z \end{cases} \tag{8.1}$$

The logical procedure for passing from the raw data to the B-spline parameters required for the optimization procedure is summarized in figure 8.4, while a detailed mathematical overview is explained in chapter 3.6.

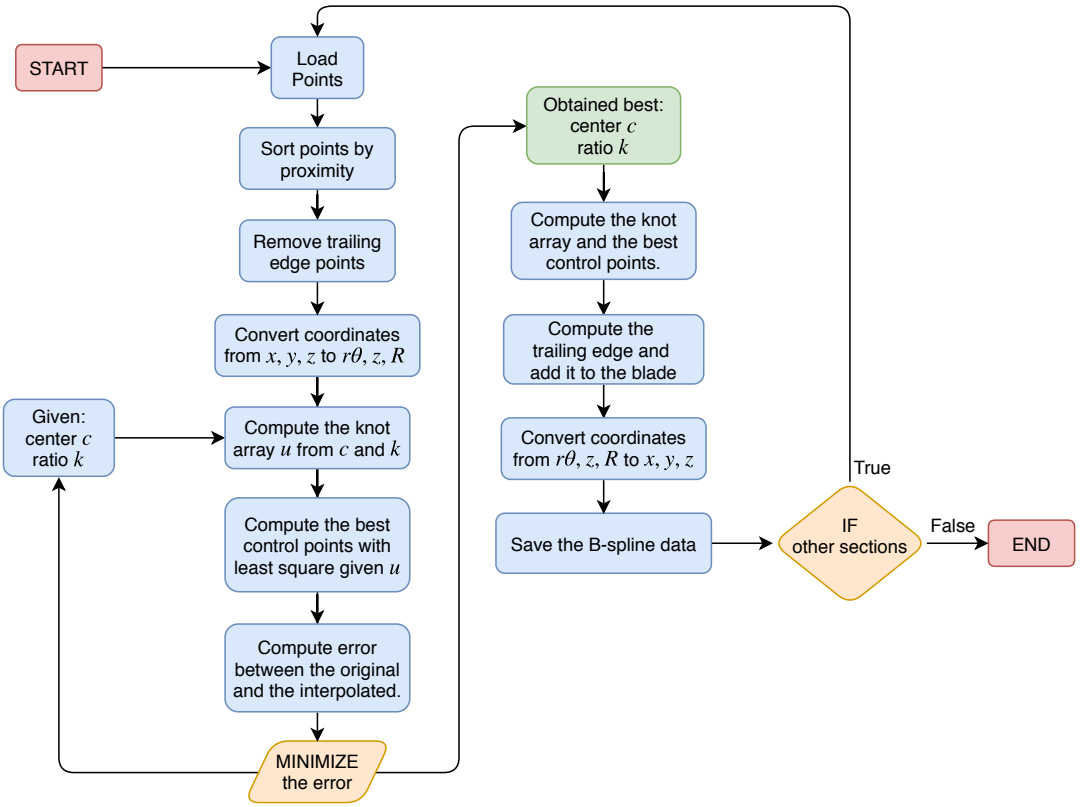
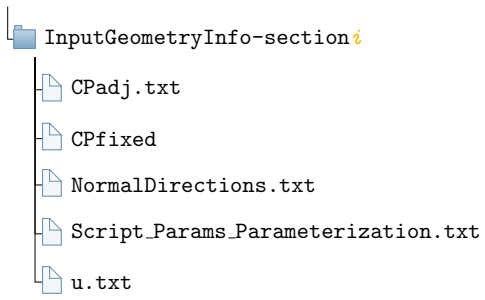


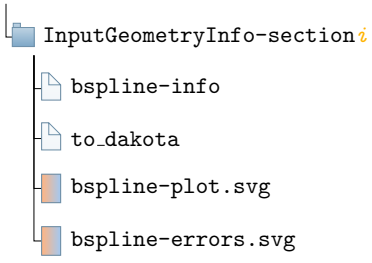
Figure 8.4: Logical flow-chart of the pre-processing of the blade data.

Parametrization folder structure

The final result of the procedure is the generation of a folder for each section given in input to the algorithm, which contains six files needed for the parametrization step, and other file that summarize the chosen value of ratio and center for the rational parametrization, the plot of the original and the interpolated blade and a sketch of the error distribution along the length of the curve. Then the file `to_dakota` contains the boundaries of the design variables that can be imported to the Dakota input file or rewritten by hand.



(a) Mandatory file for the parametrization.



(b) Information files.

In files `CPadj.txt` and `CPfixed.txt` the coordinates and the indexes of the respective control point are written; the list of knots is contained in file `u.txt`; in file `Script_Params_Parameterization.txt`, instead,not only the parameters required for the reconstruction of the baseline, like the degree and the number of control points, but also some useful information about the machine, like the hub and shroud radii and the kind of the machine, if radial or axial, are included. In the end in the `Normaldirections.txt` text file the components of the vectors normal to the B-spline in the control points are present.

8.1.2 Solver Setup

All CFD simulations in this thesis work have been performed by ANSYS CFX, which is a software suite that contains an advanced solver with powerful integrated meshing program, pre and post-processors, with a user-friendly graphical interface. The software is mainly divided in four modules:

TurboGrid It takes the parameterized blade sections and it generates the mesh.

CFX-Pre It lets to import one or more meshes (for example in the stator-rotor case), to place them in the domain and then we have to set the flow physics, the boundary and initial conditions and the solver parameters like numerical methods, tolerances and number of iterations.

CFX-Solver Ansys-CFX includes a coupled solver which is effective in terms of number of iterations and computational time in particular for compressible flows. The convective fluxes are discretized with a second-order upwind TVD scheme, and implicit time marching integration. The convergence in terms of iterations are enhanced by using algebraic multi-grid technique. The turbulence model applied is the $\kappa - \omega$ SST which performs a blending between the $\kappa - \varepsilon$ far from the walls and $\kappa - \omega$ close to the walls.

CFX-post It takes the result of the solver and it manages to generate useful information on the quantities of interest about the simulation.

A fundamental feature of the software is that two different ways of launching it are available. The common way of approaching such a commercial software is through the graphical interface, but it is clear that we cannot manage hundreds of simulations by hand with it. Actually we interface with the program through the command line with the flag called `batch` inside CFX. The code necessary to run the four phases of the CFD simulations is integrates in the Python scripts.

To setup an optimization the graphical interface is however required. A really useful available feature it the possibility to record macros, which can be divided in two kinds:

- `States`: they are a way to save all the current parameters set by the user in a simulation; we have `states` for the meshing process, and for the pre-processor;

- `Sessions`: they are actual macro, so recordings of the steps that the user execute to perform an action. we use `sessions` for the meshing and for the pre and the post-processor.

The general procedure is to run a session in batch mode which loads a state, executes the commands and save an output file. Even if theoretically just the sessions are mandatory, it is suggested to prepare a state with all the parameters already set and then uses the sessions to load it, at least edit a single parameter and save the output file. This approach has revealed it helpful in terms of easiness of manual modification of that parameter (ex. the number of iterations).

The structure of files and folders required is sketched in the following images.

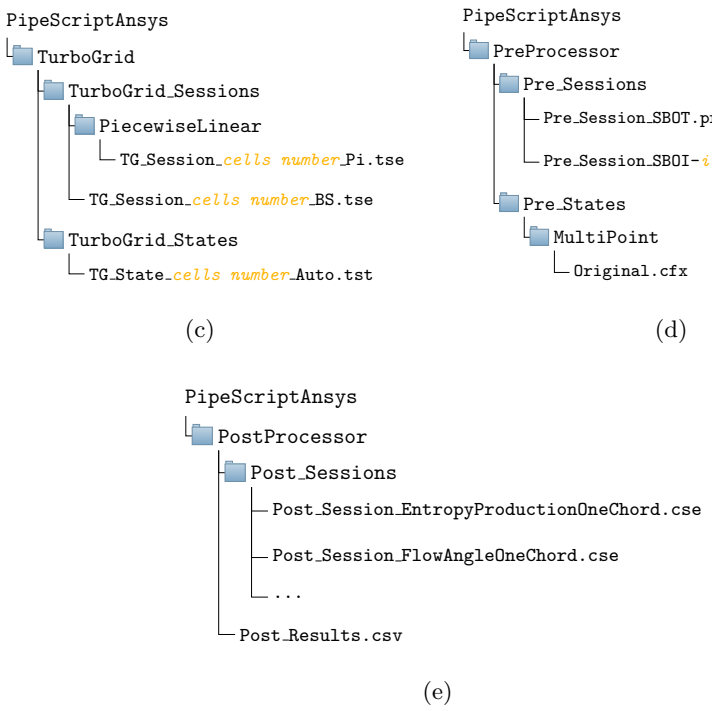


Figure 8.5: Structure of the folders for CFX solver.

8.1.3 Dakota Setup

Dakota is a command line objective oriented mathematical framework that in this work has the aim of running the surrogate-based optimization from an high level point of view. It does not have to know anything about the function that it has to optimize, and this makes it a really powerful and flexible tool in aerodynamic optimization and in many other fields.

The setup of Dakota simulation is through the writing of an input settings file called in our case `INPUT-FILE.in` which has to be referenced when launching the program. The input file is made of blocks which can be classified in six different types: Environment,

Interface, Method, Model, Variables and Responses.
In each iteration of the algorithm, a method block requests a variables-to-responses mapping from its model, which the model fulfills through an interface like in figure 8.6.

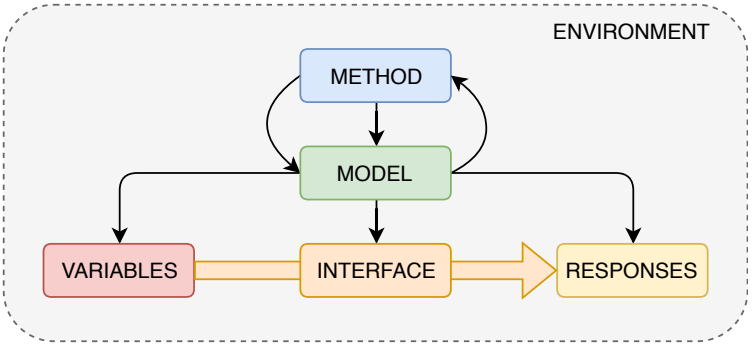


Figure 8.6: Representation of the links between blocks.

This is the easiest way to connect the blocks, but in the test case, we have more than one method and more that one model. The blocks that are actually used in the current Dakota simulation are:

Environment It manages the execution of the optimization method (SBLO or SBGO in our case) and the formatting of the result files. In general it is optional if a single method is present, but in advanced simulations like in this work we have to specify the top level method.

Surrogate Method The surrogate method is the top level algorithm, which in our project can be either the global or local surrogate-based optimization. It also contains the parameters of the method, like the maximum number of iterations, the tolerance or the trust region settings. It requires the reference of an additional method for the optimization of the surrogate model and a model for the kind of interpolation applied.

Optimization Method In a surrogate-based optimization the actual algorithm is just applied to the approximated model. We have chosen genetic algorithms. In case of single objective optimization the algorithm is the SOGA, otherwise, when multi-objective optimization is setup, the MOGA algorithm is chosen.

Surrogate Model In this block the setup of the surrogate model is performed. Among the many models available, like neural networks, polynomial, radial basis functions and others, in the current work the Kriging model has been chosen. Dakota integrates two ways of computing Gaussian processes, one in directly in the Dakota code, which is nowadays deprecated, and the other through the Surfpack library, which is the choice of this thesis.

DACE Method It contains the choice of the algorithm to generate the set of samples for the design of experiments. The number of samples can be chosen, but Dakota does not let to have less than five times the number of variables. We have chosen the Latin Hypercube Sampling as DACE method.

High-Fidelity Model It requires the reference to variables and responses.

Variables It contains the number of the variables and for each of that also a lower bound, an upper bound and a descriptor are required.

Interface The most important parameter of the interface block is the command that need to be run to process the input data to generate the output. Dakota supports natively drivers like Matlab, Python, Numpy and Scilab, but in general let to run any bash command. We manage to run the Python script through a bash call. Additionally Dakota supports concurrent multi-process simulations, whose configuration has been tested to ensure the best scaling performances.

Responses The responses block provides an abstract data representation for the response functions. In case of an optimization algorithm the objective functions and the constraints have to be written inside this block.

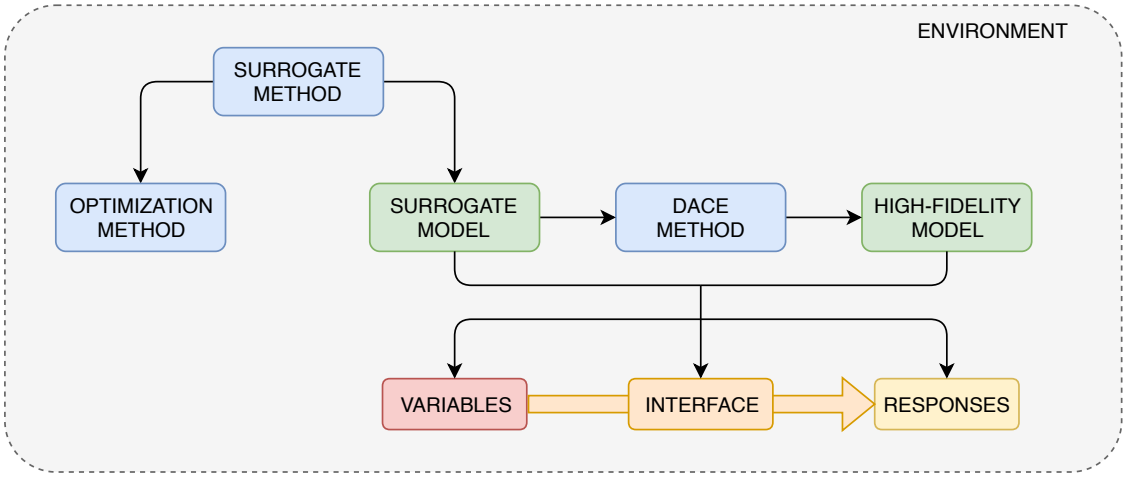


Figure 8.7: Representation of the links between blocks in the current project.

The interface between Dakota and the fitness function is made by two files, one of input and one of output. The input file `params.in` contains the value of the design variables that have to be evaluated, while the output file `params.out` have to be created with the value of every objective functions and constraints.

The passages for the optimization start with the generation of the DOE, then for each set of design variables an input file `params.in` is created, the Python main script is called, it reads the input value, it parametrizes the blade, generate the mesh, and

launch the pre-processor the solver and the post-processor, it reads the result of the simulation and prepare a `params.out` file for Dakota before to terminate the high-fidelity simulation. After the DOE Dakota computes the Kriging surrogate model and optimize it with a genetic algorithm. The optimal design variables are written to the `params.in` and the Python main file is launched again. The high fidelity value is added to the database of solution and it is used to improve the Kriging model. This procedure is iterated until the convergence condition of the surrogate evolutionary strategy is satisfied in terms of maximum number of iterations or tolerance.

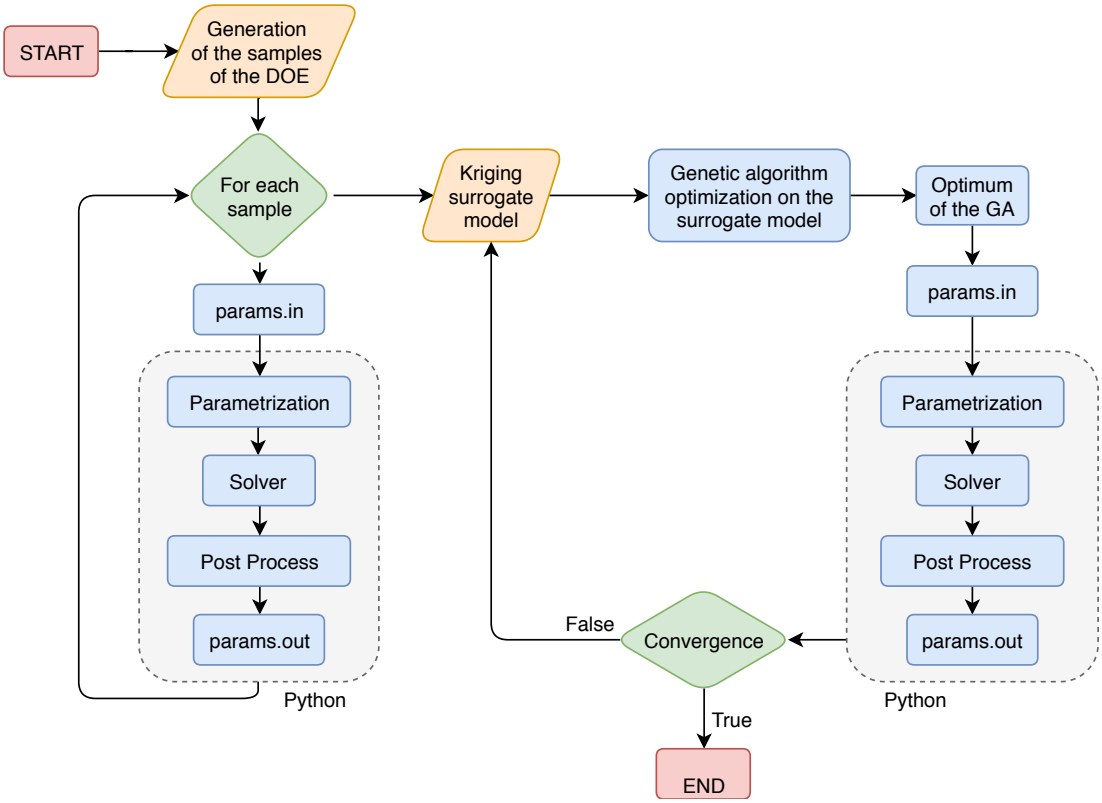


Figure 8.8: Logical flow-chart of the pre-processing of the blade data.

8.2 Post-processing

A big improvement into the Python run-time code was the introduction of the possibility to post-process the results even while the optimization is running. The code can provide additional information other then the raw data required by Dakota that easily let to have a clear idea on how the optimization is going instead of having to manually extract the few information available and re-run the CFD simulations with the extracted blade shape; this is particularly useful when something is going wrong and it is no more necessary to wait until the end to judge the results.

The most important tool that the code makes available is the chance to perform post processing on every CFD simulation result file to extract additional data respect to the

objectives and the constraints, that will not be passed to Dakota. Among the quantities that is useful to monitor but that are not involved in the interpolation there can be the residuals, the convergence trend along the iterations of a significant quantity like the entropy drop, the distribution at the outlet of dynamic or thermodynamic quantities like pressure velocity or flow angle, the blade loading and many other depending on the case of interest.

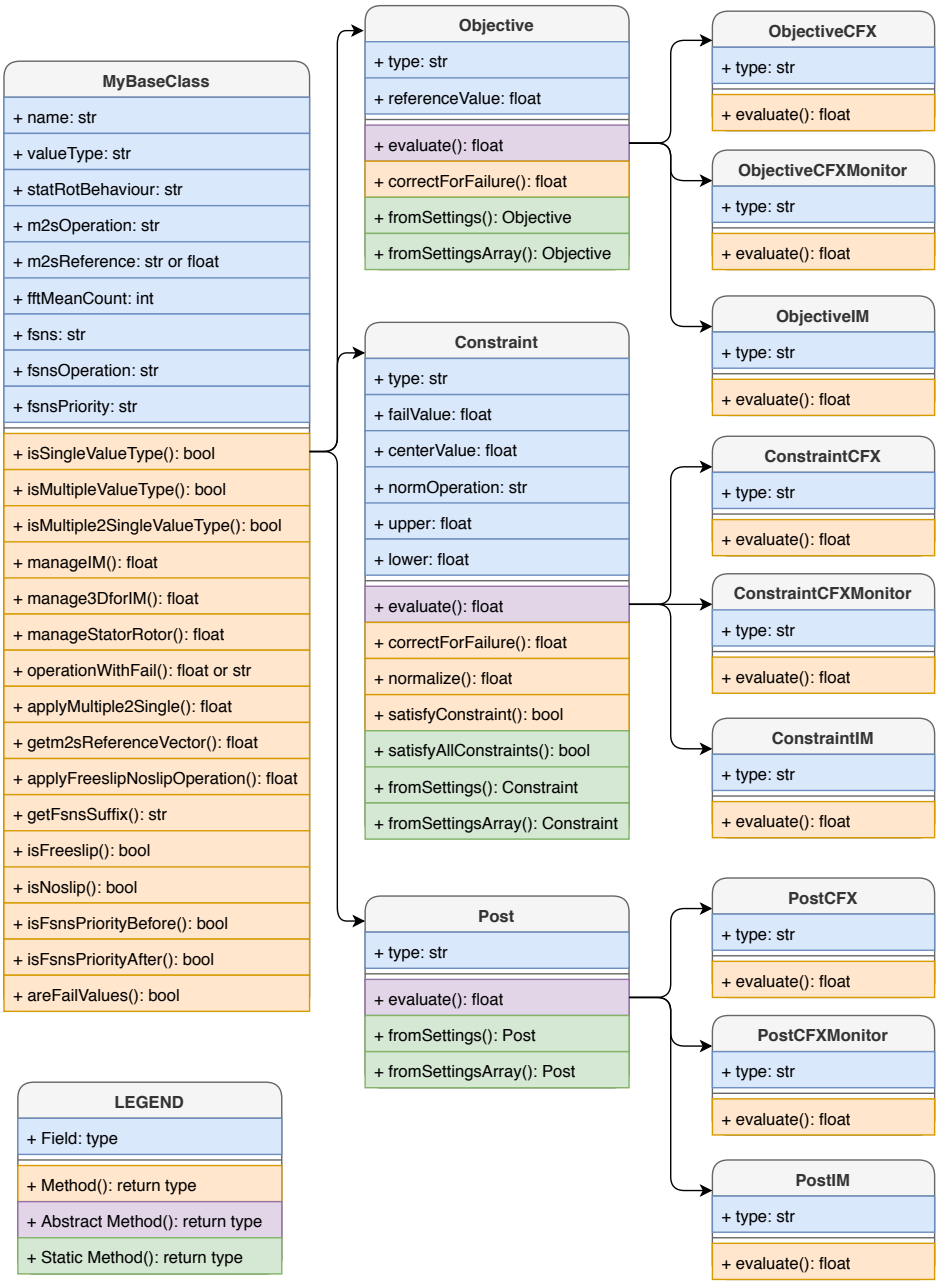


Figure 8.9: UML chart of the classes.

This additional feature is easily implemented in the code thanks to the fact that

Python is an Objected-Oriented programming language (OOP). We can build a base class that manages all the operations that are in common between `objectives`, and `constraints`, so the extension to `posts` it is just a matter of implementation of the small differences between them, like the possibility to manage a vector as a result of a `post` operation, instead of just a scalar value.

Then all the values coming from `objectives`, `constraints`, and `posts` results, are stored in a result file in tabular form, and can be easily tracked during the optimization. For what that concern `post` results that return a vector of values, they are automatically plotted and saved in an individual folder that refers to each high fidelity CFD simulation, which contains also the value of `objectives` and `constraints` and the representation of all the sections of the blade compared to the baseline profile, both in 2D and 3D optimizations.

8.3 Average Over Iterations

In the process of optimizing a 3D blade shape, the computational cost for the CFD simulation is critical; the number of cells of an even 3D coarse mesh is quite high, since we need to have a sufficient number of layers near the hub and the shroud to properly describe the boundary layer, and consequently each further iteration will increase the required time.

Dealing with blade shapes that can even be strange and cumbersome, the perfect convergence is even difficult to be reached in every simulation, but a fast and fairly accurate estimation of the performance of each blade is absolutely necessary for a proper optimization in particular when supported by a surrogate intermediate layer.

For all these reasons an average strategy is particularly meaningful in this case.

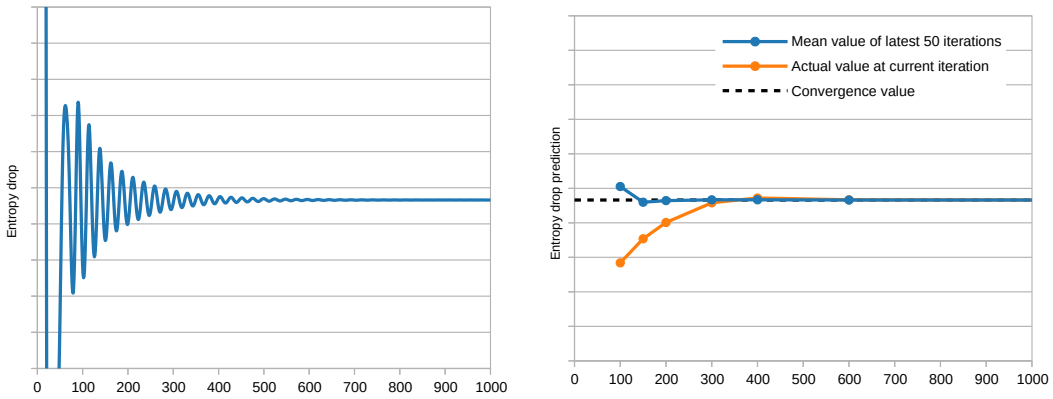
To quantify the performance of an airfoil we will use the entropy generation, in particular the difference between the outlet and the inlet mass flow average of the specific entropy s.

$$\Delta S = \int_{A_{\text{out}}} \rho s \, \mathrm{d}A - \int_{A_{\text{in}}} \rho s \, \mathrm{d}A \tag{8.2}$$

Then we will monitor this quantity along the iterations to check if the convergence is reached. What that we notice is that often, after an initial transition phase, the trend of ΔS is close to a sinusoidal function whose amplitude is decreasing along iterations. Predicting the value at which the function will tend, can prevent us to run up to convergence.

We will show this through an example: in figure 8.10(a) is represented the trend of the entropy drop along iterations during a simulation. This simulations can be considered converged above around 600 iterations.
In figure 8.10(b) are extrapolated the values of the objective function that we get if we stop at 150, 200, 300, 400, 600 and 1000 iterations, and at 200 iterations we are not really close to the real value. However if we apply an averaging to the data even at 200 iterations we get a value really close to that at which we aim to. This averaging

procedure in this case would have reduced the computational time by a factor 3 keeping almost the same accuracy.

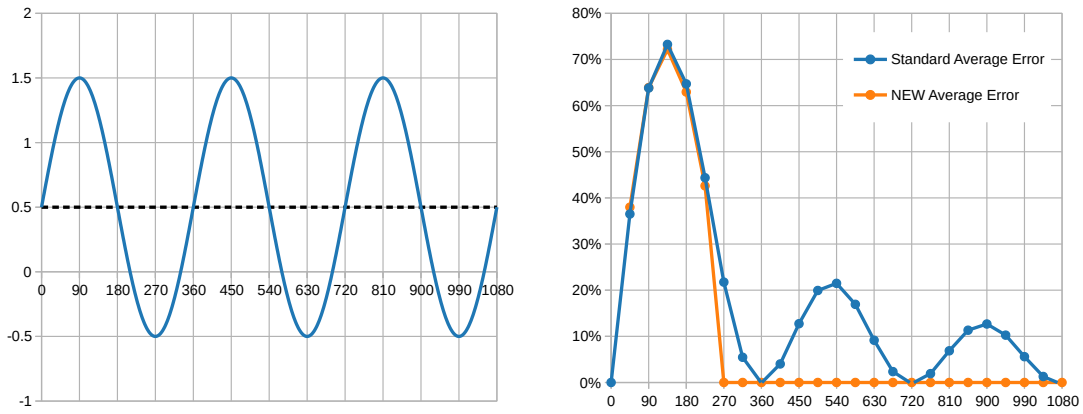


(a) Entropy drop trend along iterations.

(b) Entropy drop value comparison.

Figure 8.10: Entropy drop example.

Another modification can be done to reduce the effect of leakage. Taking the average over the same number of samples without knowing the period of the signal can in general lead to the *leakage* phenomenon, which means that we are taking the average (or in general we apply the discrete Fourier transform) to an interval which is not an integer multiple of the period. A clarifying example is shown in figure 8.11. Actually with the standard average we are estimating correctly the mean value of the signal only when the interval size is 360, 720 or 1080 and so on, and we are wrong in any other case. We can also notice that the error progressively decreases enlarging the interval size.



(a) Signal example

(b) Average comparison

Figure 8.11: Leakage example and comparison between standard average and out proposal.

Our strategy instead consists in estimating the period of the signal, and then

performing the computation of the average just on an interval multiple of the period found. It is however required to choose an initial interval that it is at least roughly large as the period, otherwise the number of samples is too small. So when the interval is larger then the period, in this simple case, we are able to exactly estimate the value of the mean, independently on the interval count.

Chapter 9

Optimization of an Axial Turbine Stator

The turbine assembly is mounted downstream of the combustor, commonly forming the rear part of a jet engine when viewed as a whole. The air entering in this system is highly compressed, mixed and vaporized with the fuel and finally ignited; the hot gases leaving the combustor are expanded to a lower pressure and temperature in the turbine. This expansion extracts energy from the hot gases used to rotate the turbine blades and the disc assembly, which then drives the compressor via a centrally rotating shaft. To produce the correct driving torque and the required efficiency, the turbine is composed by several stages; each of them employs one row of static nozzle guide vanes and one row of rotating blades. The number of turbine stages depends on the relationship between the powers required by the rotational shaft speed and the permitted turbine diameter.

Figure 9.1: Rotor and stator blades.

The machine is an high pressure axial turbine made of 22 blades for the stator and 25 for the rotor, with a mean reaction degree of around 0.3. The use of a casing for the stator is applied in the LFM laboratory in order to reduce the leakage effects. The blades are designed according to the principle of keeping constant the incidence angle along the blade height; the final shape is obtained through a tri-dimensional twisting,

with the aim of obtaining a two-dimensional flux in the midspan section. Furthermore the blades are not radial along their height but designed using the *leaning* technique.

