POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Shape Optimization of Airfoils by Machine Learning-Based Surrogate Models

Laurea Magistrale in Aeronautical Engineering - Ingegneria Aeronautica

Author: Marco Zanichelli

Advisor: Prof. Luca Dede'

Academic year: 2020-2021

1. Introduction

Shape optimization is a crucial task in Aeronautical Industry for the reduction of costs and emissions. Performing an aerodynamic optimization requires multiple calls to high-fidelity CFD software, in some contexts the computational burden can be unsustainable. It is important to reduce at most this cost, choosing the procedure that allows maximizing the results in these terms.

We perform airfoil shape optimization by integrating Fluid Mechanics with Machine Learning (ML) algorithms. The main goals of this work are represented by the generation of an appropriate data-set and the construction of a surrogate model linking design parameters to aerodynamic force coefficients. This model, based on Artificial Neural Networks (ANNs), is then used to perform the airfoil optimization in two different ways: with a Standard Surrogate Model Optimization, in which the optimization is performed on a single ANNs-based surrogate model, and with an Iterative Surrogate Model Optimization, in which the optimization is performed on a sequence of surrogate models whose design space is centred on the optimization point of the previous iteration. An example of ANN is shown in Figure 1.

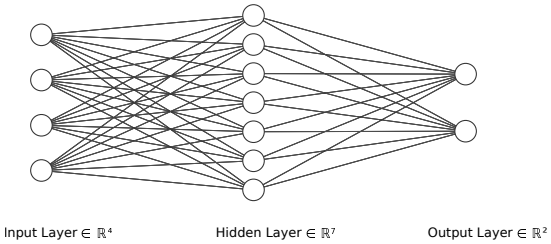


Figure 1: Example of Feed Forward ANN

2. Generation of the Data-Set

The construction of an ANN-based surrogate model requires the generation of an appropriate data-set. This set is made by input-output pairs, where the inputs are the values of some geometrical parameters that come from the shape deformation of the original airfoil, and the outputs are the aerodynamic force coefficients obtained via CFD simulation.

2.1. Numerical Simulation

The first task to be performed is to obtain an accurate CFD solution of the reference condition. This requires the generation of an appropriate mesh and the choice of suitable numerical methods for space and time discretization. In this work, the reference solution is represented by the transonic flow around a RAE2822 (air-

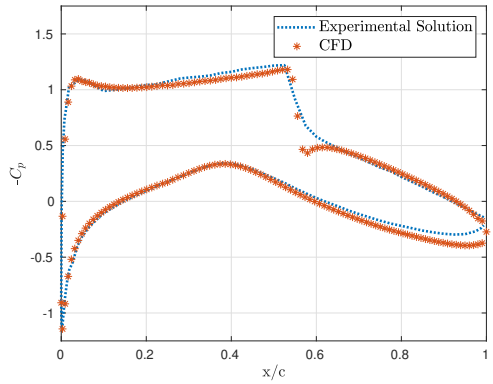
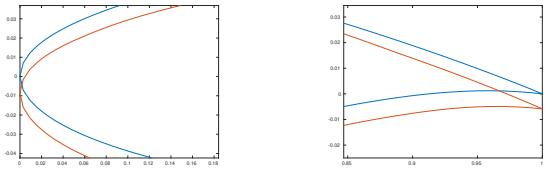


Figure 2: Pressure coefficient - Comparison with experimental solution

foil), at the following conditions: $M_\infty = 0.729$; $T_\infty = 288.15$ K; $\alpha = 2.31$ deg; $Re = 6.500.000$. Where M_∞ is the freestream Mach number, T_∞ is the freestream temperature, α is the angle of attack of the airfoil, Re is the Reynolds number $Re = \frac{\rho U_\infty c}{\mu}$, computed with the density ρ, freestream velocity U_∞, chord c, and dynamic viscosity μ. These conditions have been experimentally replicated by Cook et al. in [1]. The flow is transonic and it presents a strong shockwave on the suction side of the airfoil.
This experiment has been numerically replicated by means of the Finite Volume Method, which is implemented in the solver *rhoCentralFOAM*, whithin the open-source software *OpenFOAM*. To reduce the computational burden associated with the CFD simulation and given the high Reynolds number, the flow has been considered inviscid. In this way, the numerical solution approximates the solution of the Euler Equations. The numerical simulation has been run using a C-type grid, second order numerical methods with flux limiters for the space dicretization and Crank-Nicholson scheme for the time discretization. This allowed us to obtain a good convergence in terms of pressure coefficient behaviour along the chord, illustrated in Figure 2, with a 2.95% error on the lift coefficient and a 11.8% error on the drag coefficient. The Mach number flowfield is displayed in Figure 5(a). With these settings, the numerical solution requires 17 minutes of computation on a 6-core laptop.

2.2. Shape Parametrization and Mesh Deformation

The airfoil deformation has been performed in two ways, firstly with Radial Basis Functions-based Free Form Deformation, then with Hick-Henne functions. In both cases the mesh of the reference simulation has been deformed with Radial Basis Functions.



(a) Leading edge (b) Trailing edge

Figure 3: Deformation issues

2.2.1 Radial Basis Functions

Radial Basis Functions (RBFs) are commonly used for the interpolation of scattered data across many fields, from image manipulation to mesh deformation. In this work, RBFs are used to map a deformation vector of structural nodes on the airfoil surface to each point of the structured grid. RBFs Φ are real-valued functions whose value at the point $\mathbf{x}$ depends only on the distance from a certain control point $\mathbf{x_i^c}$, hence $\Phi(\mathbf{x}, \mathbf{x_i^c}) = \Phi(||\mathbf{x} - \mathbf{x_i^c}||)$.
The interpolation function $\mathbf{s}$ describing the displacement in the whole domain can be approximated by a sum of basis functions:

$$\mathbf{s}(\mathbf{x}) = \sum_{i=1}^{N} \gamma_i \Phi(\|\mathbf{x} - \mathbf{x_i^c}\|) + \mathbf{h}(\mathbf{x}) \qquad (1)$$

where N is the number of control points, each one associated with a coefficient γ_i, Φ is a Radial Basis Function and $\mathbf{h}(\mathbf{x})$ is a linear polynomial, with $\mathbf{x} = (x,\ y)^T$. The coefficients are recovered imposing interpolation conditions, from which a linear system is created. With the known value of the displacement on the control points it is hence possible to recover the displacement field $\mathbf{s}(\mathbf{x})$. Evaluating this field on each mesh point allows to find the final position of that point, associated with that displacement of the control points.
The deformation algorithm has been generated starting from the work of R. Lapuh [2], to which a damping function has been added as in Equation (2). By choosing appropriate radii R_1 and R_2, it is possible to obtain a smooth mesh

deformation that leaves the outer boundary elements untouched.

$$f_{\text{damp}}(r) = \begin{cases} 1 & r \leqslant R_1 \\ 1 - \frac{r-R_1}{R_2-R_1} & R_1 < r \leqslant R_2 \\ 0 & r > R_2 \end{cases} \quad (2)$$

This mesh deformation strategy has been used in two ways: i) by choosing a low number of control points and imposing their translation it is possible to obtain at the same time the deformation of the airfoil and of the mesh. This is possible because the points of the airfoil surface are treated in the same way of the other mesh points. Their displacement is hence derived from the motion of the control points. This procedure requires the tuning of a smoothing parameter, σ; ii) by choosing as control points all the airfoil boundary points, whose displacement is obtained by means of Hicks-Henne functions.
The computational power available limited the number of the control points, since an increase of a single degree of freedom required much more samples to be simulated to generate an accurate surrogate model. The maximum number of control points used has been set to eight. With such a low number of design parameters, the deformed airfoils generated with the procedure (i) showed some problems at the leading and trailing edges, as shown in figure. To obtain good results in the shape optimization, the generated shapes should show sufficient regularity. However, with such a low number of control points it was not possible to satisfy this requirement without obtaining an unwanted change in the angle of attack, as shown in Figure 3, where it can be seen that both the leading and trailing edges are shifted in the deformed configuration. For this reason it has been decided to perform the airfoil deformation with another parametrization, following the procedure (ii).

2.2.2 Hicks-Henne Functions

Hicks-Henne functions have been introduced in the context of airfoil optimization. The Hicks-Henne deformation method consists in adding a linear combination of n augmented sine functions to the original coordinates of the airfoil, these functions are in the form of a bump.

$$y_{\text{mod}} = y_0 + \sum_{i=0}^{n} a_i \sin^{w_i}(\pi x^{\ln(0.5)/\ln(x_i^M)}) \quad (3)$$

Where y_0 are the initial y-coordinates of the upper and lower surface points, y_{mod} are the final y-coordinates, n is the number of bumps for each one of the upper or lower surfaces, x_i^M is the x coordinate of the bump, w_i is the bump width, while a_i are the bumps intensities. In this work, the position and width of each bump has been fixed, and the design parameters are then a_i, for $i = 1, ..., n$. These functions allowed us to obtain a smooth deformation and at the same time a fixed angle of attack.
As mentioned above, also in this case the mesh deformation has been performed with RBFs, by using as control points all the points on the airfoil surface y_0, imposing them the deformation $g = y_{mod} - y_0$.

3. Surrogate Models

Surrogate models have been introduced in the context of shape optimization to capture the most important features of a high fidelity model at a low computational cost. The surrogates are constructed using data drawn from high-fidelity models, and provide fast approximations of the objectives and constraints at new design points. The surrogate models used in this work are based on Feed Forward Artificial Neural Networks, which have been implemented by F. Regazzoni in the *model-learning* library, explained by Regazzoni et al. in [4]. ANNs have been chosen for this work thanks to the *Universal Approximation Theorem*, which states that ANNs with a single hidden layer can approximate with arbitrarily small error any continuous function on a compact set, provided that a sufficient number of hidden neurons are employed. The algorithms present in *model-learning* library are thought for a data-driven Model Order Reduction of time-dependent problems. For computational limitations, the problem here considered is steady and hence the Machine Learning algorithms were modified to work on a steady problem. In particular, the input-output pairs were given as constant in time and the Loss function that the ML algorithm minimizes was redefined so that it was computed only at the latest time-step. The ANNs-based surrogate model thus obtained was then evaluated only at that time during the shape optimization procedures.

-	ε_{Train} [%]	ε_{Test} [%]	ε_{CV} [%]
C_l	0.0310	0.0873	0.0879
C_d	1.113	1.392	1.418

Table 1: Learning errors - 16 neurons

3.1. Results

All the optimizations have been performed using a Hick-Henne parameterization with eight bumps, five of which on the suction side of the airfoil. Figure 4 shows the design space of this test. Using the Sobol sequence to obtain the values of the bumps, a sufficient number of airfoil samples has been generated and then simulated with the CFD software. This sequence has shown to help in reducing the number of samples needed to obtain an accurate ANNs-based surrogate model [3]. With this parameterization, 500 airfoil samples were generated. The numerical values of the bumps constitute the inputs of the network, and the aerodynamic coefficients represent the output. The data-set has been divided in three sets: train set (70% of the samples), test and cross-validation sets (15% of the samples each). The training set is directly used in the training phase to evaluate the loss function and to modify the values of the neurons activations thanks to the back-propagation procedure. The test set is used, during the learning phase, to evaluate the network's accuracy on unseen data, allowing to stop the learning as soon as the error on this set starts increasing, to avoid overfitting. The cross-validation set is instead used, once the best-performing network is chosen, to evaluate the generalization error, i.e. the error that the network does on unseen data. In the learning phase, different neural networks were trained, which differ in the number of hidden neurons. The best performance in terms of error on the training set was found for the 16-neurons network, whose performances have been summarized in Table 1, where ε_{Train} is the Root Mean Squared Error (RMSE) on the train set, ε_{Test} is the RMSE on the test set, ε_{CV} is the RMSE on the cross-validation set.

4. Optimization

4.1. Objective Function

The first step associated with the aerodynamic optimization is the definition and the minimization of an objective function. In this work two different objective functions have been considered, associated with two different goals: the increase of the aerodynamic efficiency (lift-to-drag ratio), as in Equation (4) and the reduction of drag at fixed lift, as in Equation (5), where P is a penalization factor, max is the function taking the maximum value between the two inside the brackets, so that when the lift coefficient is lower than the reference the value of the objective increases.

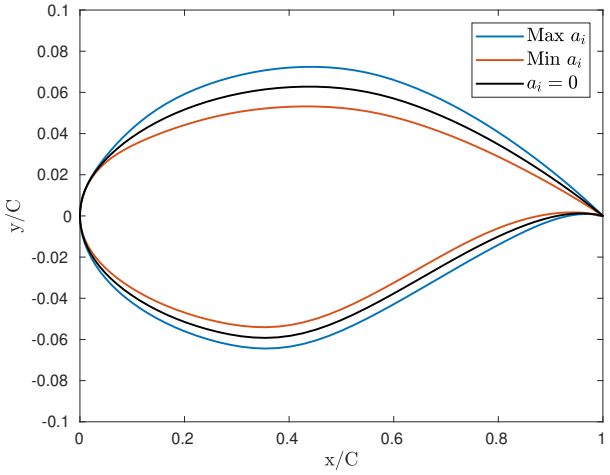


Figure 4: Design space

$$\mathcal{O}_1(E(C_l, C_d)) = \frac{E^{ref}}{E}, \tag{4}$$

$$\mathcal{O}_2(C_l, C_d) = \frac{C_d}{C_d^{ref}} + \mathcal{P}max(0, 0.999 - \frac{C_l}{C_l^{ref}}). \tag{5}$$

The optimization was then performed by means of the interior-point method, implemented in the Matlab *fmincon* function. Since the algorithm requires an objective function whose derivatives are continuous, and the function (5) has a discontinuous derivative along the line $C_l = C_l^{ref}$, the objective has been modified to go along with *fmincon*. To obtain a smooth tendency, the penalization term has been defined using an exponential function of the ratio between the reference and the actual lift coefficient.

4.2. Standard Surrogate Model Optimization

The Standard Surrogate Model Optimization (SSMO) procedure consists in finding a unique

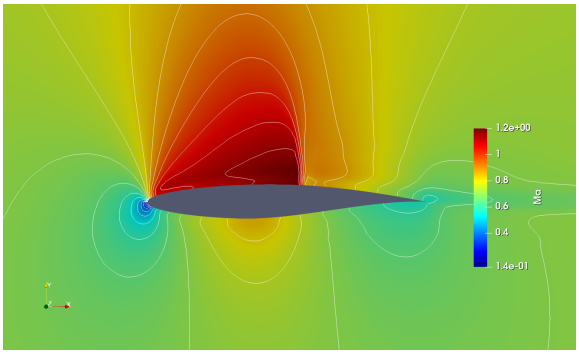
-	Ref	Opt $\mathcal{O}_1$	Opt $\mathcal{O}_2$
C_l	0.7526	0.6558	0.7588
C_d	0.01355	0.009986	0.01182
E	55.54	65.67	64.20
$\Delta\mathcal{O}[\%]$	-	15.44	12.79

Table 2: SSMO - Results

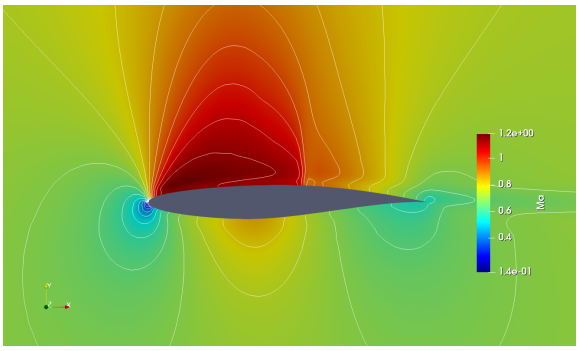
surrogate model that maps the whole design space, as the one presented in section 3.1. This model is then given to the optimization algorithm that finds a minimum in the objective function using the surrogate model to evaluate the aerodynamic coefficients.
One of the advantages associated in the generation of such a model is that, once it has been built, it is possible to inexpensively perform optimizations with different objective functions. This would not be possible using directly the CFD solver, since for each objective function it would be necessary to calculate a large amount of solutions, requiring a high computational effort. The model presented in section 3.1 has been indeed used with both the objective functions defined in section 4.1, giving the results presented in Table 2, where E is the aerodynamic efficiency, Opt $\mathcal{O}_1$ is the airfoil optimized with the goal of increasing efficiency, and Opt $\mathcal{O}_2$ is the airfoil optimized in the drag-reduction case, $\Delta\mathcal{O}$ is the percentage improvement on the case-specific objective function. In each one of the two cases, the model is able to predict the aerodynamic coefficient values at the minimum point with a C_l prevision error below the 0.3% and a C_d prevision error below the 2.5%.

4.3. Iterative Surrogate Model Optimization

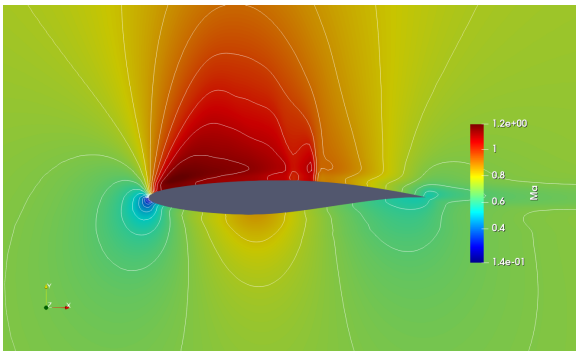
We propose then an Iterative Surrogate Model Optimization (ISMO) with the goal of reducing the number of samples required for the training of the networks, therefore reducing the computational cost of the optimization. This procedure consists in reducing the dimension of the parameters space, reducing the boundary of each design variable, generating the space S_1. Using the samples previously generated that fall inside this reduced space, an ANNs-based surrogate model is trained. At this point, if the generalization error of the surrogate model is above the accepted tolerance (1.5%) 25 new samples are



(a) Mach number contour - Original Airfoil



(b) Mach number contour - SSMO Optimized Airfoil



(c) Mach number contour - ISMO Optimized Airfoil

Figure 5: Mach number contours - Comparison

added in this space, and the training procedure is repeated. The resulting model is used as in the SSMO case to perform the optimization within the space S_1, concluding the first iterative step. The following steps k, for $k = 2, ..., N_k$ consist again in the selection of the space S_k whose center is shifted in the minimum point of the previous iteration. This procedure continues until the relative norm of the step N_k falls below the tolerance $\hat{t} = 0.5\%$. This procedure was tested only in the drag-reduction case. After the first two steps, the drag coefficient has decreased of the 13.31%, which is an increase of the 0.6% in the performance compared to the SSMO case. The research of this minimum required only 234

Step	C_l	C_d	$\Delta\mathcal{O}^k$	$\Delta\mathcal{O}^{tot}$
1	0.7537	0.01216	-	10.28
2	0.7608	0.01175	3.372	13.31
3	0.7584	0.01170	0.4274	13.67

Table 3: ISMO steps

samples, with a 54% reduction in the computational time compared to the SSMO. With the third step, the objective function improved further, obtaining in this way the best result of the whole work with this objective function. The percentage improvements on the objective function at each step are given in Table 3, where $\Delta\mathcal{O}^k$ is the improvement compared to the previous step and $\Delta\mathcal{O}^{tot}$ is the total improvement. The third iterative step required the addition of 50 samples, increasing the computational time from 65 to 78 hours, which is a 43% improvement compared to the SSMO procedure, achieving at the same time better results.
Figure 5 shows the comparison between the original airfoil simulation, and the solutions obtained in the drag-reduction case with the SSMO and ISMO procedures. It can be seen that, in both cases, the reduction in the drag coefficient is achieved with a reduction in the shock intensity. This is caused by a smooth isentropic compression that starts soon after the leading edge. This compression allows to reduce the pressure jump caused by the shock and hence the shock drag and the drag coefficient.

4.4. HF Model Optimization

To investigate the advantages associated with the surrogate model optimization, an optimization using the CFD solver to generate the force coefficients was performed. Due to computational limitations, this optimization was limited to 10 iterative steps of the interior-point method. Findings show that, with the same number of interior-point iterations and with 148 calls to the CFD solver, the results obtained are worse in terms of objective function improvement, compared to the same procedure performed with the surrogate model. Furthermore, the computational time required for the CFD simulations called from Matlab is increased of the 50%, requiring a total of 72 computer hours.

5. Conclusions

The airfoil shape optimization requires multiple calls to expensive numerical CFD solvers, the usage of surrogate models in the context of constrained optimization is an attractive proposition. This work shows that, as long as the surrogate model provides an accurate approximation of the PDE while being computationally cheap to evaluate, it can be used within standard optimization algorithms with good results, leading to computational advantages.
Among the tested procedures, the Standard Surrogate Model Optimization has the main advantage of being inexpensively used to perform optimizations with different goals. It allowed us to obtain a 15.4% improvement with the goal of increasing the aerodynamic efficiency, and a 12.8% improvement in the drag-reduction problem, requiring a total computational time of 135 hours. The Iterative Surrogate Model Optimization, instead, allowed us to obtain a 13.7% reduction in the drag coefficient, reducing the computational time required for the generation of the training set to 78 hours. With the same parameters used in the Standard Surrogate Model Optimization, the Direct Optimization using the CFD solver instead of the surrogate model has shown to significantly increase the required computational time and, with the same number of iterations, it gave worse optimization results.

References

[1] P. Cook, M. McDonald, and M. Firmin. Aerofoil rae 2822-pressure distributions, and boundary layer and wake measurements. *AGARD Report AR*, 138, 1979.

[2] R. Lapuh. Mesh morphing technique used with open-source cfd toolbox in mdo. Master's thesis, Uppsala Universitet, 2019.

[3] S. Mishra and T. Rusch. Enhancing accuracy of deep learning algorithms by training with low-discrepancy sequences. *SIAM Journal on Numerical Analysis*, 59(3):1811–1834, 2021.

[4] F. Regazzoni, L. Dede', and A. Quarteroni. Machine learning for fast and reliable solution of time-dependent differential equations. *Journal of Computational physics*, 397:108852, 2019.

POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Shape Optimization of Airfoils by Machine Learning-Based Surrogate Models

Tesi di Laurea Magistrale in
Aeronautical Engineering
Ingegneria Aeronautica

Author: **Marco Zanichelli**

Student ID: 943979
Advisor: Prof. Luca Dede'
Academic Year: 2020-21

A mio nonno Romano ,
con il quale avrei voluto condividere questo giorno

Abstract

In this thesis, airfoil shape optimization is carried out by integrating Computational Fluid Dynamics with Machine Learning algorithms. With this aim, we perform: i) the generation of an appropriate data-set and ii) the construction of a surrogate model linking design parameters to aerodynamic force coefficients. This model, based on Artificial Neural Networks (ANNs), is then used to perform the airfoil optimization.

A goal of this work is to produce an appropriate data-set that is required to train the ANNs-based surrogate models. This set is composed of input/output pairs, which the Machine Learning algorithm uses to find a model mapping the input space to the output space. To perform this task, Hicks-Henne functions are used for the perturbation of the initial airfoil geometry, generating a sufficient number of airfoil samples. The input values of the training data-set are represented by the numerical values of the Hicks-Henne bumps intensities, which define the shape of each sample. The outputs of the data-set are the airfoil lift and drag coefficients, computed through CFD simulations in OpenFOAM, an open-source software based on finite volume space discretization.

The surrogate models are then generated by means of Feed-Forward ANNs, using a sufficient number of samples so that they can accurately predict the force coefficients on unseen airfoils. The development of the ANNs-based model allows us to perform the airfoil shape optimization, with different objectives, without the need to run expensive CFD simulations, hence leading to significant savings in terms of computational time.

Keywords: Airfoil Optimization, Machine Learning, Artificial Neural Networks, Surrogate Model, Computational Fluid Dynamics.

Abstract in lingua italiana

In questa tesi l'ottimizzazione di forma di un profilo alare viene svolta integrando la fluidodinamica computazionale con algoritmi di Machine Learning. Le fasi principali di questo lavoro sono rappresentate dalla generazione di un data-set e dalla costruzione di un modello surrogato in grado di legare le variabili di progetto ai coefficienti aerodinamici. Questo modello, basato su Reti Neurali Artificiali, è quindi utilizzato per svolgere l'ottimizzazione del profilo.

Uno degli obiettivi di questo lavoro è rappresentato dalla generazione di un set di dati richiesti per la fase di apprendimento del modello surrogato. Questo set è composto da coppie input/output che vengono utilizzate dall'algoritmo di Machine Learning per trovare un modello che colleghi gli spazi delle variabili di ingresso e di uscita. Per generare questi dati sono state utilizzate delle funzioni di Hicks-Henne per la deformazione della geometria iniziale del profilo alare, producendo, così, un opportuno numero di campioni. Gli input della Rete Neurale sono costituiti dai valori delle intensità dei bump di Hicks-Henne, che definiscono la geometria dei profili perturbati. I valori di output sono rappresentati dal coefficiente di portanza e di resistenza di ogni profilo. Questi coefficienti sono ottenuti con simulazioni CFD svolte con il software open-source OpenFOAM, basato su una discretizzazione spaziale operata con il Metodo dei Volumi Finiti.

I modelli surrogati vengono quindi generati tramite Reti Neurali Artificiali con flusso in avanti ("feed-forward"), usando un numero di campioni sufficiente per poter ottenere un modello surrogato in grado di prevedere accuratamente il valore dei coefficienti aerodinamici anche su profili non utilizzati durante la fase di allenamento della Rete Neurale. Questo ha permesso di effettuare l'ottimizzazione di forma del profilo senza dover richiamare il software CFD, comportando un significativo risparmio in termini di costo computazionale.

Parole chiave: Ottimizzazione di Profili Alari, Machine Learning, Reti Neurali Artificiali, Modello Surrogato, Fluidodinamica Computazionale.

Contents

Introduction

Fluid mechanics has traditionally dealt with massive amounts of data from experiments, field measurements, and large-scale numerical simulations. In the last years, the advances in computational hardware and the reduced costs for computations have allowed to increase the volume of data easily available to the user. These improvements have fueled renewed interest in the field of Machine Learning (ML) to extract information from these data.

Artificial intelligence and ML can be applied to several different sectors of the aerospace industry such as air traffic management [1, 14], turbulence closure modelling [15, 37, 41], shape optimization [39, 44, 65] and control [6, 16].

In 2017, the aviation sector created 13.9% of the emissions from transport, making it the second biggest source of transport Greenhouse Gas (GHG) emissions after road transport. Aviation also impacts the climate through the release of nitrogen oxides, water vapour, and sulphate and soot particles at high altitudes, which could have a significant climate effect [17]. This problem is confirmed by the European Union Aviation Safety Agency report [13]. To achieve climate neutrality, the European Green Deal sets out the need to reduce transport emissions by 90% by 2050 (compared to 1990-levels). The aviation sector will have to contribute to the reduction.

Shape optimization is a crucial task in the aerospace industry for both the reduction of costs and emissions. A 1% decrease in drag coefficient on a Boeing 747 can lead to a reduction of almost 400.000 liters of fuel each year, this impacts both on the airline operating costs and on the emissions [2].

In this work, we applied ML to Computational Fluid Dynamics (CFD) to perform an airfoil shape optimization. CFD has become an important tool in the aerospace industry; it is widely used within the design and performance evaluation of aerodynamic bodies [50, 66].

Given the high computational costs associated with CFD simulations, it would be beneficial to find a direct relationship between given inputs and aerodynamic observ-

ables without explicitly solving the discretized flow equations. In this work, we use ML algorithms to find a surrogate model linking geometric parameters with aerodynamic observables, i.e. aerodynamic coefficients. To perform this task, we need to choose accurately the geometry parametrization algorithm, together with a suitable mesh deformation algorithm. These tools will be used before the Artificial Neural Network training phase to generate a training set of input/output couples, using CFD to generate the outputs. We determine the CFD numerical solution by means of the Finite Volume Method, which is implemented in the solver *rhoCentralFOAM* [22], whithin the open-source software OpenFOAM [23]. The computational grid is deformed using Radial Basis Functions [12]. After a comparison between Radial Basis Functions and Hicks-Henne functions [29], the airfoil shape parametrization is performed using the latter, since with a low number of design parameters it was possible to fix the leading and trailing edges position, and at the same time achieve a smoother shape deformation, matching the findings of Castonguay et al. [9].

Once a sufficient number of airfoils samples is generated and simulated, we train the surrogate model made by a feed forward Artificial Neural Network with a single hidden layer. This surrogate model allows a fast and accurate evaluation of the design variables and it is used to perform the shape optimization, using the interior-point method. This optimization is performed in two different ways, first with a unique surrogate model mapping a large design space (standard method) and then with an iterative method, training a sequence of surrogate models whose design spaces are centred in the minimum found by the previous step.

Then, the results obtained with the surrogate models optimization are presented. The combination between the surrogate model and the interior-point method allowed us to improve the objective function by more than the 10%.

Finally, the advantages of the surrogate model optimizations are investigated through the comparison with a direct optimization performed with the high fidelity CFD model. This is aimed to understand the advantages of the generation of such a model in terms of total computational time..

This thesis dissertation is organized into seven chapters, whose main topics are summarized below.

Chapter 1

This chapter provides basic notions of Fluid Dynamics and CFD. It starts presenting Euler equations and then it presents the main features of the Finite Volume Method

and describes the meshing options and the solver used in this work.

Chapter 2
This chapter illustrates the choices made regarding the geometry parameterization algorithms. Then, it explains the mesh deformation procedure, which based on Radial Basis Functions. Finally, it presents the main sampling possibilities and the different types of surrogate models that could be used in this work.

Chapter 3
This chapter is intended to provide the basic concepts of Artificial Intelligence and Machine Learning. The main parameters required for a Neural Network are introduced and explained in more detail, to provide the necessary theoretical knowledge to apply ANNs to a case study.

Chapter 4
This chapter presents the setup and the results of the numerical simulation of the reference airfoil, which is a RAE 2822 supercritical airfoil. The setup validity is assessed with a comparison with an experimental solution.

Chapter 5
This chapter describes further the parametrization choices, highlighting the pros and cons of each one of the two methods tested: Radial Basis Functions and Hicks-Henne functions. Then, it presents details on the implementation of mesh deformation algorithm.

Chapter 6
This chapter starts with a description of the ML models that are used in this work to obtain the surrogate model. It continues with the learning results of the most relevant tests performed.

Chapter 7
This chapter describes the the optimization strategy and the shape optimization results. Then, it presents the comparison between the standard surrogate model optimization, the iterative surrogate model optimization and the optimization made using the high fidelity CFD solver. Conclusions follow.

1 | Basic notions of the Finite Volume Method

This chapter provides basic information about Euler Equations and CFD, following [35] and [45].

1.1. Governing Equations

Euler Equations describe how the velocity, pressure, and density of a moving fluid are related, named in honour of Leonard Euler. They are a set of coupled differential equations obtained with a simplification of the more general Navier-Stokes equations of fluid dynamics and they can be solved for a given flow problem by using methods from calculus. The Euler equations neglect the effects of the viscosity of the fluid which are instead included in the Navier-Stokes equations. A solution of the Euler equations is therefore only an approximation to a real fluids problem. For some problems, like the lift of a thin airfoil at a low angle of attack, a solution of the Euler equations provides a good model of reality [25]. For other problems, like the growth of the boundary layer on a flat plate, the Euler equations do not properly model the problem. A useful feature of the Euler equations is that, in the unsteady case, they are hyperbolic, and can be written in conservation form, as follows for the 1-D case:

$$\frac{\partial}{\partial t}\mathbf{u}(x,t) + \frac{\partial}{\partial x}\mathbf{f}(\mathbf{u}(x,t)) = 0 \tag{1.1}$$

Where $\mathbf{u}$ is a vector of conserved quantities, or state variables, such as mass, momentum, and energy in a fluid dynamics problem.

The main assumption underlying (1.1) is that knowing the value of $\mathbf{u}(x,t)$ at a given point and time allows us to determine the rate of flow, or flux, of each state variable at (x,t). The vector-valued function $\mathbf{f}(\mathbf{u})$ with jth component f($\mathbf{u}$) is called the flux function for the system of conservation laws.

Once the viscosity terms from the Navier-Stokes equations are dropped, the system

becomes hyperbolic and can be written in conservation form as follows, in one space dimension:

$$\frac{\partial}{\partial t}\begin{bmatrix}\rho\\ \rho v\\ E\end{bmatrix}+\frac{\partial}{\partial x}\begin{bmatrix}\rho v\\ \rho v^2+p\\ v(E+p)\end{bmatrix}=0. \tag{1.2}$$

Where $\rho=\rho(x,t)$ is the density, v is the velocity, ρv is the momentum, E is the energy, and p is the pressure. The pressure p is given by a known function of the other state variables. Equation (1.2), can be written for a general 3D case as:

$$\frac{\partial \mathbf{u}(\mathbf{x},t)}{\partial t}+\nabla\cdot\mathbf{f}(\mathbf{u}(\mathbf{x},t))=0. \tag{1.3}$$

Where $\mathbf{f}(\mathbf{u}(\mathbf{x},t))$ is the general flux vector of the conservative quantity $\mathbf{u}(\mathbf{x},t)$.

1.2. Finite Volume Method

The Finite Volume Method, FVM, has come to play a unique role amongst the numerical methods used to implement CFD [36]. It is a technique that transforms partial differential equations (PDEs) representing conservation laws over differential volumes into discrete algebraic equations over finite volumes, also called elements or cells. The first step in the solution process is the discretization of the geometric domain, which, in the FVM, is discretized into N non-overlapping elements or finite volumes. The partial differential equations are then discretized into algebraic equations by integrating them over each discrete element. The system of algebraic equations is then solved to compute the values of the dependent variable for each of the elements, obtaining the numerical solution [35]. This process is illustrated in Figure 1.1. In the FVM, some of the terms in the conservation equation are turned into face fluxes and evaluated at the finite volume faces. Because the flux entering a given volume is identical to that leaving the adjacent volume, the FVM is strictly conservative [45].

1.2.1. Domain Discretization

The numerical solution of a PDE consists of the values of the dependent variable $\mathbf{u}$ at specified points from which its variation over the domain of interest can be constructed. These points are called grid elements, and result from the discretization of original geometry. In all the numerical methods for the approximate solution of PDEs the focus is on replacing the continuous exact solution of the partial differential

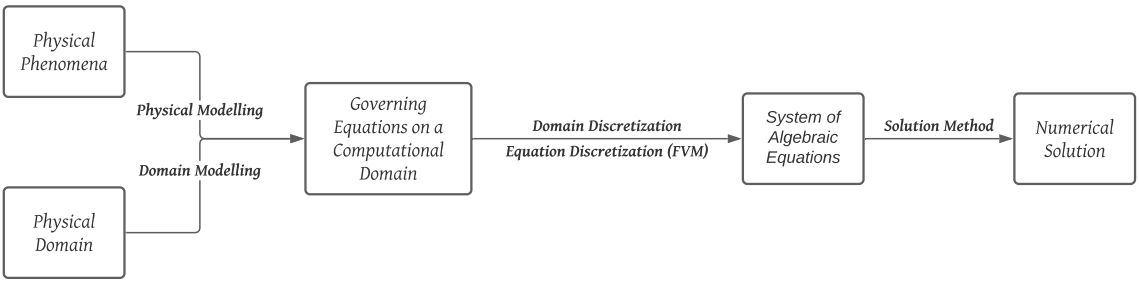


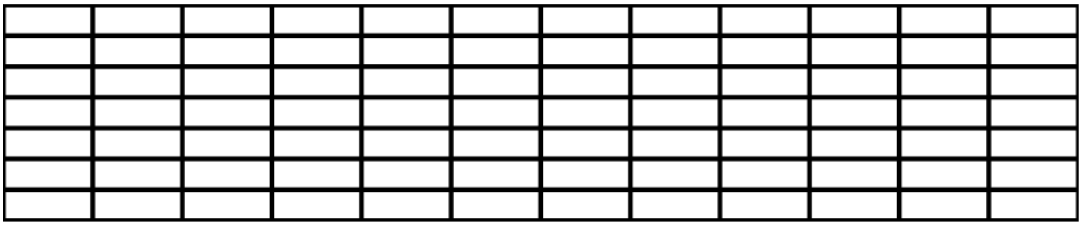
Figure 1.1: Discretization Process

equations with discrete values. The discrete values of **u** are typically computed by solving a set of algebraic equations relating the values at neighboring grid elements to each other; these discretized or algebraic equations are derived from the conservation equation governing **u** [45].

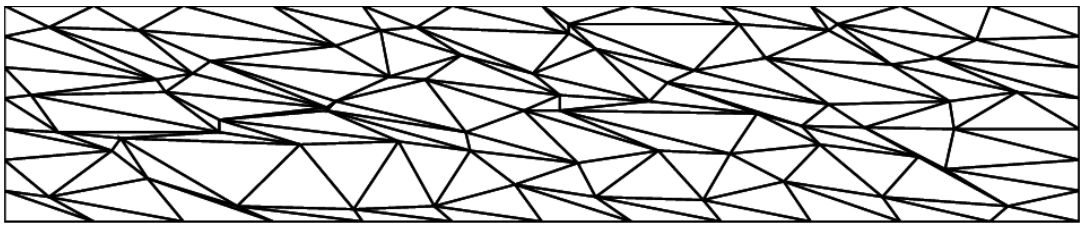
The geometric discretization of the physical domain results in a mesh on which the conservation equations are eventually solved. This requires the subdivision of the domain into discrete non-overlapping cells or elements that completely fill the computational domain to yield a grid or mesh system. For the mesh to be a useful platform for equation discretization, information related to the topology of the mesh elements, in addition to some derived geometric information, are needed. These include element to element relations, face to elements relations, geometric information of the surfaces, area and normal direction. This information is usually inferred from the basic mesh data. For certain mesh topologies, details about the mesh can be easily deduced from the element indices as in structured grids, while for others it has to be constructed and stored in lists, as for unstructured grids [45].

The advantage of a structured grid is that its connectivity is simple and the points of an elemental cell can be easily addressed by double indices (i,j). The disadvantage, particularly for more complex geometries, is the increase in grid non-orthogonality or skewness that can cause unphysical solutions due to the transformation of the governing equations.

Unstructured meshes are instead well suited for handling arbitrary shape geometries, especially for domains having high curvature boundaries. In this type of grid, an elemental cell may have an arbitrary number of neighbouring cells attaching to it, making the data treatment and connection much more complicated with respect to the structured case, as explained by J. Tu et al. in [63]. Figure 1.2 illustrates the differences between structured and unstructured grids in 2D.



(a) Structured grid



(b) Unstructured grid

Figure 1.2: Types of grids, image taken from [58]

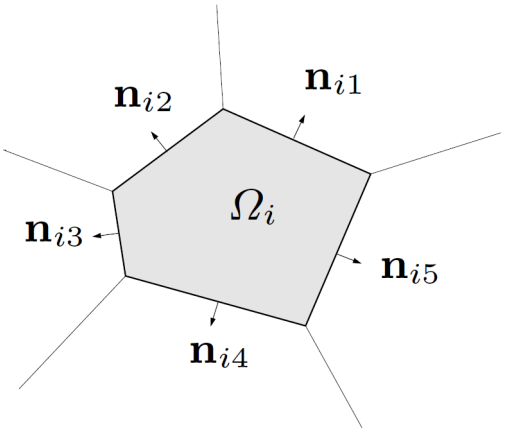


Figure 1.3: Finite Volume Ω_i. Image taken from [47]

1.2.2. Governing Equations Discretization

In the FVM, the discretization procedure of the governing equations starts from the integration of the PDEs over a each element Ω_i, represented in Figure 1.3, as explained in [47]. This, for Equation (1.3) results in

$$\int_{\Omega_i} \left(\frac{\partial \mathbf{u}(\mathbf{x},t)}{\partial t} + \nabla \cdot \mathbf{f}(\mathbf{u}(\mathbf{x},t)) \right) d\Omega = 0. \tag{1.4}$$

Which is the integral form of the conservation law, evaluated on that element. Using the divergence theorem allows to obtain a system of ODEs. Equation (1.4) becomes:

$$\frac{\partial}{\partial t}\int_{\Omega_i}\mathbf{u}(\mathbf{x},t)\ d\Omega+\int_{\partial\Omega_i}\mathbf{f}(\mathbf{u}(\mathbf{x},t))\cdot\mathbf{n_i}dl=0 \tag{1.5}$$

where the second integral represents the net flux across the surface of the control volume considered and $\mathbf{n}_i$ represents the vector of unit normal, positive outwards of $\partial\Omega_i$. Denoting with L_i the number of interfaces l_{ij} of Ω_i, and by $\mathbf{n}_{ij},\ j=1,..,L_i$ the unit vector normal to the side ij of $\partial\Omega_i$, the previous equation can be rewritten as

$$\frac{\partial}{\partial t}\int_{\Omega_i}\mathbf{u}(\mathbf{x},t)\ d\Omega+\sum_{j=1}^{L_i}\int_{l_{ij}}\mathbf{f}(\mathbf{u}(\mathbf{x},t)\cdot\mathbf{n}_{ij}dl=0. \tag{1.6}$$

For $i=1,...,\mathrm{M}$ ith M being the total number of elements. The interfaces l_{ij} can be lines (in 2D) or surfaces (in 3D).

The flux at the interfaces $\mathbf{f}(\mathbf{u})$ is replaced by a numerical flux $\mathbf{F}(\mathbf{u})$. This numerical flux can be defined according to different numerical schemes, but it needs to satisfy flux conservation at adjacent control volumes and consistency [35]. A numerical scheme is identified by the way the numerical flux approximates the physical flux across each cell face or edge. The reader is referred to [35] for more information on the numerical schemes.

The surface integral needs to be evaluated at each face of the element, which can be an interface with another element or with the domain boundary. Considering the interface between the cell i and j:

$$\int_{\partial l_{ij}}\mathbf{f}(\mathbf{u})\cdot\mathbf{n}_{ij}\ dS\simeq\mathbf{F}_{ij}\Delta l_{ij} \tag{1.7}$$

At this point, a piecewise constant cell average is introduced for each grid element Ω_i.

$$\hat{\mathbf{u}}_\mathbf{i}(\mathbf{t})=\frac{1}{|\Omega_i|}\int_{\Omega_i}(\mathbf{u}(\mathbf{x},t))\ d\Omega \tag{1.8}$$

Substituting Equation (1.8) in the first term of Equation (1.6) yields

$$\frac{\partial}{\partial t}\int_{\Omega_i}\mathbf{u}(\mathbf{x},t)\ d\Omega=|\Omega_i|\frac{\partial\hat{\mathbf{u}}_i}{\partial t}. \tag{1.9}$$

The combination of Equation (1.6), (1.7) and (1.9) leads to the space discretization

of the governing equation in conservation form, evaluated on the element Ω_i:

$$|\Omega_i|\frac{\partial \hat{\mathbf{u}}_i}{\partial t} = \sum_{j=1}^{L_i} \mathbf{F}_{ij}\Delta l_{ij} \tag{1.10}$$

To obtain a fully discrete form, it is required to introduce a time discretization. Using the forward Euler scheme on Equation (1.10) yields

$$|\Omega_i|(\hat{\mathbf{u}}_i^{n+1} - \hat{\mathbf{u}}_i^{n}) = -\Delta t \sum_{j=1}^{L_i} \mathbf{F}_{ij}\Delta l_{ik}. \tag{1.11}$$

Where time $n+1$ and n refer to two consecutive discrete time steps. The the computational domain has now been discretized, and the governing PDEs are transformed into a set of algebraic equations, one for each element in the computational domain. These algebraic equations are then assembled into a global matrix and vectors that can be expressed in the form:

$$\mathbf{A}[\hat{\mathbf{u}}] = \mathbf{b} \tag{1.12}$$

where the unknown variable vector $\hat{\mathbf{u}}$ is defined at each interior element and at the boundary of the computational domain. Boundary values for $\hat{\mathbf{u}}$ are generally obtained from the specified boundary conditions [45].

The value of $\mathbf{u}$ at a grid point influences the distribution of $\hat{\mathbf{u}}$ only in its immediate neighborhood. As the number of grid elements increases, the solution of the discretized equations is expected to approach the exact solution of the corresponding differential equation.

Steady - State Simulations

The problem here considered in this work is steady. Steady-state solutions are achieved by setting an initial condition, e.g. a uniform flow, and introducing a fictitious time $\hat{t}$. This problem is hence treated as an unsteady problem, in which the steady state solution is achieved when the variation in the solution between two following pseudo-time steps falls below some tolerance. The pseudo-time discretization can be performed, for example, with the Crank-Nicolson method, for which equation (1.13) is discretized as equation (1.14), where the subscript i refers to the spatial point i, where n is the pseudo-time step.

$$\frac{\partial u(x,\hat{t})}{\partial \hat{t}} = F(u,x,\hat{t}) \tag{1.13}$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta \hat{t}} = \frac{1}{2}[F(u_i^{n+1}, x_i, \hat{t}^{n+1}) + F(u_i^n, x_i, \hat{t}^n)] \tag{1.14}$$

1.2.3. CFD Solver: OpenFOAM

The Open Source Field Operation and Manipulation (OpenFOAM) is an open-source CFD software package, which implements the FVM. All the codes in OpenFOAM are written in C++ with an oriented programming interface [23]. It provides a variety of solvers and utilities both pre-processing, such as the meshing tools blockMesh, and snappyHexMesh and postprocessing solvers with several finite volume solvers that can work on structured and unstructured grids. These solvers are capable of solving the steady, unsteady, compressible, incompressible, viscous, inviscid, laminar, and turbulent flows using finite volume numeric that solve a system of PDEs within up to three space dimensions.

Any OpenFOAM case structure contains three major folders called *0*, *constant*, and *system* respectively. The *0* folder contains all the initial field definitions like pressure, temperature, velocity, turbulent energy, dissipation rate etc. The *constant* folder contains full information about the geometry and boundary conditions. The *system* folder contains information about the solver control. Figure 1.4 depicts the case structure of OpenFOAM.