These specific blades are obtained with a tangential translation of the profile of a cylindrical blade of a quantity proportional to the blade height. The resulting shift is sufficient for obtaining an inclined profile of a constant angle, called stack angle, maintaining a linear longitudinal axis. For this blade a stronger difference of pressure between the pressure and the suction side at the tip compared to the hub is present, due to an uneven distribution of the load.

The blade was designed by skilled engineers and has pretty good performances, so an optimization of this profile is a tough challenge. In this chapter we will focus on the optimization of the shape of the stator blade.

9.1 Boundary Conditions

We will optimize the stator blade subjected to boundary conditions called *OP1* or operating conditions 1. These are the most severe conditions, with the maximum rotational speed, and flow speed.

In particular the rotational speed is equal to 11100 rpm, and the mass flow rate 6.05 $\frac{kg}{s}$. In this conditions the dimensionless parameters ϕ [1] and ψ [2] values respectively 2.09 and 5.00.

We set the inlet total pressure and temperature and outlet static pressure, and their value can be found in table 9.1. The inlet conditions are set as uniform along the span, while the outlet static pressure is set with an admissible blending of 5 % respect to the radial equilibrium (Appendix A).

	Inlet	Outlet
Static pressure p		125723 Pa
Total pressure p_{T}	192070 Pa	
Total temperature T_{T}	323 K	

Table 9.1: OP1 boundary conditions.

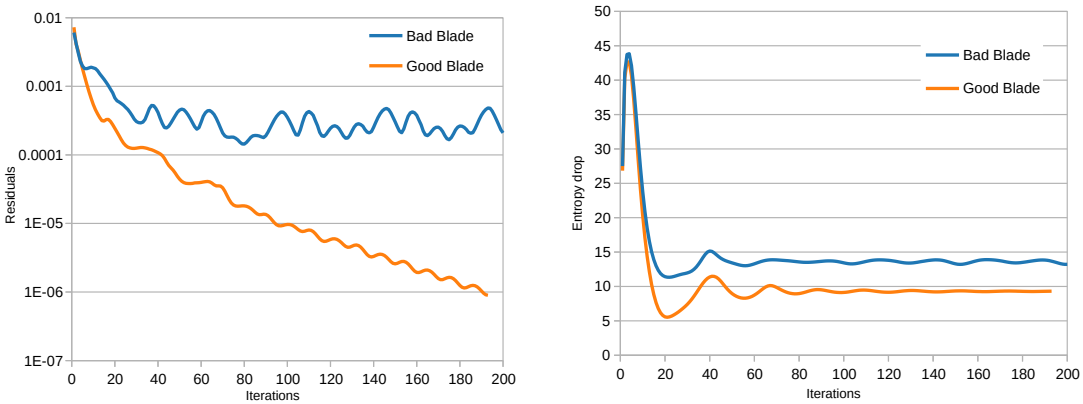
As an alternative the experimental inlet total pressure could have been applied, instead of the constant value along the span. However this caused issues in terms of convergence when dealing with profiles modified by the optimization procedure. Actually the baseline showed a good convergence trend, but probably that total pressure conditions are appropriate only when dealing with that specific profile.

With the boundary conditions chosen we have still observed variable levels of convergence, but we can expect this behavior with such strange blade shapes, and

[1] The flow coefficient $\phi = \frac{v_a}{U}$

[2] The loading coefficient $\psi = \frac{\Delta h_{\mathrm{T}}}{U^2}$

however what that most matters is that none of the simulations showed completely useless results, and we are always able to estimate approximately how a blade would perform respect to another.



(a) Residuals

(b) Entropy drop

Figure 9.2: Comparison between a bad and a good blade.

As we can see in figure 9.2, even if the residuals in the bad profile case are not perfect, and consequently the entropy drop shows oscillations, we are absolutely sure that it performs worst that the other example shown, and this reflects the preliminary analysis on the shape of both blades.

9.2 Fluid model

The fluid that we set for the simulation is the proprietary implementation of air as ideal gas of CFX which represents a fluid with constant isobaric heat capacity C_p. The chemical and thermo-physical properties of air are summarized in table 9.2.

Property	Symbol	Value
Isobaric heat capacity	C_p	$1.0011\ \frac{kJ}{kg \cdot K}$
Molar mass	M_M	$28.96\ \frac{kg}{kmols}$
Dynamic viscosity	μ_d	$1.831 \cdot 10^{-5}\ \frac{kg}{m \cdot s}$
Thermal conductivity	k	$0.0261\ \frac{W}{m \cdot K}$
Reference pressure	p_{ref}	$1\,atm$
Reference temperature	T_{ref}	$25°C$

Table 9.2: Air properties.

The ideal gases undergoes to the *ideal gas law* written in equation 9.1, whose

particular simplicity explain why this kind of modeling is so widespread.

$$pV = nRT \qquad \text{or} \qquad \frac{p}{\rho} = R^\star T \tag{9.1}$$

where: p is the static pressure;
V is the volume taken by the gas;
ρ is the density;
n is the number of moles;
T is the temperature;
R is the gas constant which is equal to 8.31 $\frac{\text{J}}{\text{K}\cdot\text{mols}}$;
$R^\star$ is the gas constant divided by the molar mass.

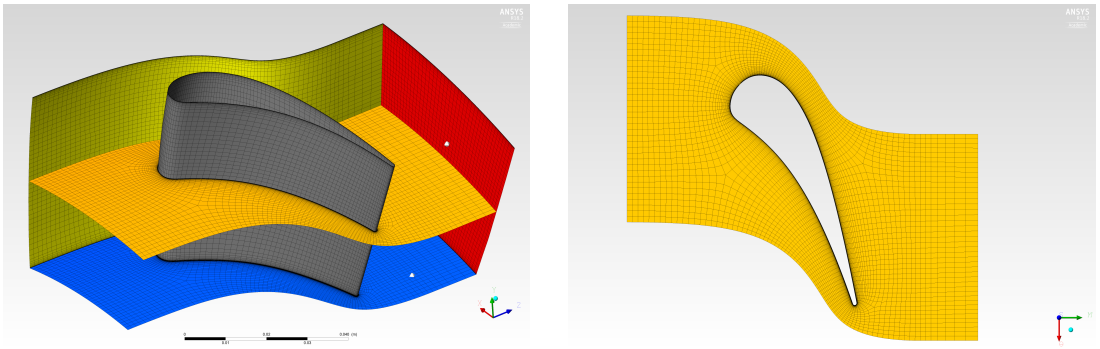
For the ideal gas model it is straightforward to compute the most important thermodynamic quantities as:

$$u = u(T) = (c_p - R^\star) \cdot T + u_0 \tag{9.2}$$
$$h = h(T) = c_p\, T + h_0 \tag{9.3}$$
$$s = s_0 + c_p \log T - c_p \log T_0 - R^\star \log p - R^\star \log p_0 \tag{9.4}$$

9.3 Base case

Before to run the optimization in important to have the results of the original blade, in terms of mesh sensitivity and performances.

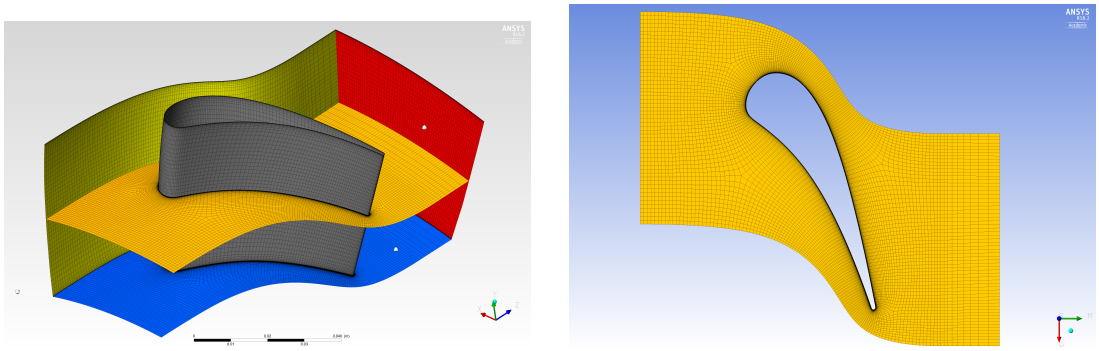
We have performed the simulations with 3 different meshes, from 500 000 to 2 000 000 cells. The different meshes are represented in figures 9.3, 9.4 and 9.5.



(a) 3D view of mesh with 500 000 cells

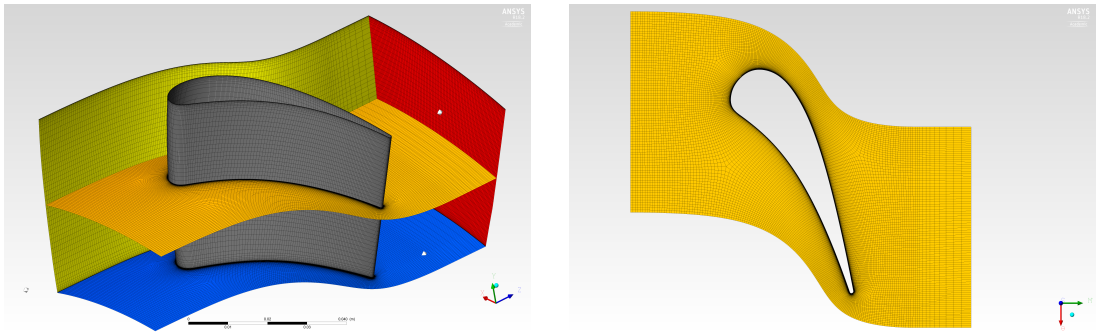
(b) Midspan view of mesh with 500 000 cells

Figure 9.3: Mesh with 500 000 cells.



(a) 3D view of mesh with 1 000 000 cells (b) Midspan view of mesh with 1 000 000 cells

Figure 9.4: Mesh with 1 million cells.



(a) 3D view of mesh with 2 000 000 cells (b) Midspan view of mesh with 2 000 000 cells

Figure 9.5: Mesh with 2 million cells.

The rule applied to build the set of meshes is to keep the value of y^+ close to 1 for all the walls, meaning the blade, the hub and the shroud, to guarantee a sufficient accuracy in the application of the $\kappa - \omega$ SST turbulence model. This is done manually by adjusting the parameters inside the TurboGrid software in order to match such condition.

Along the span, the mesh is divided into 3 sections of respectively 20, 30, 20 cells in all the three cases, where the first and the last sections are used as boundary layer for the hub and the tip. Keeping constant the number of layers along the span, increasing the number of cells let to improve accuracy of the simulations in the blade to blade plane.

9.3.1 Mesh sensitivity analysis

We have performed the simulations for the three meshes previously defined, and we compare them mainly in terms of entropy difference between outlet and inlet as defined in equation 8.2. The results are reported in figure 9.6.

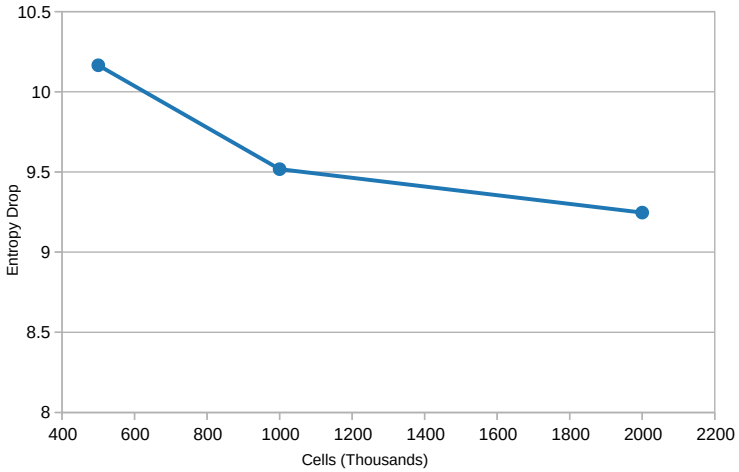
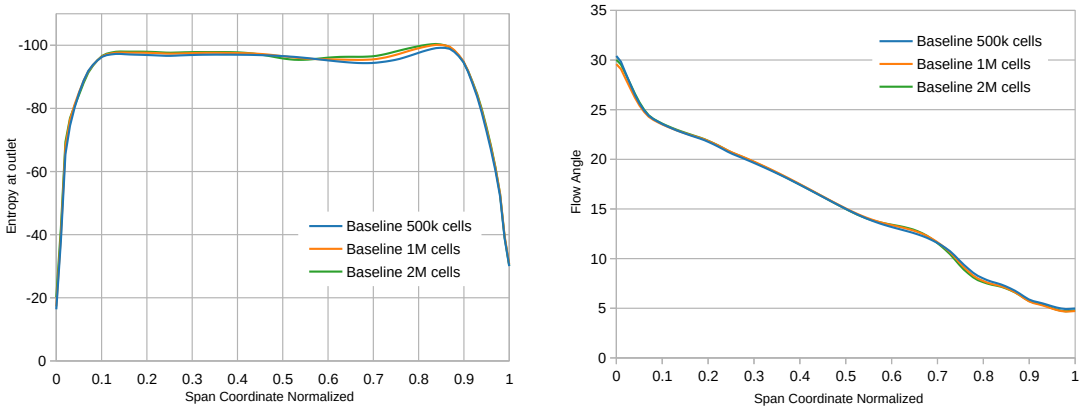


Figure 9.6: Entropy drop increasing the mesh refinement.

As we can appreciate, increasing the number of cells, the entropy difference decreases. This is a typical behavior in CFD simulations since the numerical diffusivity decreases with smaller cells, and so on the entropy generation.

For sake of completeness we also report the distribution of flow angle and entropy at outlet for all the three cases in figure 9.7.



(a) Outlet entropy

(b) Outlet flow angle

Figure 9.7: Comparison of the baseline case with 500 000*, 1 million and 2 millions cells.*

Then we perform the comparison for all the quantities that we have computed at outlet, so static and total pressure, flow angle, entropy and modulus of the velocity. To understand the results in figure 9.8, we have taken the difference point by point for each quantity respect to the reference value, that we considered the case with 2 millions cells; then we have summed the squared of all that differences. Finally, noting that the "error" decreases passing from 500 000 to 1 million cells, we have normalized it respect

to the value at 500 000 cells.

$$\text{error} = \sum_i \left(y_i - y_i^{\text{ref}} \right)^2 \qquad \text{error}_{\text{normalized}} = \frac{\text{error}}{\text{error}_{500\text{k}}} \cdot 100\,\% \tag{9.5}$$

Except the flow angle all the other quantities show almost the same trend. The flow angle shows a slightly different behavior, but given that the absolute value of the error for the coarser case is around 3.5 which is equivalent to a mean error over 100 points of less than 0.2°, which is the value considered as threshold below which two angles are equivalent.

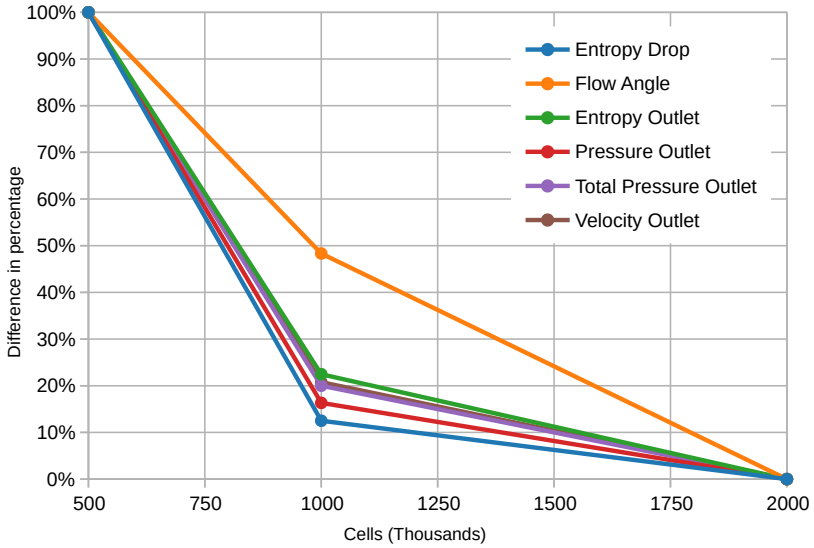


Figure 9.8: Normalized comparison of point to point error between 500 000, 1 million and 2 millions cells.

All the optimizations are performed with a mesh with 500 000 cells which is not a convergence simulation, but take this choice assuming that two blades tested at 500 000 and 2 000 000 that would have different performances, has the same ranking at both levels of fidelity. Having said that we will validate this assumption for all the optimal blades computing them at 2 000 000 and comparing the result to the baseline.

9.3.2 Selection of the number of sections

The 3D blade is reconstructed with 21 profiles from radius 150 mm to 200 mm. Optimizing the blade with such a high number of sections would require an enormous amount of time and computational power and probably it not even significantly improves the result quality. To decide how many sections to use in the optimization phase we will perform a comparison study. The tested cases are reported in table 9.3. Firstly we have simulated evenly spaced sections, then we have concentrated more profiles or close to the hub or close to the tip.

Case	Number of sections	Sections index
Reference	21	All the sections
1	3	1-11-21
2	5	1-6-11-16-21
3	7	1-4-7-11-15-18-21
4	11	1-3-5-7-9-11-13-15-17-19-21
5	6	1-6-11-14-18-21
6	6	1-4-8-11-16-21

Table 9.3: Testing cases for the choice of number of sections.

Increasing the number of sections dramatically increase the computational cost, since we introduce many other design variables. For example adding 1 more sections and considering 10 movable control points in each section would require 100 more CFD simulations just for the design of experiment procedure.

Luckily, for this profile even the most simple case with 3 section evenly distributed, interpolated with B-splines along the span is able to provide enough accurate results. In figure 9.9 it is shown the outlet entropy along the span with 3, 6 and all 21 sections to reconstruct the blade.

Even if the case with six sections is more accurate than the simpler case, the difference between and two, and most importantly the difference between them and the reference case is almost negligible.

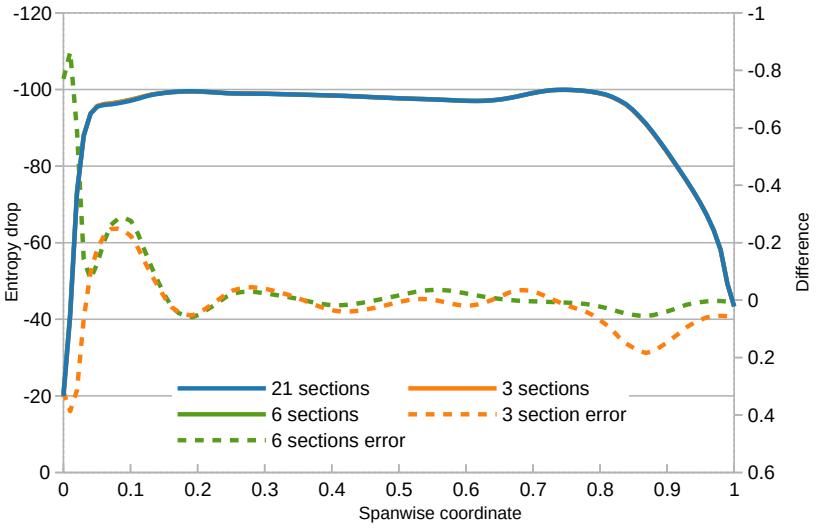


Figure 9.9: Example of unordered profile

For this reason we have chosen to optimize the blade dividing it in three sections, the hub, the mid-span and the tip, and interpolate along the span with the B-spline feature already provided in the TurboGrid mesher tool.

9.3.3 Choice of the number of iterations

To run the optimization algorithm is required to find a trade off between accuracy and computational time of each high fidelity simulation. Actually CFD simulations generally do not have a perfect linear convergence trend, and above a certain number of iterations often a stable condition is reached. In general we don't want to spend to much time for just a small improvement in terms of residuals reduction and solution accuracy, at least in the optimization phase, while we will care more on accuracy in the validation procedure.
The number of iterations is chosen analysing the residuals and the convergence trend of the entropy production for the reference case.
The residuals trend is shown in figure 9.10.

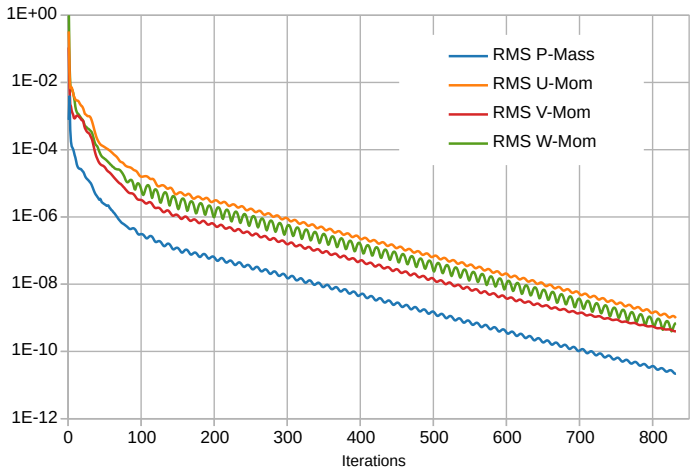
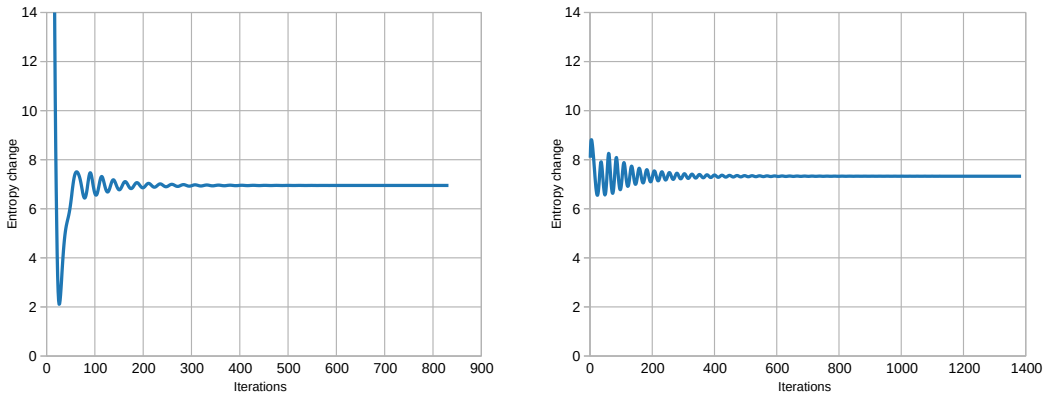


Figure 9.10: Residuals of the reference case.



(a) Without initialization.

(b) With initialization.

Figure 9.11: Comparison of the entropy production with and without initialization.

Actually the optimization process will be run with initialization to speed up the convergence process and reduce the number of iterations required, so it is interesting to show the entropy production trend also when we initialize the case with a previous result. The comparison is drawn in figure 9.11 and it shows that in the initial iterations the result without initialization is much worse than the other, but, after the starting phase, the trends are quite similar.

In both cases to reach a proper convergence value the simulations have to run more than 600 iterations, which seems the right number that we have to choose for the optimization. In reality we will run all the simulations for just 200 iterations, which accordingly to chart 9.11 seems to produce inaccurate results. We will explain in chapter 8.3 how we can reduce so much the number of iterations without losing accuracy.

9.4 System setup and parallelization

One of the most important requirements of any CFD simulation is the computational power. As the necessity of reliable results, the number of cells of the domain and the number of iterations of the solver increase. This requires more and more system memory (RAM) and CPU power.

The machine that we use in this thesis work is a node of a cluster at Politecnico di Milano.
The heart of the system is made by an Intel Xeon E5 2630 v3. It is an octa-core CPU with sixteen thread thanks to the hyper-threading technology.
The available random access memory is 64 GB large.

Before to run the optimization simulations we have studied the behavior of the system with different parallel strategies. This process and its results are explained in the next sections.

9.4.1 Parallel simulations

The scaling of the CFD code with the number of cores it is not always linear, meaning that doubling the resources in terms of cores does not always bring to a reduction of a factor two of the time required. Furthermore the physical cores are 8 but we can use up to sixteen concurrent threads together thanks to the hyper-treading technology, which let to exploit as much as possible the hardware capabilities, but it is not able to double the real performances.

Moreover, the first step of the optimization procedure is the generation of the surrogate model through a design of experiments algorithm (DOE), which is in general highly parallelizable, since we can theoretically launch all high fidelity simulations at the same time, since they are independent one of the others, given enough computational power.

Having said this, due to limited resources and a highly scalable problem, a proper balance of resources distribution in terms of core per simulation can reduce the time required at least for the DOE generation.

First at all we will study the scaling of the case of interest just one simulation at a time, testing different numbers of cores, measuring the scaling of a single CFD simulation; Then we will test many simulations at the same time with the same number of cores, to saturate the 16 threads available with different combinations of core count per simulation.

9.4.2 CFD cores scaling

To highlight the scaling of the simulation of interest, we will run the turbine stator simulation, with 500 000 cells for 150 iterations, with different number of cores.
Each simulation is running alone, meaning that for example for the 1 core case, all the other 15 threads available on the machine are at idle.
It will bench the total time required to build the mesh with the TurboGrid software, to prepare the simulation, to effectively solve it and to compute the values required with the post-processor. It is not just the run time which is the only one influenced by the parallelization, but the full time required to run the simulation, since is that which we are interested in. To be as fair as possible, no initialization is applied so that does not influence the resulting time.

As we can appreciate in figure 9.12 we have an acceptable scaling performance ($> 80\,\%$) up to 4 cores; beyond that, the scalability drops, and with 16 threads we even have worse computational time respect to the case with just 12.

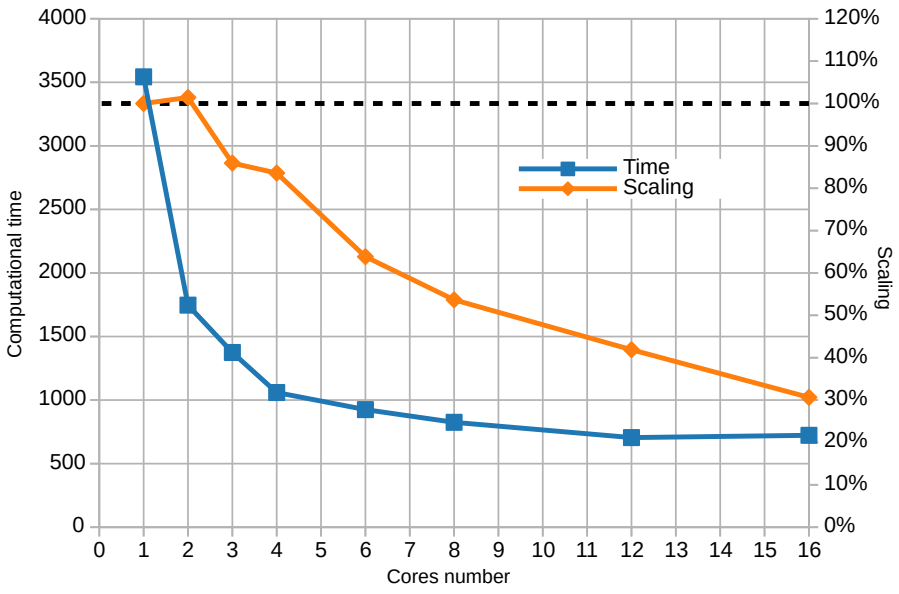


Figure 9.12: Computational time with different number of cores with 500k cells.

9.4.3 Performance comparison

In previous section we have tested just one case at a time without filling the CPU power, which is what happens during the GA optimization. In the case of parallelized DOE generation, however, we will run as much parallel multi-core simulations together as possible, so we have also tested how much the real hardware scales when fully utilized. The testing process contains five different configuration, shown in table 9.4.

	Cores	Parallel Simulations	Total Threads
Case 16x1	1	16	16
Case 1x16	16	1	16
Case 2x8	8	2	16
Case 8x2	2	8	16
Case 4x4	4	4	16

Table 9.4: Tested cases.

The logic behind that is to perform simulations changing the number of cores, but keeping the total number of required thread to the maximum available in our CPU. For example for the dual core case, we will run 8 of that simulations in parallel, to max out the power of our hardware.
The result is shown in figure 9.13, and it accounts for the normalized time required for performing 16 simulations.

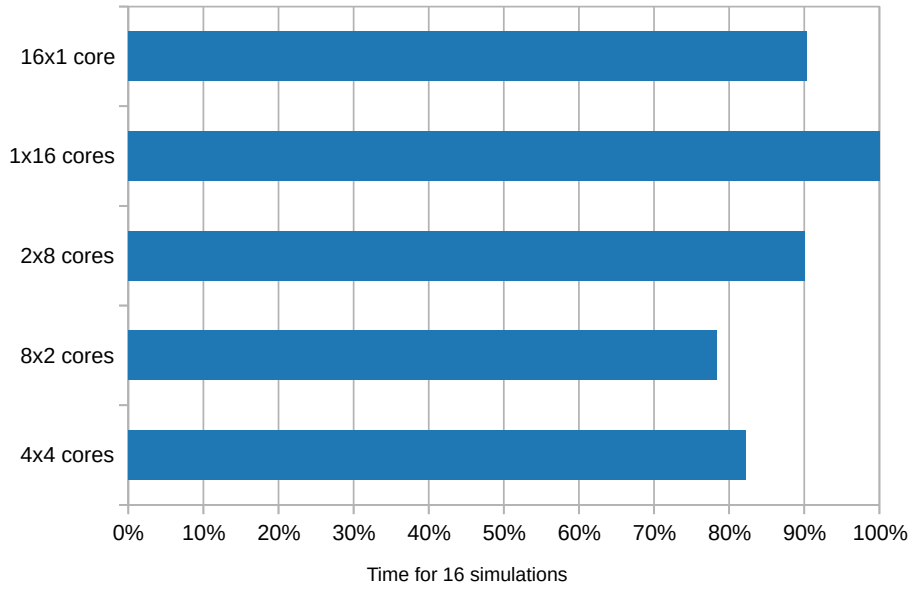


Figure 9.13: Computational time with different number of cores with 500k cells.

It is interesting to notice that the hierarchy is not the same as figure 9.12. The

biggest difference is the single core case; actually running 16 single core simulation together dramatically drops the performance respect to run it alone. The mid multi-cores simulations (dual and quad cores) are confirmed as the best ones, but the high multi-cores cases reduces significantly the distance with the best overcoming the single core simulation.

This apparently strange behavior can be partially explained searching into Intel specifications. The processor is indicated to have a base frequency of 2.4 GHz and a Turbo Boost one up to 3.2 GHz. In general the performances scales approximately linearly with the frequency; between the base and the maximum frequency there is a difference of 50 %, so the actual frequency is highly affecting computational time.

The Turbo Boost technology increases the frequency of the processor depending on the thermals and the number of cores used. In the Intel data sheet is explicated that the maximum frequency depends on the number of cores currently used. For the Xeon E5 2630 v3 the values are reported in table 9.5.

Used Cores	1	2	3	4	5	6	7	8
Max Frequency (GHz)	3.2	3.2	3.0	2.9	2.8	2.7	2.6	2.6

Table 9.5: Turbo Boost frequencies of the Xeon E5 2630 v3.

Investigating the actual frequency of the system in all the situations is probably out of the scope of this work, but even the nominal maximum frequency can explain the behavior that we have observes in our testing.

9.4.4 Scaling with different number of cells

The previous results show that probably 500 000 cells are not enough to saturate the 8 cores 16 threads Xeon processor. Actually the best solution was to run 8 parallel simulations with 2 cores each.

So our interest is to understand if this is also true with 1 000 000 and 2 000 000 cells or if to have more cells in each core guarantees a better scaling of performances.

We have performed both the case with a single simulation running on the cluster with a different number of cores, and concurrent simulations with the same number of cores that take advantage of the 16 threads available.
Results of the case with 1 000 000 cells are shown in figures 9.14 and 9.15.

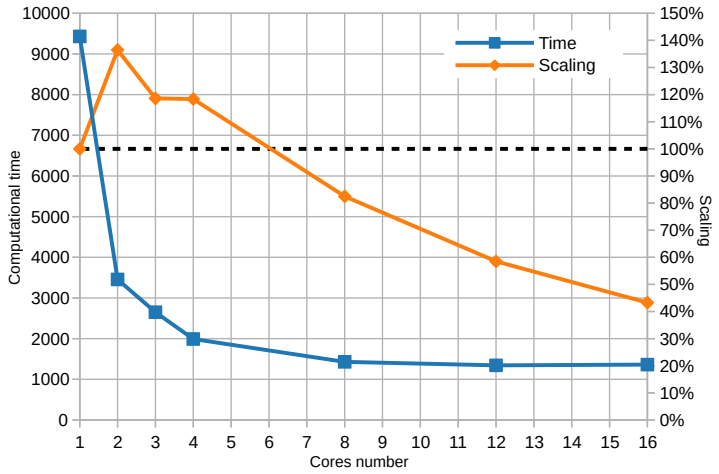


Figure 9.14: Computational time with different number of cores with 1M cells.

From the point of view of the single simulations at a time test case, the results are quite similar between 500 000 and 1 000 000 cells, with the difference that the single core performances are quite bad, but the dual core remains the best case and reasonable scaling performances are kept up to 4 cores, with a drop beyond that.

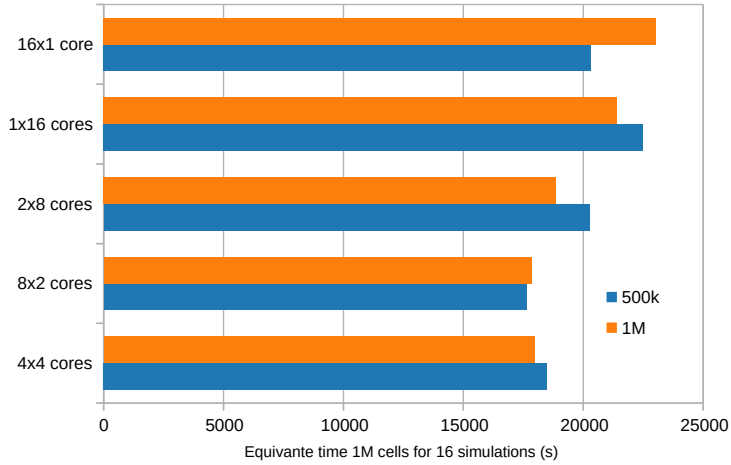


Figure 9.15: Computational time with different concurrent simulations with 1M cells [3].

When we consider the more realistic load case during the DOE generation, in figure 9.15, we can appreciate that we have worsen scaling with a high core count (1 and 2 cores), a comparable with mid core count (4 cores) and a better one with high core count (8 and 16 cores); the optimal configuration is with 2 concurrent simulations of 8 cores each. And probably the most significant change is that the serial procedure, consisting in assigning all the cores for a single run performs drastically better doubling

[3] It means that the time of the 500 000 cells mesh is multiplied by two.

the number of cells.

Increasing the number of cells to 2 000 000 the situation is similar to the 1 000 000 case but even more accentuated. In the single concurrent simulation results, shown in figure 9.16, we can see that the best case has become the four core simulation, and the single core one has become even worsen.

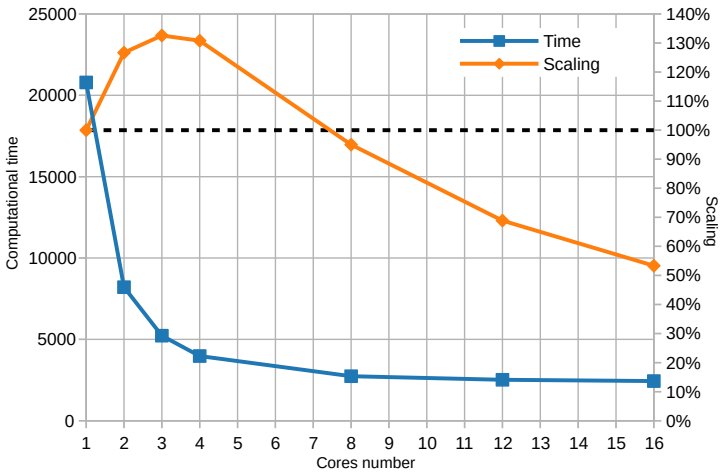


Figure 9.16: Computational time with different number of cores with 2M cells.

The more realistic load case in figure 9.17 highlights better scaling with high cores count (greater than 8) and worsen when the cores number is lower than 8; the better simulations are 4X4 cores and 8X2 cores and 2X8 cores with very similar results.

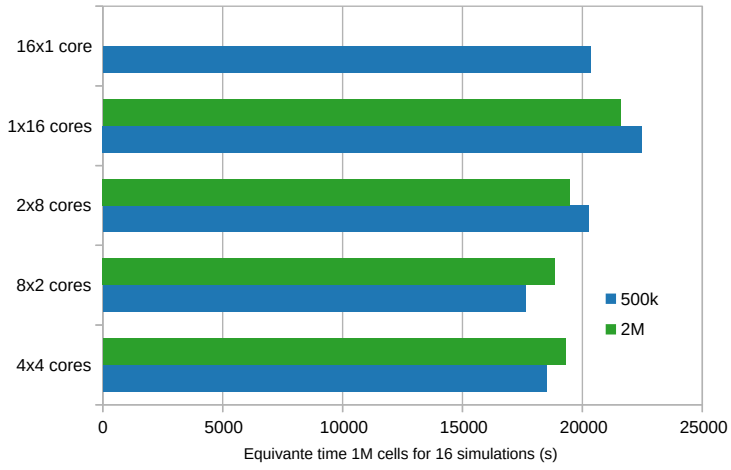


Figure 9.17: Computational time with different concurrent simulations with 2M cells.

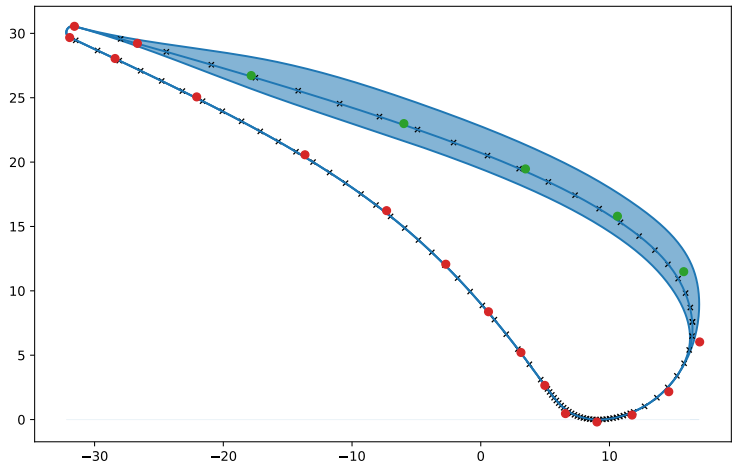
The sixteen single core simulations have had problems probably due to the size of the case in terms of RAM usage, requiring a RAM quantity bigger than that installed on the system.

Even with smaller mesh sizes (500 000 and 100 000 cells) the 16 concurrent single core simulations show a weird behaviour in terms of difficulty in the balance of the load between the cores. This has the consequence that some simulations are much faster then others, even of more than two times. If we measure just the total time we have misleading results, and we have overcome it taking the average of the sixteen simulations and adding to it the difference between the total time and the maximum solver running time, as a strategy to account for pre and post processing.

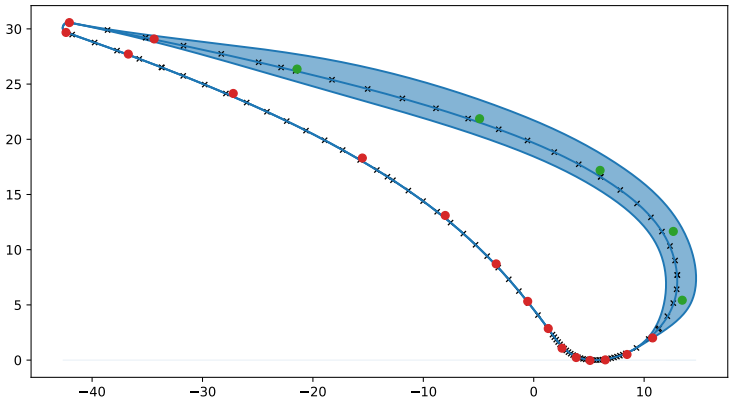
9.5 Blade parametrization

As described in chapter 8.1.1 where we are listing all the procedures necessary to setup the optimization, the most tricky part is probably the interpolation of the blade and the definition of the of the design space.

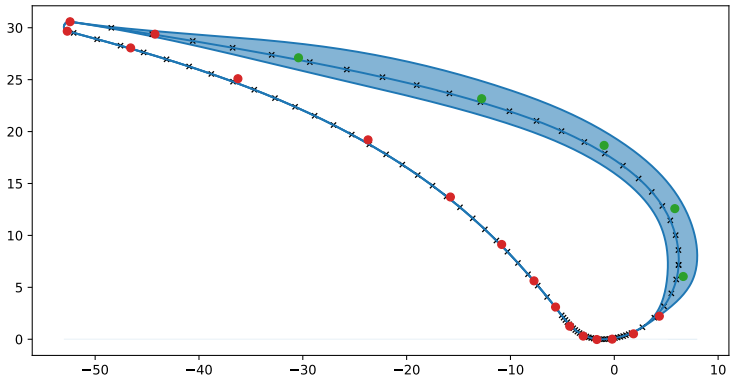
For the first cases we have chosen to divide the blade in three sections along the span, one at hub, one at mid-span and one at tip, and to use 21 control points for the interpolation with a B-spline of degree equal to 3. The knot array is composed by two geometrical progressions with ratio and center optimized with the procedure previously explained in chapter 3.6.3.
In figure 9.17 we have reported for the three sections the original points, the movable and the fixed control points, the trailing edge, the interpolated baseline and the maximum and minimum blades that can come from the optimization.



(a) Hub



(b) Mid-span



(c) Tip

Figure 9.17: Blade interpolation in three sections.

Some additional information for the building of the interpolating B-splines can be found in table 9.6. As we can appreciate there, with 21 control points the mean error is quite small, meaning that the interpolation is pretty good.

Section	Knot Ratio	Knot Center	Mean Error
Hub	1.24	0.57	$7.46 \cdot 10^{-3}$
Midspan	1.43	0.57	$6.44 \cdot 10^{-3}$
Tip	1.43	0.57	$4.43 \cdot 10^{-3}$

Table 9.6: Additional information on the construction of the B-splines.

9.6 Design of experiments

The choice of the number of samples in the design of experiments is an extremely important parameter in surrogate-based optimization algorithms in particular when dealing with expensive objective functions.
Too many samples, and the computational cost is unnecessarily unbearable, too few samples and the surrogate model is not able to interpolate correctly the high-fidelity objective function, not being able to exploit the global optimum.

Given the available computational power, we have tested the same optimization case with two different DOE size, the former with 75 samples and the latter with 150. his numbers are not randomly chosen, but 75 is the five times the number of design variable, which are 5 control points for each one of the 3 sections along the span, and it is the smaller number of samples that Dakota let to use in surrogate-based optimizations. Then we have chosen to double the minimum number to have 10 samples for each design variable.

In image 9.18 we compare the objective function value along the iterations; apparently the image represents just a mess; and probably it is the case for both 75 and 150 DOE samples. Actually there is no difference in terms of absolute value of objective inside the DOE, and we can notice that the theoretical worst case it is even better the the other. This behavior can be easily explained by the fact the in this phase we are simply randomly sampling a complex function with the only aim of obtaining as much data as possible to build an accurate surrogate model, and no real effort on the optimum direction is done.

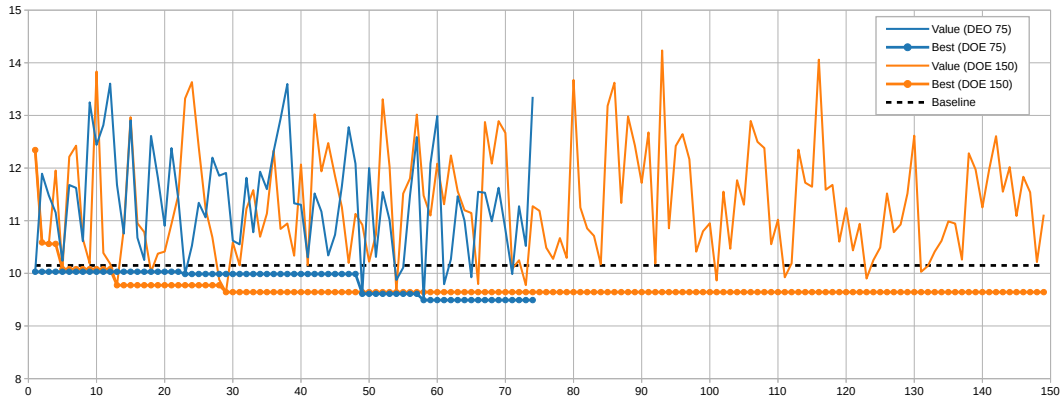


Figure 9.18: Comparison of the objective function values in the DOE iterations

The situation is instead completely reversed when we include also the real optimization process in the analysis, on which is focused figure 9.19. After the end of the design of experiments, both are able to improve the optimum value of the DOE, which is a symptom of the fact the optimization algorithm is going toward the right direction.

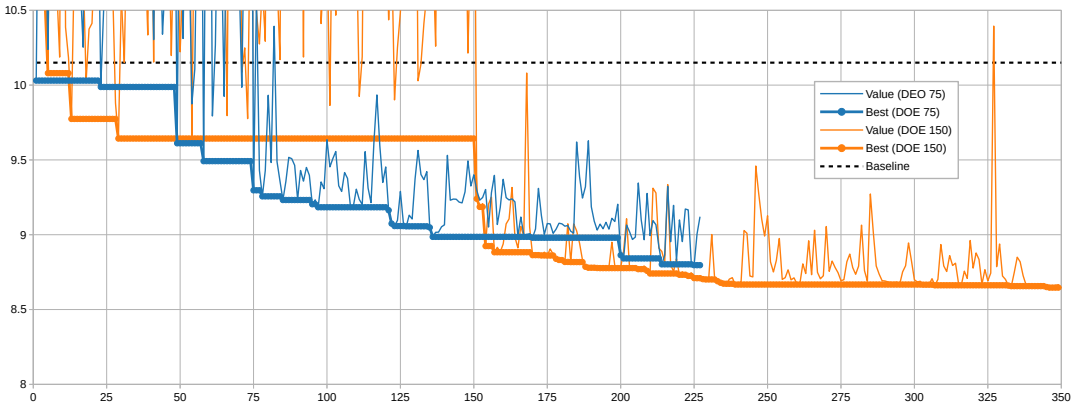


Figure 9.19: Comparison of the objective function values along iterations

However the case with 75 samples it is not really able to fully reconstruct the shape of the objective function, and we can infer this highlighting the fact that just small improvements in the optimization are found after the first phase, and it is able to find so optimal region, but the exploitation is slow and progressive. In the other case instead it is clear the big step that the optimum undergoes just few steps after the end of the DOE. And then the exploration starts with small progressive improvements on the optimal solution, like the 75 case but with better performances.

We have run the better case for more iterations, but comparing at the same number of iteration of the DOE-75 we can notice a significant difference in terms of the shape of the optimal blade, even if apparently this the entropy drop values are similar.

Comparing the blade shapes in figures 9.20 and 9.21 it is clear that the former has not reached convergence, and the hub section shows oscillations which probably are going to disappear increasing the number of iterations of the optimization, while the latter has more smooth profiles in all the three sections.

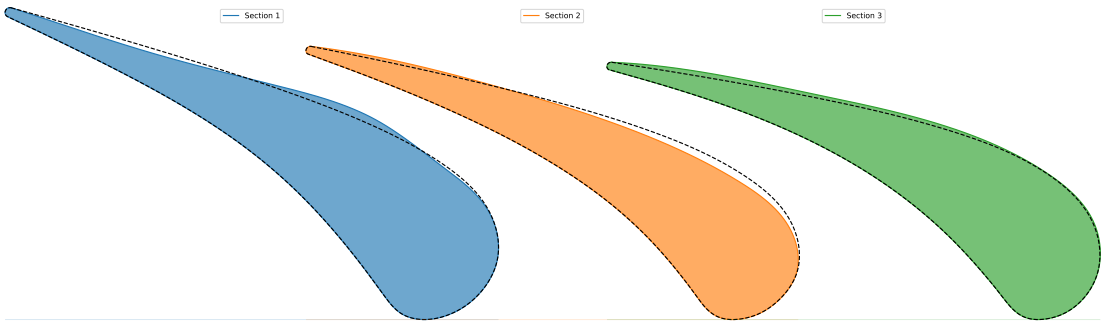


Figure 9.20: Optimal hub mid tip sections with 75 samples in the DOE

The blade sections represented below in figure 9.21 refer to the optimal solution up to the same number of iterations, so are perfectly comparable with figure 9.20.

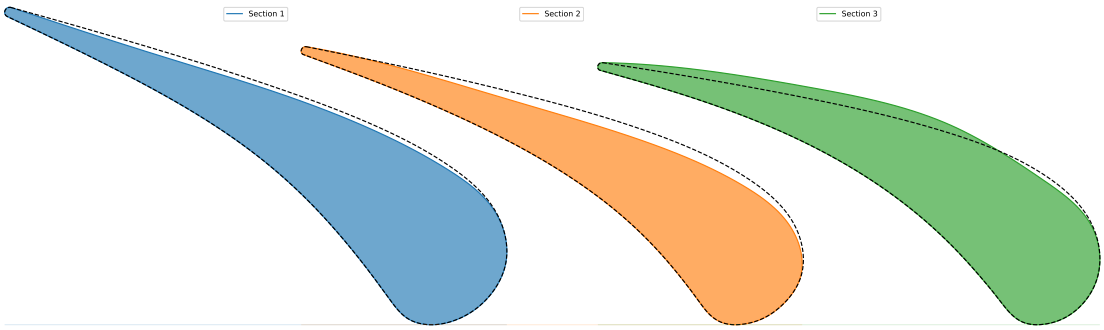


Figure 9.21: Optimal hub mid tip sections with 150 samples in the DOE

However since we have even better solutions it is interesting also to see the direction of the optimization. The optimal blade is really similar to that sketched in 9.21 simply accentuating the direction already taken, meaning that it simply digs more the hub and mid sections, and this seems to improve the performances.

9.7 Global and Local Surrogate-based Optimization

As explained in chapter 5 two kind of approaches can be applied to implement surrogate models, one called *global* that consists in building the interpolation model just once at the beginning of the optimization and after that one simulation at a time is performed, and the other called *local* that instead consists in running the DOE to build the surrogate model after each iteration. It is trivial that the latter approach requires much more computational time (one order of magnitude accordingly to Fernandez thesis [34]) but has the great advantage of being highly parallelizable having enough cores available.

In our case the local surrogate-based optimization with 150 samples is probably too heavy with the resources available, so we have compared local and global approaches with 75 items in the design of experiments. The resultant entropy drop along the iterations is shown in figure 9.22.

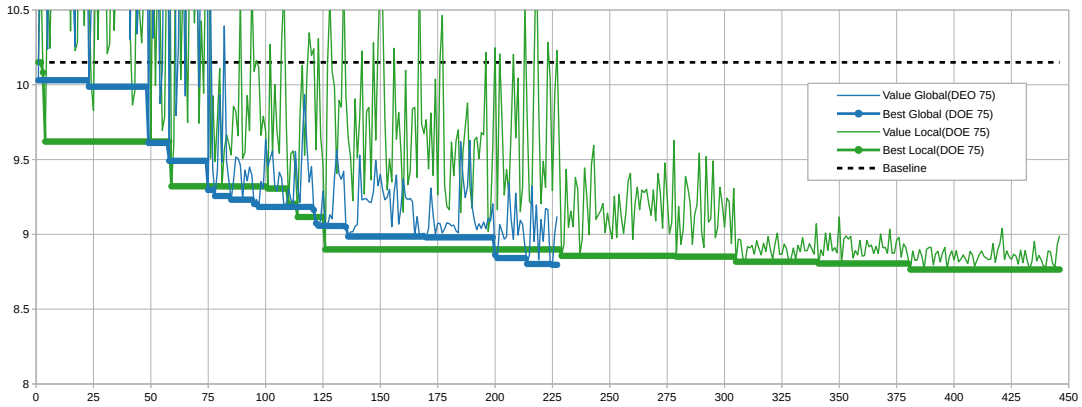


Figure 9.22: Comparison of the objective function values between SBGO and SBLO.

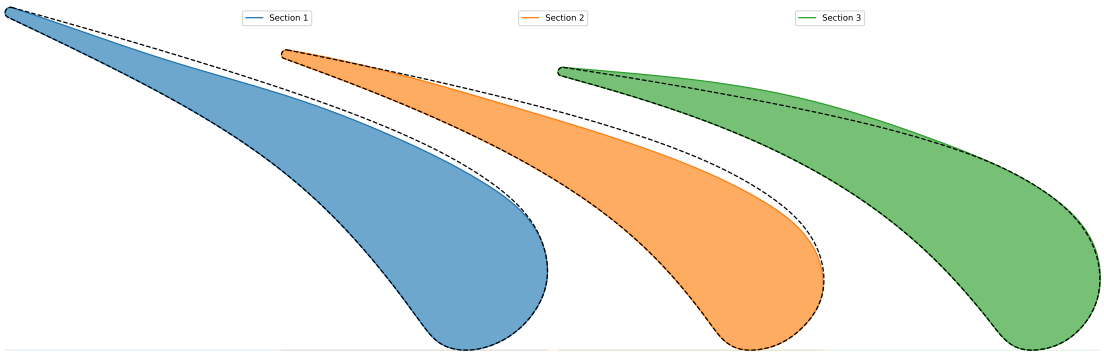


Figure 9.23: Optimal hub mid tip sections with 75 samples in the DOE with local SBO

Apparently both the methods lead to almost the same solution, but the local approach requires much more evaluation of the expensive objective function. If this is true in terms of entropy production, it is no more true when analyzing the resultant blade shape.

Actually comparing the figure 9.23 with 9.20 we can notice that they are quite different, highlighting that the optimization are directed to different solutions.

What that it is most interesting is that the direction taken in almost the same as that of global surrogate-based optimization with 150 samples in the design of experiments, and comparing figures 9.23 and 9.21, they look pretty similar.

Mathematically it is known that SBLO it is better in terms of convergence, being theoretically able to reach the absolute optimum. The fact that the global optimization with a smaller DOE is directed to a different solution can be interpreted as a symptom that for this actual case 75 samples are not enough.

9.7.1 High fidelity validation

Since all the simulations are performed at 500 000 cells, which is a pretty coarse mesh, and up to 200 iterations, it is important to be sure that the blades found by the optimization algorithm are really better than the baseline. Since cannot test tens or hundred of blades, we have decided to validate the optima of the previous simulations, meaning the cases of SBGO with 75 and 150 samples in the DOE and the SBLO with 75, other that the baseline.
The validation has been performed with a mesh of 2 millions cells with 400 iterations. The results are represented in figure 9.24.

What that most matters from this analysis is that the same ranking is valid in both low and high fidelity CFD simulations, even if the absolute value are predictably slightly different.

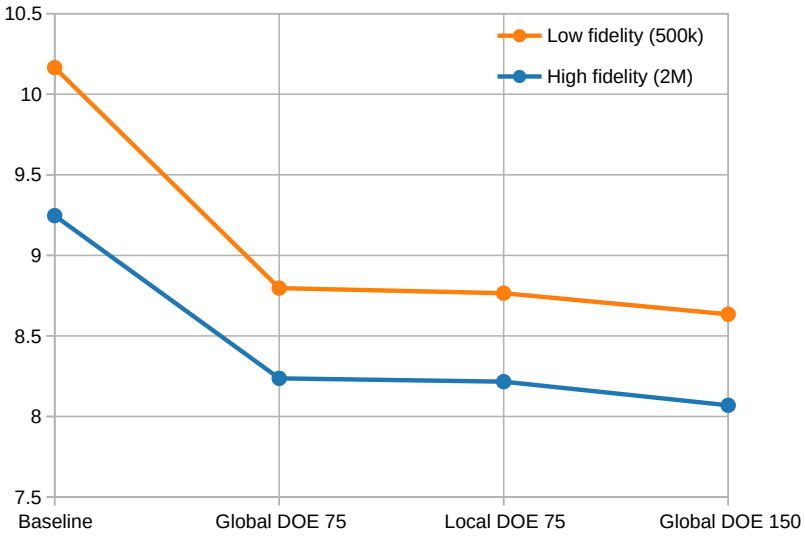
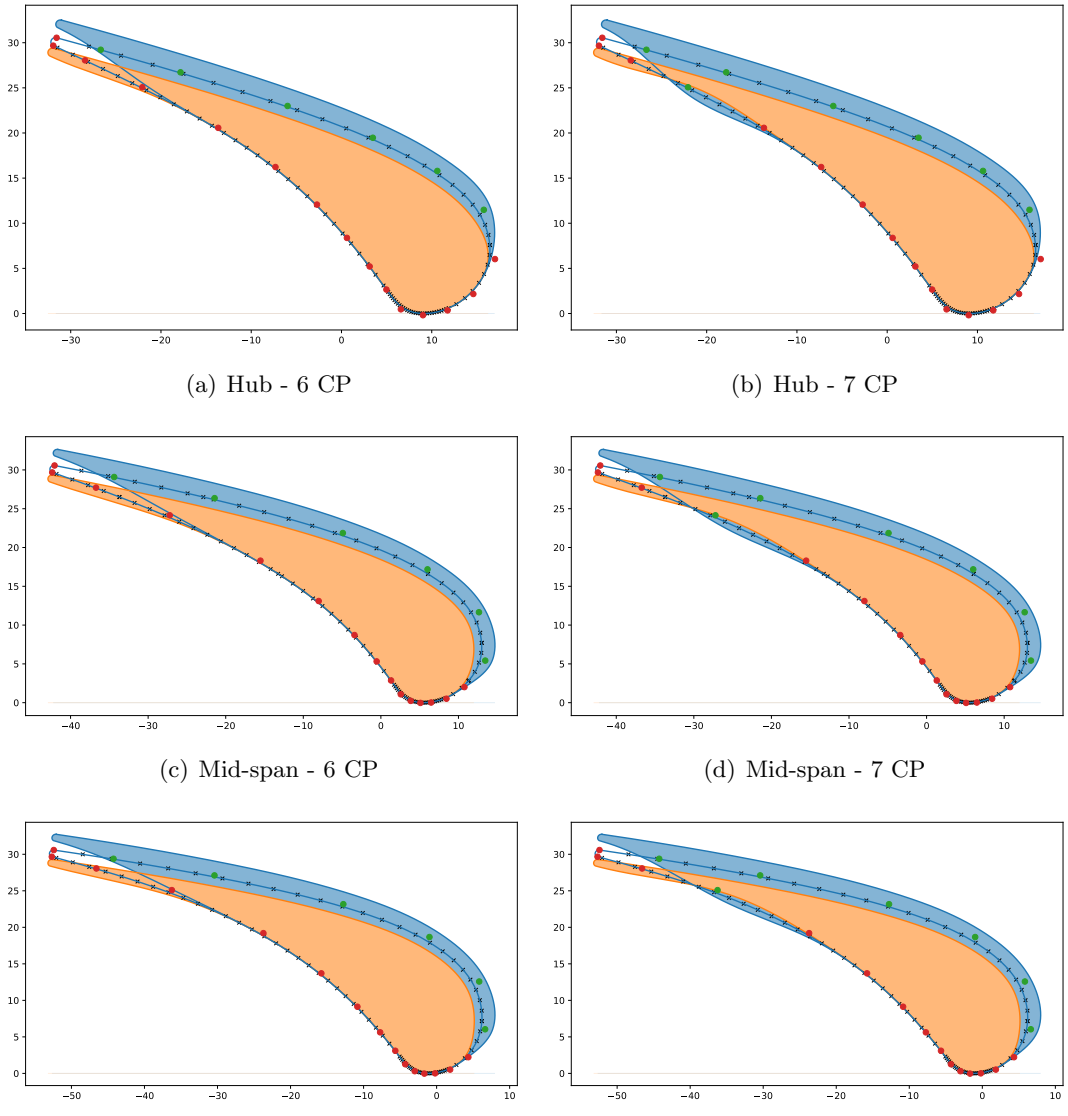


Figure 9.24: Comparison of the entropy production with a low-fidelity and an high-fidelity CFD simulation.