A central theme of the OpenFOAM design is that the solver applications, written using the OpenFOAM classes, have a syntax that closely resembles the partial differential equations being solved. For example the equation:

$$\frac{\partial \rho \mathbf{v}}{\partial t} + \nabla \cdot \phi \mathbf{v} - \nabla \cdot \mu \nabla \mathbf{v} = -\nabla p \tag{1.15}$$

Is represented as:

```
    solve
(
    fvm::ddt(rho, v)
  + fvm::div(phi, v)
  - fvm::laplacian(mu, v)
    ==
  - fvc::grad(p)
);
```

This operation casts the PDE into a matrix system of the form $[\mathbf{A}][\hat{\mathbf{u}}] = [\mathbf{b}]$, where

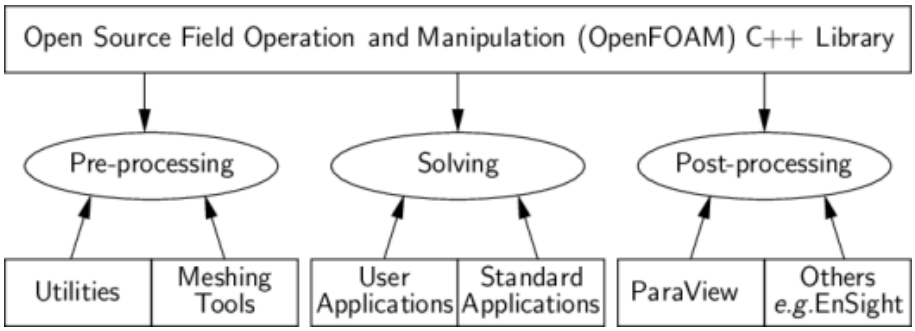


Figure 1.4: OpenFOAM case structure. Image taken from [23]

[**A**] is composed of the algebraic coefficients derived from the discretization of the convective, gradient and Laplacian terms; [$\hat{\mathbf{u}}$] is the matrix of the dependent variables and [**b**] is the matrix resulting from the source terms [57].

rhoCentralFoam

The finite volume numerical solution of compressible fluid flow equations can be addressed using different approaches i.e, pressure-based and density-based solvers with the corresponding governing equations solved either in segregated or coupled manners [22]. Within OpenFOAM, both approaches are implemented, for instance, the density-based *rhoCentralFoam* and the pressure-based *sonicFoam*.

rhoCentralFoam solves explicitly the inviscid part of governing equations and then corrects them implicitly to account for viscous terms. It has been chosen for this work since it contains the inviscid flow option, which, if activated, allows to avoid the calculation of the viscous correction, allowing for faster computation.

2 | Design Variables and Reduced-Order Modelling

The aerodynamic shape optimization will be performed in Chapter 7 by using a surrogate model that links some design variables to the force coefficients. The construction of a surrogate model that allows a reliable evaluation of the output is crucial for an accurate investigation of the parameters space.

The model construction is based on the results obtained by the evaluation of a significant number of samples via the High Fidelity model that the surrogate model approximates. In this case, those samples consist in input/output pairs, where the inputs are given by the geometrical parameterization and the outputs by the CFD solution of several different airfoils. To produce different airfoils and meshes, a suitable parameterization and mesh deformation algorithm need to be chosen. Then, after the selection of an appropriate sampling sequence and surrogate model, shape optimization can be performed.

2.1. Choice of the Design Variables

There are many ways to parametrize an airfoil, such as Bezier Curves [68], NURBS [19, 46] , B-Splines [40]. This works uses as initial geometry the RAE 2822 supercritical aerofoil, set in transonic flow conditions. Since in this case we are referring to a specific reference airfoil of which we know the exact coordinates, we do not want to find a parametrization that resembles the base airfoil, because this would entail a difference between the original and the parametrized airfoil. In airfoil optimization, a crucial aspect is the shape deformation and the mesh morphing technique. Between the possibilities Free Form Deformation (FFD) [56], Hicks-Henne functions [38, 39, 67], Radial Basis Functions [37] need to be considered. The idea of FFD parameterization is to embed a flexible object into a parallelepiped lattice of control points that define the parametric space. By modifying the lattice, a deformation is transmitted to the included object similarly, creating a new shape. FFD is

known to be reliable with small boundary deformations. The Hicks Henne formulation consists instead in using a base airfoil to which is added a linear combination of trigonometric "bump" functions to perturb the upper and lower surface of the initial geometry. Radial Basis Functions (RBF) can also be used to deform the original coordinates of the airfoil. This can be achieved, in the same way of the free form deformation, by linking the movement of some points of the domain (that can be some points on the airfoil surface) with a part or all of the grid points, via Radial Basis Functions interpolation.

According Castonguay et al. [9], B-splines and Hicks Henne are among the best solutions to perform shape parameterization with a low number of design variables. Thanks to its simplicity and for the possibility to obtain a smooth surface deformation, RBFs deformation has been tested at first. This method allows to modify the initial shape using as design variables some of the points on the airfoil. Then, after a comparison with the Hicks-Henne formulation, the latter has been used for the following studies, since it gives more freedom on surface deformation with a low number of design variables.

The mesh deformation will be performed in all cases using Radial Basis Functions. When RBFs are also used for the geometry deformation, both the modification of the airfoil shape and of the mesh are performed at the same time, whereas when Hicks-Henne functions are used for the airfoil deformation, the two steps remain distinct, as further explained in the next section.

In the following sections, the theory behind Hicks - Henne and RBFs will be reviewed.

2.1.1. Radial Basis Functions

In this work, the mesh created for the simulation of the RAE 2822 airfoil is deformed to account for the deformations in the shape of the airfoil. This mesh deformation approach is based on Radial basis function interpolation of the displacements of some control points. This method requires knowing the initial position of each one of the mesh nodes, the position and the displacement of the control points. From the displacement of these control points, the displacement of all the mesh points is derived. The initial mesh for CFD can be structured or unstructured. This method can deal with large shape deformation with good quality, excellent versatility, and robustness, according to [12].

This method has been used in two ways:

1. By choosing a low number of control points and imposing their translation it is possible to obtain at the same time the deformation of the airfoil and of the mesh. This is possible because the points of the airfoil surface are treated in the same way of the other mesh points. Their displacement is hence derived from the motion of the control points. This procedure requires the tuning of a smoothing parameter, σ, that will be introduced in the next section.
2. By choosing as control points all the airfoil boundary points, whose displacement is obtained by means of Hicks-Henne functions, as explained in Section 2.1.2.

Mathematical Representation

This section is inspired by the work of De Boer et al. in [12].

Radial Basis Functions Φ are real-valued functions whose value at the point $\mathbf{x}$ depends only on the distance from a certain control point $\mathbf{x_i^c}$, hence $\Phi(\mathbf{x}, \mathbf{x_i^c}) = \Phi(||\mathbf{x} - \mathbf{x_i^c}||)$.

The interpolation function $\mathbf{s}$ describing the displacement in the whole domain in the x, y and z directions, can be approximated by a sum of basis functions:

$$\mathbf{s}(\mathbf{x}) = \sum_{i=1}^{N} \gamma_i \Phi(\|\mathbf{x} - \mathbf{x_i^c}\|) + \mathbf{h}(\mathbf{x}) \tag{2.1}$$

where $\mathbf{x_i^c} = (x_i^c,\ y_i^c)^T$ are the control points in which the displacement values are imposed, N is the number of control points, each one associated with a coefficient γ_i, Φ is a given Radial Basis Function that can assume different expressions, as the ones in Tables 2.1 and 2.2, and $\mathbf{h}(\mathbf{x})$ is a linear polynomial that enables the rigid motion of the point $\mathbf{x}$:

$$\mathbf{h}(\mathbf{x}) = \beta_1 + \beta_2 x + \beta_3 y, \tag{2.2}$$

with $\mathbf{x} = (x,\ y)^T$. $\mathbf{s}(\mathbf{x})$, γ_i, $\mathbf{h}(\mathbf{x})$ are vectors, where the number of elements depends on the dimensionality of the case, assuming one element for each space dimension. The coefficients γ_i and the polynomial coefficients of $\mathbf{h}(\mathbf{x})$ are determined by interpolation conditions, imposing that $\mathbf{s}(\mathbf{x_i^c}) = \bar{\mathbf{s}}_\mathbf{i}$ with $\bar{\mathbf{s}}_\mathbf{i}$ being the known value of the displacement of the i-th control point. An additional requirement is that the total

contribution of the polynomial in the control point is null. This condition reads:

$$0 = \sum_{i=1}^{N} \gamma_i q(\mathbf{x_i^c}). \tag{2.3}$$

for all polynomials q with a degree less or equal to the degree of the polynomial $\mathbf{h}$.

Table 2.1: RBF with global support

Name	$\Phi(\mathbf{x})$
Spline type (n odd)	$\sigma\|\mathbf{x}\|^n$
Multiquadric	$\sqrt{1+\sigma\|\mathbf{x}\|^2}$
Gaussian	$1-e^{-\sigma\|\mathbf{x}\|^2}$

Table 2.2: RBF with local support

Name	$\Phi(\eta)$
CP C^0	$\sigma(1-\eta)^2$
CP C^2	$\sigma(1-\eta)^4(4\eta+1)$

From the previously explained conditions one can obtain a the following system of equations:

$$\begin{bmatrix} \bar{\mathbf{s}} \\ 0 \end{bmatrix} = \begin{bmatrix} M_{\mathrm{c,c}} & P_{\mathrm{c}} \\ P_{\mathrm{c}}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\gamma} \\ \boldsymbol{\beta} \end{bmatrix} \tag{2.4}$$

Where $M_{\mathrm{c,c}}$ is a $N \times N$ matrix such that $M_{ij} = \Phi(||\mathbf{x_i^c} - \mathbf{x_j^c}||)$ for $1 < i, j < N$ and P_c is the matrix that contains the control points coordinates, with row j given by $(1,\ x_j^c,\ y_j^c)$ Once the linear system is solved, one can obtain:

$$\mathbf{s}(\mathbf{x}) = \begin{cases} s_x(\mathbf{x}) = \sum_{i=1}^{N} \gamma_i^x \Phi\left(\|\mathbf{x} - \mathbf{x_i^c}\|\right) + \beta_1^x + \beta_2^x x + \beta_3^x y \\ s_y(\mathbf{x}) = \sum_{i=1}^{N} \gamma_i^y \Phi\left(\|\mathbf{x} - \mathbf{x_i^c}\|\right) + \beta_1^y + \beta_2^y x + \beta_3^y y \end{cases} \tag{2.5}$$

This system allows to find the displacement $\mathbf{s}(\bar{\mathbf{x}})$ of the generic mesh point $\bar{\mathbf{x}}$, such that:

$$\bar{\mathbf{x}}_{\mathbf{deformed}} = \bar{\mathbf{x}} + \mathbf{s}(\bar{\mathbf{x}}). \tag{2.6}$$

There are several types of RBFs presented in the literature, the main division that can be done is between RBFs with global and local support. When an RBF with local support is used, mainly the mesh points inside a circle in 2D (or a sphere in 3D) with radius r and centre $\mathbf{x_i^c}$ are influenced by the movement of the centre. The main

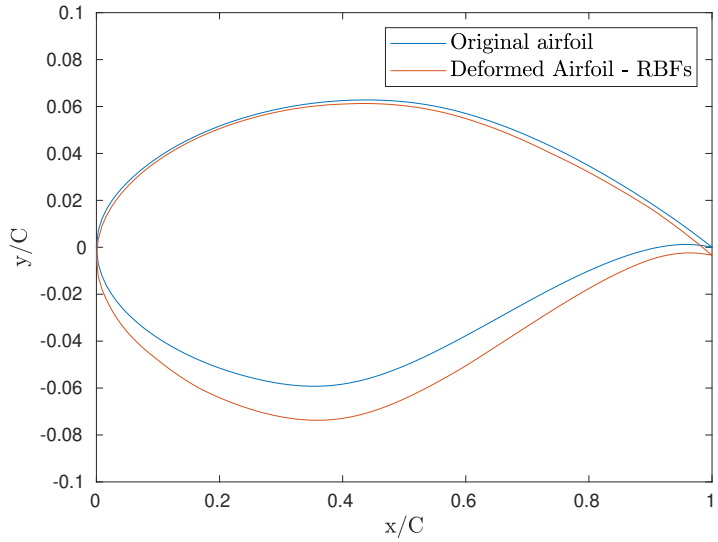


Figure 2.1: Example of RBFs-deformed airfoil

advantage of RBF with local support is that the system of equations that needs to be solved (2.4) becomes sparse, whereas with global support it is full. Global support, on the other hand, provides more accurate solutions.

In Tables 2.1 and 2.2, some of the most common RBFs are presented, where $\eta = ||\frac{\mathbf{x}-\mathbf{x_i^c}}{R_s}||$, with R_s being the support radius of the local RBFs.

With this procedure, the displacement of each grid point can be computed individually starting from its original position and the displacement of the boundary points, without needing any information on the connectivity structure of the grid.

From this formulation, it is easy to obtain the airfoil deformation by choosing as control points $\mathbf{x_i^c}$ some of the points on the airfoil surface. Imposing a movement on those points allows to obtain new airfoil geometries. Increasing the parameter σ allows to obtain a smooth surface deformation. This methods works with both two and three dimensional meshes. In this work, in particular, the mesh is two-dimensional and the boundary movement has been given only in the y direction, which is perpendicular to the airfoil's chord.

2.1.2. Hicks-Henne Deformation

These alternative parameterization has been introduced to overcome some issues found with the RBFs parameterization, that are shown in Section 5.2.1. Hicks-Henne functions have been chosen since they have shown good results in airfoil

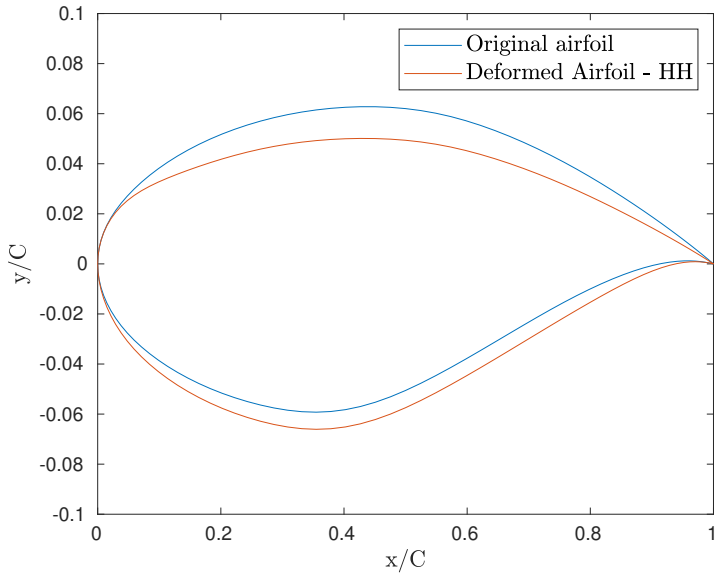


Figure 2.2: Example of Hicks-Henne functions deformed airfoil

parameterization using a low number of design parameters [9]. This is a crucial point of this work since the increase in the dimension of the problem has shown to dramatically increase the computational costs associated with the shape optimization. Figure 2.2 shows an example of airfoil obtained with this procedure using eight design parameters.

These functions have been introduced by R. Hicks and P. Henne in 1978 [29] in the context of airfoil optimization. This method consists in adding a linear combination of n augmented sine functions to the original coordinates of the airfoil, these functions are in the form of a bump.

$$y_{\text{mod}} = y_0 + \sum_{i=0}^{n} a_i \sin^{w_i}(\pi x^{\ln(0.5)/\ln(x_i^M)}) \tag{2.7}$$

Where y_{mod} are the final coordinates of the upper or lower surface of the airfoil, n is the number of bumps for each one of the upper or lower surfaces, xmi is the x coordinate of the bump, w_i is the bump width, while a_i are the bumps intensities. Each bump can hence be defined by three variables: a_i, w_i, x_i^M. In this work, w_i and x_i^M are fixed, while the a_i are the parameters of the optimization. Figure 2.1 shows an example of airfoil obtained with this procedure using eight design parameters, while Figure 2.3 shows the shapes of the bumps for $n = 16$, $t = 4$, $x_i^M = 0.5(1 - sin(\theta_i))$, with $\theta_i = \pi \frac{i}{n}$, for $i = 1, 2, ..., n$.

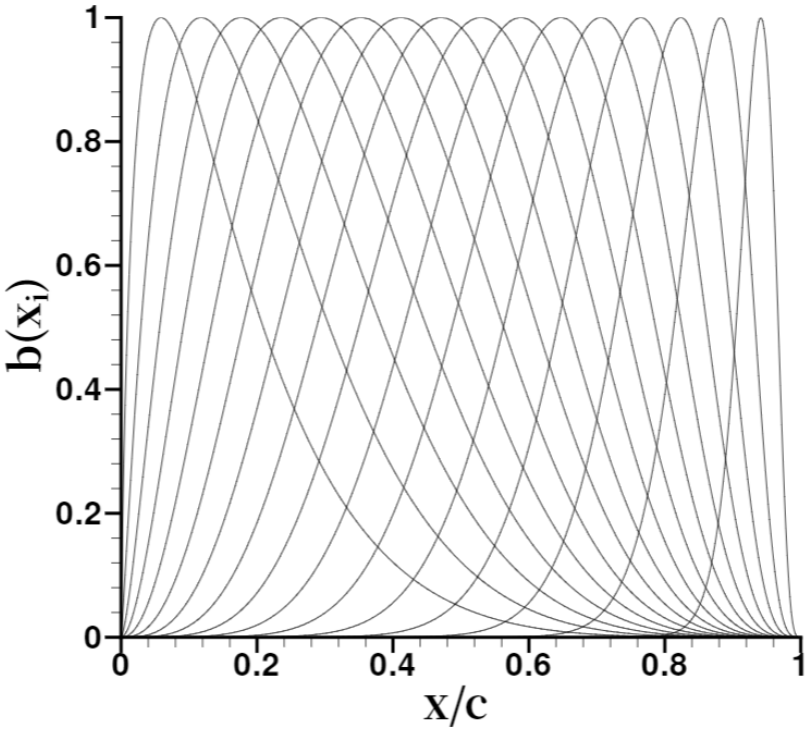


Figure 2.3: Hicks Henne bumps

Once the coordinate of each bump has been chosen, the y-coordinate of each airfoil point is modified according to equation (2.7). The difference between the final y-coordinate y_{mod} and the initial coordinate y_0 are then given to the RBFs mesh deformation algorithm as a deformation vector, which is used to determine the motion of each one of the mesh points.

It should be noticed that in this case the dimension of the deformation vector is far greater than the Free-Form RBFs case, this causes a much higher computational time required for the mesh deformation. This cost remains negligible compared to the computational cost of the CFD simulation.

2.2. Shape Optimization and Surrogate Models

Shape optimization is a crucial task in Aeronautical Industry, being able to reduce drag on a civilian airliner can increase the profits of that aeroplane by several percentage points. Performing an aerodynamic optimization requires multiple calls to high-fidelity CFD software, in some contexts the computational burden can be unsustainable. It is important to reduce at most this cost, choosing the procedure that allows maximizing the results in these terms.

Before choosing the optimization procedure it is important to define an objective function, that can be for example the drag coefficient at fixed lift or the lift-to-drag ratio. Once this has been done, the first possibility is to start from an initial geometry and numerically compute the derivatives of the objective functional with respect to any design parameter. This method can be costly since it requires at least one call to the CFD solver for each derivative evaluation.

Another option that must be considered is to construct a surrogate model. Surrogate models have been introduced in the context of shape optimization to capture the most important features of a high fidelity model at a low computational cost. The surrogates are constructed using data drawn from high-fidelity models, and provide fast approximations of the objectives and constraints at new design points, thereby making sensitivity and optimization studies feasible [49]. The computational time required for the generation of the data set and the construction of the surrogate model should be lower than the time required for the optimization performed directly with the HF software. Once the surrogate model has been built it is easy to perform parametric studies and optimizations with different objectives which would be infeasible if they were to be performed with the HF model.

The construction of a surrogate model usually involves the following tasks, described also in Figure 2.4:

1. Design space sampling. This can be done using one of the methods described in Section 2.3;
2. Numerical simulation of the selected location via High Fidelity model;
3. Construction of a surrogate model;
4. Evaluation of the generalization error made by the model, using unseen data. If the model does not satisfy the accuracy requirements, the previous tasks need to be repeated.

Surrogate models can be divided into three categories: reduced-order models, data fit models, multi-fidelity models.

2.2.1. Reduced-Order Models and Multi-Fidelity Models

A reduced-order model can be derived from the HF model using a projection technique. These methods require the generation of a data set and the computation of a set of basis functions, such as eigenvectors, and the identification of how many of

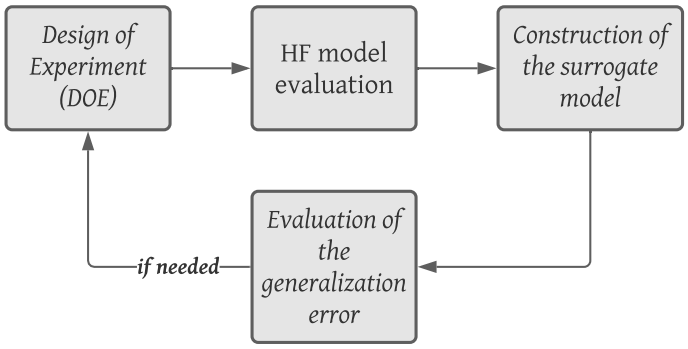


Figure 2.4: Stages of the surrogate model construction

them are required to capture well enough the dynamics of the problem.

One example of these methods is the Proper Orthogonal Decomposition (POD) [28, 48], which has been used in the approximation of Euler [21] and Navier-Stokes equations [7]. This method uses a series of snapshots generated with the HF model and computes the left singular vectors corresponding to the most dominant singular values of the matrix having the snapshot vectors as its columns. The POD basis can then be used to determine an approximate solution for different values of the design parameters.

In the case of multi-fidelity surrogate models, the model is a lower fidelity approximation of the HF model, but it is still based on the same physical considerations. This can be done for example in CFD using a coarser domain discretization.

Data Fit Models

Data fitting methods involve the construction of a surrogate model using data generated with the high fidelity model. These methods involve the evaluation of the HF model on many points spread over the parameters ranges. Using these data it is possible to generate a response surface [18] that can be obtained using different methods, such as:

- Polynomial regression, the surrogate is built using of n^{th} order with least-square regression;
- Radial Basis Functions, the surrogate is built using a summation of local supported RBFs such as the ones presented in Table 2.2;
- Artificial Neural Networks, which consists of an interconnected architecture of artificial neurons that changes some parameters as information flows in the network during the learning phase. In this work, as mentioned above, Artificial

Neural Networks will be used to produce a surrogate model linking the design parameters to the aerodynamic force coefficients.

2.3. Design Space Sampling

Once the design variables and the type of surrogate model have been defined, one important step is to choose how to vary the parameters in the design space. The choice of the sampling type is crucial for the success of the generation of a working surrogate model. It influences both the result and the number of samples needed.

2.3.1. Grid Search and Random Search

The easiest way to vary the design parameters is called "grid search", this method consists in the creation of a Cartesian grid of points that will then be evaluated with the High Fidelity model. This method is exhaustive but requires the evaluation of many points to obtain good results in terms of accuracy of the surrogate model.

The Random Search method consists in choosing random points in the parameters space where the High Fidelity model is evaluated. This method does not guarantee good results but, for a sufficient number of points, it is more promising than grid search [5].

2.3.2. Latin Hypercube

Latin Hypercube sampling has been introduced in the context of numerical integration over multi-dimensional domains [61]. It is one of the most common sampling techniques, as it allows in many cases to reduce the number of samples needed for a good approximation of the High Fidelity model. This strategy allows for a more efficient search using the same number of points, it examines more values for each parameter and ensures that each value shows up only once in randomly blended combinations. This method is more likely to give better results with respect to grid and random search.

Figure 2.5 shows the different samples chosen with the three methods that have just been explained, showing that for those two variables X_1 and X_2 both grid search and random search do not sample the peak in the corresponding function, whereas with the Latin Hypercube Sampling technique those functions are sampled correctly and it is more likely to obtain a better approximation.

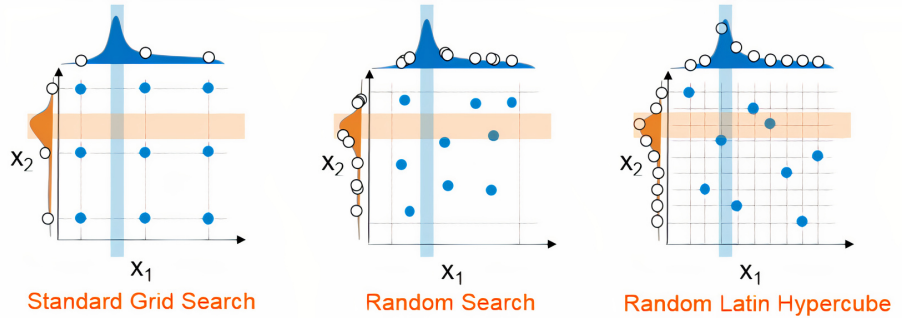


Figure 2.5: Different sampling techniques. Image taken from [54]

2.3.3. Low Discrepancy Sequences

Low discrepancy sequences are Quasi-Montecarlo sequences that allow a faster rate of convergence with respect to standard Montecarlo algorithms [30]. In the context of numerical integration, the rate of convergence of a standard Montecarlo algorithm is of the order $\mathcal{O}(1/\sqrt{N})$, whereas with a low discrepancy sequence such as Sobol sequence the rate of convergence is close to $\mathcal{O}(1/N)$ [33]. In the same context, the LHS technique can outperform the Standard Montecarlo techniques, but only for specific types of functions [33]. Mishra et al. [43] motivate the use of low-discrepancy sequences to train neural networks was based on the equidistribution property of these sequences, i.e. these sequences fill the underlying domain more uniformly than random points and hence can better represent the underlying map. Sobol sequence has been successfully used to generate samples for Deep Learning-based surrogate models in [39]. For these reasons, Sobol sequence has been chosen in this work.

3 | Basic Concepts of Machine Learning

"Machine learning (ML) is the study of computer algorithms that can improve automatically through experience and by the use of data."

Mitchell, Tom (1997). Machine Learning.

This chapter aims to provide a broad introduction to Machine Learning and Neural Networks' history and fundamentals.

3.1. Artificial Intelligence and Machine Learning

Artificial Intelligence made its first steps at the beginning of the 20th Century. In 1936, the English mathematician Alan Touring developed a code-breaking machine for the British government, aiming to decipher the Enigma code used by the German army in the Second World War. This machine, *The Bombe*, is generally considered the first working electro-mechanical computer [24]. Touring published in 1950 the article "Computing Machinery and Intelligence" [64] describing how to create intelligent machines, providing also a test: the machine can be considered if it can fool people into thinking it is a person. The word Artificial Intelligence was coined in 1956 by M. Minsky and J. McCarthy. In 1969, D. Hebb started a research on the imitation of the process of neurons in the human brain, which led to the creation of the research on Artificial Neural Networks [24]. The evolution of this field was slowed down by the moderate computing capabilities available at that time. Through the years, many challenges were beaten as when in 1997 the IBM computer *DeepBlue* beat the chess world champion, Gary Kasparov. In the last years, we have entered the age of Big Data, in which we can collect huge amounts of data, too much for a person to process. The application of AI has been fruitful in many industries, such as banking, marketing, and entertainment [3]. It is now part of our everyday life inside our mobile phones, for speech recognition, image recognition, and much more. In the future, we can expect that ML will have a growing impact in most of the

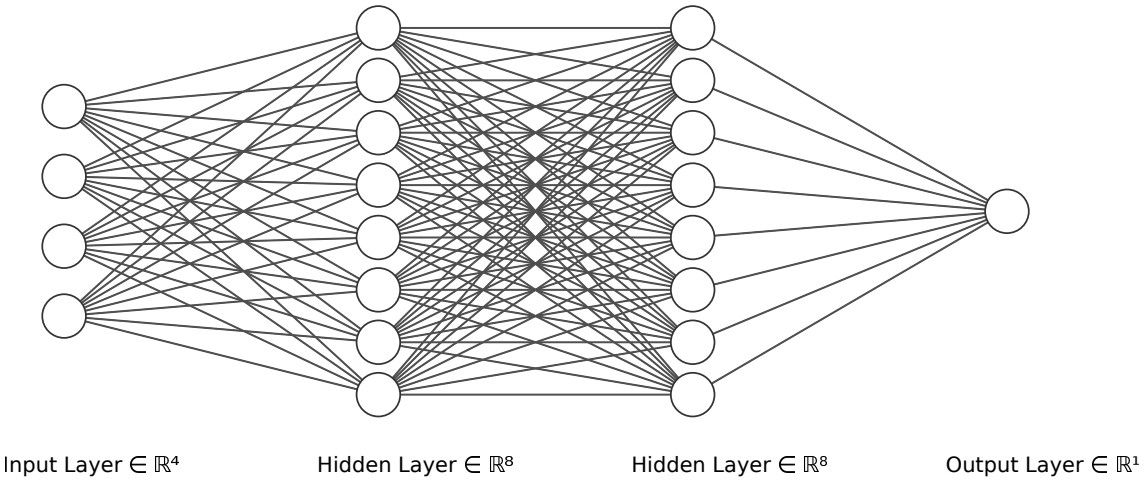


Figure 3.1: Artificial neural network

fields, among the most promising projects, the development of quantum computing can be mentioned [4]. To better understand the differences between AI, Machine Learning, and Artificial Neural Networks some definitions are needed.

Artificial Intelligence is the field of study that aims to enable a machine to simulate human behaviour, with a very wide range of scope. Machine Learning is a subset of AI that focuses on enabling computers to perform tasks without explicit programming. The goal of ML is to allow a machine to automatically learn from data so that it can give accurate predictions for future outputs. A Neural Network is a collection of algorithms used in Machine Learning for data modelling using graphs of neurons. Figure 3.1 shows an example of Artificial Neural Network.

Machine Learning algorithms can be categorized into two classes: supervised and unsupervised learning.

3.1.1. Supervised Learning

Supervised Learning uses labelled data sets to train algorithms to classify data or predict outcomes accurately. This training data set includes inputs and correct outputs, which allow the model to learn over time, measuring accuracy through a loss function. Supervised Learning can be divided into two types of problems:

- **Classification**: the algorithm deals with discrete output values i.e. categories. It aims to find which labels need to be assigned to input data in order to divide it into those categories. One example of a classification algorithm is logistic regression.

- **Regression**: the algorithm is used to find a continuous relation between input and output data. It can be used to make projections. One example of regression is polynomial regression.

3.1.2. Unsupervised Learning

In Unsupervised Learning the task is to extract features from data, discovering hidden patterns or group data without human supervision. One of the most important classes of these algorithms is represented by Clustering algorithms, which are used to find structures within data that share similar features. These clusters can be exclusive or overlapping and their number can be pre-assigned (e.g. K-means algorithm [27]) or can be automatically decided by the algorithm [31].

3.2. Neural Networks

(Artificial) neural networks are information processing systems, whose structure and operation principles are inspired by the nervous system and the brain of animals and humans. They consist of a large number of fairly simple units, the so-called neurons, which are working in parallel. These neurons communicate by sending information in the form of activation signals, along with directed connections, to each other [32]. Between the many useful properties of ANNs, the *Universal approximation theorem* [11] deserves to be reported. This theorem states that ANNs with one hidden layer (provided that there is a sufficient number of neurons) can approximate any continuous function to an arbitrarily small error.

3.2.1. Biological Background

This section is taken from [32] and it is intended to explain the biological structures which ispired ANNs.

The effective structure of artificial neural networks is an oversimplification of the biological process happening in our brain, which is depicted in Fig 3.2.

A neuron is a cell that collects and transmits electricity pulses. The cell body, which contains the nucleus is called soma. From the cell body extend several branches that are called dendrites and a long extension called axon, that can be up to 1mt long. The axons are the paths along which neurons communicate with each other. The axon of a neuron is connected to the dendrites of other neurons. At the end of the axons, there are ramifications that end in terminal buttons. Each button

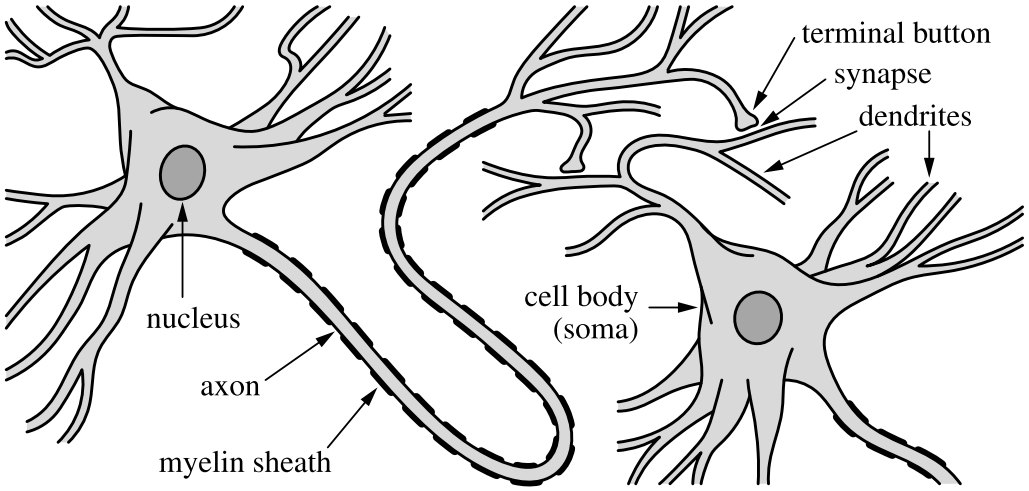


Figure 3.2: Neuron structure. Image taken from [32]

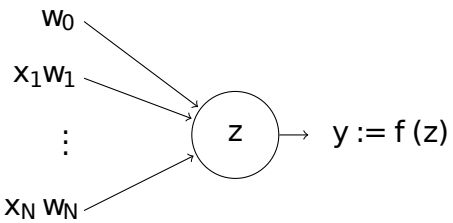


Figure 3.3: Artificial Neuron

almost touches a dendrite of another neuron. That gap is called synapse. The most common form of communication between neurons is that a terminal button of the axon releases chemicals that act on the receiving dendrite and change its electric potential. If the sudden change of potential caused by the interaction of the neuron with many others, is large enough, above a certain threshold, the impulse propagates along the axon. In this way, information is transmitted.

3.2.2. Artificial Neuron - Mathematical Modelling

We model a neuron as a threshold logistic unit, that takes as an input n real-valued inputs x_i and returns as output a real number y. A weight w_i is assigned to each input x_i. The inputs x_i and the weights w_i with $1 < i < n$ are combined into an input vector $\mathbf{x}$ and a weight vector $\mathbf{w}$ with the addition of a bias unit w_0, as in Figure 3.3.

The inner state of the neuron z is then computed as $z = \sum_{i=1}^{n} w_i x_i + w_0$. Then the activation function $f(z)$ can be calculated. This function determines the output value of the neuron and can be for example the sigmoid function or the Heaviside

step function.

Neural Networks consist of multiple connected layers of neurons. Each layer contains one or more neurons. The neurons of each layer can be connected to some or all the neurons of the previous and the following layer, depending on network architecture. The network depicted in Figure 3.1 has one input layer, two hidden layers, one output layer.

All neural networks have one hidden layer and one output layer, while the number of hidden layers can change depending on the problem requirements and complexity. Networks in which all the neurons of one layer are connected to all the neurons of the following layer are called fully connected ANNs.

The working principle is the following: the output $y_{ij} = f(z_{ij})$ of the neuron i in layer j is fed as input to all the neurons of the layer $j + 1$ to which that neuron is connected. This value will be multiplied by the appropriate weight in each one of those neurons of the layer $j + 1$. This process is called forward propagation. The learning problem can be summarized as the process of making estimations of the relationship between inputs and outputs. This approximation is stochastic, as the core of a ML process is the minimization of a loss function $J(\theta)$:

$$J(\theta) := \sum_{x \in \mathcal{S}} |\mathcal{L}(x) - \mathcal{L}_\theta(x)|^p \tag{3.1}$$

For any $x \in \mathcal{S}$, with $\mathcal{L}(x)$ being the true output and $\mathcal{L}_\theta(x)$ being the estimated output. At each iteration of the learning algorithm, the weights of each unit are updated according to the gradient of the loss function that measures the distance between the actual and the predicted output through the back-propagation algorithm. The name comes from the fact that errors are propagated backwards through the network to iteratively find optimal values for the weights and biases [55].

3.2.3. Types of networks

Among the many types of neural networks, the most common are:

- The **feed forward NN**, also known as multilayer perceptrons, consists of a network where each neuron of a given layer is connected with each neuron of the next layer, as in figure 3.4;
- The **radial basis NN** are particular multilayer-type perceptrons where the

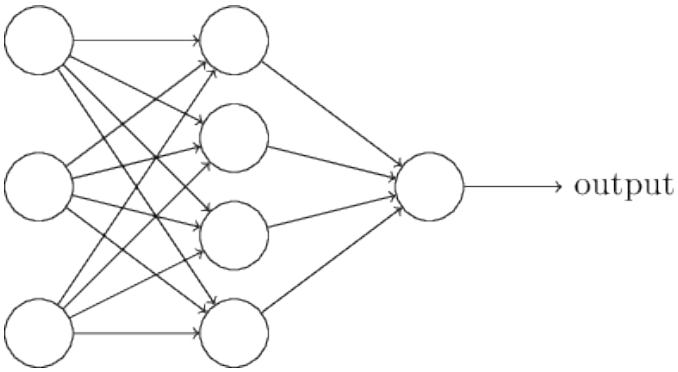


Figure 3.4: Feedforward network

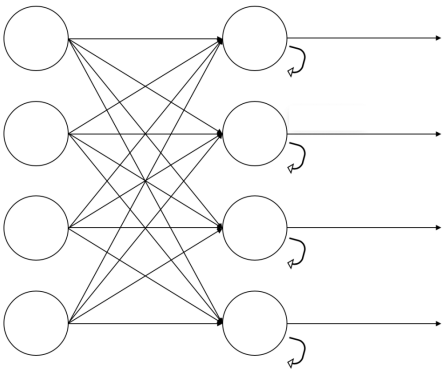


Figure 3.5: Recurrent network

activation function is a radial basis function instead of a logistic function;

- The **recurrent NN** (Figure 3.5) are characterized by their memory as they take information from prior inputs to influence the current input and output. This is done with the introduction of a different cell that receives its output within a fixed delay or following a different law [69]. This kind of network is used for example for natural language processing;
- The **deep feedforward NN** are FFNN that have more than one hidden layer, as in Figure 3.6, this causes a higher computational burden but can lead to better results;
- The **convolutional NN** perform the convolution of the information given in the input layer. They are commonly used in image recognition.

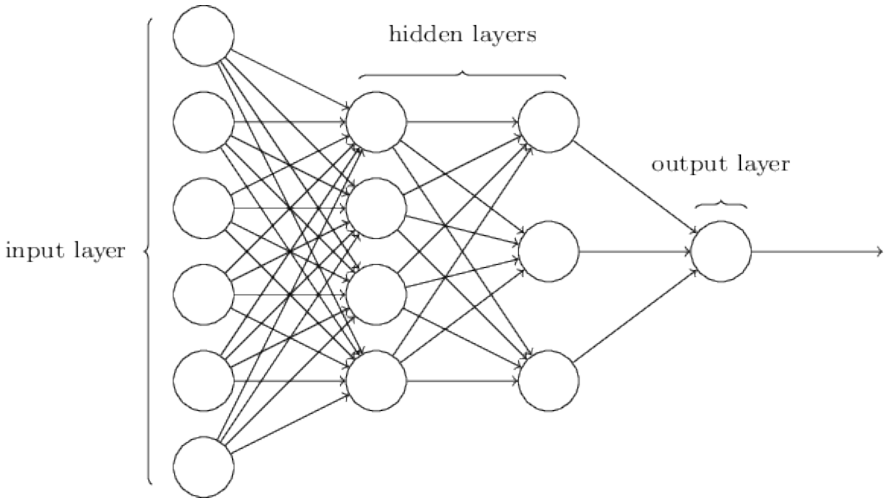


Figure 3.6: Deep feed-forward network

3.2.4. Machine Learning Features

These sections are intended to provide background on some of the most important parameters and problems that can be found in ML problems.

Data Sets

In Supervised Learning problems it is common practice to divide the complete data set into three subsets:

- **Training set**: this is the largest set of data and it is used to fit the model during the learning phase. This set usually contains 60 or 70% of the total samples.
- **Test set**: this set is used to select the model that works best on unseen data. This is done by repeating the training phase for more than one model, varying for example the number of neurons or a regularization parameter. This set usually contains 15 or 20% of the total samples.
- **Cross - validation set**: this set is commonly used to get an estimate of the generalization error, which will be further explained in the next section. This set usually contains 15 or 20% of the total samples.

Loss Function

The goal of the learning phase, as previously stated, is the minimization of a loss function, which in supervised learning is a measure of how well the learning machine

is approximating the I/O relationship. This function can be defined in many ways, depending on the problem goal. In this section the two most common possibilities are presented:

- Mean squared error MSE:

$$MSE = \frac{1}{n}\sum_{i=1}^{N}(y_i - \bar{y}_i)^2 \tag{3.2}$$

- Mean absolute error MAE:

$$MAE = \frac{1}{n}\sum_{i=1}^{N}|y_i - \bar{y}_i| \tag{3.3}$$

Another measure that can be used is the Root Mean Squared Error that is $RMSE = \sqrt{MSE}$.

Model Selection

An important part of the learning phase, where some of the model parameters still need to be selected is called model selection. In this phase, it can be useful to study the evolution of some error measures. The training error is the error that the model does on the training set. This measure usually decreases with the increase of the model complexity, this does not necessarily mean that the model is more accurate.

To have insights on the accuracy, the model needs to be tested on unseen data, in the model selection phase this data is represented by the test set. This error usually assumes a "U" shape as in Figure 3.7. This behaviour is due to the fact that the model can tend to show an oscillatory behaviour outside of the points of the training set, as shown in Figure 3.8 representing a polynomial regression task performed on eleven points with a ten-degree polynomial. This behaviour is known as overfitting and coincides with the right part of Figure 3.7. Overfitting can also happen if the number of epochs, i.e. the number of times the training error is fed back to the network, is too high.

The left part of Figure 3.7 is referred to as underfitting i.e. both the errors on the train and test set are high because the model is too simple to fit the train set data. In that case, the model continues to learn from train data but does not generalize well using unseen (test) data.

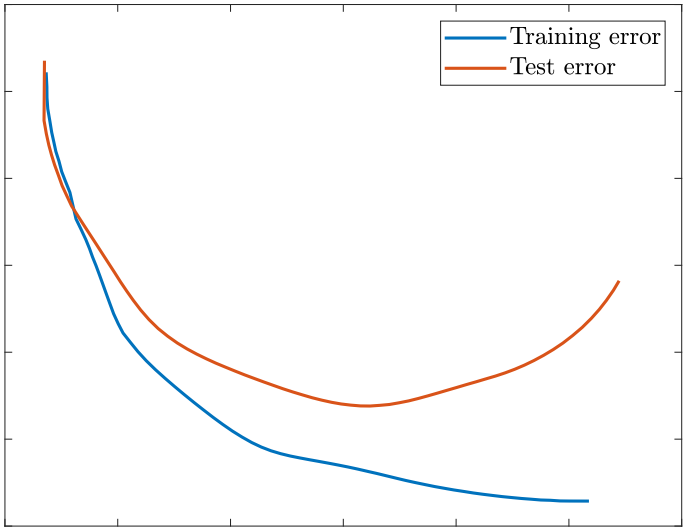


Figure 3.7: Learning curves

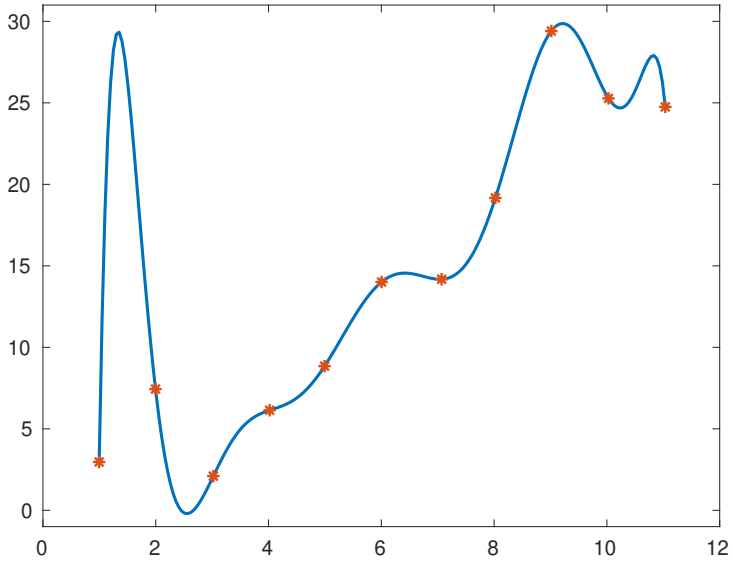


Figure 3.8: Overfitting - polynomial regression

Once the model has been selected, the error on the training set cannot be considered a correct estimate of the accuracy since it has been used in the selection of the model. The accuracy of a supervised learning task is often measured employing the so-called generalization error that is the error on the estimate that is made on unseen data [42] that is the cross-validation set.