9.8 Enhancement on the control points

Analyzing the resulting blade in figures 9.23, 9.21, we can notice that at the hub the optimizer is trying to dig more the blade close to the trailing edge. For this reason we are going to give more degrees of freedom to the trailing edge, fixing the shape but keeping the ability to rigidly move it.

We have optimized two different design spaces, both with 6 and 7 control points. The difference between the two is a control point on the pressure side of the blade. The two blade parametrization is represented in figure 9.25.



(a) Hub - 6 CP

(b) Hub - 7 CP

(c) Mid-span - 6 CP

(d) Mid-span - 7 CP

(e) Tip - 6 CP

(f) Tip - 7 CP

Figure 9.25: Blade interpolation in three sections for the two cases.

We have seen in the previous chapter that taking ten times the number of design

variables for the DOE process, offers better performances respect to using just five times. For this reason we run 180 samples with 6 control points and 210 with 7 control points. The result along the iterations in terms of entropy production is represented in figure 9.29.

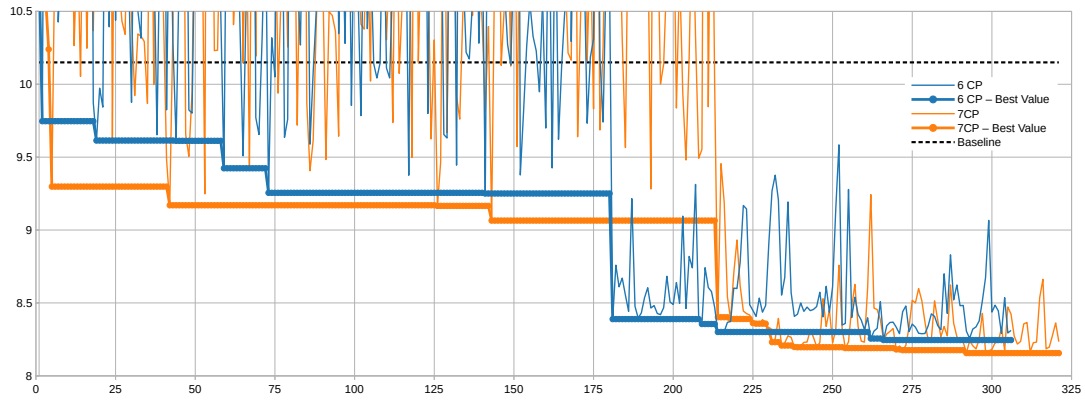


Figure 9.26: Comparison of the objective function values along iterations

Both the 2 simulations are run for a number of iterations not really extensive, since our aim was not to get the best blade directly from one of this optimizations, but they produce similar blades, drawn in figures 9.27 and 9.28.

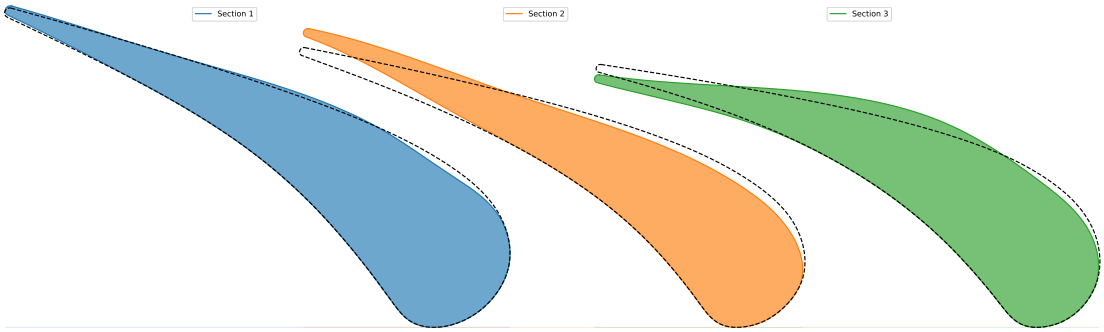


Figure 9.27: Optimal hub mid tip sections with 6 control points.

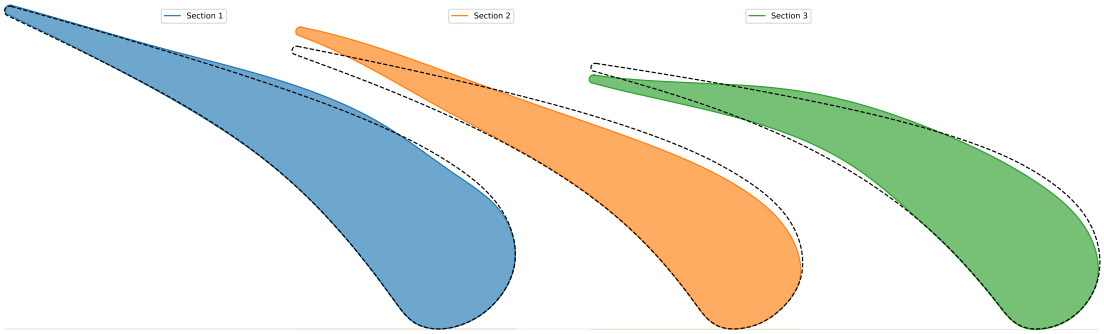


Figure 9.28: Optimal hub mid tip sections with 7 control points.

Actually up to now we have neglected the flow angle at the outlet focusing just on the minimization of the entropy production, however it is true that we want improve efficiency reducing the entropy drop, but we need also to keep the flow angle as close as possible to the designed profile.
So the next step will be to run a constrained optimization limiting the maximum difference with the outlet flow angle of the baseline case. The Dakota framework already implements the possibility to run constrained optimization, and our job was to run the post process to extract the required data from the CFD simulation, to compute the maximum difference with the baseline and the pass the data back to Dakota.

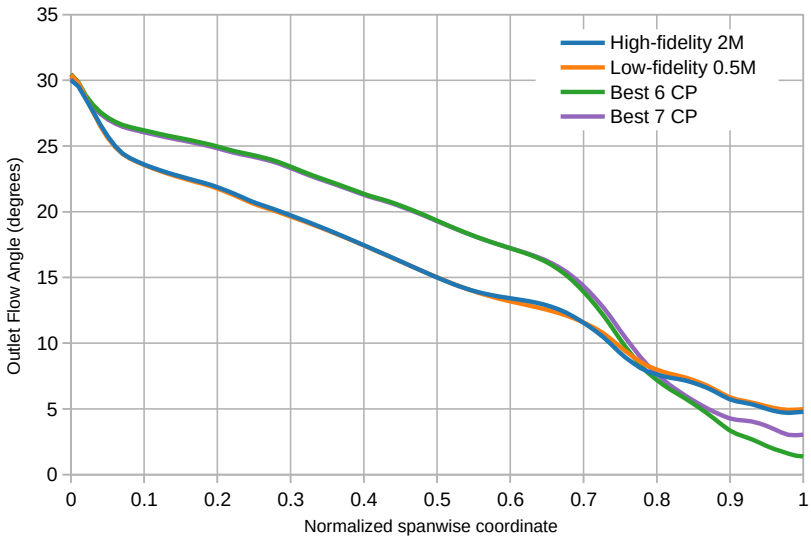


Figure 9.29: Comparison of the objective function values along iterations

9.8.1 High fidelity validation

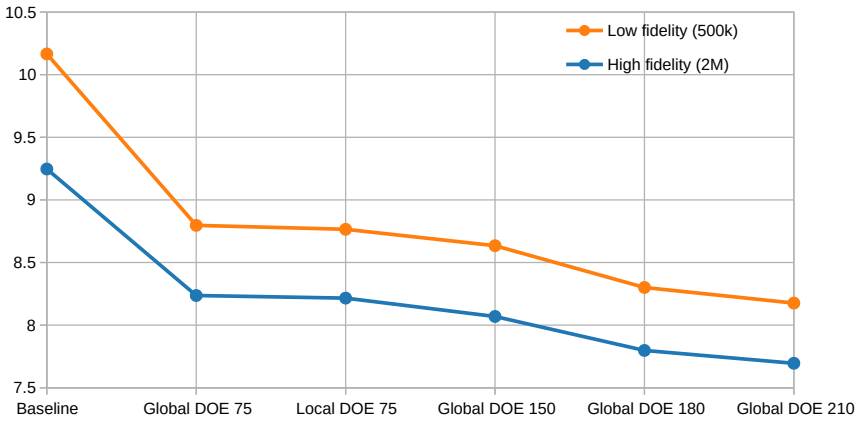


Figure 9.30: Comparison of the flow angle between optimal blades and baseline.

We need to validate the optimal solutions found by the two optimizations running an high-fidelity simulation with 2 millions cells. The result of the validation is drawn in figure 9.30.
As expected the validation has provided a positive result keeping the same rank in both high and low-fidelity simulations.

Chapter 10

Analysis of Randomness

When dealing with optimization of computational expensive functions, it is not common to launch the optimization many times with different random seeds. The seed of a random generator is that number that uniquely determine the sequence of pseudo-random number generated.

For direct genetic optimization, it is a knowledge in the energy department at Politecnico di Milano that around 10 different optimization varying the seed are required to be reasonably sure to find a good solution, since a single run it is not robust enough to guarantee to find the optimum. No information though are available for surrogate-based optimization.

In surrogate-based optimizations two steps of the process require a random numbers generator:

- the building of the design of experiments;
- the genetic algorithm for the global optimization of the surrogate.

First at all we will focus on the former which we consider the most significant, since the genetic algorithm is applied to optimize an interpolation function, and should be more robust to the random initialization, at least for unconstrained objective functions.

A multiple optimization of a 3D blade would probably be out of our opportunities in terms of available computing power, so we have carried out the seed analysis for a 2D blade for both free and constrained optimization.

The mesh ANSYS CFX is only able to solve 3D problems so the case has been modeled as quasi-2D by imposing two cells along the span-wise direction (it has been demonstrated that the flow lines do not have axial components [35] and [36]).

Passing from 70 to 2 cells along the span dramatically reduces the computational cost, and this let us to perform the optimization directly with the same mesh density of the 3D case with two millions cells, which we have considered as high-fidelity in the mesh sensitivity analysis. So the 2D mesh obtained has around 57 000 cells and it is sketched in figure 10.1.

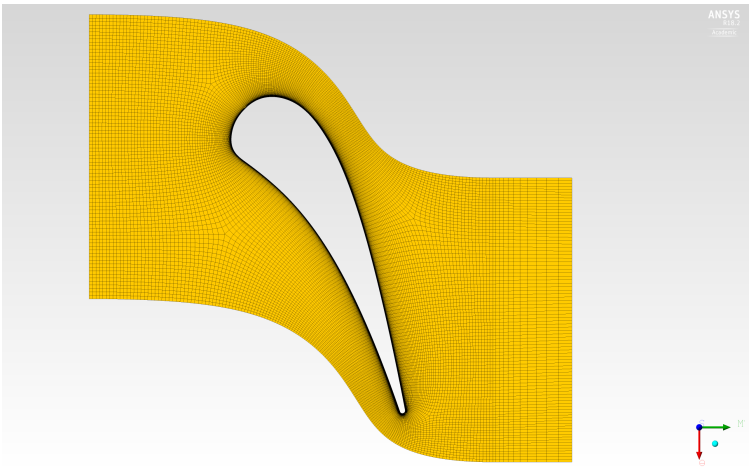


Figure 10.1: Mesh for the 2D optimization.

10.1 Seed of the DOE

We have decided to perform the comparison in both constrained and unconstrained way.

Constrained

The formulation of the constraint explained in chapter 6.1 can actually be quite challenging for the optimizer and may vary the results.

We have taken as constraint the difference between the outlet flow angle respect to the baseline case. Since we have 2 cells along the span but we want to perform a 2D optimization we take the difference at midspan.

$$\begin{aligned} &0^\circ < \delta_\alpha < 0.5^\circ \qquad \text{where:} \\ &\delta_\alpha = \left| \alpha^{(\text{mid})}_{\text{baseline}} - \alpha^{(\text{mid})} \right| \end{aligned} \tag{10.1}$$

We perform the optimization in two sets of five, for a total of ten constrained optimizations. All the simulations are exactly the same, with the only difference in the seed applied for the generation of the design of experiments. As previously seen, ten times the number of the design variables is the proper number of samples for the DOE. Having just one profile with six control points, we keep 60 iterations for the DOE and then we keep they run for a reasonable amount of time (200 iterations in total).

In figures 10.2 and 10.3 are represented the evolution of the entropy of the best solution that satisfies the constraint for the two sets of five constrained optimizations. Two aspect can be highlighted from that results: first at all we notice that all the optimizations are able to improve the baseline performances independently on the seed, but the final blade quality slightly depends on the the random parameter.

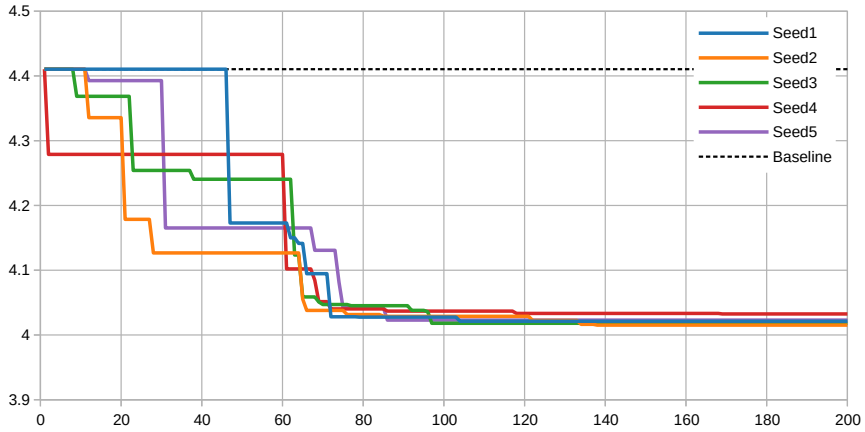


Figure 10.2: First set of 5 optimizations with different seeds (Constrained).

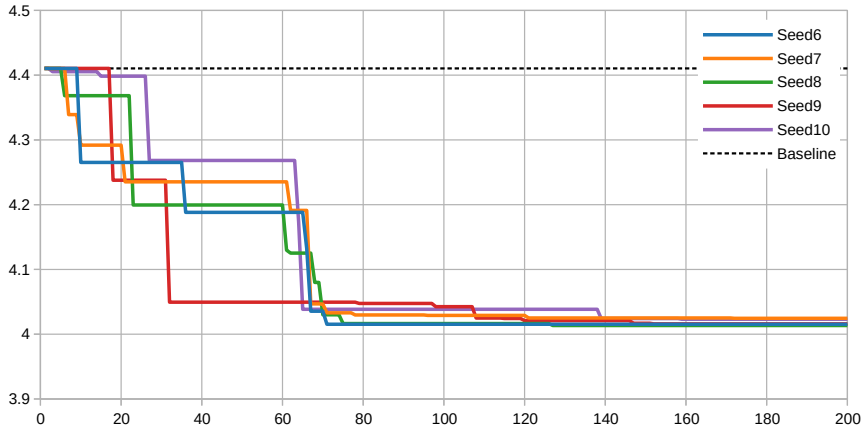
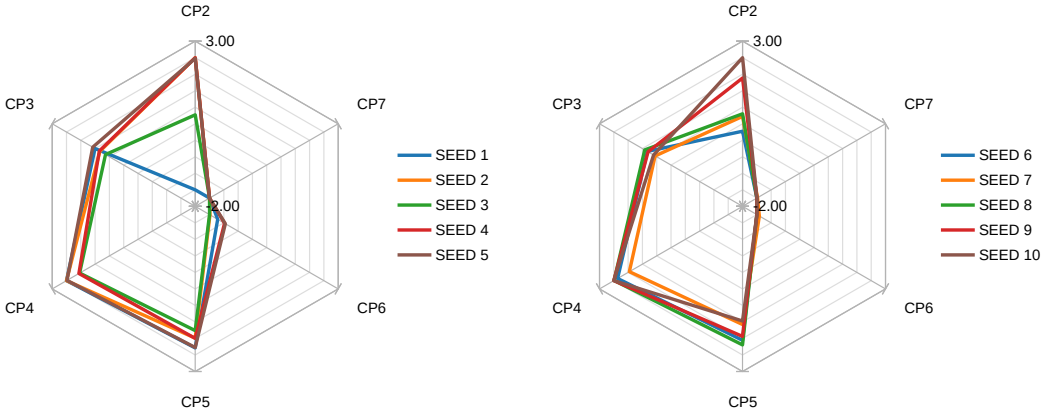


Figure 10.3: Second set of 5 optimizations with different seeds (Constrained).



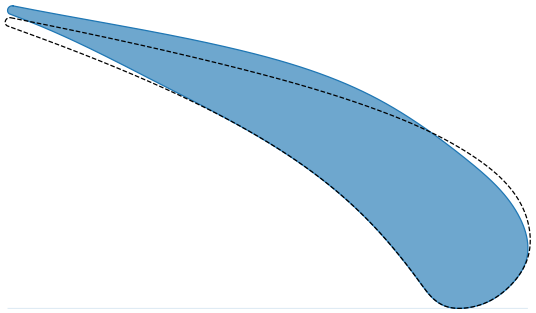
(a) First set

(b) Second set

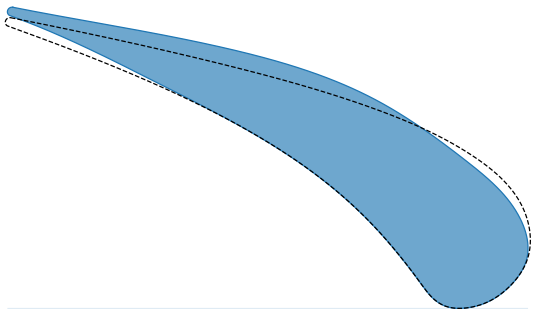
Figure 10.4: Design variables of the optimal solution with different seeds (Constrained)

The difference between the solutions can be also pointed out observing the design variables that represents the blades for all the seed, drawn in figure 10.4. Basically the blades are slightly different one each other, and this reflects to the entropy drop they produce.

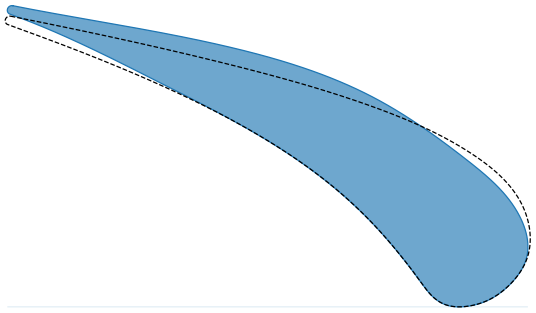
Finally the shape of the blades is drawn in figure 10.4



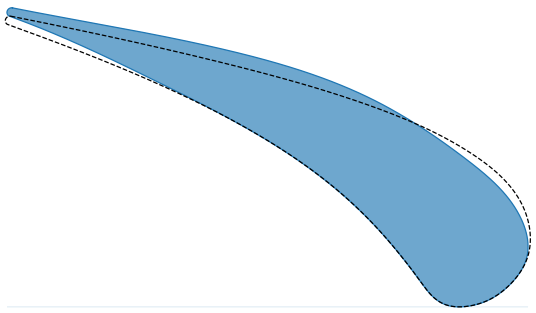
(a) Optimal blade seed 1



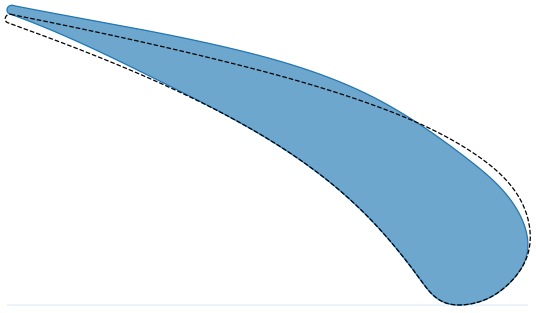
(b) Optimal blade seed 6



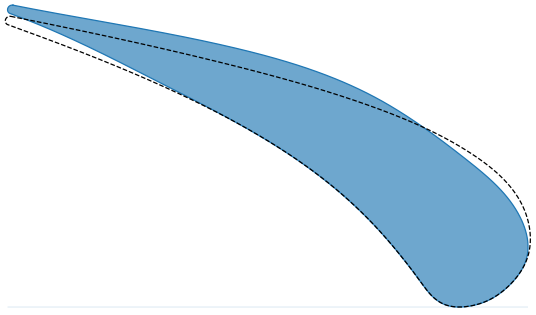
(c) Optimal blade seed 2



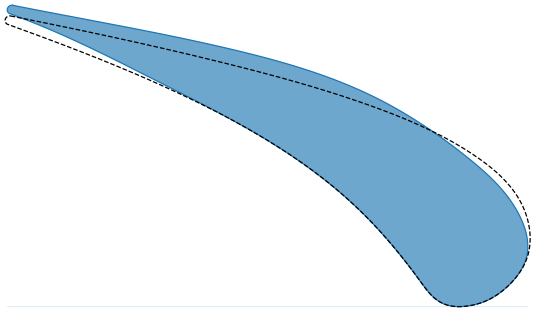
(d) Optimal blade seed 7



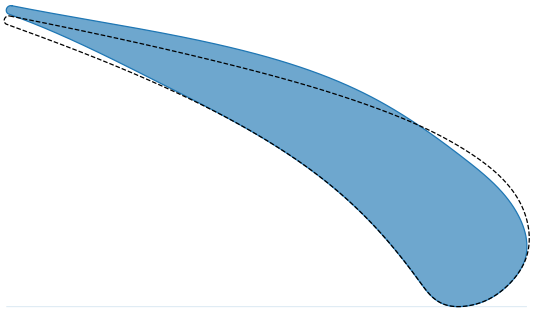
(e) Optimal blade seed 3



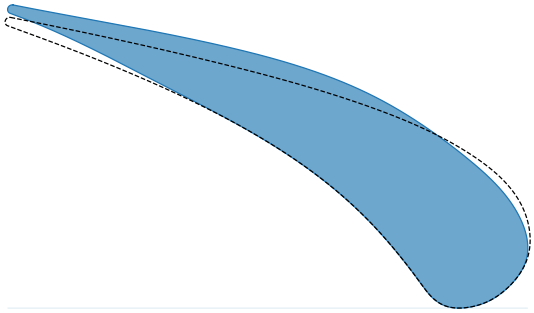
(f) Optimal blade seed 8



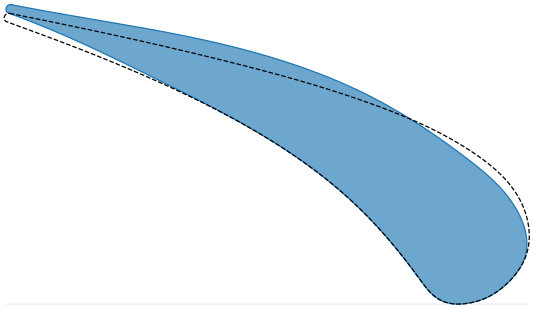
(g) Optimal blade seed 4



(h) Optimal blade seed 9



(i) Optimal blade seed 5



(j) Optimal blade seed 10

Figure 10.4

More rigid constraint

To have an idea if something changes with a more tough constraint, we have performed a set of five optimizations, in which we have changed the value of the constraint from 0.5° to 0.25°. This value should be feasible, since in the previous cases some blades better than the baseline that satisfy this constraint exist.

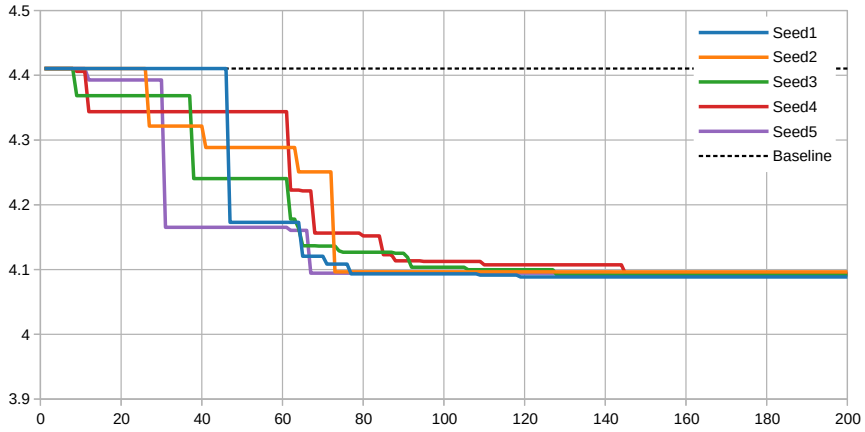


Figure 10.5: Set of 5 optimizations with different seeds (Constrained at $0.25°$*).*

In figure 10.5 are plotted the trends of the objective function value for blades that respect the new threshold. All the five optimizations are able to find a blade better than the baseline and the convergence trend are quite similar to that with a softer constraint in figures 10.2 and 10.3. Other than that clearly a more rigid constraint makes the performances of the optimal blade worsen.

For sake of completeness the design variables are drawn in figure 10.6.

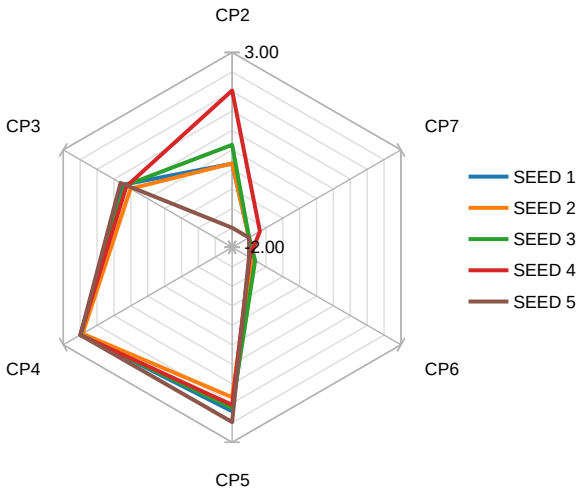


Figure 10.6: Design variables of the optimal solution with different seeds (Constrained at $0.25°$)

Unconstrained

Removing the constraint in a surrogate-based optimization the quality improves, and we can clearly appreciate this fact comparing the previous case with exactly the same case at which we have removed the boundaries on the outlet flow angle. We have run the optimizations starting from the analogous DOE.

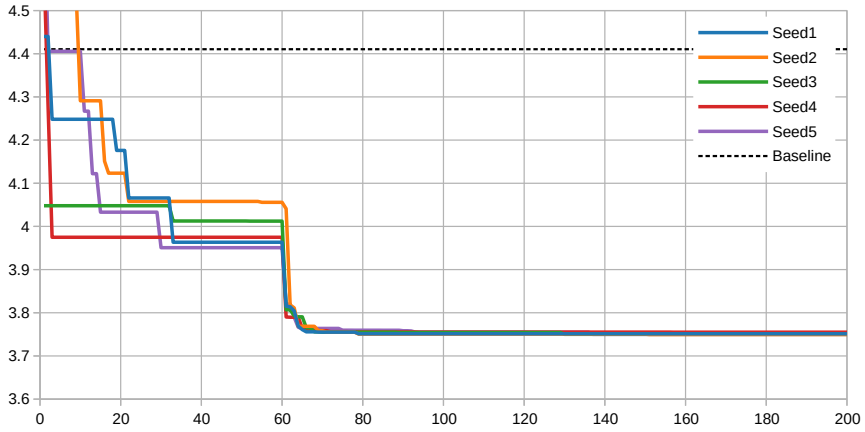


Figure 10.7: First set of 5 optimizations with different seeds (Unconstrained).

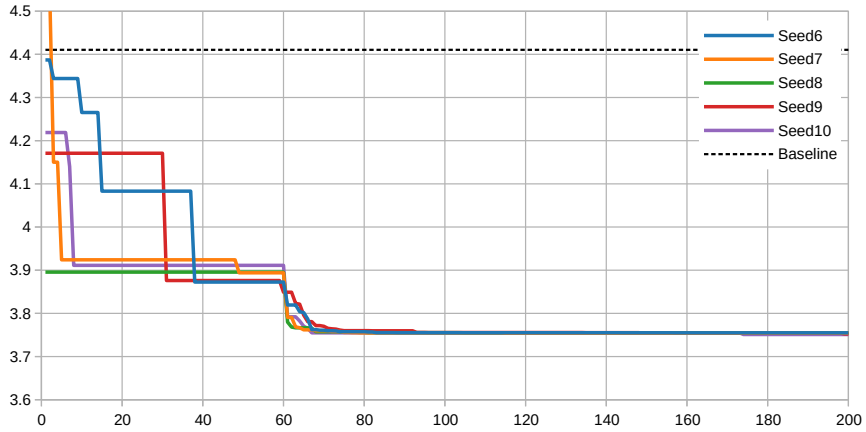
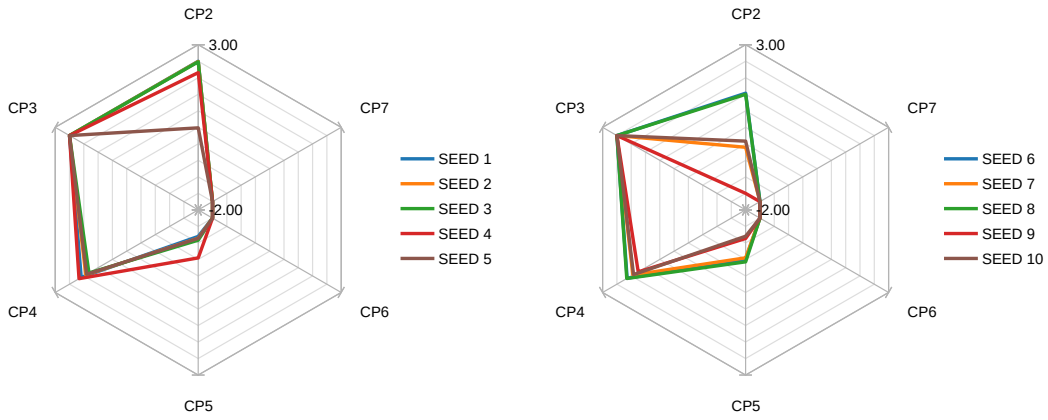


Figure 10.8: Second set of 5 optimizations with different seeds (Unconstrained).

The evolution of the entropy drop with 5 different seed shows in figures 10.7 and 10.8 that all the optimizations converge around the same value of the objective function.



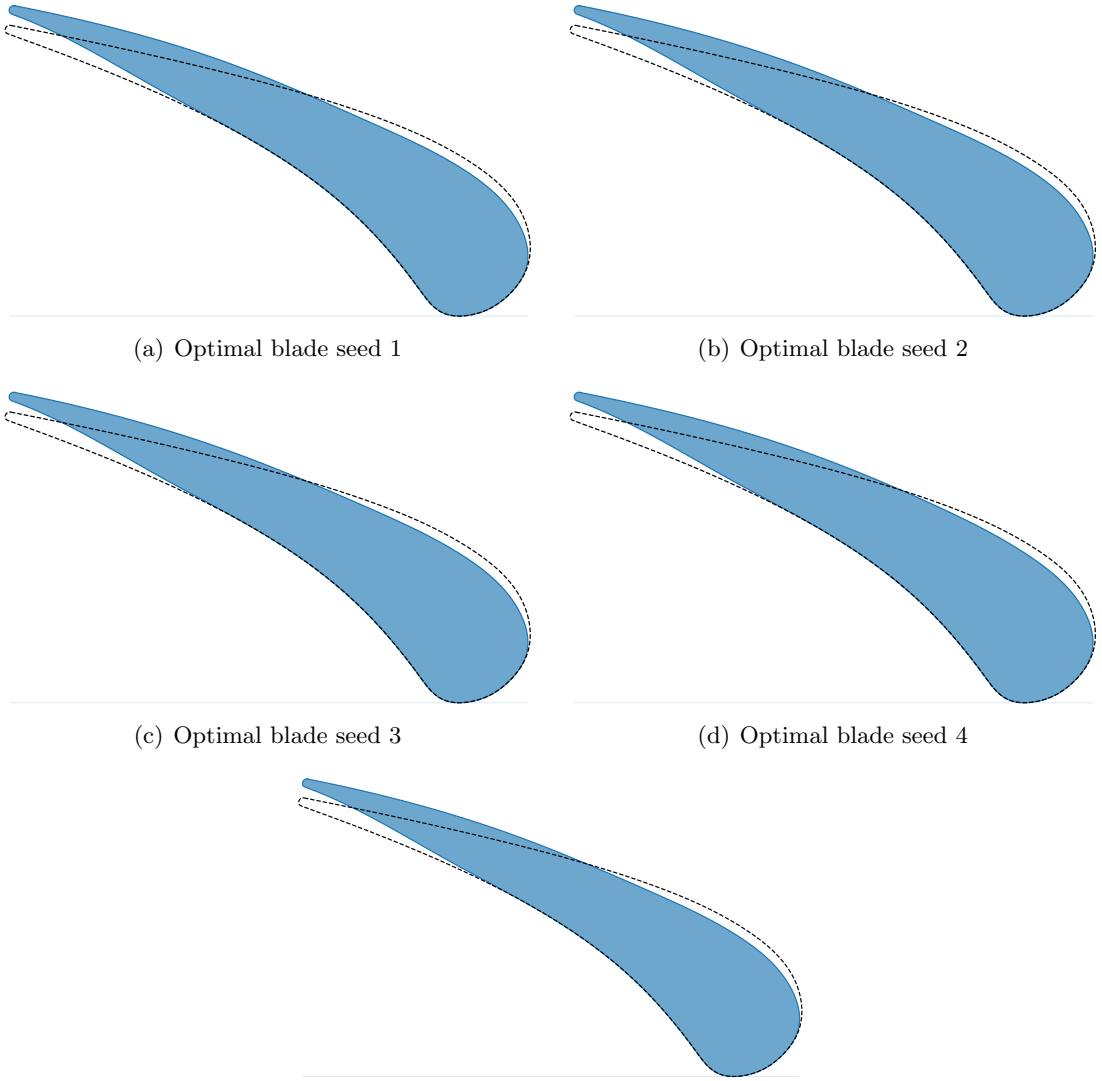
(a) First set

(b) Second set

Figure 10.9: Design variables of the optimal solution with different seeds (Unconstrained)

Also the design variables are much similar, in particular in figure 10.9 can be noticed that control points 3, 4, 5, 6, 7 are almost in the same region, while the greatest differences can be found in the displacement of control points 2, which spread in the whole available range.

Even if the difference respect to the constrained case seems negligible, in this case the blades are the same for both the leading and the trailing edge, and the only difference is in the middle of the suction side. All the blades are reported in figure 10.10.



(a) Optimal blade seed 1

(b) Optimal blade seed 2

(c) Optimal blade seed 3

(d) Optimal blade seed 4

(e) Optimal blade seed 5

Figure 10.10: Blades shape of the unconstrained seed optimization.

The unconstrained optimization is much more robust to the randomness in the determination of the samples in the DOE, and just minor differences exists between solutions. This cannot clearly be an absolute general statement, but it is a reasonable conclusion based on our knowledge of the optimization process.

10.2 Seed of the Genetic algorithm

For the analysis of the effect that the randomness has in genetic algorithm of a surrogate-based optimization we have performed two additional set with five optimization each:

- starting from the samples from the design of experiments, the optimizations are performed with a different seed for the GA;

- starting from five different DOEs, each optimization is run with two GA's seeds.

Since the constrained optimization is more critical we have been focused on that.

Same DOE - different GA seed

We have started from the doe of the seed 1 case and we have performed 4 additional optimizations changing the other random parameter. The results in terms of evolution of the best solution are represented in figure 10.11

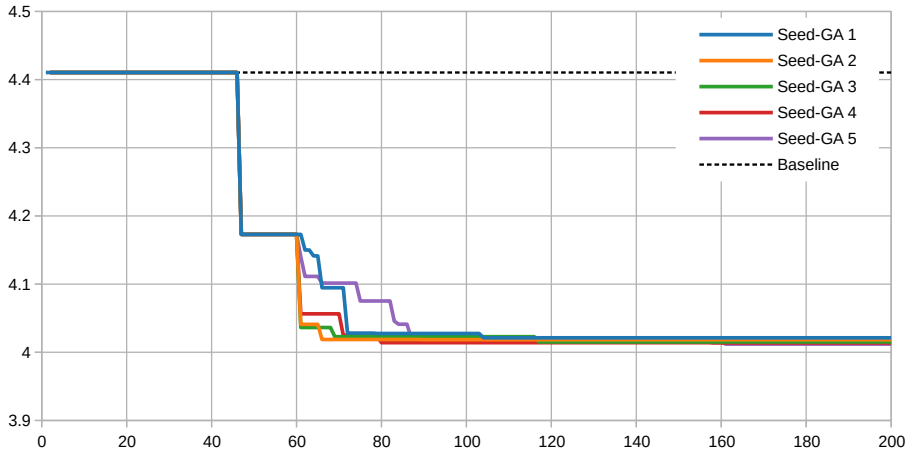


Figure 10.11: Set of 5 optimizations with different seeds for the GA (Constrained).

Then we have extrapolated the value of the design variables of the optimal solution for the set of optimizations, reported in figure 10.12.

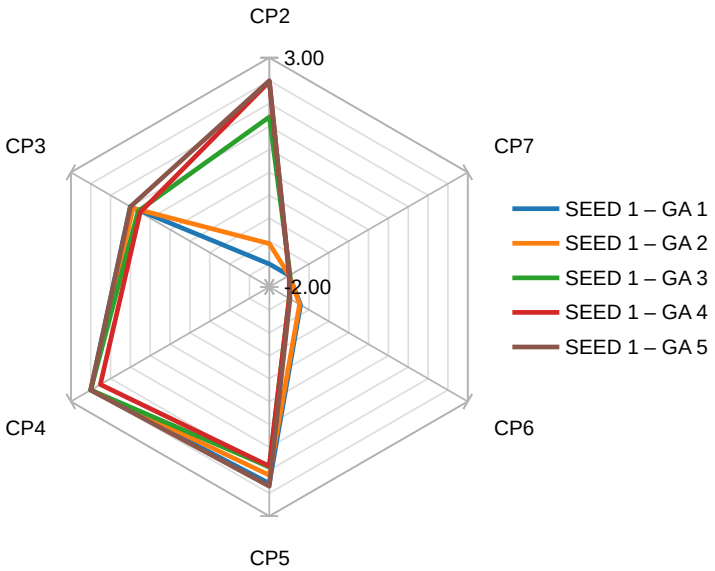


Figure 10.12: Design variables of the optimal solution with different seeds (Unconstrained)

Comparing figure 10.11 to figures 10.2 and 10.3, can be noticed that the randomness in the design of experiment step has a bit more effect compared to that of the genetic algorithm. Actually the latter shows two effects; the more evident is to change the convergence trend and the other is to slightly modify the value of convergence. However no one of the seed is able to change the convergence or not of the optimization, in any of the cases.

Different DOE and GA seeds

We have taken 5 of the constrained optimizations and we have re-run that after the DOE with a different seed of the genetic algorithm. The results of the convergence trends are reported in figures 10.13 and 10.14.

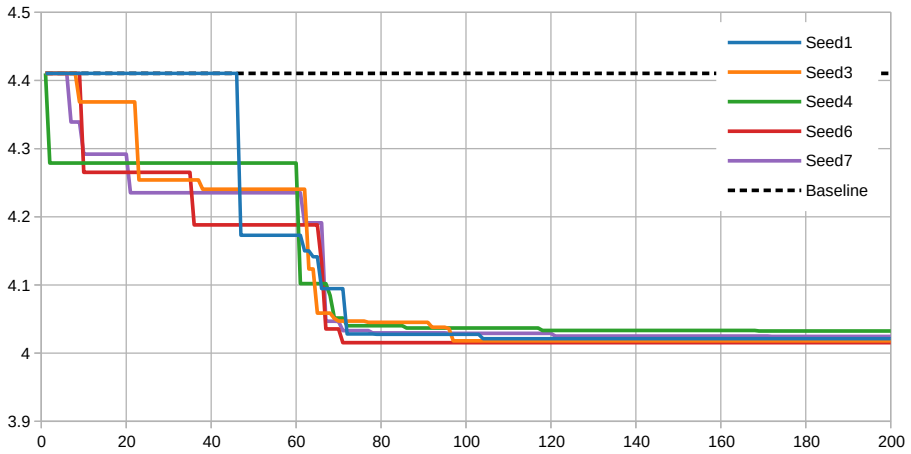


Figure 10.13: First set of 5 optimizations with different seeds (Constrained).

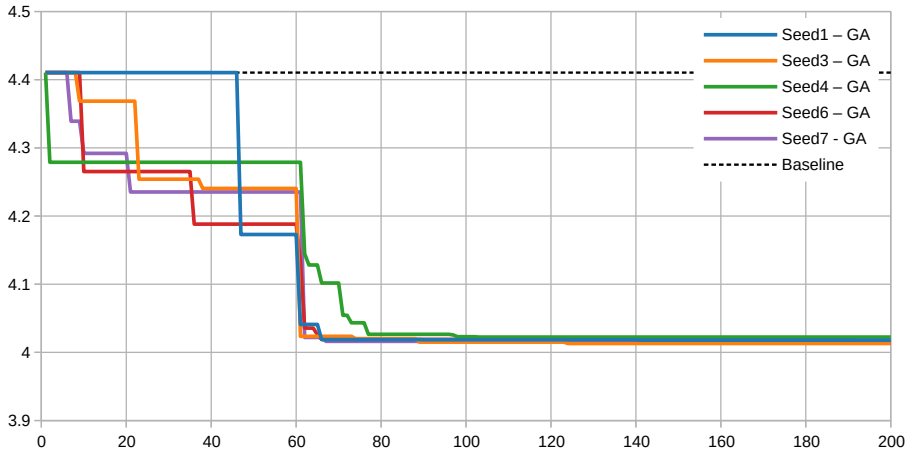


Figure 10.14: Second set of 5 optimizations with different seeds (Constrained).

To highlight the differences, in figure 10.15 is plotted the objective function value for

both the seeds. From that can be seen that a different seed guarantees slightly better performances for all the five optimizations, and a minor variance among the cases.

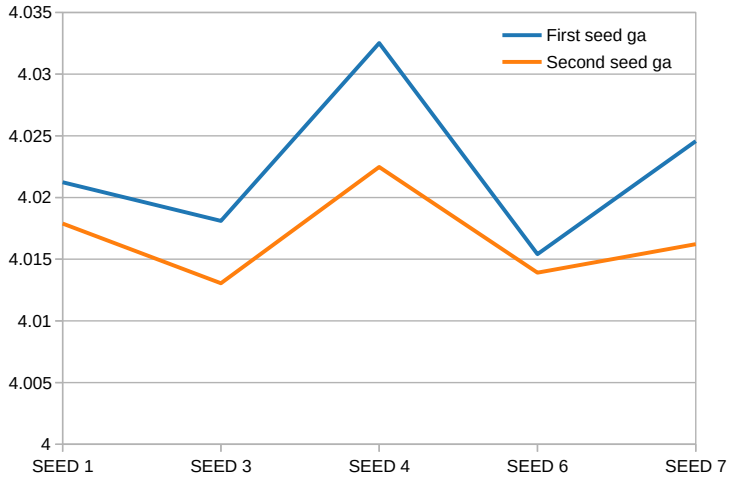


Figure 10.15: Objective function value at convergence.

Chapter 11

Optimization of the reference flow

In the following optimizations we have changed the boundary conditions of the CFD simulations. We have replaced the no-slip boundary condition for the hub and shroud surfaces with the free-slip condition. We have however kept the viscous contribution into the Navier-Stockes equation and the boundary layers around the blade.

Persico and Al. [2] actually proposed this procedure to isolate the contribution of the reference flow from the secondary flows. They shows that the secondary quantities like the flow angle are much more appropriate when projected to the reference frame defined with a 3D free-slip fluidynamic simulation.
In this way we can distinguish between 2D and secondary losses. Actually in the blade shape optimization field the may interest is to minimize 2D losses, while secondary losses can be reduced with other strategies (i.e. Denton [37] and D'Ippolito [38]) after the definition of the optimal blade.
Keeping secondary flows directly inside the optimization can bring to misleading results, in particular for what that concern the flow angle, since the secondary flows may change it of even few degrees, a much larger value that the constrain that we have set($0.5°$).

With the change in the boundary conditions also the mesh size has been adapted; it is no more necessary to reduce the cells size close to the walls, so we keep the same density of layers along the span that we had at mid-span, but we extend it to the whole span, reducing from 70 to 38 layers. Consequentially the total number of cells reduces from 500 000 to around 270 000 which reduces by a factor two the computational cost, while keeping the same blade to blade resolution (Figure 11.1).

To understand how the optimizer behaves, and the direction that it takes, we have performed many simulations changing the starting blade, the number of control points and the constraints. The result was a set of nine different optimizations, which can roughly be classified in three categories:

- original 3D blade with six control points in each of the three layers;
- cylindrical 3D blade with six control points in each of the three layers which

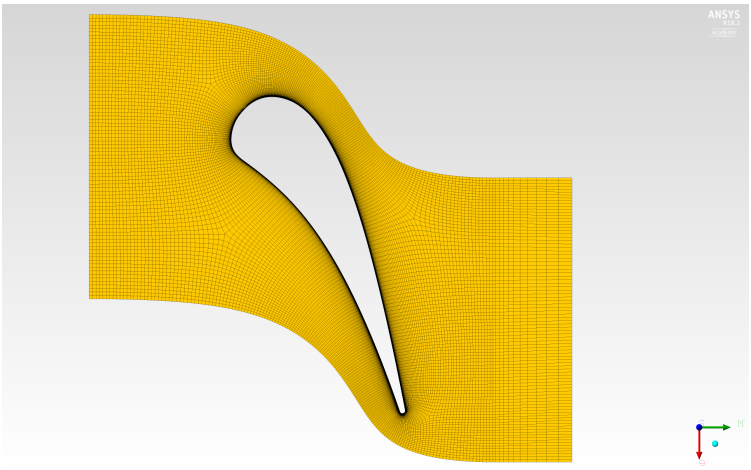


Figure 11.1: Blade to blade mesh.

results in a non-cylindrical solution;

- cylindrical 3D blade with six control points; all the three section shares the same shape.

The purely cylindrical blade has been tested just with the constraint on the angle at mid-span respect to the cylindrical baseline. The 3D original blade instead has been run both without and with constraint on the maximum difference of the outlet angle between the blade and the baseline both just at mid-span and at hub mid and tip. Finally the other case starts from the cylindrical blade as baseline but let to change independently all the three sections of hub mid and tip.

11.1 Cylindrical blade

The first and simplest test case is to optimize a purely cylindrical blade. To generate the blade, we have extrapolated the mid-span profile in the $r\theta, z$ flat plane and we have converted into x, y, z coordinates in the hub, mid and tip sections accordingly to the proper radius, as explained in chapter 8.1.1, then interpolated along the span with a B-spline.
The resultant design space has a total of 6 design variables, and accordingly to previous considerations, the design of experiments has a total of 60 samples.

We have considered as constraint the outlet flow angle at mid-span respect to the cylindrical baseline, with a margin of 0.5°. We have chosen not to take the whole distribution along the span since we have no direct effect other that at mid-span.

The convergence trend in figure 11.2 shows an improvement over the baseline while satisfying the constraint. After around 80 high-fidelity evaluations, the algorithm almost stagnates.

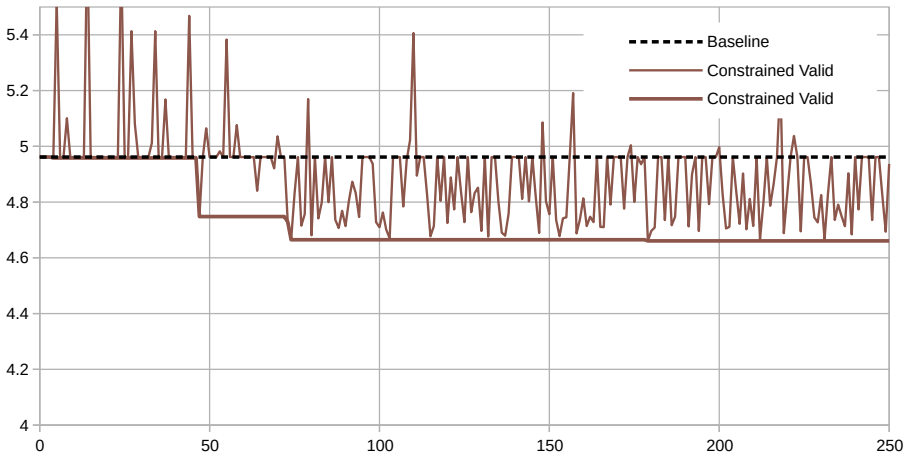


Figure 11.2: Convergence trend of the cylindrical case with 6CP.

The optimal mid-span blade profile is shown in figure 11.3.

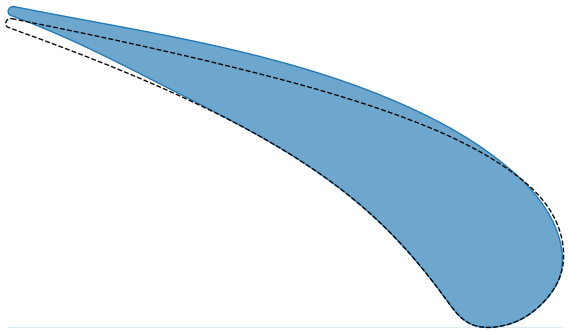


Figure 11.3: Convergence trend of the cylindrical case with 6CP.

The same optimization has been tested also with a constant outlet flow angle fixed at hub, mid and tip.

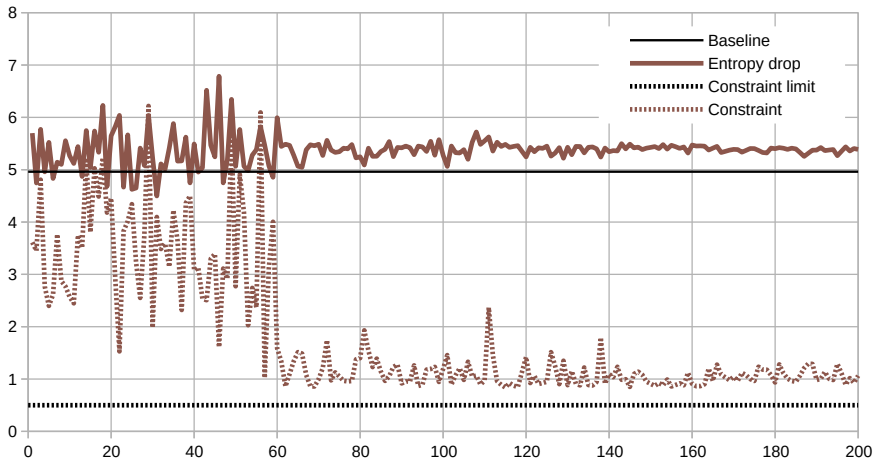


Figure 11.4: Convergence trend of the cylindrical case with constant angle constraint.

This simulation has also the aim to check if we were able to obtain a constant angle with a purely cylindrical blade which performs better than the baseline. Unfortunately as shown in figure 11.4 the optimizer was not able to find a blade with a difference smaller that 0.5°, and even increasing the threshold to 1°, there is no solution which can improve the performances of the baseline.

11.2 Cylindrical baseline

Starting from a baseline cylindrical blade we have set up an optimization with three sections at hub, mid and tip, each one with 6 control points, for a total of 18. Accordingly to that the number of samples of the design of experiments procedure is chosen as ten times the design variables so 180 in total.

In this case we have applied different constraints:

- unconstrained;
- flow angle at mid-span;
- flow angle at hub, mid and tip;
- constant flow angle at outlet;
- flow angle equal to that of the 3D original blade.

1) Unconstrained

The unconstrained optimization has been performed for sake of comparison, to understand where the blade shape should go if the outflow angle is free to change. The convergence trend of the optimal entropy is shown in figure 11.5.

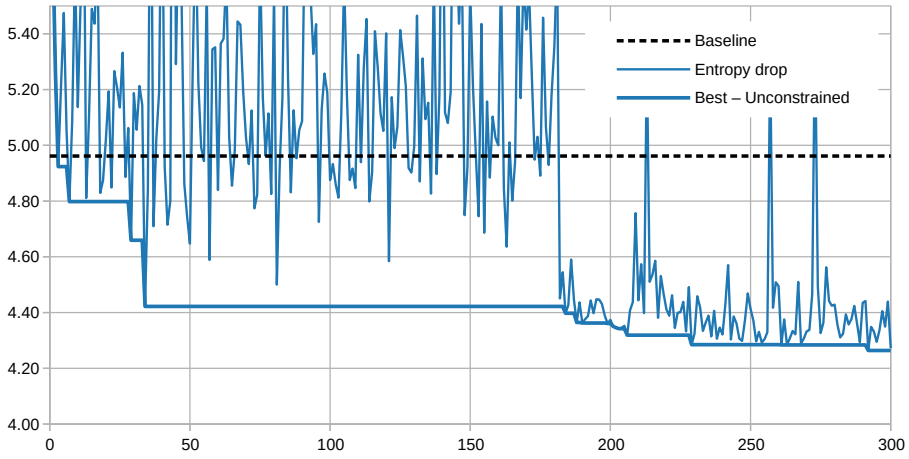


Figure 11.5: Convergence trend of the unconstrained cylindrical case with 18CP.

Analyzing the outlet flow angle compared to the baseline we can understand how the solver is able to reach such level of entropy production on a blade which already performs quite well; in figure 11.6 we can see that the optimal blade deflects the flow almost 3° less than the baseline, and with less deflection the efficiency increases. This however is a prove that the algorithm is working well.

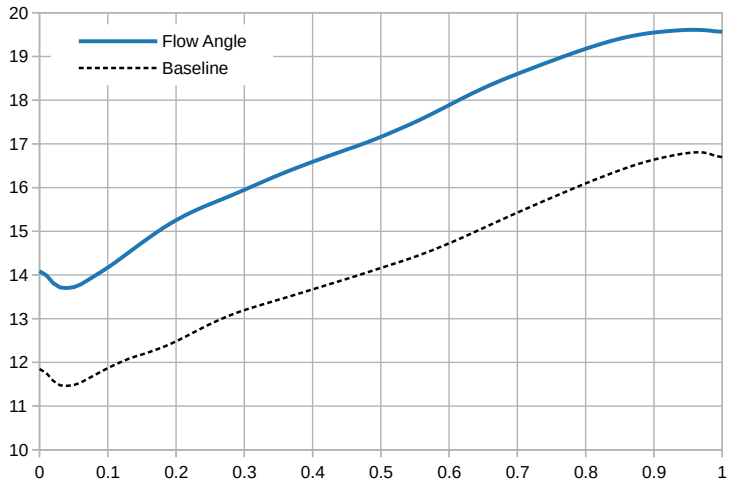


Figure 11.6: Outlet flow angle along the span for the unconstrained optimization.

The hub, mid and the tip sections are represented side by side in figure 11.7 from left to right.

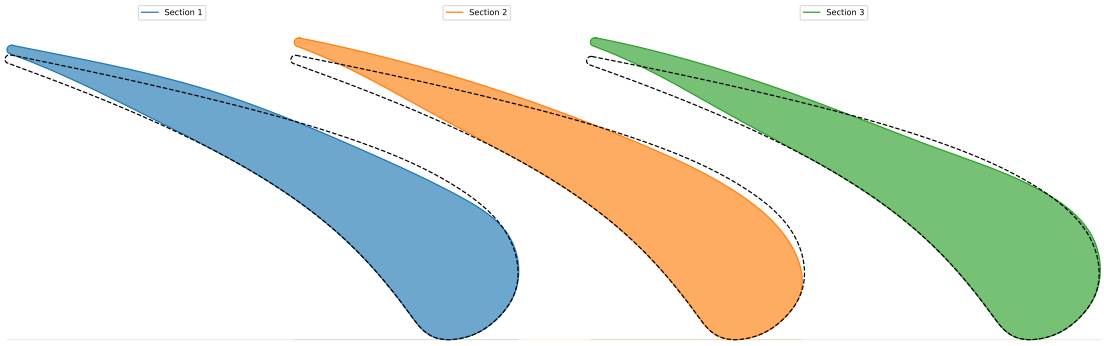


Figure 11.7: Hub, mid and tip sections of the optimal blade (Side by side).

2) Mid-span constraint

Another simulation run for the purpose of better understanding both the blade and the algorithm was with the constraint on the outlet flow angle applied just at mid-span, leaving it free in the other parts along the span For the simulations that did not satisfied the constraint of 0.5° at mid-span the value has been substituted with the entropy production of the baseline.

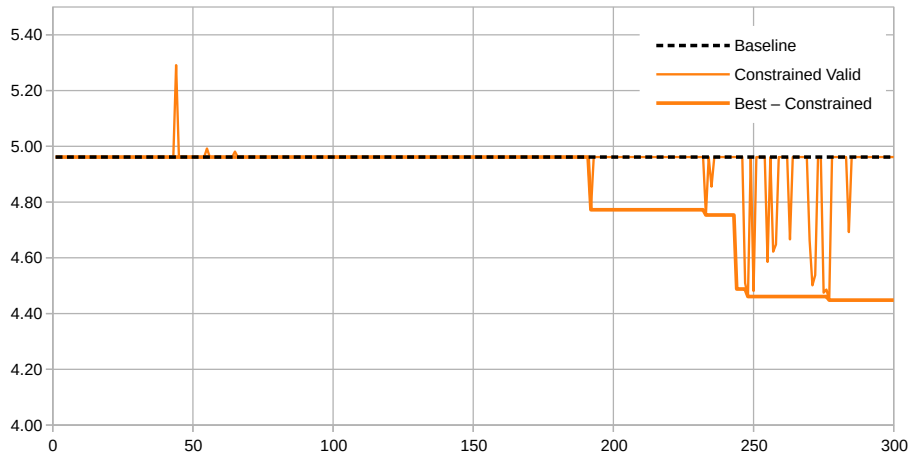


Figure 11.8: Convergence trend of the unconstrained cylindrical case with 18CP.