Activation Function

One of the most important parameters of the network is the so-called activation function. This function determines the activation (output) value of each neuron, starting from its inner state z and must be computationally effective since it is recalled many times during each step of the learning phase.

The purpose of the activation function is to add a nonlinear behaviour to the network, this is crucial since without it the output of the NN would trivially be a linear transformation of the inputs.

Among the many possibilities, the following deserve to be presented:

- **Heaviside step function**: this is the most simple possibility. The inner state z is compared to a certain threshold θ, if z is greater than θ the neuron will be activated (i.e. the output will be one). The strongest limitation of this function is that it cannot provide multi-valued output. Another problem is that its gradient is null, and this causes troubles in the back-propagation algorithm. The Heaviside step function is illustrated in Figure 3.9.
- **Linear activation function**: the activation is proportional to the inner state. As well as the previous one, this function causes troubles in the back-propagation process as all the derivatives of f are equal and unrelated to the inputs and the inner state of the neuron. The linear activation function is illustrated in Figure 3.10.

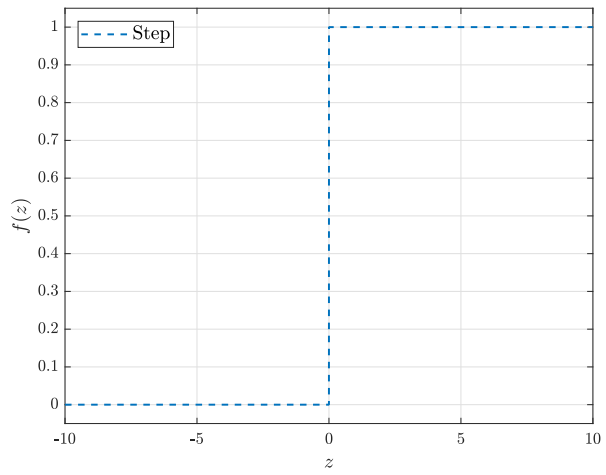


Figure 3.9: Step function

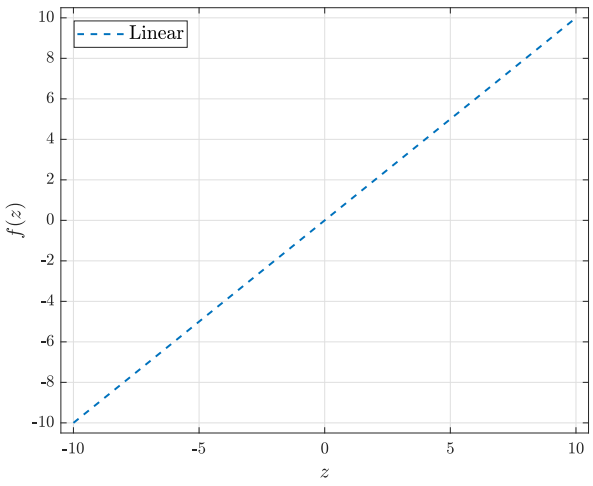


Figure 3.10: Linear function

- **Sigmoid/Logistic activation function**: this is the first nonlinear function here presented and it is one of the most widely used. As illustrated in Figure 3.11, it can assume any value between 0 and 1. The larger the inner state, the larger the activation will be, whereas the smallest (i.e. the more negative) the inner state, the more the activation will be near zero. The main advantage of this function, aside from being nonlinear, is that the output is limited between 0 and 1 and could be seen as a probability value. Furthermore, it has $\mathcal{C}^\infty$ continuity, which ensures the continuity of its derivative. The downsides of this function are its computational cost, involving an exponential function, and the so-called "vanishing gradient" problem [20]. As it can be seen in Figure 3.12 the function's derivative assumes really low values for inner states whose absolute value is greater than 3.

- **Hyperbolic tangent**: this function has similar properties to the previous one. It is shown in Figure 3.13 Its main advantage with respect to the standard sigmoid is that it is symmetric with respect to both the x and y axes, as its output is between -1 and +1. This is the activation function chosen for this work.

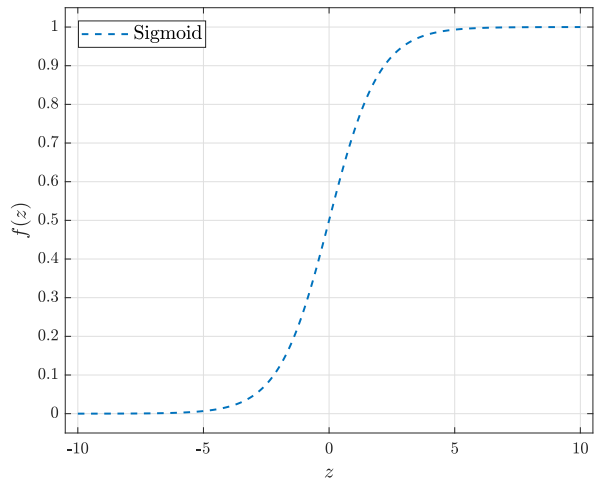


Figure 3.11: Sigmoid function

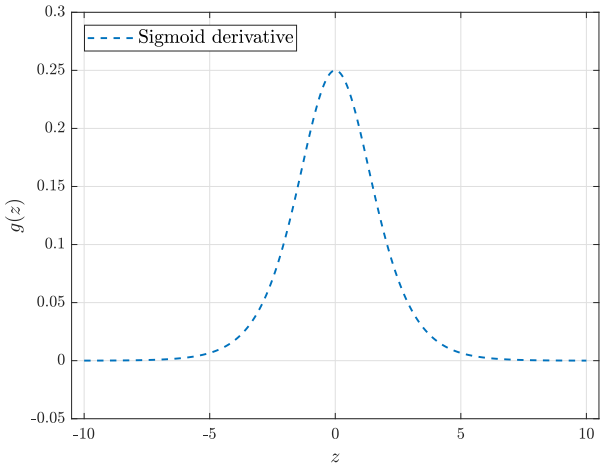


Figure 3.12: Sigmoid derivative

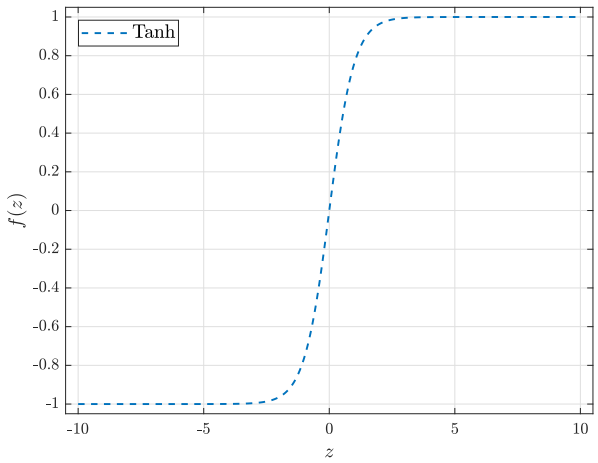


Figure 3.13: Hyperbolic tangent function

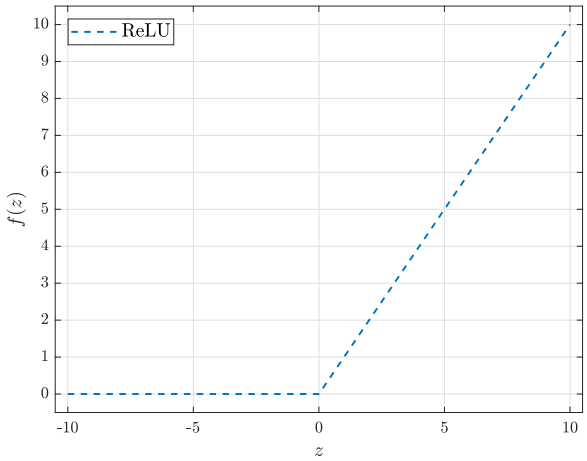


Figure 3.14: ReLU function

- **ReLU activation function**: its name stands for Rectified Linear Unit. As shown in Figure 3.14, it is a simple nonlinear function that activates the neuron only if its inner state is greater than zero. Thanks to its simplicity, this function is computationally effective and can lead to a reduction in the training time.

Back-propagation Algorithm

The process that starts with the inputs assigned to the input layer, that continues in the hidden layer and ends producing the output is called forward propagation. During training, forward propagation is used to get the output estimate needed for the computation of the loss function. Back-propagation instead, allows the information to flow back to compute the gradient needed to modify the weights and biases. In Figure 3.15 the black arrows represent the forward-progagation process, the red dashed arrows represent the back-propagation process

Back-propagation refers to the method for computing the gradient, while another algorithm, such as stochastic gradient descent, is used to perform learning using this gradient [20].

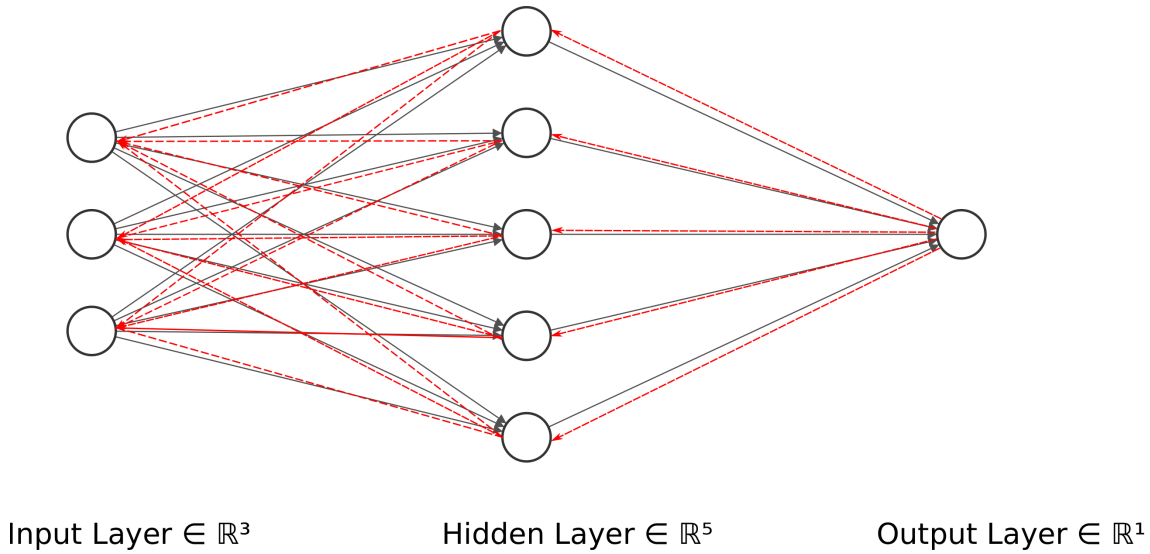


Figure 3.15: Forward propagation and Back-propagation

4 | Numerical Simulations

In this chapter the steps leading to the flow simulation of the baseline case will be presented.

4.1. CFD Airfoil Simulations

The airfoil selected for the optimization is the RAE2822, which has been the object of many well-known studies about transonic flows and model validations. In particular, this work takes as reference the AGARD Report AR 138 by Cook et al [10], from which empirical data has been obtained. Figure 4.1 shows the airfoil geometry.

Figure 4.1: RAE2822 Airfoil

This is a supercritical airfoil which has a maximum thickness of the 12.1% at 37.9% chord and maximum camber of the 1.3% at 75.7% chord. Cook et al. in [10] present a wide range of transonic conditions: from subcritical flow to conditions where a comparatively strong shock wave exists in the flow above the upper surface of the airfoil. The particular solution taken as reference in this work shows a shock

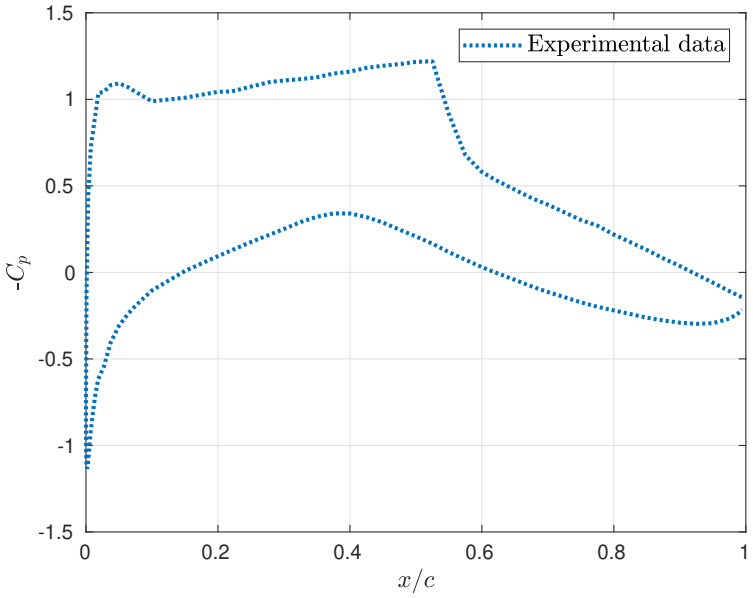


Figure 4.2: Pressure coefficient along the chord

Table 4.1: Force and moment coefficients

Coefficient	Value
C_l	0.7310
C_D	0.01212

wave on the suction side of the airfoil, which is set at the following flow conditions:

$$M_\infty = 0.729 \text{ [-]};$$
$$T_\infty = 288.15 \text{ [K]};$$
$$\alpha = 2.31 \text{ [deg]};$$
$$Re = 6.500.000 \text{ [-]}.$$

Where M_∞ is the freestream Mach number, T_∞ is the freestream temperature, α is the angle of attack of the airfoil, Re is the Reynolds number $Re = \frac{\rho v_\infty c}{\mu}$, computed with the density ρ, freestream velocity v_∞, chord c, and dynamic viscosity μ.

The behaviour of the pressure coefficient along the chord is shown in Figure 4.2. In these conditions, the force coefficients assume the values reported in Table 4.1.

The Mach number contours are presented in Figure 4.3, showing the supersonic bubble formed above the upper surface of the airfoil that ends with a shock wave.

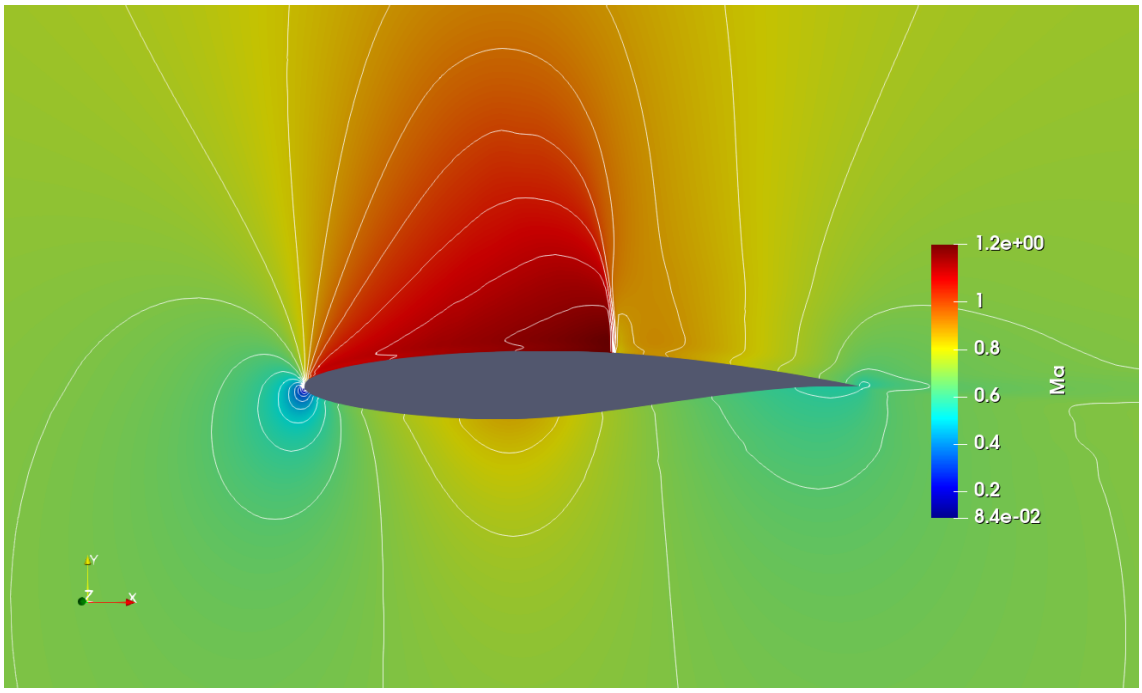


Figure 4.3: Mach contours

4.2. Computational Model

As stated in Chapter 1, the governing equations considered for the solution of this case are the Euler Equations, which are a simplification of the Navier-Stokes equations that would provide more accurate results. This choice has been made to account for computational limitations, since the solution of the viscous equations even with a simple RANS model would have exceeded the computational power available. All the computations are performed on a Lenovo Legion laptop with a 9th Generation Intel Core i7-9750H Processor (2.60GHz, up to 4.50GHz with Turbo Boost, 6 Cores, 12MB Cache).

4.2.1. Mesh Generation

The computational grid has been generated with the Python utility *CuriosityFluids* [60], which allows generating a structured C-shaped grid for the CFD simulation of airfoils with OpenFOAM. This utility requires setting some parameters to produce a grid that fits the user's requirements, such as the height of the first element near the airfoil boundary and the progression of the elements' dimensions along some lines. This python script has been edited to further increase the dimension of the elements in the wake.

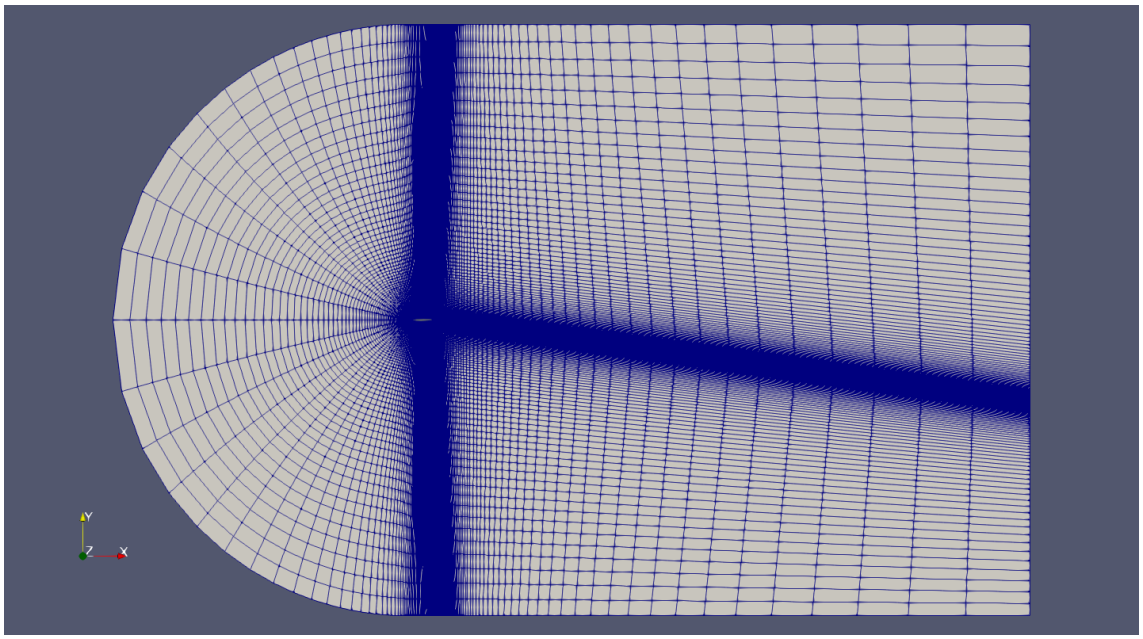


Figure 4.4: Farfield mesh

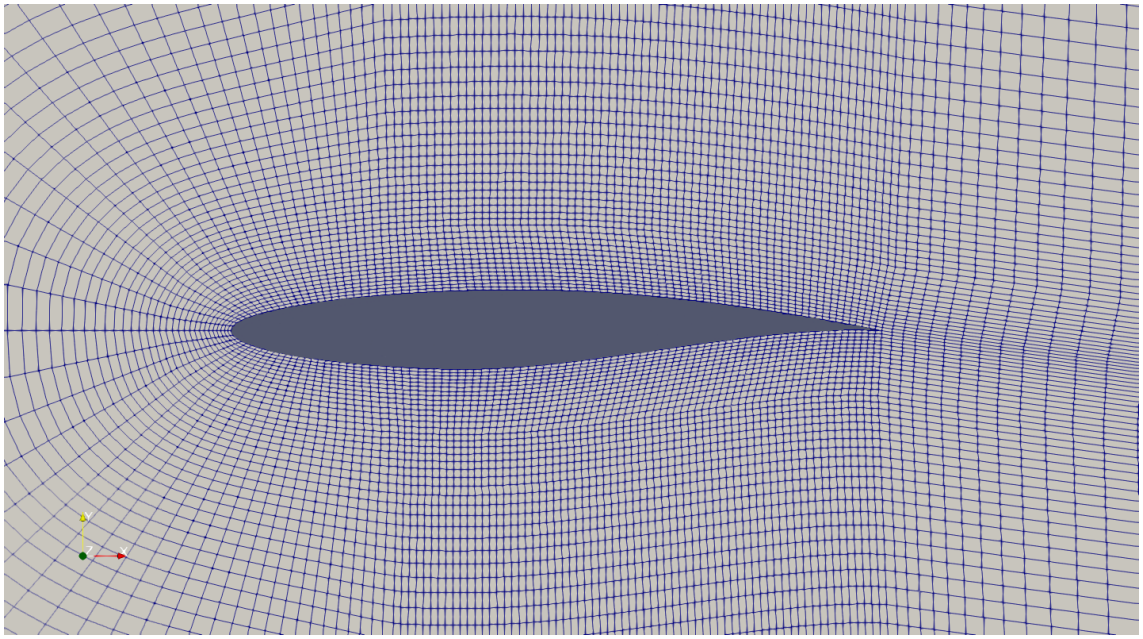


Figure 4.5: Mesh near the airfoil

The computational grid is represented in Figures 4.4 and 4.5, the dimension of the elements decreases approaching the airfoil, with high density in the part of the domain where the physical solution presents a shock wave. The computational domain is made of a semi-circular part which has a radius of $10c$, with c being the

length of the airfoil's chord, and of a square part with a is $20c$ long edge, so that the total length of the domain is $30c$ and its height is $20c$.

4.2.2. Boundary Conditions

In OpenFOAM, initial and boundary conditions are set in the *0* folder, which includes several files that depend on the problem. In this case, four files are present, that refer to temperature (T), pressure (p), Mach number (Ma), velocity (U). These files are reported in Appendix A for completeness. The full external boundary is labelled as farfield, and there the freestream velocity is set as well as the static temperature and pressure, and the Mach number. The boundary condition set on the airfoil is a slip condition.

4.2.3. Stability

One common problem that occurs with *rhoCentralFoam* is that the solution oscillates around a mean value, even if dissipative first order numerical schemes are used for the space discretization. According to Greenshields et al. [22], this problem can be solved by imposing a strict limitation on the Courant number, for this problem the value that allowed to obtain a stable solution is 0.1.