The distribution of the outlet flow angle in figure 11.9 shows an interesting and hopefully expected behavior, since the angle modifies it to satisfy the constraint at mid-span while trying to replicate the unconstrained solution at hub and tip.

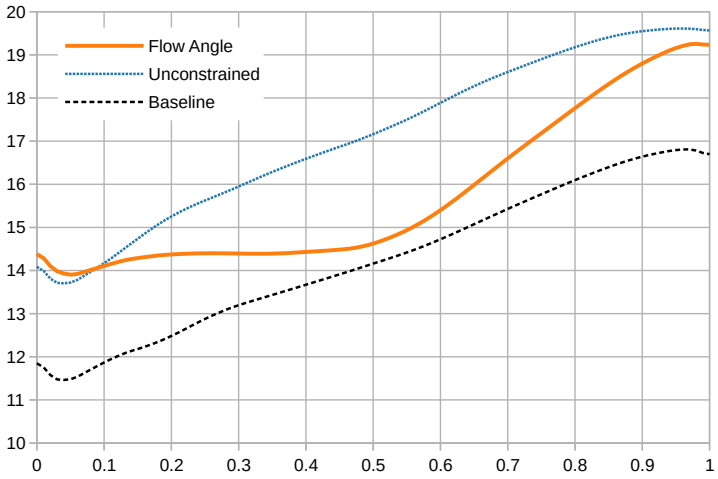


Figure 11.9: Outlet flow angle along the span for the mid-constrained optimization.

The same behavior can be highlighted also comparing figures 11.10 and 11.7; the hub and the tip profiles are quite similar in both cases, while the mid one, due to the constraint, is really different.

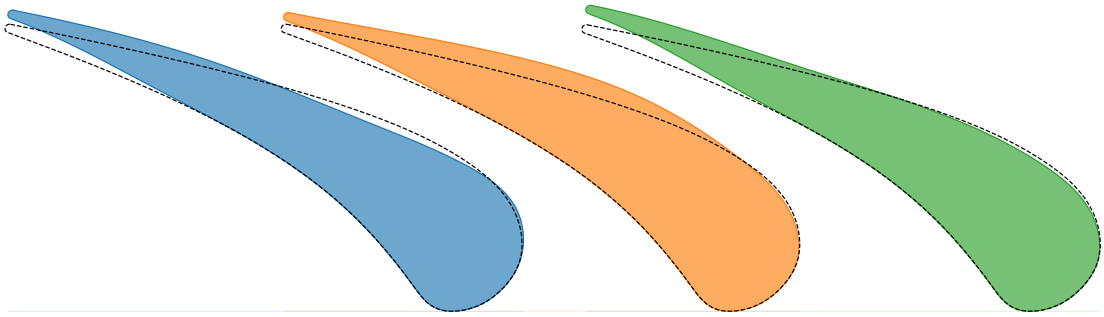


Figure 11.10: Hub, mid and tip sections of the optimal blade (Side by side).

3) Full-span constraint

To perform an optimization with an outlet flow angle similar to that of the baseline along the whole span we have decided to bind the angle in the 3 sections where we have the control on the shape, meaning hub, mid and tip, so the constraint is formulated as:

$$\begin{aligned} &0^\circ < \delta_\alpha < 0.5^\circ \qquad \text{where:} \\ &\delta_\alpha = \max_i \left| \alpha^{(i)}_{\text{baseline}} - \alpha(i) \right| \quad \text{for } i \text{ in hub, mid and tip} \end{aligned} \tag{11.1}$$

This case is much more realistic since we are applying a quite strong constraint on the geometry, and the optimization is much tougher. The convergence trend in figure 11.11 shows less simulations that satisfy the constraint, in particular in the design of experiments phase, but we are however able to reach a better value for the entropy production than the cylindrical baseline with a similar flow angle.

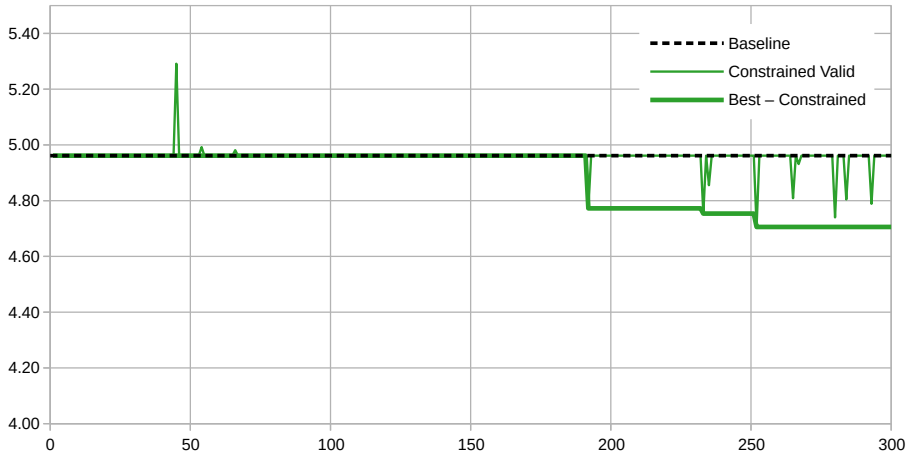


Figure 11.11: Convergence trend of the full-constrained cylindrical case with 18CP.

Analyzing the outlet flow angle drawn in figure 11.12 we can highlight that the three-points constraint is almost able to replicate the baseline flow angle with good accuracy providing better efficiency.

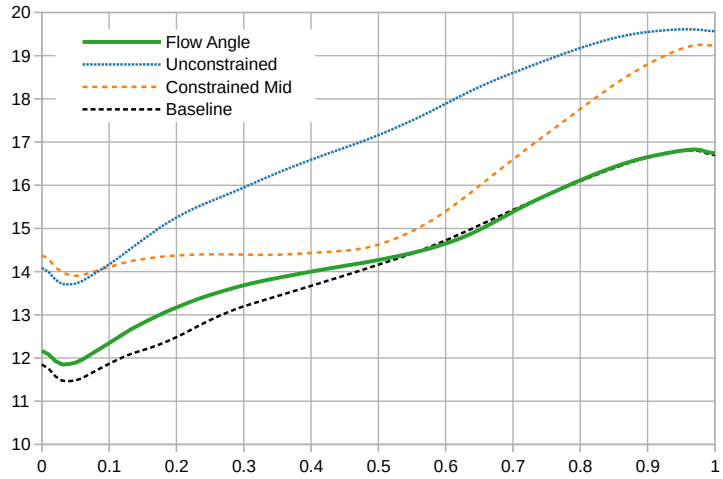


Figure 11.12: Outlet flow angle along the span for the full-constrained optimization.

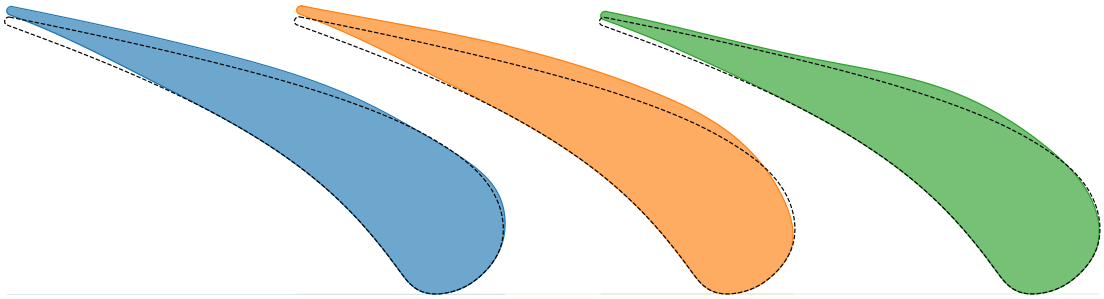


Figure 11.13: Hub, mid and tip sections of the optimal blade (Side by side).

As expected the mid-span profile is very similar to that in figure 11.10, while the hub and shroud sections are quite different due to the constraint applied.

4) Constant angle constraint

Additionally to the previous optimizations we have also tried to obtain a constant flow angle from at the of the stator, setting it equal to that at mid-span of the baseline case. This constraint is applied both at hub, mid and tip, like in the previous section, to guarantee the correct angle along the span.

The convergence trend is similar to that of paragraph 3), but with even more difficulties to satisfy the constraint; however the entropy production of the optimal solution is quite similar to that.

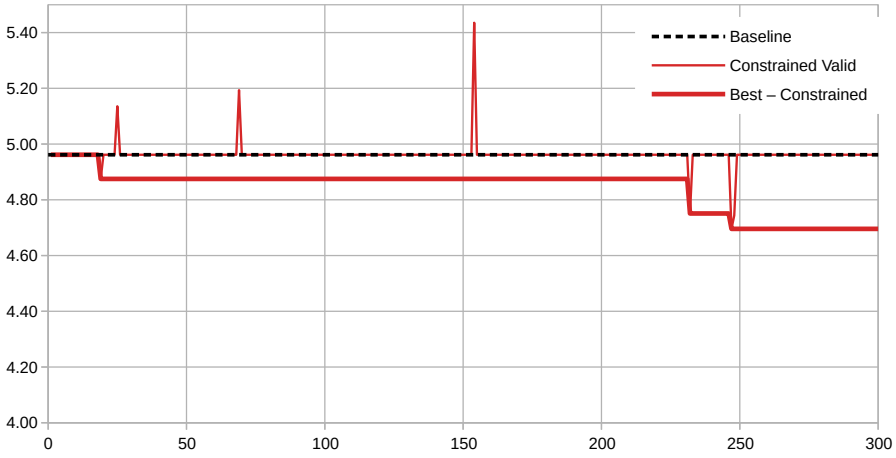


Figure 11.14: Convergence trend of the full-constrained cylindrical case with 18CP.

Then the flow angle is represented in figure 11.15; performing the average distance between the reference value and actual outlet angle the resulting value is around 0.48°, just above the limit of 0.5° that we have set at the three relevant sections.

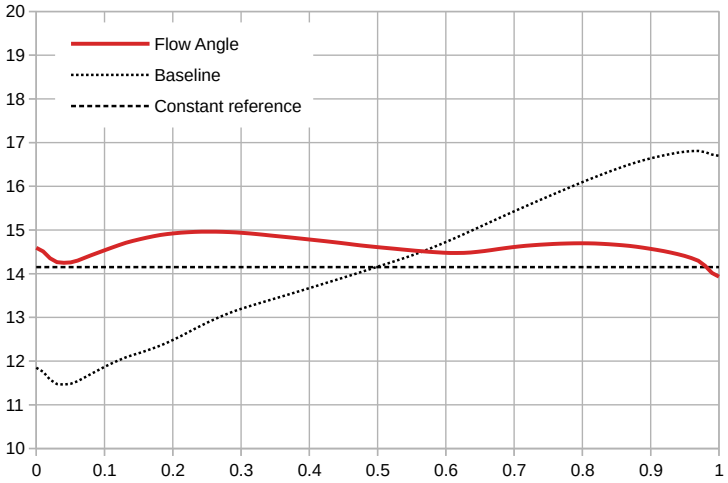


Figure 11.15: Outlet flow angle along the span for the full-constrained optimization.

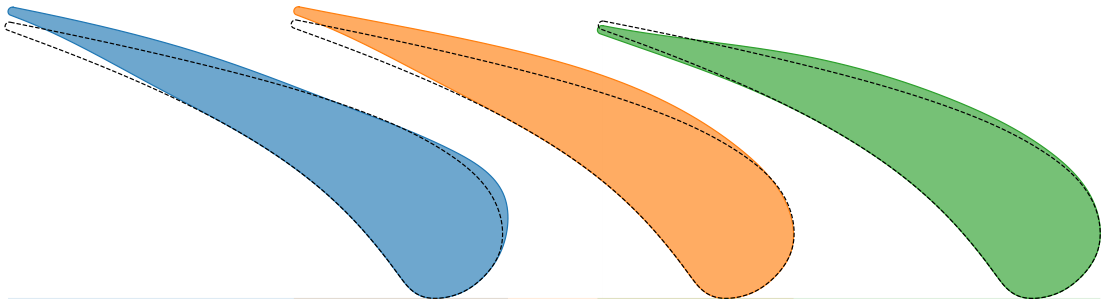


Figure 11.16: Hub, mid and tip sections of the optimal blade (Side by side).

The shape of the blade, drawn in figure 11.16, at mid-span is very similar to figures 11.10 and 11.13.

5) Twisted blade constraint

The latest optimization that starts from the cylindrical blade has been performed trying to verify if we were able to replicate the flow angle of the twisted blade. Unfortunately the test has been unsuccessful, since any blade is not able to satisfy the constraint, and the least difference was around two degrees. For sake of completeness we report in figure 11.17 the evaluation of the objectives and the constraints along the iterations.

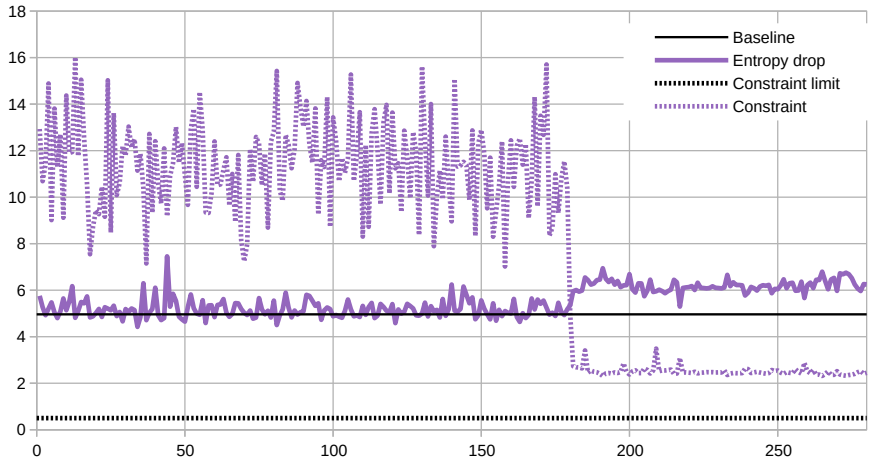


Figure 11.17: Convergence trend of the constrained cylindrical case with 18CP.

11.2.1 High-fidelity validation

All the optimizations are performed with a blade to blade density equivalent to the 500 000 cells mesh, which is enough for that process, but we can now verify that the progresses in terms of optimal blades are still valid with a mesh with higher fidelity. In this case we replicate the density of the 2 000 000 cells mesh for validation purpose, so the number of cells for the high-fidelity validation passes for 270 000 cells to 1 085 000.

We have simulated the optimal blades of all the optimizations (except paragraph 5) in which no blades satisfy the constraint) to be sure that the ranking between them are maintained also with a finer mesh, and from figure 11.18 this is clearly valid.

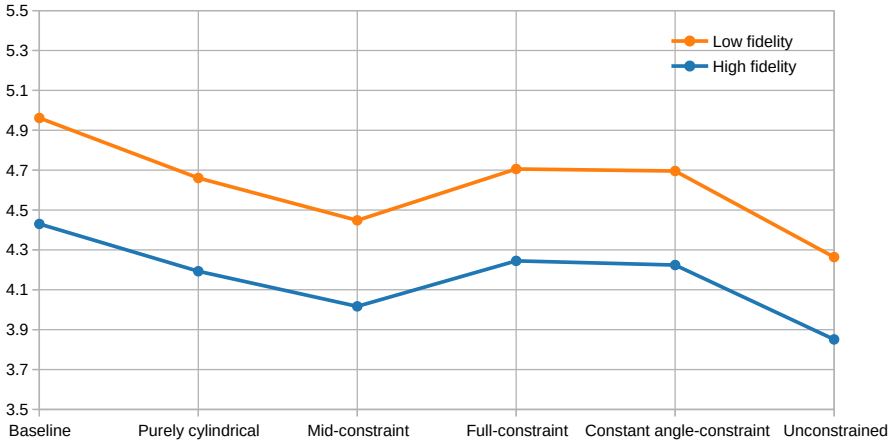


Figure 11.18: Comparison between 270 000 *and* 1 085 000 *cells.*

11.3 Twisted baseline

Starting from the original blade instead of the cylindrical we have performed a smaller set of the most significant optimizations. We have kept the same number of control points and samples for the generation of the design of experiments.

In this case the constraints that we have applied are:

- unconstrained;
- flow angle at mid-span;
- flow angle at hub, mid and tip;

6) Unconstrained

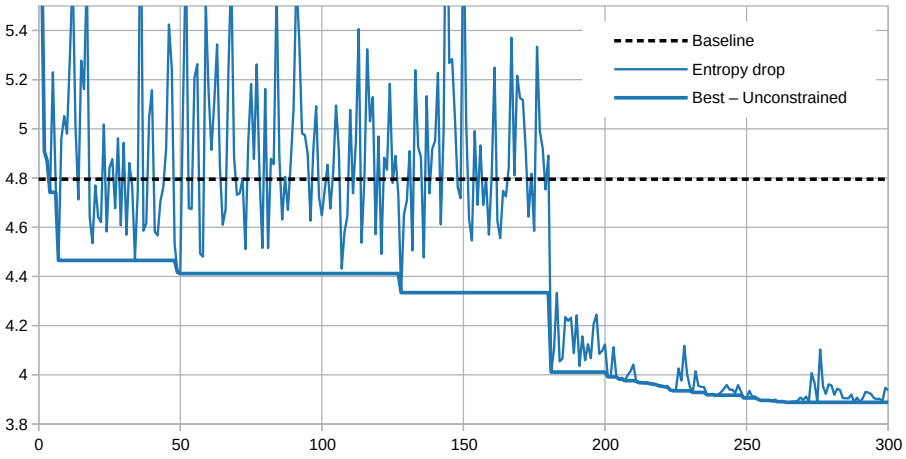


Figure 11.19: Convergence trend of the unconstrained case with 18CP.

The unconstrained optimization has been performed for sake of comparison, to understand where the blade shape should go if the outflow angle is free to change. The convergence trend of the optimal entropy is shown in figure 11.19, and it shows pretty good results, since after the DOE the performances progressively increases; the optimizer finds a region of the design space with good blades and then it improves the surrogate and it explores over there.

The outlet flow angle compared to the baseline is quite different from the previous case, but it is conserved the fact that the deflection on average is reduced.

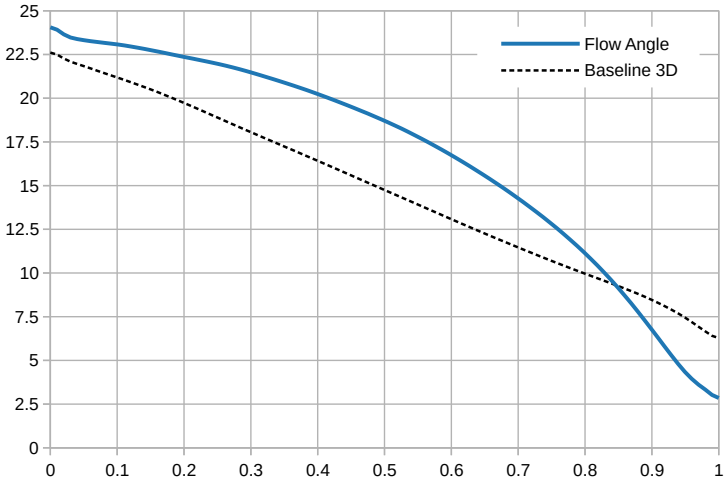


Figure 11.20: Outlet flow angle along the span for the unconstrained optimization.

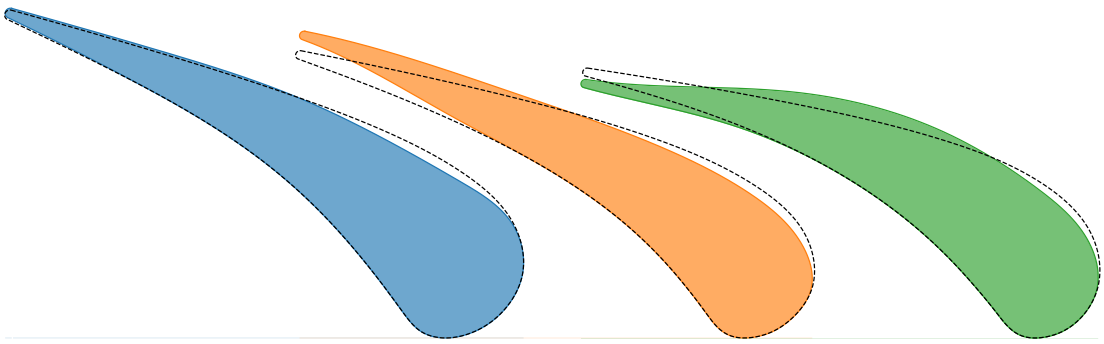


Figure 11.21: Hub, mid and tip sections of the optimal blade (Side by side).

For what that concern the blade representation in this case the baseline is no more cylindrical so we will provide two different kind of figures, 11.21 and 11.22 where the former just contains the three relevant sections side by side, just for sake of simplicity in the identification of the profiles characteristics, while the latter shows the real section positioning one to each other from a radial point of view.

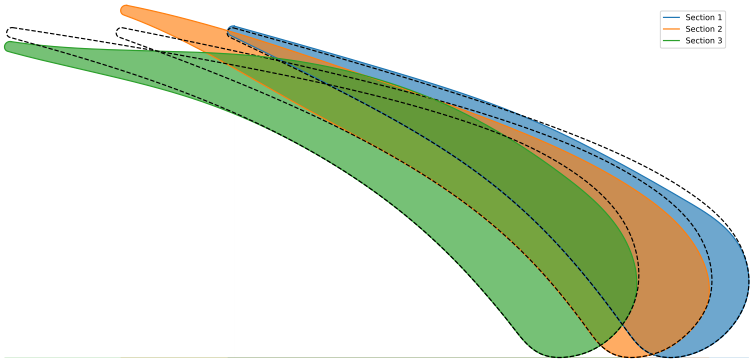


Figure 11.22: Hub, mid and tip sections of the optimal blade from a radial point of view.

7) Mid-span constraint

Similarly we have applied the constraint of 0.5° of difference respect to the baseline only at mid-span with the same representation rules as before, substituting with the baseline all the simulations that did not satisfied the constraint.

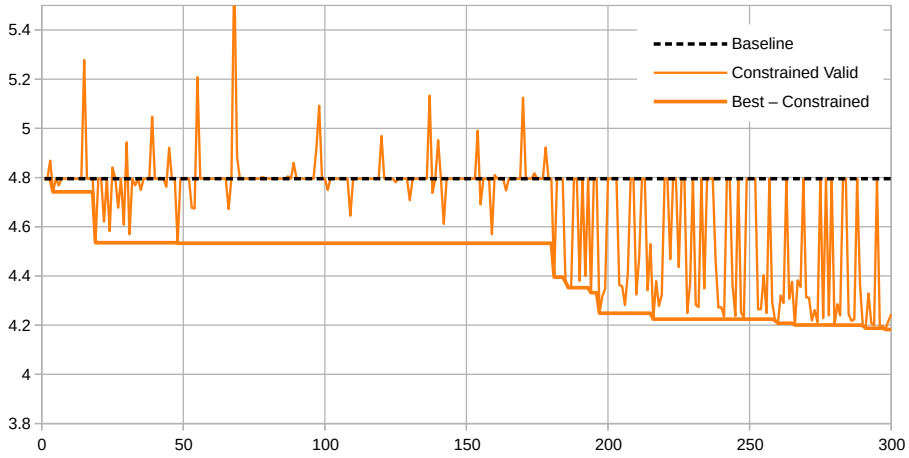


Figure 11.23: Convergence trend of the entropy production for mid-constrained twisted blade.

The distribution of the outlet flow angle in figure 11.24 having to satisfy the constraint at mid-span it not able to match the angle distribution of the unconstrained case at the boundaries of the domain.

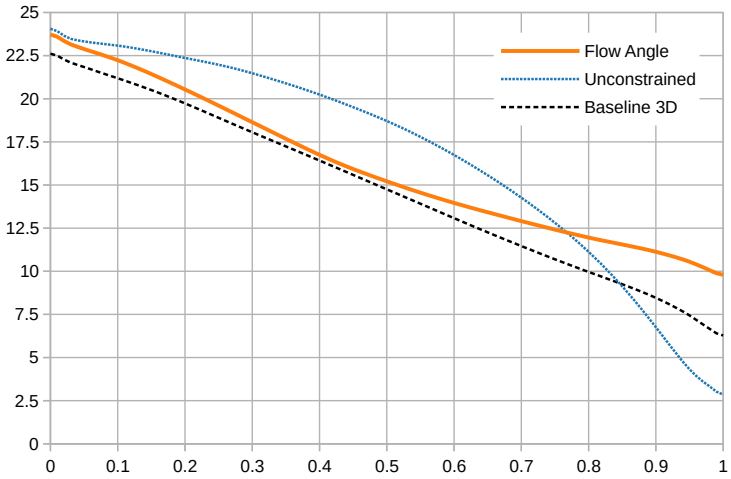


Figure 11.24: Outlet flow angle along the span for the mid-constrained optimization.

The same behavior can be highlighted also comparing figures 11.25 and 11.21, where the optimal blades are quite different, in particular in the tip section. In figure 11.26 it is also provided the physical positioning of the sections.

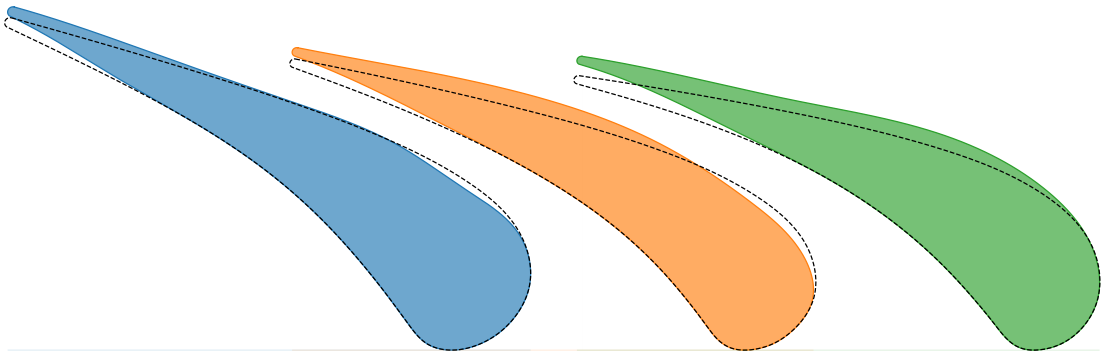


Figure 11.25: Hub, mid and tip sections of the optimal blade (Side by side).

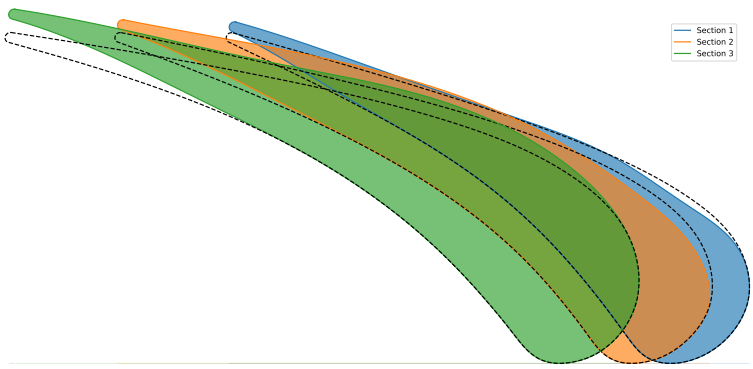


Figure 11.26: Hub, mid and tip sections of the optimal blade from a radial point of view.

8) Full-span constraint

To perform an optimization with an outlet flow angle similar to that of the twisted baseline along the whole span we have decided to bind the angle in the 3 sections where we have the control on the shape, meaning hub, mid and tip, so the constraint is formulated as in equation 11.1.

The convergence trend in figure 11.27 shows less simulations that satisfy the constraint, but the optimizer is however able to obtain a blade better that the baseline with similar flow angle.

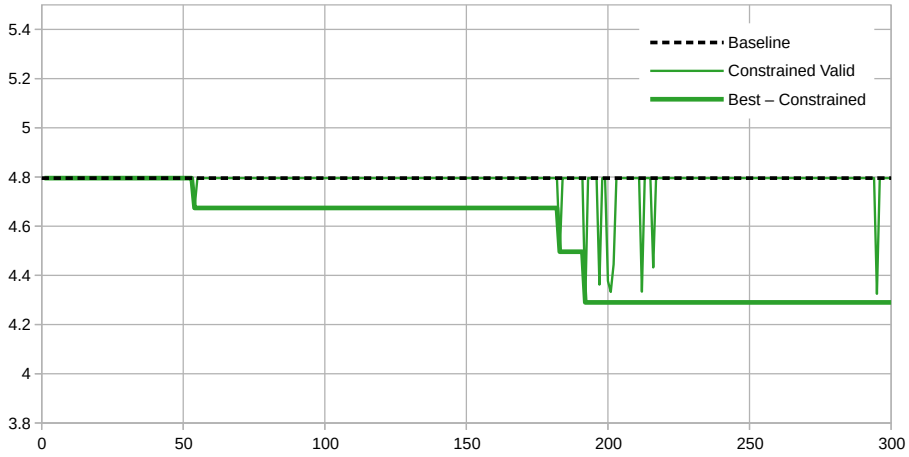


Figure 11.27: Convergence trend of the entropy for the full-constrained twisted case.

Analyzing the outlet flow angle drawn in figure 11.28 we can highlight that the three-points constraint is almost able to replicate the baseline flow angle with good accuracy providing better efficiency.

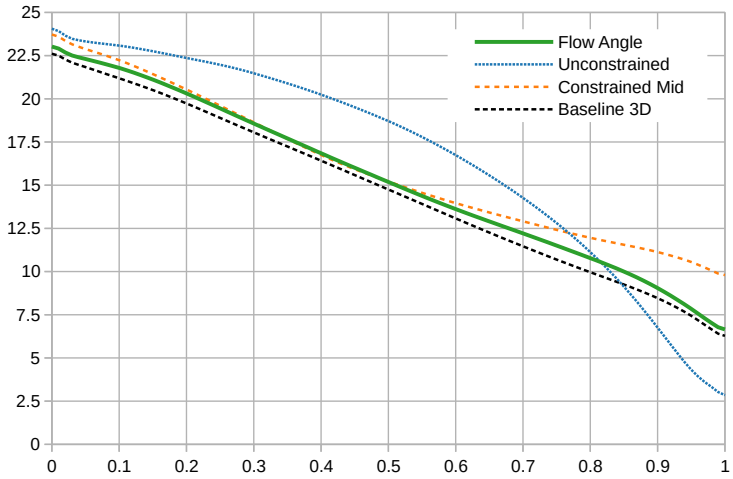


Figure 11.28: Outlet flow angle along the span for the full-constrained optimization.

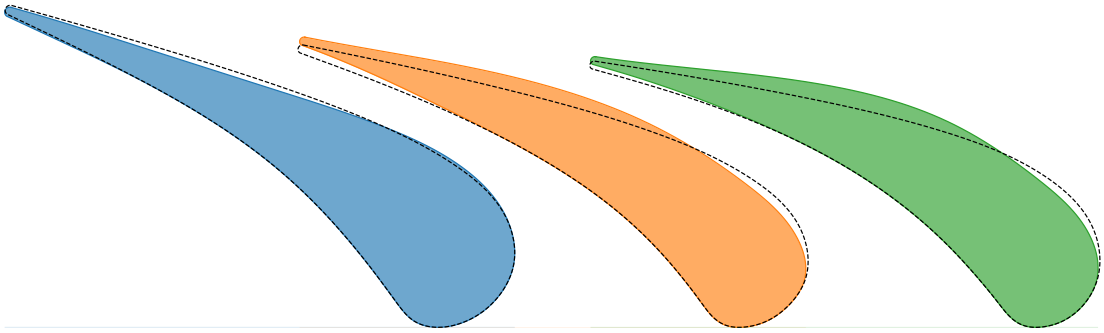


Figure 11.29: Hub, mid and tip sections of the optimal blade (Side by side).

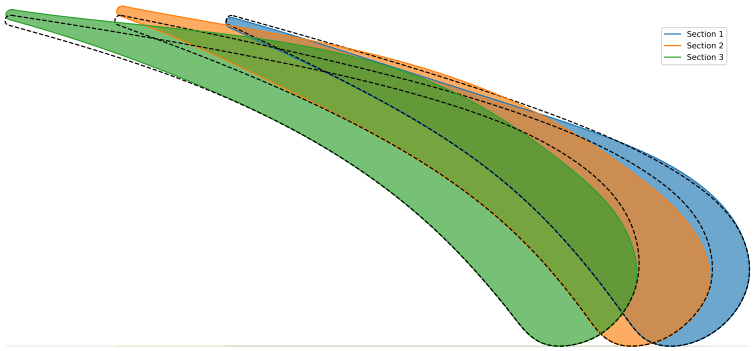


Figure 11.30: Hub, mid and tip sections of the optimal blade from a radial point of view.

As expected the mid-span profile is very similar to that in figure 11.29, while the hub and shroud sections are quite different due to the constraint applied.

11.3.1 High-fidelity validation

At the end of every set of optimizations with a low-fidelity mesh, the high-fidelity validation of the optimal solutions is required. As before, all the optimizations are performed with a blade to blade density equivalent to the 500 000 cells mesh, which is enough for that process, but the validation is performed with a mesh of blade to blade plane density equivalent to the 2 000 000 cells mesh, which corresponds to an actual number of cells of 1 085 000.

We have simulated the optimal blades of all the optimizations to check that the ranking between them is maintained also with a more accurate mesh; from figure 11.31 this is actually valid.

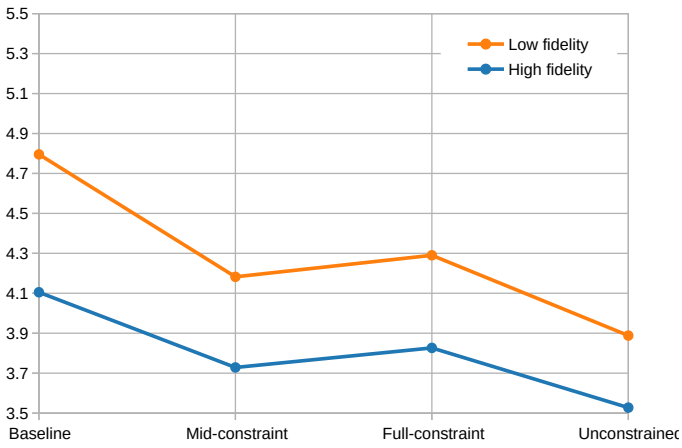


Figure 11.31: Comparison between 270 000 *and* 1 085 000 *cells meshes.*

11.4 Design of experiments - random analysis

To complete the set of free-slip optimization, we want to exploit what appends to the optimal solution with 3D simulations. We have extensively studied the optimizer behavior in chapter 10, but it was for 2D CFD cases only. A significant difference could exists given that we are comparing optimization with 6 for the 2D versus 18 design variables for the 3D. To have an idea, the number of samples required to perform a full factorial design of experiments is respectively 64 and 262144, while our strategy to set the sample ten times the number of variables leads to actual 60 and 180 samples computed, so this may be the reason why for the 2D case the seed is not able to significantly affect the results of the optimization.

For the 3D free-slip optimizations, we have decided to perform a set of 5 simulation with different seed of the design of experiments, for what that we consider the most significant case, in paragraph 8). We optimize the original blade constrained at hub, mid and tip, with a threshold of 0.5°.

The convergence trend of the optimizations is reported in figure 11.32

The first thing that it is important to notice is that all the optimizations are able to find a solution which is better than the baseline satisfying the constraint on the flow angle. However we can highlight a small but not negligible difference between the solutions, larger than the 2D case.

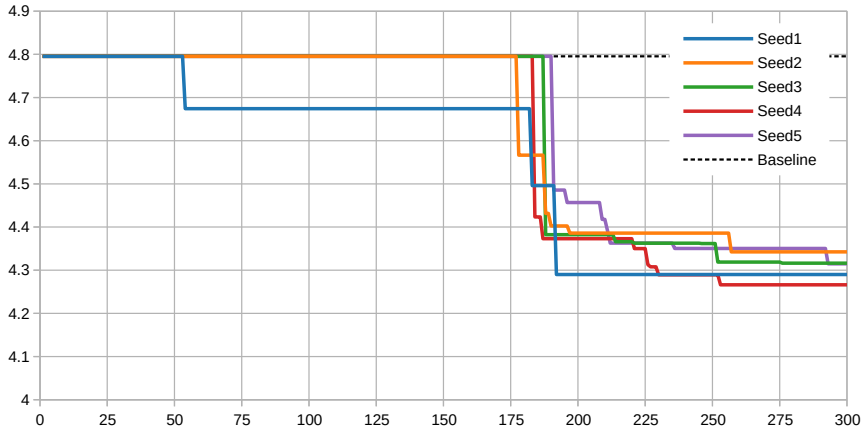
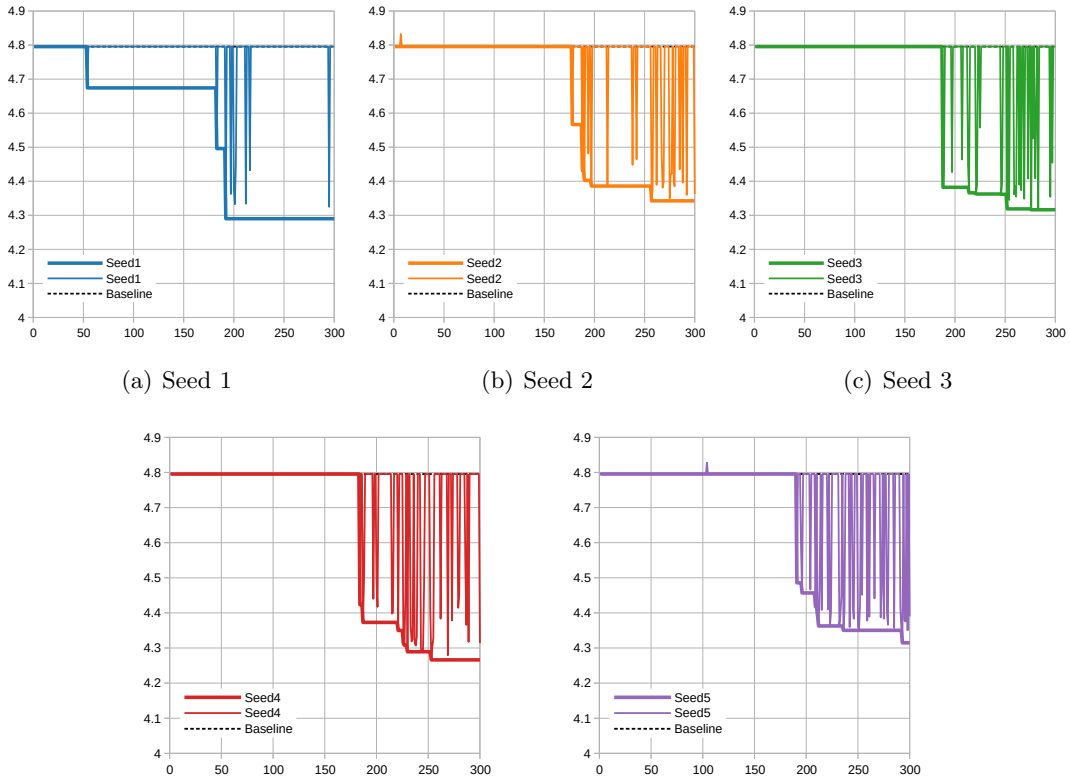


Figure 11.32: Set of 5 optimizations with different seeds (Constrained).

It is also interesting to show the convergence trends of the simulations that satisfy the constraint for each seed, represented in figure 11.33. There exists a great variability in the number of iteration that can satisfy the constraint. From just few in optimization 1, to most in optimization 5. This behavior however seems not to significantly influence the optimal entropy production in a positive way, since the one with a smaller number

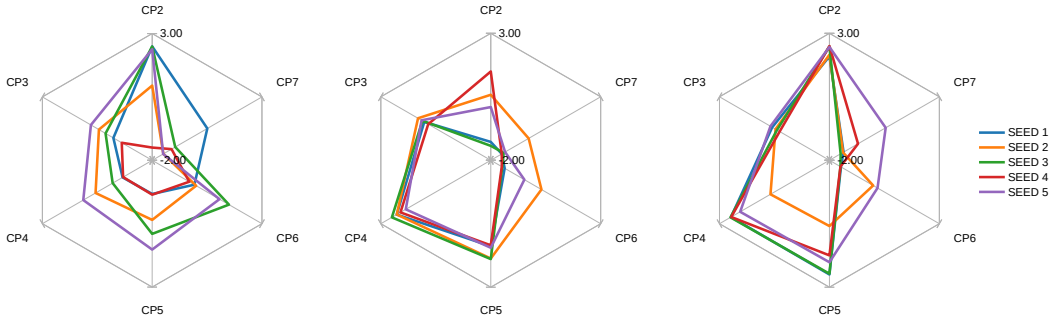


(a) Seed 1 (b) Seed 2 (c) Seed 3

(d) Seed 4 (e) Seed 5

Figure 11.33: Convergence trend for all the simulations.

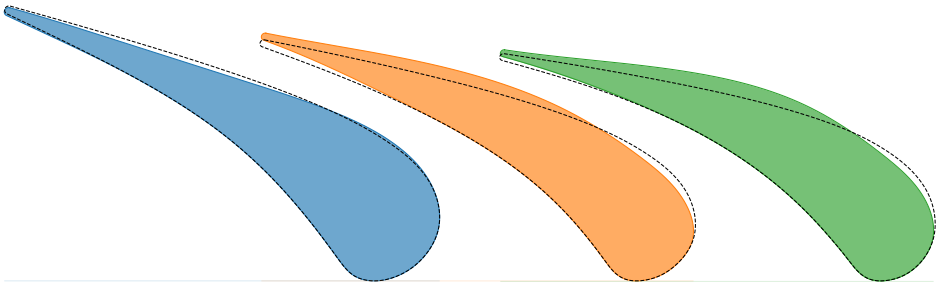
The design variables instead are quite different one from another, in particular for the hub sections (Figure 11.34). The seed 2 case is quite different from the others in terms of design variables, and this reflects to the performances of the blades.



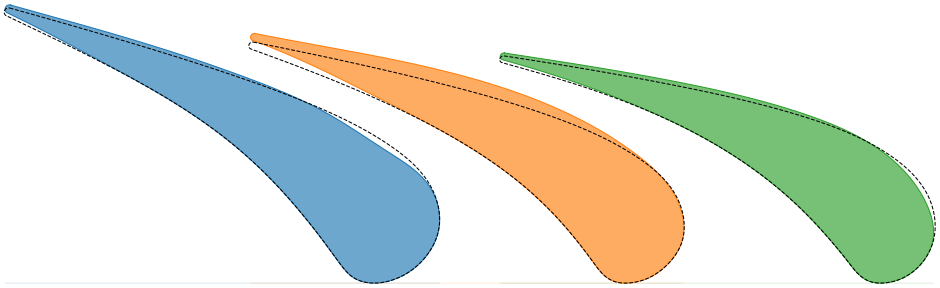
(a) Design variables - hub (b) Design variables - mid (c) Design variables - tip

Figure 11.34: Design variables for the three sections hub, mid, tip.

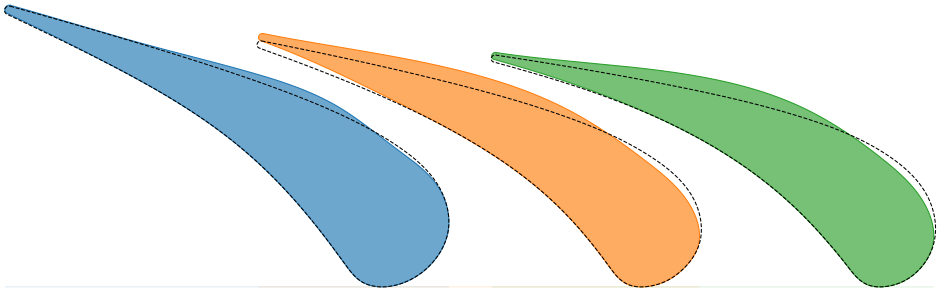
Finally the shape of the optimal blades for the set of optimizations are drawn in figure 11.34. The hub profile, as seen in the previous figures has the largest variability, which means that probably it has a smaller contribution to the entropy production respect to mid-span and tip.



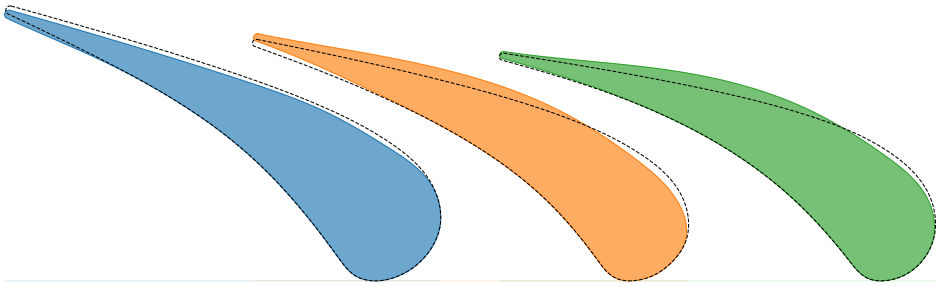
(a) Blade - seed 1



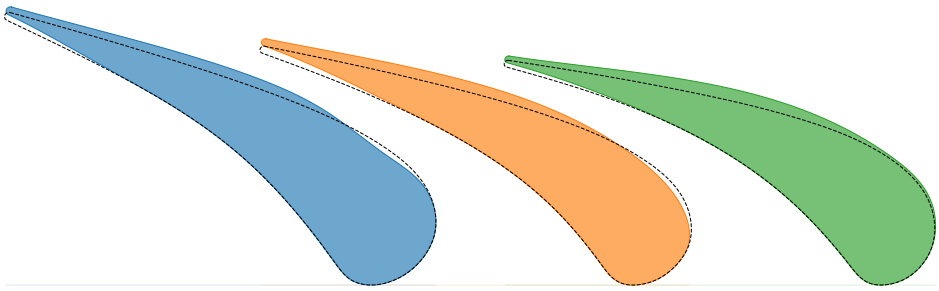
(b) Blade - seed 2



(c) Blade - seed 3



(d) Blade - seed 4



(e) Blade - seed 5

Figure 11.34: Design variables for the three sections hub, mid, tip.

11.4.1 Large DOE comparison

For sake of completeness we compare the 5 optimizations obtained changing the seed of the DOE with a single optimization in which we include all the samples of the design of experiments of the seed analysis, obtaining an initial database made by 900 samples. In this way the surrogate model should be more accurate, due to the much higher number of samples, and maybe this can improve the optimal performances. Actually this does not happen, and the results are in the average of the results with just 180 samples in the DOE.
The convergence entropy trend is drawn in figure 11.35

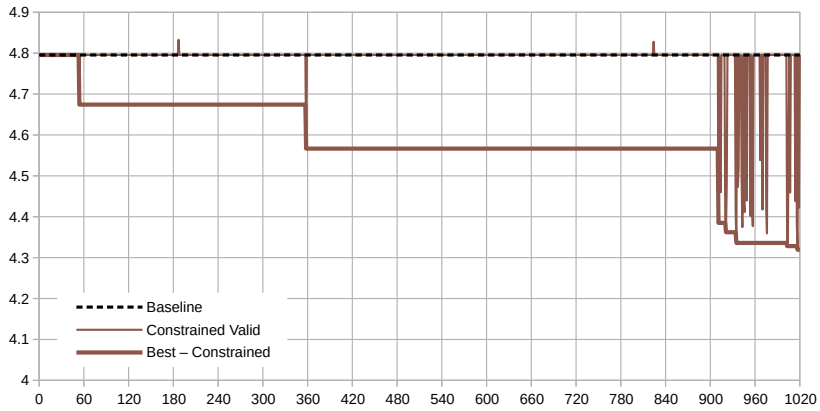


Figure 11.35: Constrained optimization with 900 samples of the DOE.

To properly position the results among that available from the seed analysis it is interesting to show all the convergence trends together, and this comparison is shown in figure 11.36. There, it is clear that having so much samples does not improve significantly the performances that the optimization is able to reach.

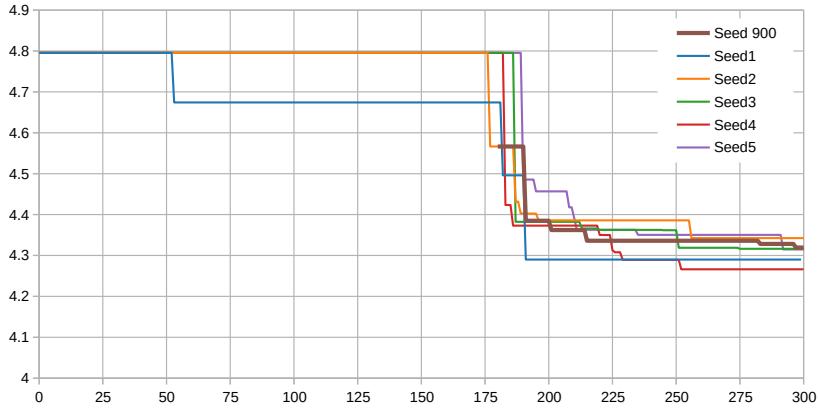


Figure 11.36: Comparison between the optimization with 900 samples and all the others.

Finally the blade shape at hub, mid and tip is represented in figures 11.37 and 11.38.

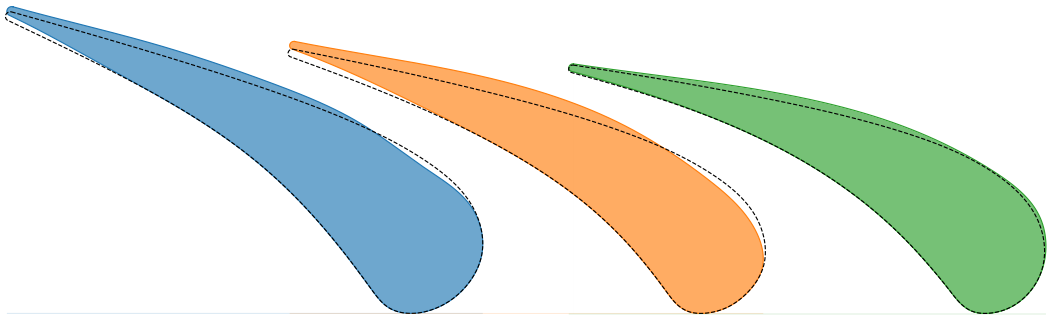


Figure 11.37: Hub, mid and tip sections of the optimal blade (Side by side).

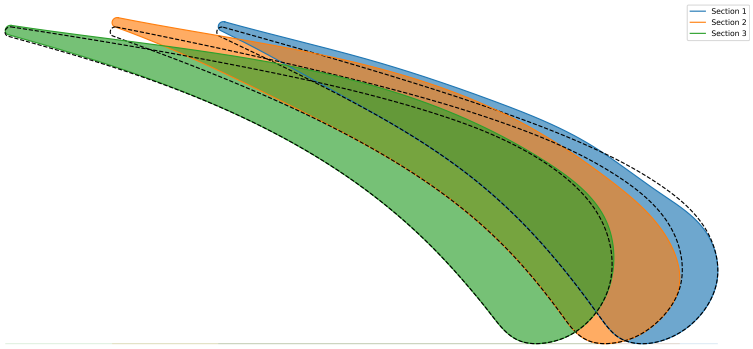


Figure 11.38: Hub, mid and tip sections of the optimal blade from a radial point of view.

The shape of this case is quite similar to seed 5 case in figure 11.34, symptom that improving the quality of the surrogate in the whole domain it is not able to guarantee a better exploration. To finally check if a uniform big exploration of the domain guarantees better results, a much expensive test case has to be setup, performing a LHS design of experiments with 900 samples and then starting from that database the optimization process. Such kind of analysis would have been much more computationally expensive then the current one, so, for this reason, we have decided to perform just that.

11.4.2 High-fidelity validation

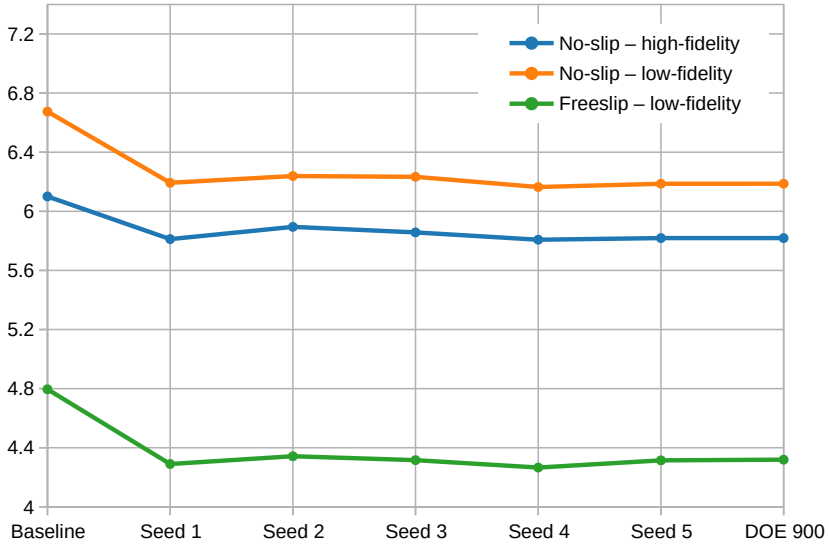


Figure 11.39: Comparison of the entropy production with a low-fidelity and an high-fidelity no-slip CFD simulations, and the free-slip one as reference.

As usual, we want to be sure that the optimal blades generated by the algorithm are really better that the baseline for the five blades of the seed analysis and the one with the larger DOE. In this case, however, during the optimization we have neglected the secondary flows, which can have a relevant effect in the entropy production of the stators.

For this reason we have decided to perform the high fidelity validation including the secondary flows, which means running the CFD simulations with the no-slip boundary condition on the hub and the shroud. In particular we have performed two different no-slip simulations, the former with 500 000 cells and the latter with 2 millions cells.

The entropy production of the set of simulations is represented in figure 11.39.

As a first view is clear that the secondary flows are not completely changing the performances of the blades, and all those bring improvements compared to the baseline; even though the ranking is not exactly kept fixed, and they are able to penalize more or less some blades respect to the others.

Chapter 12

Reference flow and real flow combination

An interesting idea is to combine together both free-slip and no-slip simulations together. Actually it is much better to refer to the reference flow for the computation of the outlet angle, as explained in chapter 11. In that chapter we have however minimized the entropy drop of the free-slip simulation, which does not include the secondary flows. But they can be responsible for losses, so would have been of interest to minimize the entropy production of the simulation including the effect of both the reference flow and the secondary losses.

An alternative procedure could be for example the minimization of the difference between the actual flow and the reference one, which should be equivalent to minimize secondary flows accordingly to Persico et Al. approach.
This kind of approach has been implemented inside our in-house code, and it let to choose for both objectives, constraints and posts at which of the two simulations to apply the CFX post-process function, and eventually even at a combination between the two(like difference, ratio or more).

The current case that we have set up consists in minimizing the entropy production of the real simulation which includes the the secondary flows while constraining the outlet reference flow angle to that of the baseline, with a threshold of $0.5°$.
For the simulation with no-slip boundary conditions on the hub and the shroud we have extended the inlet domain of 60 mm to allow the development of the boundary layers from a uniform inlet boundary condition.

In figure 12.1 the trend of the optimal entropy production is plotted.

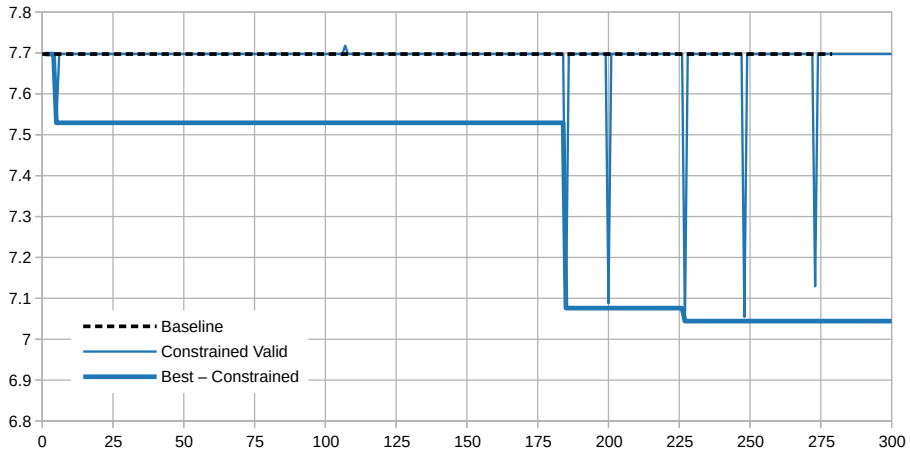


Figure 12.1: Convergence trend of entropy production the no-slip simulation.

It is also of interest to show how the flow angle changes when including the secondary flows. In figure 12.2 both the reference-flow angle and that which also includes the secondary flows are shows in comparison with the baseline blade. We can highlight that even if the baseline blade and the optimal one delivers almost the same angle when comparing the reference flow, the situation slightly changes when considering the actual angle.

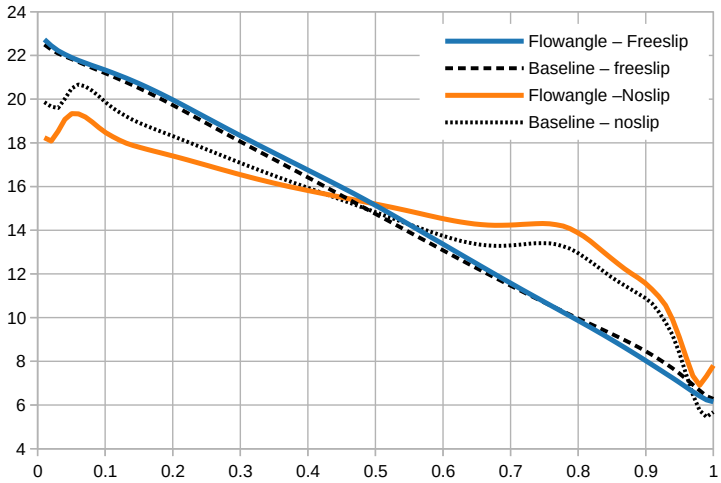


Figure 12.2: Outlet flow angle along the span for both simulations compared to the baseline.

The optimal shape of the blades in shown as common in two different ways, the former in figure 12.3 has the aim to highlight the shape at hub, mid and tip, while the latter, in figure 12.4, shows the real positioning of the blade to see the actual leaning. Comparing this solution with the equivalent optimization but where we have minimized the entropy of the free-slip simulation in figure 11.29, they seems pretty different one to each other in both hub and tip, and quite similar at mid-span. This seems to prove

that considering the secondary flows into the entropy production can have a big impact on the optimal blade shape. However this effect is much smaller when considering the seed 5 in figure 11.34; there, the three sections are much more similar to the current case, even if not exactly the same.