4.2.4. Space and Time Discretization

To give an accurate representation of the discontinuity that is present in the reference flow field, and to reduce oscillations across discontinuities that are associated with second-order schemes, the numerical schemes that were chosen for the space discretization in this work are second-order accurate with flux limiters. The full list of numerical methods is given in Appendix A.

As anticipated in Chapter 1, in a steady-state simulation a fictitious time is introduced to allow the solution to evolve from an initial condition, given in the 0 folder, to the steady-state. In this work, the pseudo-time discretization has been performed using the Crank-Nicolson method reported in Section 1.2.2.

4.2.5. Results

Grid independence has been assessed by looking at the lift and drag coefficients, as can be seen Figures 4.8 and 4.6 showing the drag and lift coefficient errors compared to the reference solution.

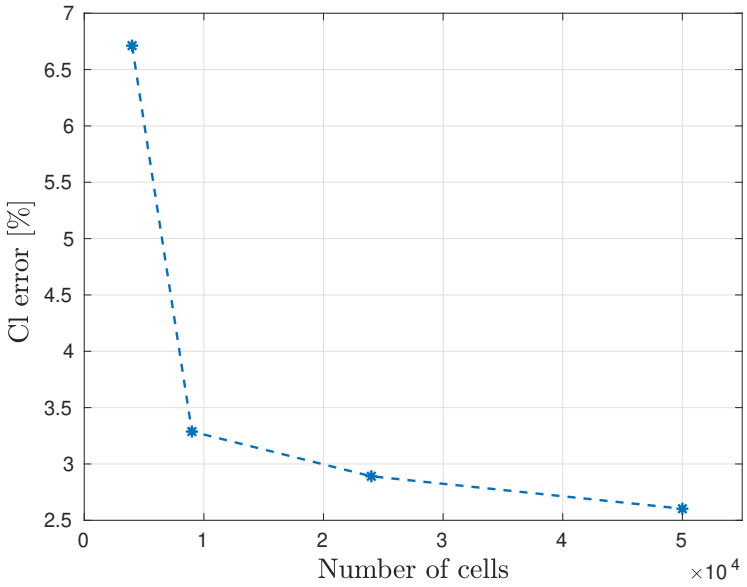


Figure 4.6: Lift coefficient error

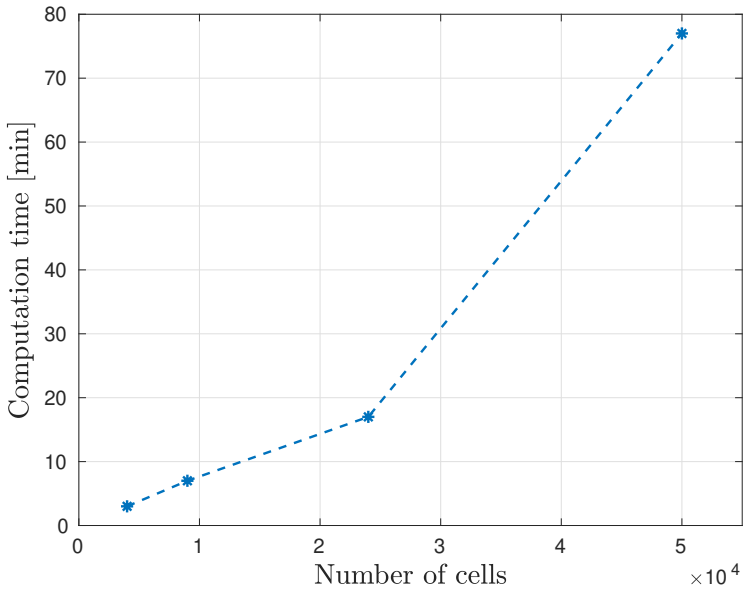


Figure 4.7: Computation time

Figure 4.8 shows that the error on the drag coefficient decreases with the increase of the number of elements. The main disadvantage associated with the finest grid is its computational time, as shown in Figure 4.7.

Making a trade-off between accuracy and computation time, the grid with 24000 elements has been chosen for the rest of this work. The results obtained with the selected grid are summarized Table 4.2.

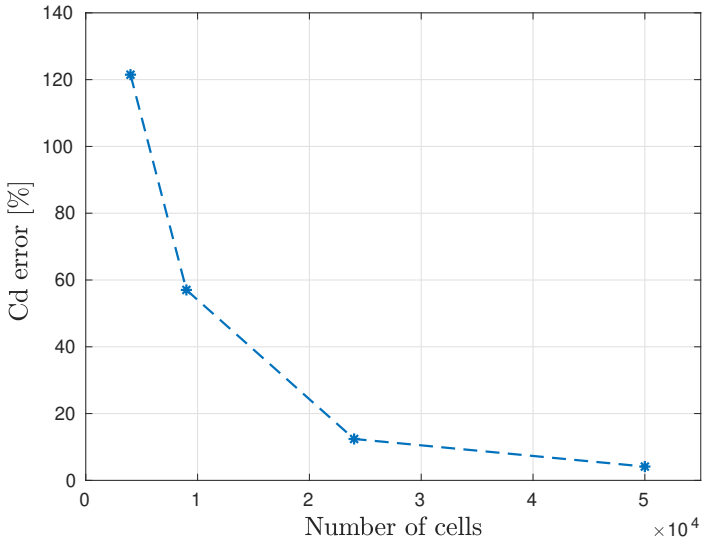


Figure 4.8: Drag coefficient error

Table 4.2: Force coefficients computed with OpenFoam

Coefficient	Value	Error [%]
C_l	0.7526	+2.955
C_d	0.01355	+11.80

The error on the drag coefficient is still significant, but since all the computations will be performed using the same grid, it has been assumed that approximately the same error will be introduced in the other computations. Figure 4.9 shows the pressure coefficient behaviour along the chord compared with the experimental solution. Since the behaviour follows almost exactly the experiment, the mesh selected has been kept for the remaining part of this work. In Figures 4.10 and 4.11 the Mach and pressure contours obtained with the numerical solution are shown.

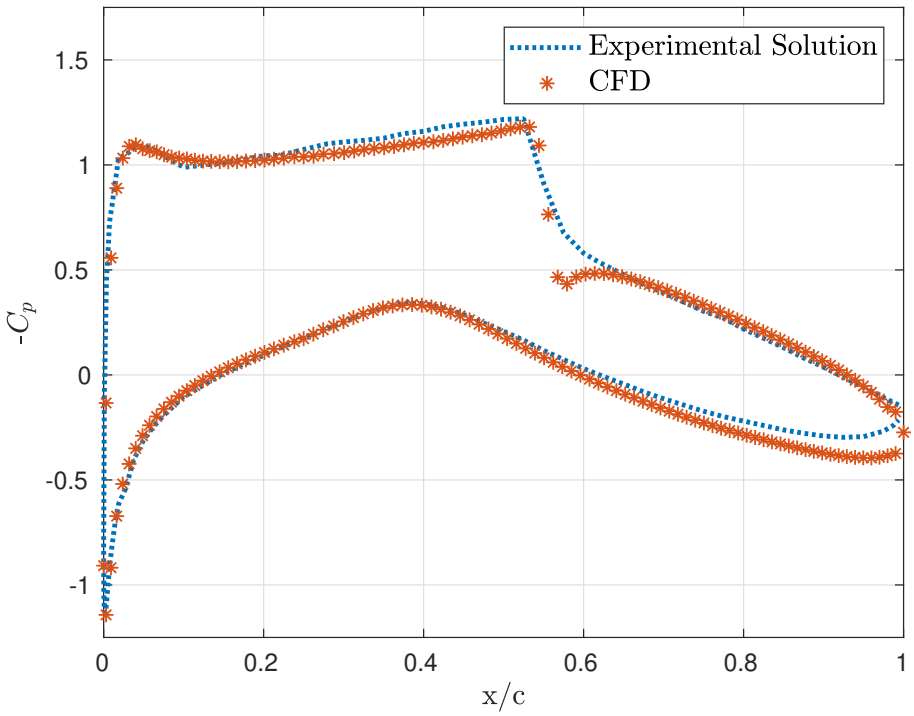


Figure 4.9: Pressure coefficient - comparison with reference solution

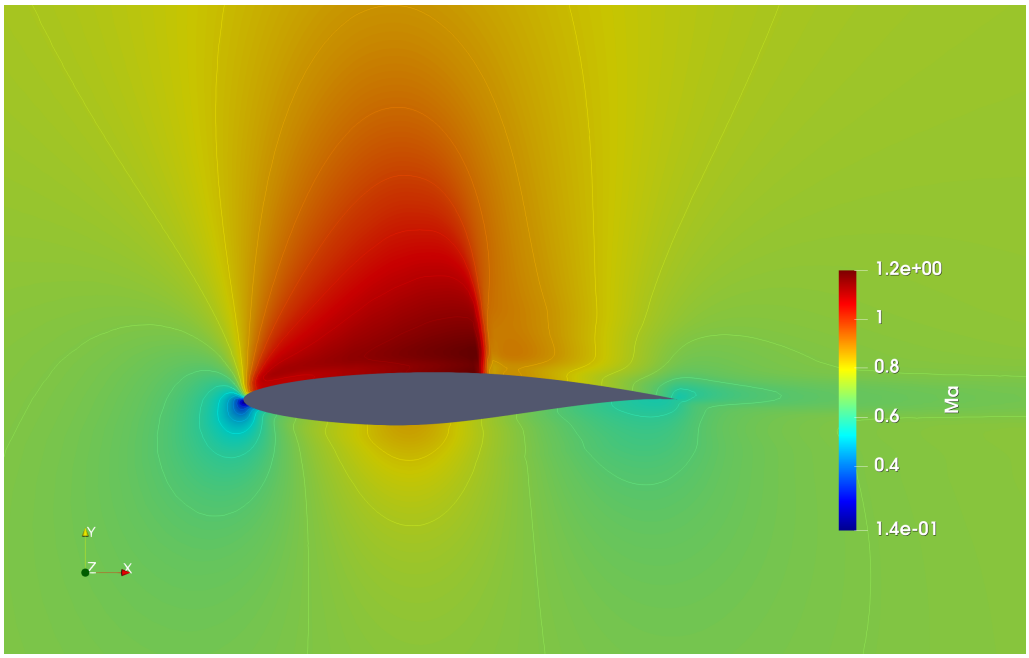


Figure 4.10: Mach contours

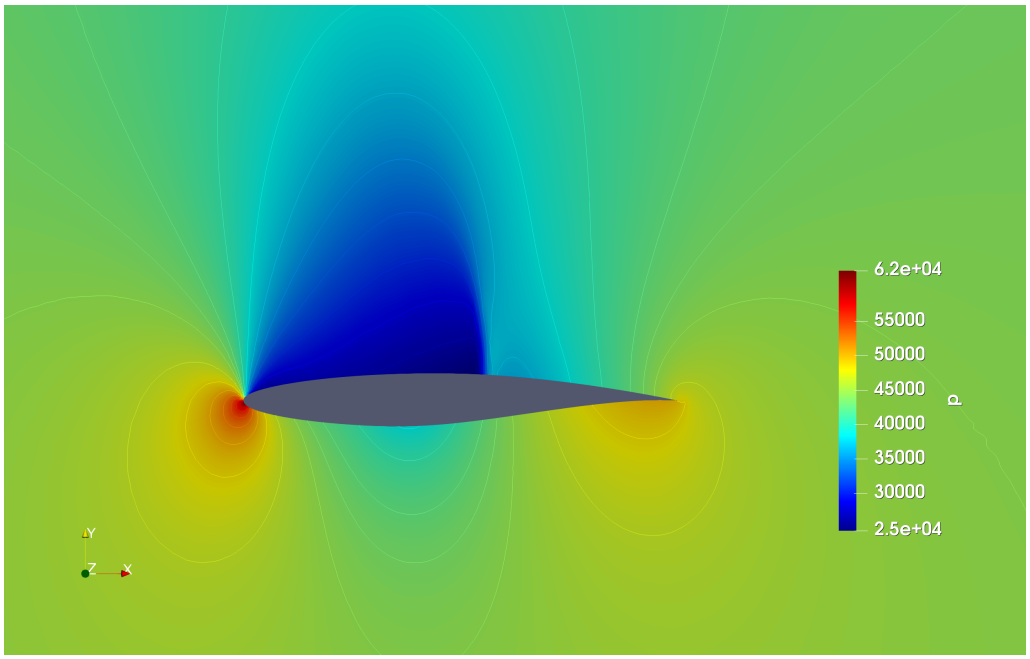


Figure 4.11: Pressure contours

5 | Data-Set Generation

In this chapter, the work behind the generation of an appropriate number of airfoil samples is presented, from the deformation of the airfoil and of the computational grid to the sampling type and the CFD simulation of each airfoil. The numerical values assigned to the parameters are given in Chapter 6.

5.1. Design Parameters

As mentioned above, two types of parametrizations have been tested. The first method consists in selecting an appropriate number of points N on the upper and lower surface of the airfoil and imposing a vertical displacement to those control points, as explained in Section 2.1.1. The design parameters of this method are the vertical displacement of the control points. Interpolating those displacements via Radial Basis Functions with the appropriate parameter σ allows obtaining the deformed shape of the airfoil.

The second method uses Hicks-Henne bump functions to deform the original surface of the airfoil. With this method, the design parameters are the values of the intensities a_i (see section 2.1.2) which, together with the bumps position x_i^M and their width w_i determine the final shape.

In both cases, mesh deformation is obtained using Radial Basis Functions interpolation. That algorithm will be explained in section 5.3

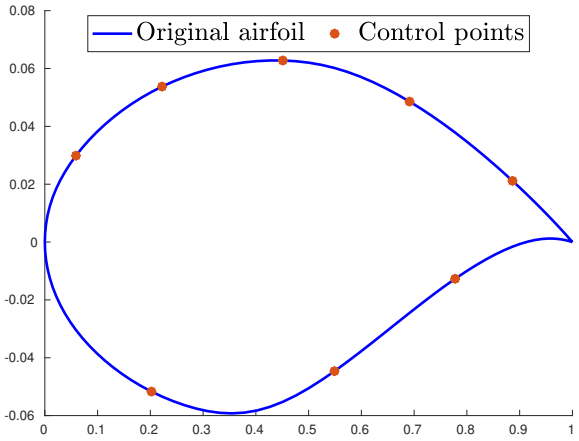


Figure 5.1: RBFs Control points - $N = 8$

5.2. Airfoil Deformation

The generation of the training data-set for the ANNs-based surrogate model requires the generation and the simulation of deformed airfoils. The next sections illustrate the two different parameterizations that have been tested in this work, explaining why it was necessary to introduce the Hicks-Henne functions deformation procedure.

5.2.1. Radial Basis Functions

The airfoil deformation based on RBFs requires the definition of the number of control points, their position, the smoothing parameter σ and the upper and lower bound for each control point. The learning task will be performed for different numbers of parameters nU, for each case the parameter σ is chosen to obtain a smooth surface of the airfoil since a "bumpy" airfoil could perform better than the original with these particular boundary conditions but would reasonably underperform the original with only a small change in the flow conditions, as stated by Painchaud-Ouellet et al. in [46]. By increasing this parameter, the influence of the motion of the control points enlarges, leading to a smoother deformation.

Figure 5.1 shows the original airfoil and an example of 8 control points, while Figure 5.2 shows the first nine airfoils generated with the Sobol sequence using those points. These airfoils have been generated limiting the maximum deformation of each control point to the $\pm 20\%$ of its y-coordinate.

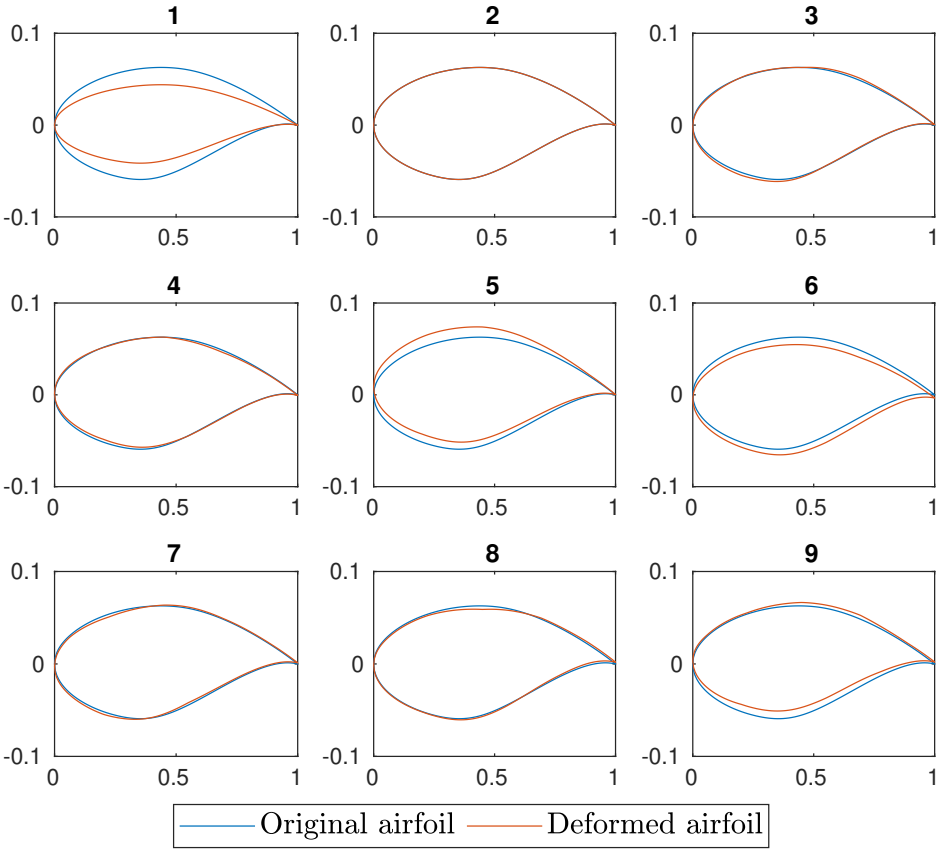
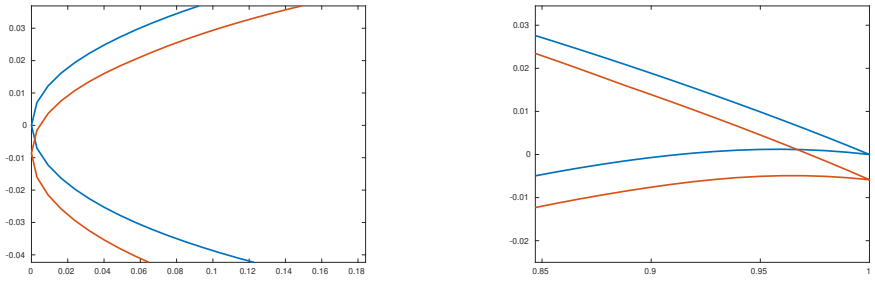


Figure 5.2: Deformed airfoils example



(a) Leading edge deformation (b) Trailing edge deformation

Figure 5.3: Deformation issues

The airfoils generated with this method seem to be sufficiently smooth, but the position of the leading and trailing edges is not fixed, as shown in Figure 5.3. This leads to a change in the angle of attack of up to 0.1°. This issue is strongly mitigated when the number of control points N is sufficiently high ($N \geq 25$) but in this work

such a high number of degrees of freedom can not be used for the computational cost associated with such a high dimensional design problem. For this reason, an alternative parametrization has been tested.

5.2.2. Hicks-Henne Functions

To generate the deformed airfoil samples with Hicks-Henne functions, it is necessary to define which of the parameters are fixed and which ones are varied. In this work, the bumps widths and positions defined in Equation (2.7) are fixed, while the bumps intensities will be used as design parameters and hence will be varied, according to the Sobol sequence, to obtain those samples. With this method, by properly varying the value of the width of the bumps, it is possible to control the smoothness of the airfoils.

For the case illustrated in Figures 5.4 and 5.5, the bump heights have been limited to the 10% of the airfoil's thickness, the bump widths w_i have been set to 2 for every $i = 1, ..., N$ and the x_i^M are the x coordinates of the points in Figure 5.4.

With this parametrization, the upper and lower surfaces of the airfoil are deformed independently. The leading and trailing edge are fixed for each choice of N. The bounds set in this case are on the maximum absolute value of the bump intensity a_i.