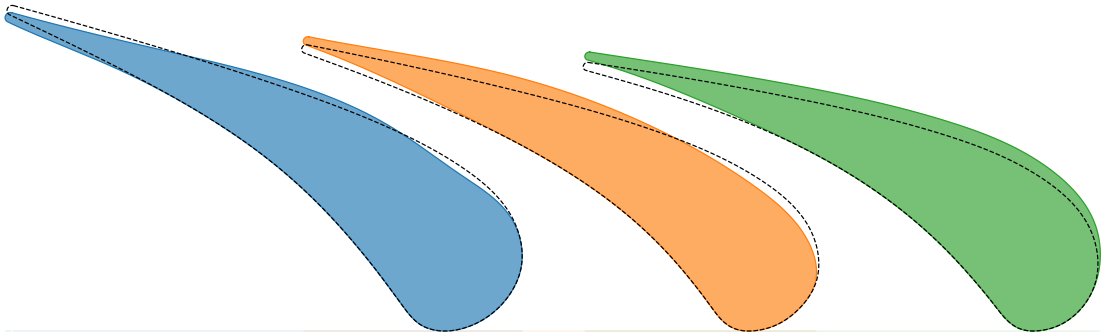


Figure 12.3: Hub, mid and tip sections of the optimal blade (Side by side).

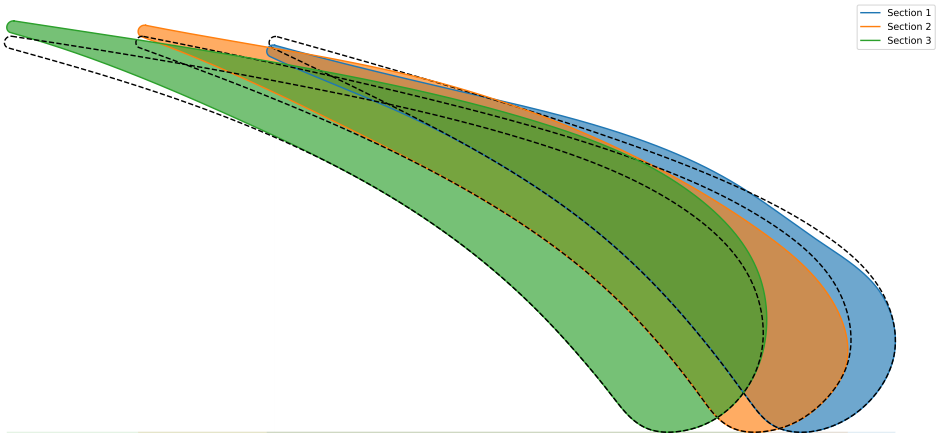


Figure 12.4: Hub, mid and tip sections of the optimal blade from a radial point of view.

From this analysis we can deduce that separating the objective and the constraints in two different CFD simulations, one free-slip and the other no-slip, of the same blade, it is able to find consistent results respect to the purely free-slip optimization case, and furthermore that with such a high number of design variable a seed analysis is suggested to be performed.

12.1 High-fidelity validation

The validation with a high fidelity mesh is shown in comparison with the baseline blade and with the results of the free-slip only optimization, and everything reported in figure 12.5.
Other than small differences, the purely free-slip optimizations are getting almost the same performances in all the conditions, meaning in free-slip mode and in both low and

high-fidelity no-slip mode. Probably running many times this optimization changing the seed would lead to similar results as the free-slip case.

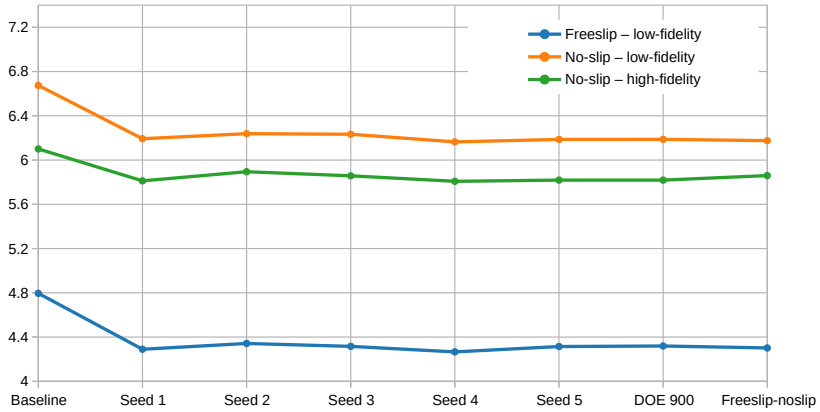


Figure 12.5: Comparison of the entropy production with a low-fidelity and an high-fidelity no-slip CFD simulations, and the free-slip one as reference.

Chapter 13

Conclusions

In this work we have performed the constrained optimization of three-dimensional transonic stator blade through a surrogate-based genetic algorithm. The first aim has been to develop a robust and flexible tool which is able to support the designer from the automatic parametrization of the baseline shape to the post-processing directly while the optimization is running.

We have gone across all the steps of the shape-optimization process, outlining the whole procedure; first at all it is necessary to prepare the shape of the baseline for the interpolation, then to adjust the parameters of the interpolation balancing accuracy and computational cost. After interpolating all the sections available, a set of simulations is required, with the aim of determining how many of that profiles have to be modified during the optimization. In this case the use of the hub, mid-span and tip sections has revealed itself good enough to represent the original blade, and in general it is suggested to reduce the number of sections as much as possible, since any additional one, increases significantly the computational time.

After the parametrization of the baseline, the number of free control points and the treatment of the trailing edge have to be decided. This choice can be rigorously done with an ANOVA analysis, that let to select only the control points that have an effect on the objective function. However it is extremely expensive, and the alternative approach is to free the points accordingly to experience. Actually in the current work, we have kept fixed all the pressure side, and part of the leading edge, since we expected that this portion of the blade should have a negligible effect.

Before running the optimization, is necessary to decide the number of cells of the mesh and the maximum number of iterations, and those are set through a mesh sensitivity analysis. Probably that number required by a three-dimensional CFD simulation in combination with the number of iterations, is too high to be manageable by the optimization, but some averaging strategies can be applied to accelerate the convergence. Furthermore the validation in high-fidelity always leads to comforting results, so the use of a lower-fidelity mesh during the optimization seems able to guarantee reliable results.

For what concern, instead, the kind of boundary conditions, the free-slip CFD simulations show consistent results with both the combination of free-slip for the angle

constraint and the no-slip one for the entropy production, and the high-fidelity validation with no-slip hub and shroud.
It is not recommended to use 5 times the number of design variables as size of the design of experiments, and it is suggested to sample at least ten times them.
From the seed analysis, for unconstrained 2D simulations, a single run is probably sufficient, and a larger number of iterations of the optimization algorithm may lead to better results. For 3D simulations (or in general when the number of design variables increases), or when the constraint is tough to satisfy, it is always suggested to perform more than one run, maybe with a lower number of iterations, and then exploit only the most promising one.

Among the many chances offered by the tool developed for this work, we have focused more on outlining the procedure for a single profile, and exploiting that profile with different analysis, respect of testing all the features of the software. For future works could be interesting to improve the parameterization in terms of alternative kind of analysis. For example after optimizing the shape of the profiles along the span, may be interesting to optimize the stacking of the sections to obtain a leaned blade, or even the combination of both together. Then a more refined mechanical analysis tool could be integrated in combination with the fluid-dynamical simulation for performing fully coupled aero-structural optimization.

Bibliography

[1] Guido Rossum. Python Reference Manual. Technical report, Amsterdam, The Netherlands, The Netherlands, 1995.

[2] G. Persico, P. Gaetani, V. Dossena, G. D'Ippolito, and C. Osnaghi. On the definition of the secondary flow in three-dimensional cascades. *Proceedings of the Institution of Mechanical Engineers, Part A: Journal of Power and Energy*, 223(6):667–676, 2009.

[3] S. Pierret, R. Filomeno Coelho, and H. Kato. Multidisciplinary and multiple operating points shape optimization of three-dimensional compressor blades. *Structural and Multidisciplinary Optimization*, 33(1):61–70, 2007.

[4] Matteo Pini Relatore, Vincenzo Dossena Coordinatore, and Carlo Bottani. *Turbomachinery Design Optimization using Adjoint Method and Accurate Equations of State*. PhD thesis, Politecnico di Milano, 2013.

[5] Nassim Razaaly, Giacomo Persico, and Pietro Marco Congedo. Uncertainty Quantification of an ORC turbine blade under a low quantile constrain. *Energy Procedia*, 129:1149–1155, 2017.

[6] Shinkyu Jeong and S. Obayashi. Efficient Global Optimization (EGO) for Multi-Objective Problem and Data Mining. *2005 IEEE Congress on Evolutionary Computation*, 3(May):2138–2145, 2014.

[7] Shinkyu Jeong, Mitsuhiro Murayama, and Kazuomi Yamamoto. Efficient Optimization Design Method Using Kriging Model. *Journal of Aircraft*, 42(2):413–420, mar 2005.

[8] René A. Van den Braembussche. *Numerical Optimization for Advanced Turbomachinery Design*, pages 147–189. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[9] James Ferguson. Multivariable Curve Interpolation. *Journal of the ACM*, 11(2):221–228, 1964.

[10] Dominic A. Masters, Nigel J. Taylor, T. Rendall, Christian B. Allen, and Daniel J. Poole. Review of Aerofoil Parameterisation Methods for Aerodynamic Shape Optimisation. *53rd AIAA Aerospace Sciences Meeting*, (January), 2015.

[11] Hilton Reno, Nevada Reno, and Brenda M Kulfan. AIAA-2007-0062 45th AIAA Aerospace Sciences Meeting and Exhibit A Universal Parametric Geometry Representation Method-"CST" A Universal Parametric Geometry Representation Method-"CST". pages 1–36, 2007.

[12] Shinkyu Jeong, Mitsuhiro Murayama, Kazuomi Yamamoto, Timothy W Simpson, Timothy M Mauery, John J Korte, Farrokh Mistree, N Alexandrov, E Nielsen, R Lewis, W Anderson, C Gumbert, L Green, P Newman, B BALDWIN, H LOMAX, Helmut Sobieczky, Raymond M Hicks, Preston A Henne, Prabhat Hajela, David Pasquale, Giacomo Persico, Stefano Rebay, G Gary Wang, S Shan, Marian Nemec, David W Zingg, Thomas H Pulliam, T D Robinson, M S Eldred, K E Willcox, R Haimes, Dirk Büche, Gianfranco Guidati, and Peter Stoll. Surrogate-Based Optimization Using Multifidelity Models with Variable Parameterization and Corrected Space Mapping. *Journal of Aircraft*, 42(7):2814–2822, jan 2000.

[13] Helmut Sobieczky. *Parametric Airfoils and Wings*, pages 71–87. Vieweg+Teubner Verlag, Wiesbaden, 1999.

[14] Gerald Farin. *Curves and Surfaces for Computer Aided Design, A Practical Guide*. 2002.

[15] E T Y Lee. Choosing nodes in parametric curve interpolation. *Computer-Aided Design*, 21(6):363–370, 1989.

[16] Nestor V Queipo, Raphael T Haftka, Wei Shyy, Tushar Goel, Rajkumar Vaidyanathan, and P Kevin Tucker. Surrogate-based analysis and optimization. *Progress in Aerospace Sciences*, 41(1):1–28, 2005.

[17] Bernard Widrow, David E Rumelhart, and Michael A Lehr. Neural Networks: Applications in Industry, Business and Science. *Commun. ACM*, 37:93–105, 1994.

[18] Alexander I.J. Forrester, Neil W. Bressloff, and Andy J. Keane. Optimization using surrogate models and partially converged computational fluid dynamics simulations. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 462(2071):2177–2204, 2006.

[19] N. Alexandrov, R. Lewis, C. Gumbert, L. Green, and P. Newman. Optimization with variable-fidelity models applied to wing design. *38th Aerospace Sciences Meeting and Exhibit*, (c), 2000.

[20] N. Alexandrov, E. Nielsen, R. Lewis, and W. Anderson. First-order model management with variable-fidelity physics applied to multi-element airfoil optimization. *8th Symposium on Multidisciplinary Analysis and Optimization*, (September), 2000.

[21] Slawomir Koziel and Leifur Leifsson. Multi-level CFD-based Airfoil Shape Optimization With Automated Low-fidelity Model Selection. *Procedia Computer Science*, 18:889–898, 2013.

[22] Prabhat Hajela. Nongradient Methods in Multidisciplinary Design Optimization-Status and Potential. *Journal of Aircraft*, 36(1):255–265, jan 1999.

[23] Sergey Peigin and Boris Epstein. Robust optimization of 2D airfoils driven by full Navier-Stokes computations. *Computers & Fluids*, 33(9):1175–1200, 2004.

[24] John Eddy and Kemper Lewis. Effective Generation of Pareto Sets Using Genetic Programming. *ASME 2001 Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, (1):1–9, 2001.

[25] T Murata and H Ishibuchi. MOGA: multi-objective genetic algorithms. *Evolutionary Computation, 1995., IEEE International Conference on*, 1:289, 1995.

[26] Joel H. Ferziger and Milovan Peric. *Computational Methods for Fluid Dynamics*. 2002.

[27] Burton Wendroff. Difference methods for initial-value problems (robert d. richtmyer and k. w. morton). *SIAM Rev.*, 10(3):381–383, July 1968.

[28] P K Kundu and I M Cohen. *Fluid Mechanics*, volume 80. 2002.

[29] A N Kolmogorov. The local structure of turbulence in incompressible viscous fluid for very large Reynolds numbers. *Proceedings of the Royal Society of London. Series A-Mathematical and Physical Sciences*, 434(1890):9–13, 1991.

[30] B BALDWIN and H LOMAX. Thin-layer approximation and algebraic model for separated turbulentflows. In *16th Aerospace Sciences Meeting*, Aerospace Sciences Meetings. American Institute of Aeronautics and Astronautics, jan 1978.

[31] Philippe Spalart and Steven Allmaras. A One-Equation Turbulence Model for Aerodynamic Flows. *AIAA*, 439, 1992.

[32] F R Menter. Two-equation eddy-viscosity turbulence models for engineering applications. *AIAA Journal*, 32(8):1598–1605, aug 1994.

[33] Brian M Adams, Keith R Dalbey, Michael S Eldred, David M Gay, Laura P Swiler, William J Bohnhoff, John P Eddy, Patricia D Hough, and Sophia Lefantzi. DAKOTA , A Multilevel Parallel Object-Oriented Framework for Design Optimization , Parameter Estimation , Uncertainty Quantification , and Sensitivity Analysis Version 5 . 1 User ' s Manual. *Analysis*, (December 2009), 2011.

[34] Pablo Rodriguez-Fernandez. Development of shape-optimization tools for the aerodynamic design of turbomachinery blades. March 2015.

[35] Giacomo Persico, Matteo Pini, V Dossena, and Paolo Gaetani. *Aerodynamic Design and Analysis of Centrifugal Turbine Cascades*, volume 6. jun 2013.

[36] Giacomo Persico, Matteo Pini, Vincenzo Dossena, and Paolo Gaetani. Aerodynamics of Centrifugal Turbine Cascades. *Journal of Engineering for Gas Turbines and Power*, 137(11):112602–112611, nov 2015.

[37] J. D. Denton and L Xu. The exploitation of three-dimensional flow in turbomachinery design. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 213(2):125–137, 1998.

[38] Gabriele D'Ippolito, Vincenzo Dossena, and Alessandro Mora. The Influence of Blade Lean on Straight and Annular Turbine Cascade Flow Field. *Volume 6: Turbomachinery, Parts A, B, and C*, 133(January 2011):1379–1389, 2008.

Appendix A

Radial Equilibrium

The streamlines in an annular duct cannot be counted as parallel to axis, due to the geometry of the meridional flow path and the strong bending caused by it. Thus it is important to introduce the definition of Radial Equilibrium that is the governing law for the flow behavior in annular turbomachinery channels.

According to radial equilibrium, when a flow turns a pressure gradient is necessary to balance centrifugal forces.

We obtain the formulation, starting from the force balance on an infinitesimal fluid element sketched in figure A.1.

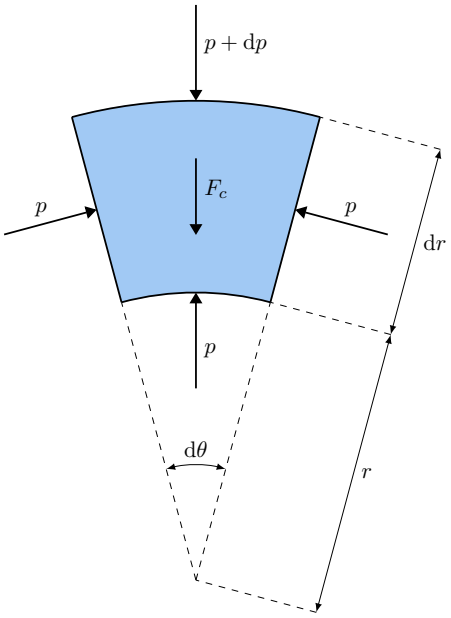


Figure A.1: Infinitesimal fluid volume.

The equilibrium in the radial direction can be written in a generic way applying the Dalambert principle as:

$$\sum \mathbf{F}_r - m\mathbf{a}_c = 0 \tag{A.1}$$

Replacing the summation with all the respective terms, and the centripetal acceleration as function of the tangential velocity and the radius, we obtain:

$$pr\,\mathrm{d}\theta\,\mathrm{d}z - (p+\mathrm{d}p)\,(r+\mathrm{d}r)\,\mathrm{d}\theta\,\mathrm{d}z + 2p\sin\frac{\mathrm{d}\theta}{2}\,\mathrm{d}r\,\mathrm{d}z + \rho r\,\mathrm{d}r\,\mathrm{d}\theta\,\mathrm{d}z\,\frac{v_t^2}{r} = 0 \tag{A.2}$$

Equation A.2 can be finally simplified neglecting second order infinitesimal terms, and replacing the sin function with its first order Taylor expansion, and the result is reported in equation A.3.

$$\cancel{r}\,\mathrm{d}p\,\cancel{\mathrm{d}\theta}\,\cancel{\mathrm{d}z} = \rho\cancel{r}\,\cancel{\mathrm{d}\theta}\,\cancel{\mathrm{d}z}\,\mathrm{d}r\,\frac{v_t^2}{r} \qquad \Rightarrow \qquad \frac{1}{\rho}\frac{\mathrm{d}p}{\mathrm{d}r} = \frac{v_t^2}{r} \tag{A.3}$$

The interpretation of equation A.3 is that any time we have a tangential velocity component a pressure gradient arise to balance the centripetal forces.
This aspect has to be taken into consideration when designing the blade evolution along the span since in turbomachinery (axial) the work is exchanged through the variation of the tangential velocity according to the Euler principle [1].

For an iso-entalphic and isoentropic flow, $\mathrm{d}h_T$ and $\mathrm{d}s$ are equal to zero, so we can write:

$$\begin{cases} \mathrm{d}h_T &= \mathrm{d}h + \mathrm{d}\left(\dfrac{v^2}{2}\right) = 0 \\ \mathrm{T}\,\mathrm{d}s &= \mathrm{d}h - \dfrac{\mathrm{d}p}{\rho} = 0 \end{cases} \qquad \Rightarrow \qquad \frac{\mathrm{d}p}{\rho} = \mathrm{d}\left(\frac{v^2}{2}\right) \tag{A.4}$$

Now differentiating equation A.4 respect to the radius r and substituting it inside equation A.3 we obtain an equation only function of the velocity, which is written in equation A.5.

$$\frac{1}{\rho}\frac{\mathrm{d}p}{\mathrm{d}r} = -\frac{\mathrm{d}}{\mathrm{d}r}\left(\frac{v^2}{2}\right) = -v\frac{\mathrm{d}v}{\mathrm{d}r} = \frac{v_t^2}{r} \tag{A.5}$$

Alternatively, remembering the relationship between the modulus of the velocity and its components $v^2 = v_a^2 + v_t^2$ we obtain the most common expression for the radial equilibrium in turbomachinery.

$$\frac{\mathrm{d}}{\mathrm{d}r}\left(\frac{v^2}{2}\right) = -\frac{\mathrm{d}}{\mathrm{d}r}\left(\frac{v_a^2+v_t^2}{2}\right) = -v_a\frac{\mathrm{d}v_a}{\mathrm{d}r} - v_t\frac{\mathrm{d}v_t}{\mathrm{d}r} = \frac{v_t^2}{r} \tag{A.6}$$

$$v_a\frac{\mathrm{d}v_a}{\mathrm{d}r} + \frac{v_t}{r}\left(r\frac{\mathrm{d}v_t}{\mathrm{d}r} + v_t\right) = 0 \qquad \Rightarrow \qquad v_a\frac{\mathrm{d}v_a}{\mathrm{d}r} + \frac{v_t}{r}\frac{\mathrm{d}\,(rv_t)}{\mathrm{d}r} = 0 \tag{A.7}$$

To solve Equation (A.3), either one of the two unknown velocity components v_a and v_t or a relation between them is required, which leads to two different approaches that can be applied: *direct problem*, where the whole geometry is known, and the pressure distribution is evaluated solving the radial equilibrium equation, or *indirect problem*,

[1] $l_{\mathrm{Euler}} = \overline{\overline{u_2 v_{2_t}}} - \overline{\overline{u_1 v_{1_t}}}$

where one of the fluid dynamics variable is assigned and the radial equilibrium equation allows the solution of the other variables and the geometry.

With it, different blade span design strategies exist. The most common are:

- Constant angle which considers constant the angle along the span, so the ratio between axial and tangential components of the velocity;

$$\frac{v_t}{v_a} = \tan\alpha = \text{const} \qquad \Rightarrow \qquad \frac{v_a}{v_a^{\text{REF}}} = \frac{v_t}{v_t^{\text{REF}}} = \left(\frac{r_{\text{REF}}}{r}\right)^{\sin^2\alpha} \tag{A.8}$$

- Free vortex design which considers constant the work along the span assuming $rv_t = \text{const}$ along the radius; the blade loading χ decreases with the radius.

- General whirl distribution defines the tangential components of the velocity at inlet and outlet of the blade as:

$$v_t = Ar^n \pm \frac{B}{r} \tag{A.9}$$

Appendix B

Structural constraints

The design of a turbomachinery is a multidisciplinary process. Actually the concern is mainly on the fluidynamic aspect, but the structural becomes more and more important when the process of optimizing the performance leads to the increase of the rotational speed and to the decrease of the blade thickness.
To properly ensure mechanical reliability a proper finite element analysis is probably required, but a coupling between the finite element method (FEM) solver and the CFD simulation would have been required. This kind of considerations are out of the scope of this work, but additional simplified consideration can be done.
Actually the optimization process starts from a baseline blade shape, which has usually been mechanically validated, and stating from that we can ensure that the optimal blade is similar in terms of mechanical properties to the reference blade.

In the solution of the structural equations for a 2D case, three are the geometrical quantities that influences the strain and the stresses onto the blade and are:

- the area A;
- the minimum principal moment of inertia $\mathrm{I}_{\min}$;
- the maximum principal moment of inertia $\mathrm{I}_{\max}$;

The stresses along the direction orthogonal to the section can be computed with equation B.1 for a frame of reference $(x',\ y')$ centered in the centroid of the profile and aligned with x and y axis.

$$\sigma_{zz} = \begin{bmatrix} 1 & x' & y' \end{bmatrix} \cdot \begin{bmatrix} A & 0 & 0 \\ 0 & \mathrm{I}_{y'y'} & \mathrm{I}'_{xy} \\ 0 & \mathrm{I}_{x'y'} & \mathrm{I}_{x'x'} \end{bmatrix}^{-1} \cdot \begin{bmatrix} N_z \\ -M_{y'} \\ M_{x'} \end{bmatrix} \tag{B.1}$$

In the case of an aerodynamic machine, the momentum are due to the dynamic action of the fluid to exchange momentum and work. In particular for the stators the normal force N_z is actually null while for the rotor it is one of the most critical contribution, the centrifugal force.

So it is required to perform the calculation of these quantities directly in the Python tool to pass them back to Dakota for the definition of the constraints.

Area

Starting from the 2D profile we will compute its area assuming that is it a polygon made of a lot of sides. The alternative was to integrate directly the analytical expression of the B-spline, but given that we are taking 1100 points over the curve the modeling effort for the calculation of that complex integral is probably over-killed respect to the good accuracy that we can reach approximating it as a polygon of 1100 points.
The area of a generic 2D shape can be computed as:

$$A = \iint_R \mathrm{d}x\,\mathrm{d}y \quad \overset{\text{if } y=f(x)}{=} \quad \int_a^b f(x)\,\mathrm{d}x \tag{B.2}$$

The area of a polygon defined by n points $\mathbf{P}_i = (x_i,\ y_i)$ can be computed as:

$$A = \sum_{i=0}^{n-1} A_i = \frac{1}{2}\sum_{i=0}^{n-1} (x_i\, y_{i+1} - x_{i+1}\, y_i) \tag{B.3}$$

Centroid

The calculation of the centroid is required since the inertia tensor is usually defined respect to it. It is defined as:

$$\mathbf{G} = \frac{\iint_R \mathbf{x}\,\mathrm{d}x\,\mathrm{d}y}{A} \tag{B.4}$$

In equation B.5 is reported the calculation for a polygon.

$$\mathbf{G} = \frac{1}{3A}\sum_{i=0}^{n-1} (\mathbf{P}_i + \mathbf{P}_{i+1}) \cdot A_i \tag{B.5}$$

Second moment of area

The moment of inertia and the moment of area are related by the density; for constant density $\mathrm{J} = \rho\,\mathrm{I}$.
The area tensor of a 2D section respect to the frame $(x,\ y)$ is defined as:

$$\begin{bmatrix} \mathrm{I}_{xx} & \mathrm{I}_{xy} \\ \mathrm{I}_{yx} & \mathrm{I}_{yy} \end{bmatrix} = \iint_R \begin{bmatrix} x^2 & xy \\ yx & y^2 \end{bmatrix} \mathrm{d}x\,\mathrm{d}y \tag{B.6}$$

However it is convenient to compute the tensor respect to the centroid of the blade, so first at all we perform the translation of all the points in the frame of reference where the centroid is the origin.

$$\mathbf{P}'_i = (x'_i,\ y'_i) = \mathbf{P}_i - \mathbf{G} \tag{B.7}$$

The inertia (or area) tensor for a 2D problem is a 2X2 symmetric matrix which has 3 distinct components: $\mathrm{I}_{x'x'}$, $\mathrm{I}_{y'y'}$ and the cross term $\mathrm{I}_{x'y'}$ which is equal to $\mathrm{I}_{y'x'}$.

$$\boldsymbol{I} = \begin{bmatrix} \mathrm{I}_{x'x'} & \mathrm{I}_{x'y'} \\ \mathrm{I}_{y'x'} & \mathrm{I}_{y'y'} \end{bmatrix} \tag{B.8}$$

Then the expression for the three components for a polygon are:

$$\mathrm{I}_{xx} = \frac{1}{6} \sum_{i=0}^{n-1} \left(y_i^2 + y_i \, y_{i+1} + y_{i+1}^2 \right) \cdot A_i \tag{B.9}$$

$$\mathrm{I}_{yy} = \frac{1}{6} \sum_{i=0}^{n-1} \left(x_i^2 + x_i \, x_{i+1} + x_{i+1}^2 \right) \cdot A_i \tag{B.10}$$

$$\mathrm{I}_{xy} = \frac{1}{12} \sum_{i=0}^{n-1} \left(x_i \, y_{i+1} + 2x_i \, y_i + 2x_{i+1} \, y_{i+1} + x_{i+1} \, y_i \right) \cdot A_i \tag{B.11}$$

In equation B.11 it is important to specify that for sake of readability we have omitted the $'$ symbol, but all the x and y coordinate are actually x' and y' in the meaning that they are referred to the centroid **G**.

Principal moment of area

In equations B.11 we have expressed the moment of area assuming a frame of reference oriented with x and y axis centred in the centroid of the profile. It is however most important to compute the principal moment of inertia (or area), and the correspondent frame of reference respect to which the cross term disappears. The most important property of them is that, for a given 2D shape, they are independent on the frame of reference.

From the mathematical point of view the principal moment of area are the eigenvalues of the tensor B.8. We can compute them as:

$$\lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} \mathrm{I}_{xx} & \mathrm{I}_{xy} \\ \mathrm{I}_{yx} & \mathrm{I}_{yy} \end{bmatrix} = 0 \qquad \Rightarrow \qquad \begin{bmatrix} \lambda - \mathrm{I}_{xx} & \mathrm{I}_{xy} \\ \mathrm{I}_{yx} & \lambda - \mathrm{I}_{yy} \end{bmatrix} = 0 \tag{B.12}$$

Then we set to 0 the determinant of the matrix:

$$\begin{vmatrix} \lambda - \mathrm{I}_{xx} & \mathrm{I}_{xy} \\ \mathrm{I}_{yx} & \lambda - \mathrm{I}_{yy} \end{vmatrix} = (\lambda - \mathrm{I}_{xx}) (\lambda - \mathrm{I}_{yy}) - \mathrm{I}_{xy}^2 = 0 \tag{B.13}$$

Finally we solve the eigenvalues quadratic equation obtaining λ_1 and λ_2.

$$\lambda_{1,2} = \frac{-(\mathrm{I}_{xx} + \mathrm{I}_{yy}) \pm \sqrt{(\mathrm{I}_{xx} + \mathrm{I}_{yy})^2 - 4 \left(\mathrm{I}_{xx} \mathrm{I}_{yy} - \mathrm{I}_{xy}^2 \right)}}{2} \tag{B.14}$$