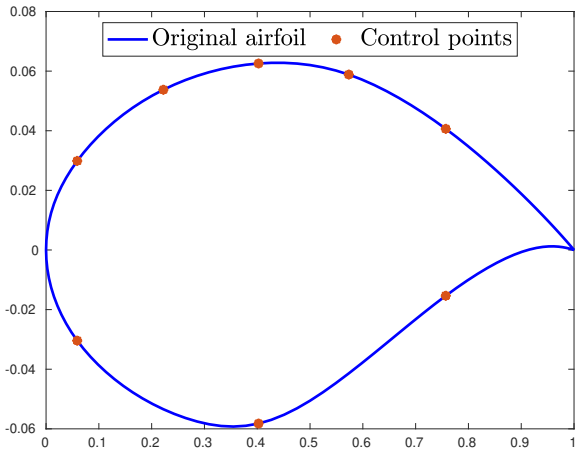


Figure 5.4: Control points - $N = 8$

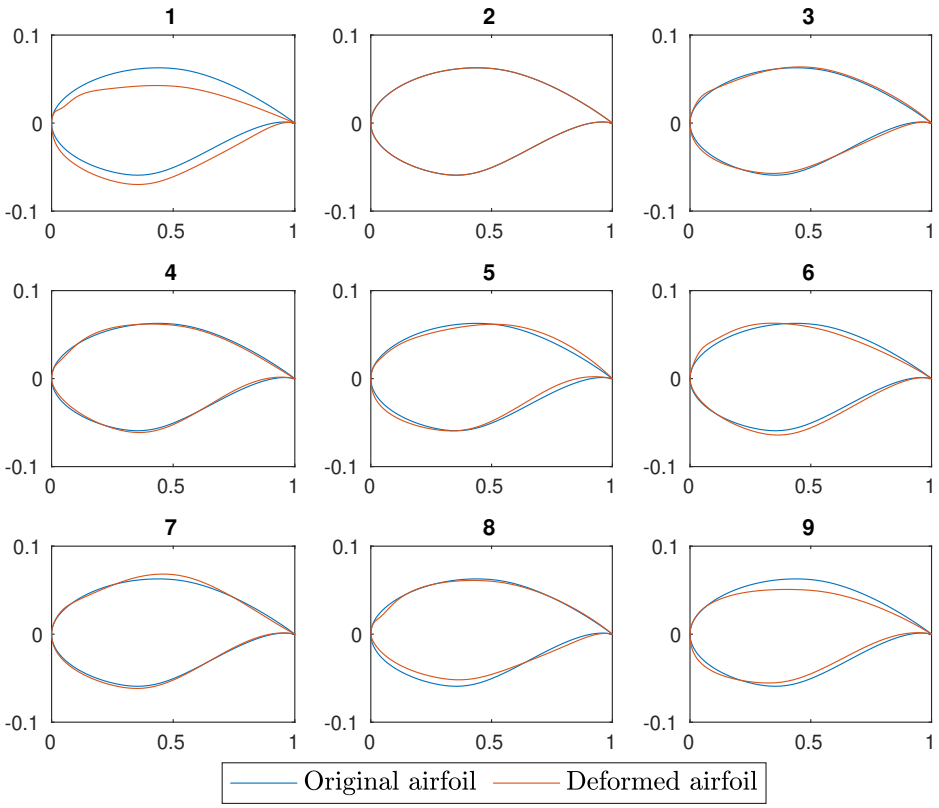


Figure 5.5: Deformed airfoils example

5.3. Mesh Deformation

The mesh deformation method explained in Section 2.1.1 was implemented as an algorithm called *deform* that has been built starting from the implementation given in [34], where only linear spline-type RBF was implemented. To increase the freedom in the smoothness of the deformed airfoil in the case of RBF airfoil deformation, the parameter σ was introduced. Furthermore, to impose that the mesh points near the outer boundary are not deformed, the damping term in equation (5.1) has been added.

$$f_{\text{damp}}(r) = \begin{cases} 1 & r \leqslant R_1 \\ 1 - \frac{r-R_1}{R_2-R_1} & R_1 < r \leqslant R_2 \\ 0 & r > R_2 \end{cases} \tag{5.1}$$

The *deform* algorithm requires as input the initial position of the control points, the complete list of grid points together with their coordinates, and the vector $\bar{\mathbf{s}}$ of

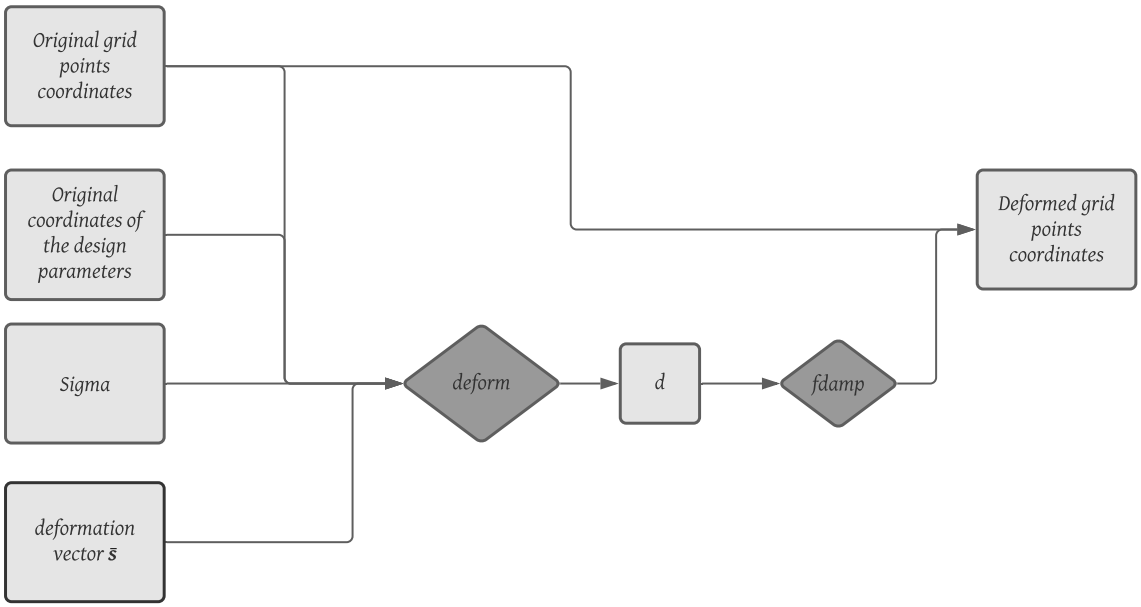


Figure 5.6: Mesh deformation algorithm flow chart

vertical displacements of the control points. This function returns a deformation vector d for each grid point. Once the vector d is multiplied by the f_{damp} function, it is summed to the original vector of grid points to obtain the deformed mesh associated with the displacements $\bar{\mathbf{s}}$.

The complete algorithm is depicted in the flow chart 5.6. Once the mesh has been deformed, its quality is assessed through the pre-processing tool *CheckMesh*, looking in particular at the maximum skewness and mesh non orthogonality. With all the cases tested, both the linear and gaussian global RBFs give satisfactory results, with this values showing moderate changes compared to the base grid.

Figures 5.7 and 5.8 (a) show the comparison between the original and the deformed grid points, associated with the airfoil deformation shown in Figure 5.8 (b).

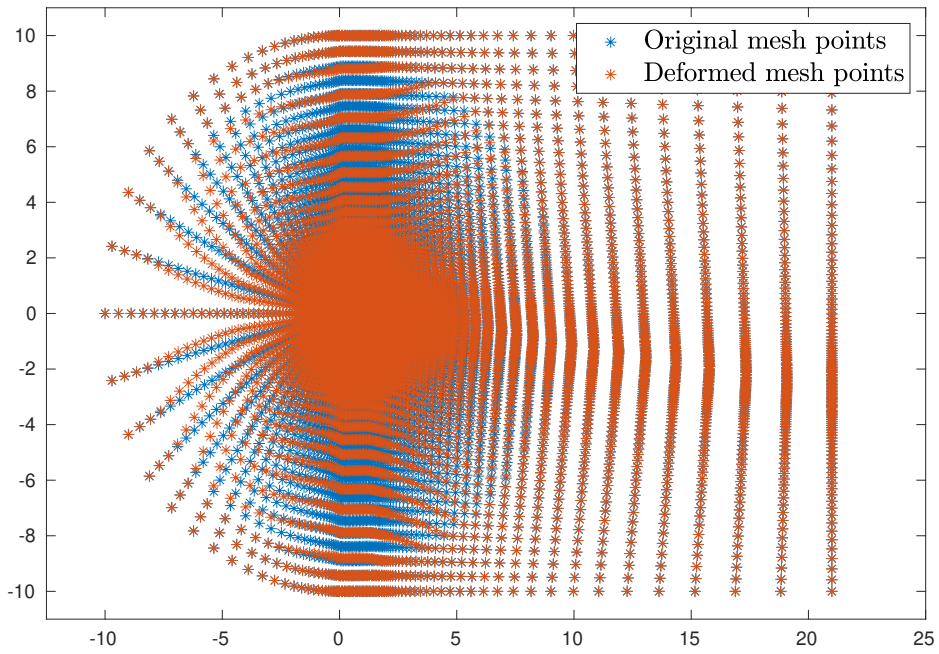
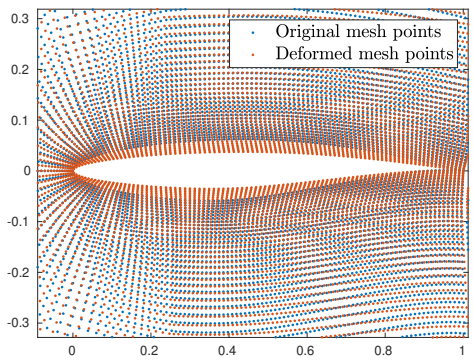
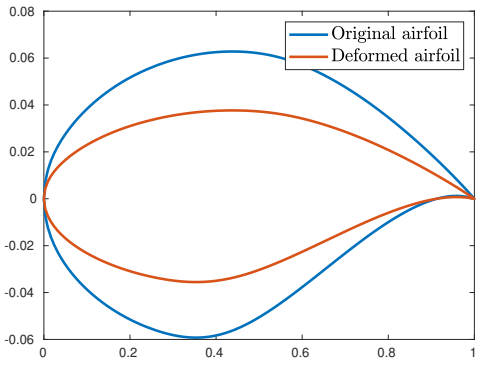


Figure 5.7: Mesh points before and after deformation



(a) Mesh deformation near the airfoil



(b) Original and deformed airfoil

Figure 5.8: Original and deformed airfoil

5.3.1. Sampling Procedure

The sampling procedure is a crucial choice in the design of the experiment, a bad sequence could drastically increase the number of simulations required in the learning phase. This could cause a *bottleneck*, i.e. to get an accurate surrogate model using the machine learning algorithm, one would need to use a large number of training samples making the training problem computationally expensive [43].

As previously explained in section 2.3, low discrepancy sequences such as Sobol [59] or Halton [26] can help in keeping low the number of samples. In this work, the Sobol sequence, which is implemented in *Matlab* [62] with the function *sobolset*, has been chosen. The y-coordinate of each design point is deformed according to the Sobol sequence, choosing in each test the most appropriate upper and lower bound for each design variable.

5.3.2. Sampling Algorithm

To ease the generation of samples, a bash script has been created that automates the generation of samples and their computation, taking as only input the number of samples to be generated. This *Create_samples.sh* algorithm performs also some post-processing tasks, such as the mean of the force coefficients on the last iterations, the generation of the pressure coefficient along the chord, the acquisition of flowfield screenshots through a python script for *Paraview*.

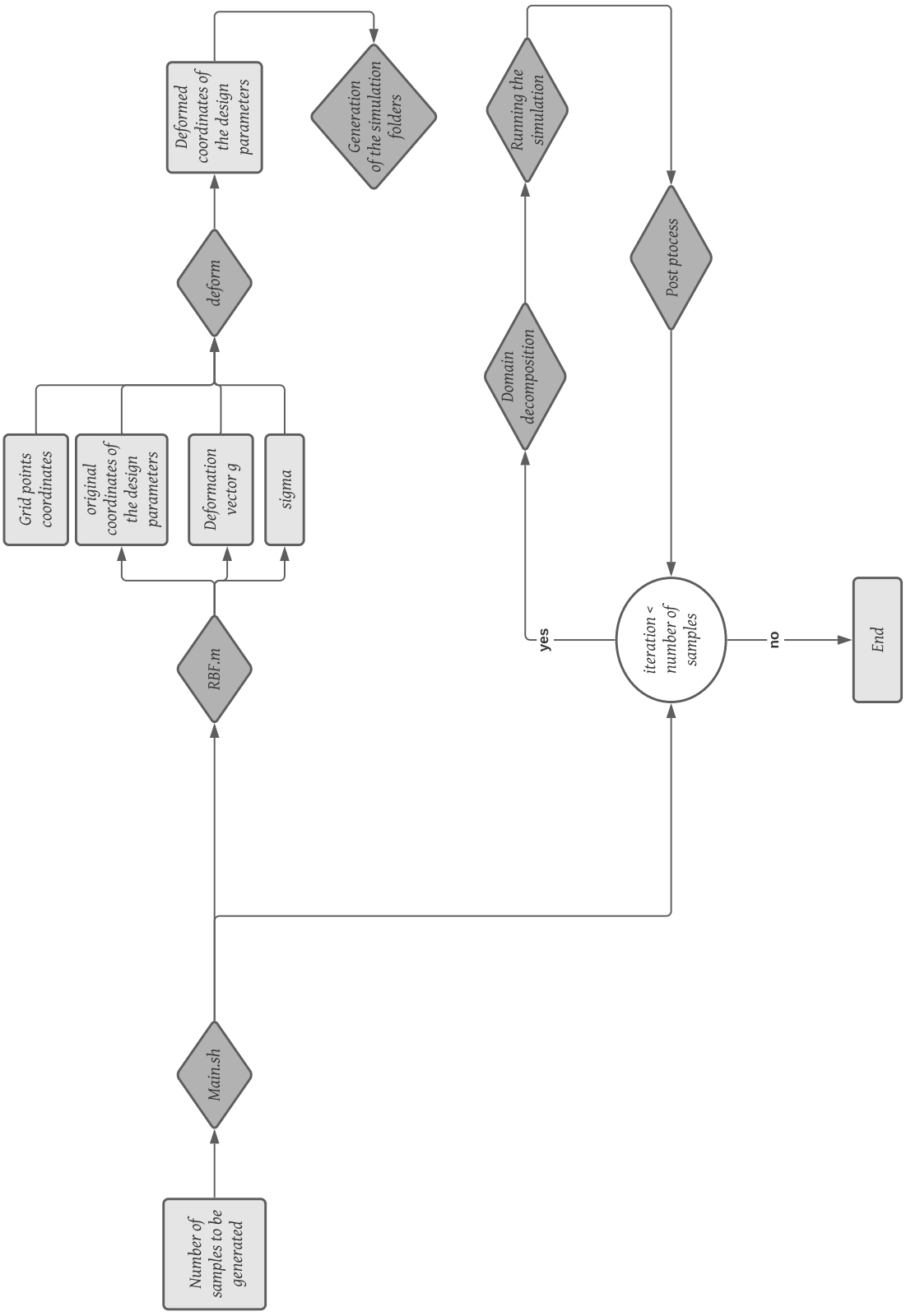


Figure 5.9: Create_samples.sh algorithm

6 | Machine Learning and CFD

6.1. Model-Learning

In this work, the Machine Learning algorithms are taken from the work of Regazzoni et al. in [52] and [53], implemented in the *Model-Learning* repository by F. Regazzoni [51].

The purpose of Regazzoni et al. in [52] is to find a data-driven Model Order Reduction based on ANNs which applies to dynamical systems arising from Ordinary Differential Equations or time-dependent Partial Differential Equations.

The models generated have been shown to approximate every time-dependent model described by ODEs with any desired level of accuracy. These data-driven reduced-order models, thanks to their black-box nature, can be applied when the state of the high fidelity model is unknown and/or one may not be interested in the explicit definition of a reduced model. These models could also be used in a system identification problem where nor the high fidelity model F nor the full-order state X is known, but only input-output pairs.

In this work, time dependence is not considered for computational limitations.

6.1.1. Building a Surrogate Model

This section derives from the adaptation of [52] to a steady problem. The HF model (in this case the CFD solver) can be seen as a map $\boldsymbol{\varphi} : \mathcal{U} \longrightarrow \mathcal{Y}$ from the space of input signals (the design variables) $U = \mathbb{R}^{N_u}$ to the space of output signals $Y = \mathbb{R}^{N_y}$, which in this work is the space of the force coefficients. To build the surrogate model, N_s simulations are performed with the HF model, collecting a set of input-output pairs:

$$\{(\widehat{\mathbf{u}}_j, \widehat{\mathbf{y}}_j)\}_{j=1,\dots,N_s} \subset \mathcal{U} \times \mathcal{Y} \tag{6.1}$$

Then after selecting a subset of candidate models, $\widehat{\Phi} \subseteq \Phi$ the best-approximation problem with respect to the HF model is defined in the least square sense:

$$\boldsymbol{\varphi}^* = \underset{\boldsymbol{\varphi}\in\widehat{\Phi}}{\operatorname{argmin}}\ J(\boldsymbol{\varphi}) \tag{6.2}$$

Where the loss function is defined as the squared distance between estimated and true output:

$$J(\boldsymbol{\varphi}) = \frac{1}{2}\sum_{j=1}^{N_s} \left|\widehat{\mathbf{y}}_j - (\boldsymbol{\varphi}\widehat{\mathbf{u}}_j)\right|^2 . \tag{6.3}$$

6.1.2. Network Optimization Strategy

The surrogate model is created by means of ANNs, structured in a Feed Forward Neural Network, where each neuron is connected to each neuron of the next layer, such as the one presented in Figure 6.1. The input neurons represent the value of the design variable, the output neurons represent the values of the lift and drag coefficients.

A feed-forward neural network represents a class of functions written as:

$$\mathbf{f}(\mathbf{p};\boldsymbol{\mu}) = \mathrm{W}_{n_L-1} f_{\text{act}}\ (\ldots \mathrm{W}_2 f_{\text{act}}\ (\mathrm{W}_1\mathbf{p} - \vartheta_1) - \vartheta_2 \ldots) - \vartheta_{n_L-1} \tag{6.4}$$

Where $\mathbf{p} \in \mathbb{R}^{N_u}$ is the input vector and $\boldsymbol{\mu} \in \mathbb{R}^{N_f}$ is the parameters vector, collecting weights W_i and biases ϑ_i. The activation function adopted is the hyperbolic tangent. The optimization is made by the *Levenberg-Marquardt* method, which finds the descent direction as a combination of the gradient descent direction with the Gauss-Newton direction, which is an approximation of the Newton direction obtained by neglecting the quadratic term in the computation of the Hessian [52].

Thanks to the possibility to compare the $\mathcal{L}^2$ error on the train and test set during the computation, it is possible to prevent overfitting by stopping the learning process as soon as, while the error on the train set keeps decreasing, the error on the test set starts increasing. In this work, the test set has been used to control overfitting by early stopping and to choose the network architecture in terms of the number of hidden neurons, whereas the cross-validation set has been used to estimate the generalization error made by the surrogate model.

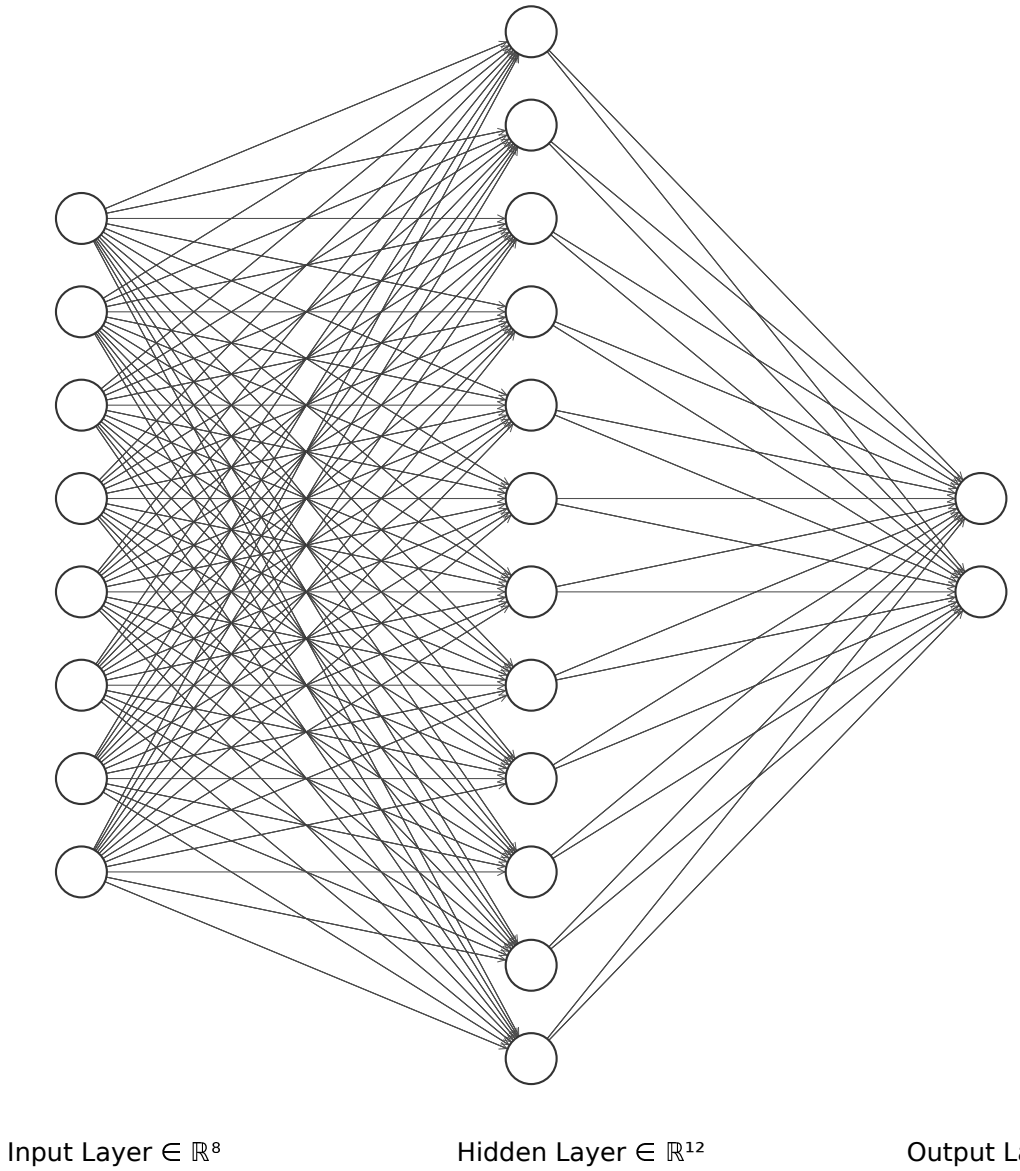


Figure 6.1: Example of feed forward ANN

Normalization

To increase the learning performance, before starting the training phase, all inputs have been normalized so that they assume values in the range $[-1,\ 1]$.

6.1.3. Original Contributions

The model-learning library is thought for time-dependent problems and hence to use it for this work it needed to be modified to focus on a steady solution. For further information on the original codes, the reader is redirected to the original library documentation that can be found at `https://model-learning.readthedocs.io/`

`en/latest/`.

To reduce at most the implementation complexity, the input and output variables have been given to the learning machine as constant signals on a time-domain $t \in [0, 1]$, and the loss function has been modified to obtain a network that predicts the correct values at the latest time step. This modification has been performed in the *model_learn.m* script. The resulting code learns to predict the lift and drag values at the latest time and, as a consequence, the following algorithms will consider the output of the network only at that time step.

6.2. Surrogate Model Generation

In this section, the results obtained with the ANNs-based surrogate model training will be discussed for both the parametrizations tested. For each parameterization, the initial number of degrees of freedom has been set to two, i.e. one design parameter on the upper and one on the lower surface of the airfoil. This choice allowed us to verify that the modifications presented in Section 6.1.3 succeed in obtaining a suitable surrogate model.

The results displayed here start from five degrees of freedom since this is the minimum number of design parameters for which the deformation complexity has been considered sufficient.

For all the cases here presented, 70% of the samples have been used as train-set, while the remaining 30% has been equally split into test and validation sets. The learning phase has been performed changing the number of hidden neurons, selected within a suitable range outside of which the results are not compatible with the following tasks. Once the networks have been trained, the one that performs best on the test set is chosen.

We are presenting here four different tests, two of them are obtained with Radial Basis Functions and the remaining two are generated with Hicks-Henne functions.

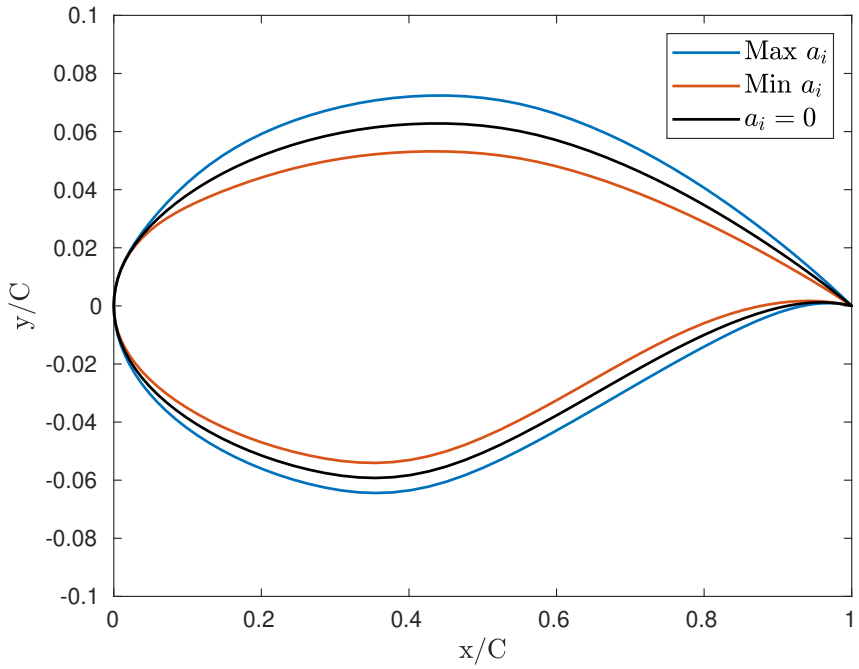
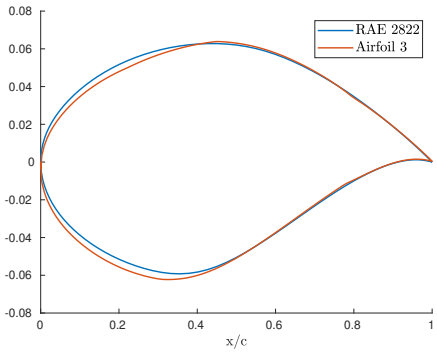


Figure 6.2: Design space - Test 1

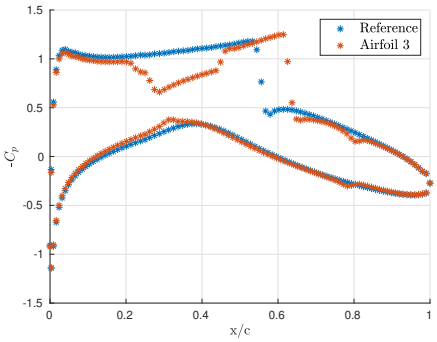
6.2.1. Test 1

The Test 1 refers to a set of 100 samples and five design parameters, where the shape deformation is performed with RBFs. The design parameters are points on the airfoil surface taken at the 20%, 45%, 70% of the chord on the suction side and at the 30% and 60% of the chord on the pressure side. To these points it has been assigned a deformation between the ±15% of the y-coordinate of the original airfoil point, the value of each parameter in each sample is given according to the Sobol sequence. Figure 6.2 illustrates the design space.

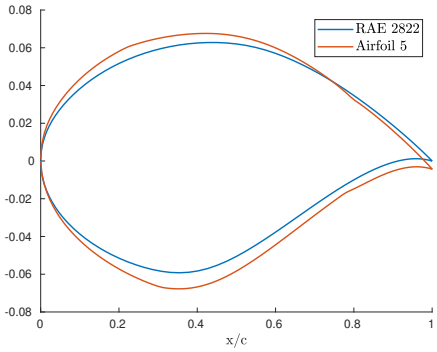
Figure 6.3 (a-h) shows the behaviour of the pressure coefficient along the chord for four sample airfoils, compared to the reference simulation. From the on the left it can be seen that the deformed airfoil show some irregularities. These cause the jumps in the pressure coefficients that can be seen in all the figures on the right. These irregularities are harmful to the performance of the airfoils and should be avoided.



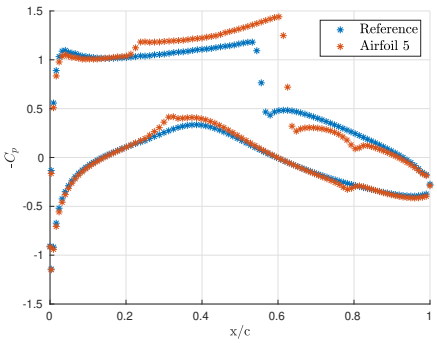
(a) Shape comparison - Airfoil 3



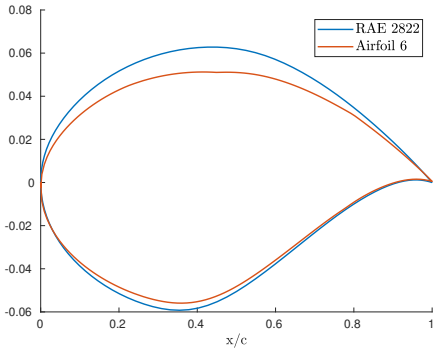
(b) Cp comparison - Airfoil 3



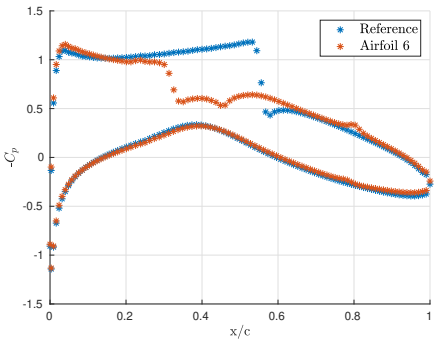
(c) Shape comparison - Airfoil 5



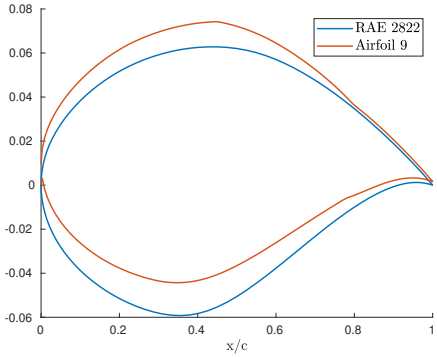
(d) Cp comparison - Airfoil 5



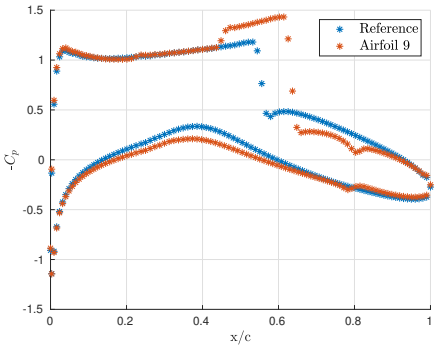
(e) Shape comparison - Airfoil 6



(f) Cp comparison - Airfoil 6



(g) Shape comparison - Airfoil 9

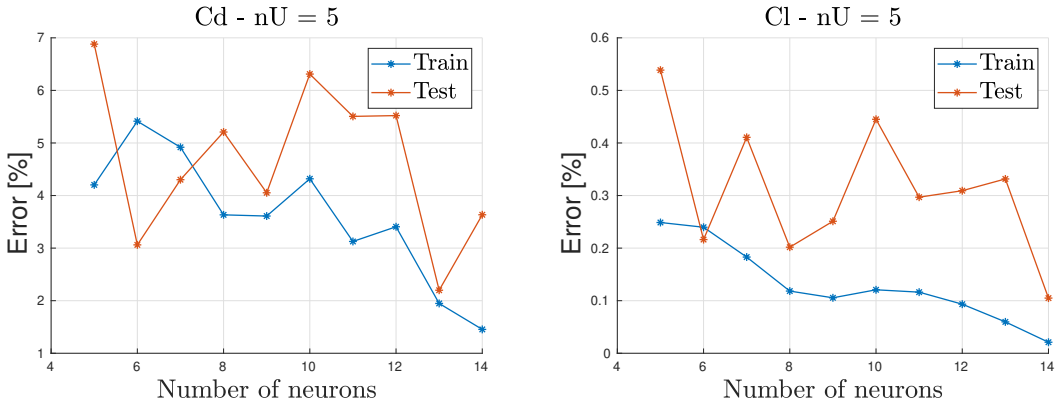


(h) Cp comparison - Airfoil 9

Figure 6.3: Airfoils shapes and pressure coefficients comparison - Test 1

Table 6.1: Learning errors - 13 neurons

-	ε_{Train} [%]	ε_{Test} [%]	ε_{CV} [%]	ε_G [%]
C_l	0.0597	0.332	0.346	0.286
C_d	1.947	2.198	2.307	0.360



(a) Drag coefficient RMSE

(b) Lift coefficient RMSE

Figure 6.4: Learning errors - Test 1

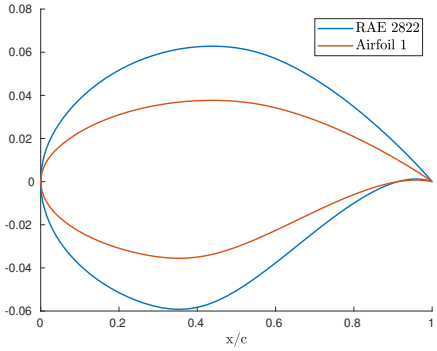
Figure 6.4 (a-b) shows the force coefficients root mean squared error (RMSE) on the train and test sets with the number of hidden neurons.

Figure 6.4 (a) and (b), shows that the error on the lift coefficient is always lower than the error on the drag coefficient. Furthermore, the lift coefficient shows some overfitting, as there is a distance between the errors on the train and test set of almost one order of magnitude also in the best architecture, which is the one with thirteen hidden neurons. The numerical values of the RMSE are reported in Table 6.1, where ε_{Train} is the RMSE on the train set, ε_{Test} is the RMSE on the test set, ε_{CV} is the RMSE on the cross-validation set, ε_G is an estimate of the percentage generalization gap, which is the difference between ε_{CV} and ε_{Train}.

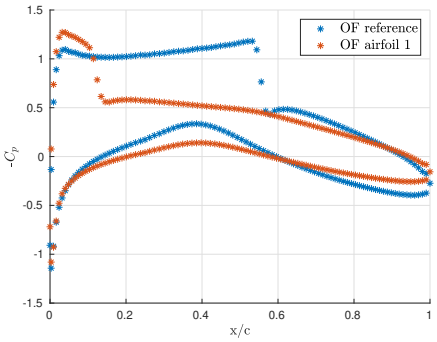
Test 2

This section presents the learning results related a RBFs-based parameterization with eight design parameters. The position of the control points is shown in Figure 5.1.

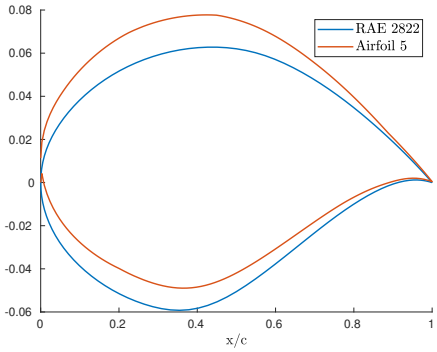
Figure 6.5 shows the first four airfoils together with the behaviour of the pressure coefficient along the chord on four selected airfoils.



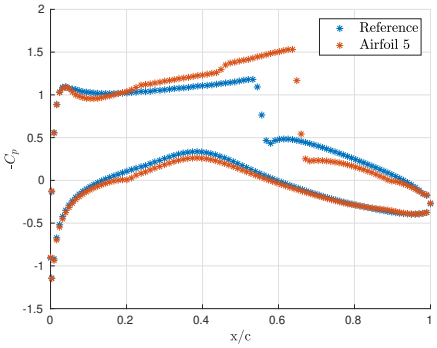
(a) Shape comparison - Airfoil 1



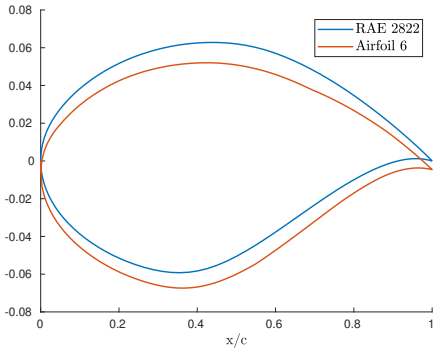
(b) Cp comparison - Airfoil 1



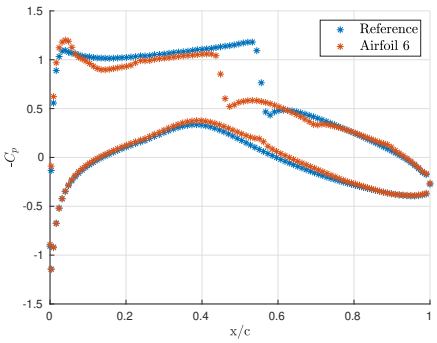
(c) Shape comparison - Airfoil 5



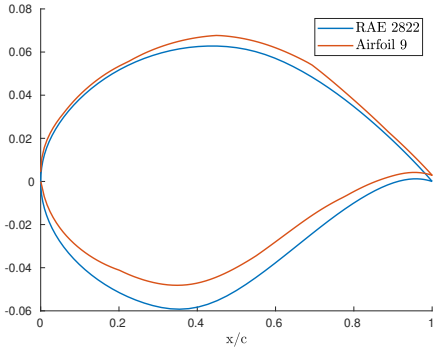
(d) Cp comparison - Airfoil 5



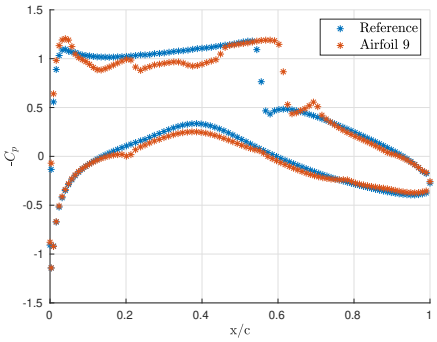
(e) Shape comparison - Airfoil 6



(f) Cp comparison - Airfoil 6



(g) Shape comparison - Airfoil 9

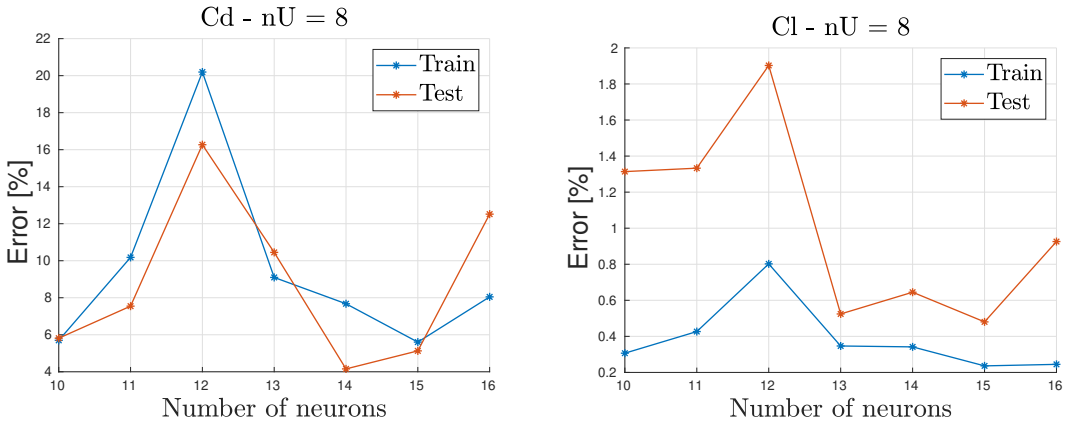


(h) Cp comparison - Airfoil 9

Figure 6.5: Airfoils shapes and pressure coefficients comparison - Test 2

Table 6.2: Learning errors - 15 neurons

-	ε_{Train} [%]	ε_{Test} [%]	ε_{CV} [%]	ε_{G} [%]
C_l	0.2362	0.4803	0.5313	0.2951
C_d	5.129	5.603	5.734	0.605



(a) Drag coefficient RRMSE (b) Lift coefficient RRMSE

Figure 6.6: Learning errors - Test 2

From Figure 6.6 it can be seen that the shape deformation succeeds in changing the flow behaviour, but the pressure coefficient plots show several oscillations that could be avoided with a better parametrization.

The increase in number of the parameters, leads to an increase in the number of samples needed to obtain an accurate ANNs model. The learning results displayed below are related to a total number of samples N_s equal to 300. Table 6.2 gives the numerical values of the errors made by the 15-neurons network. Despite the number of samples being three times higher with respect to the previous case, the errors made by the ANNs on these data is significantly higher. This is due to the increase in the dimension of the parameters space, this is a common problem in ML, often referred to as *the curse of dimensionality*.

6.2.2. Test 3

In this test, eight Hicks - Henne functions are used to parametrize the airfoil surface, five of which are set on the upper surface of the airfoil. The control points are shown in Figure 5.4, the bump intensities have been limited to the 15% of the thickness and the bump widths have been set to 2. Figure 6.7 shows the design space for this case, presenting the airfoils that correspond to the maximum and minimum values

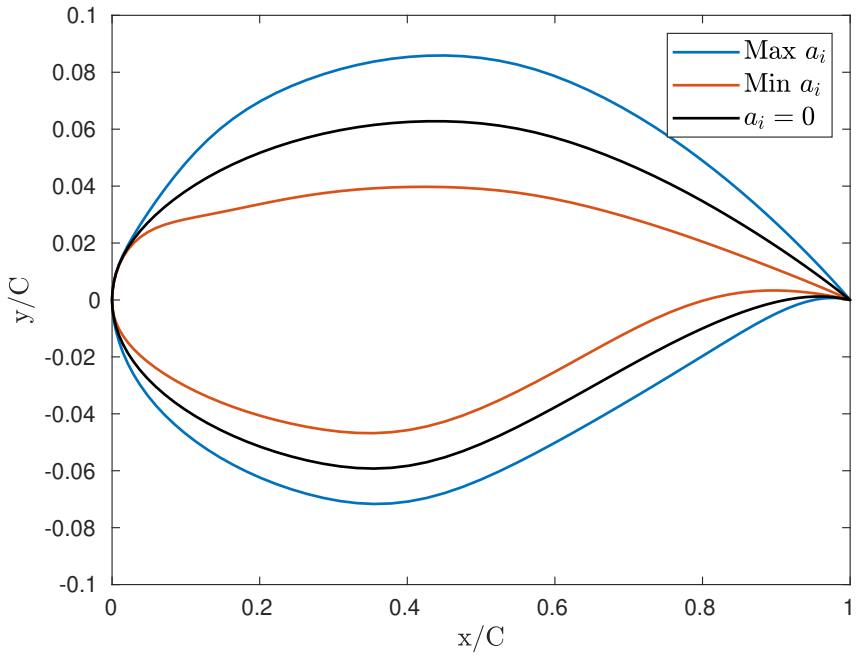
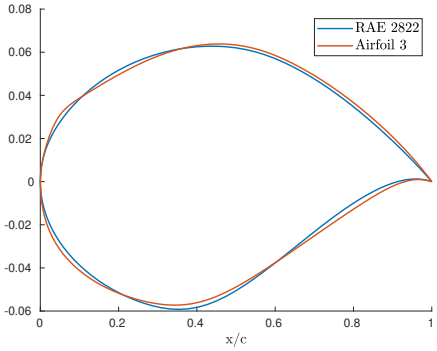


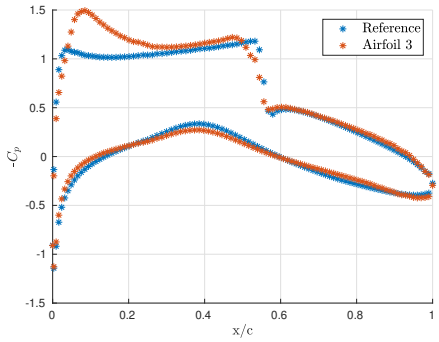
Figure 6.7: Design Space - Test 3

of the parameters a_i, together with the original airfoil, that corresponds to $a_i = 0$, for $i = 1, 2, ...8$.

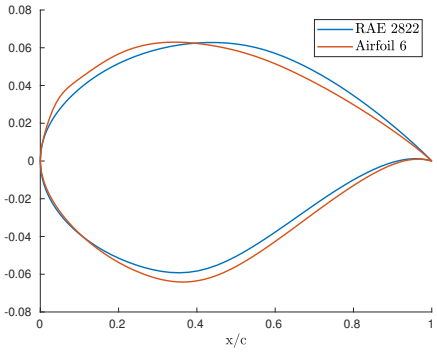
Figure 6.8 shows the results of the simulations on four airfoil samples together with their shapes. This parametrization allows to create a smoother tendency of the pressure coefficient plots with respect to the previous tests.



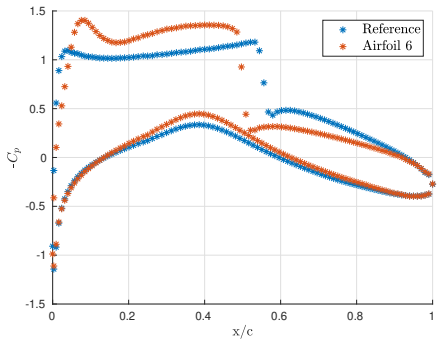
(a) Shape comparison - Airfoil 3



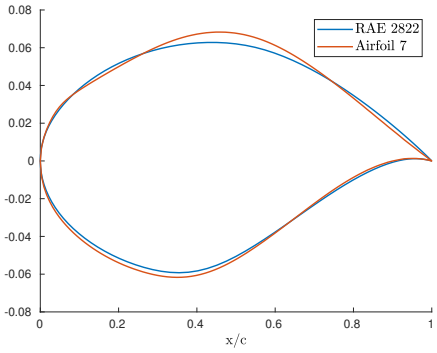
(b) Cp comparison - Airfoil 3



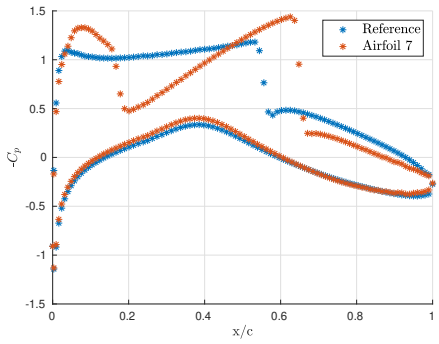
(c) Shape comparison - Airfoil 6



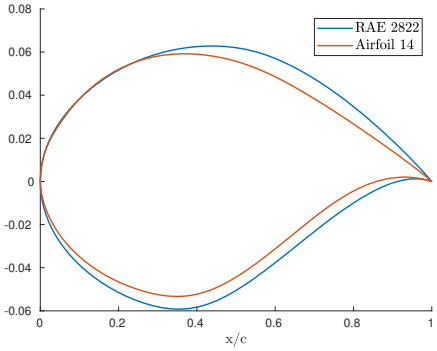
(d) Cp comparison - Airfoil 6



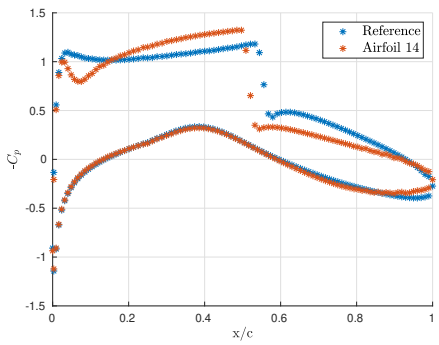
(e) Shape comparison - Airfoil 7



(f) Cp comparison - Airfoil 7

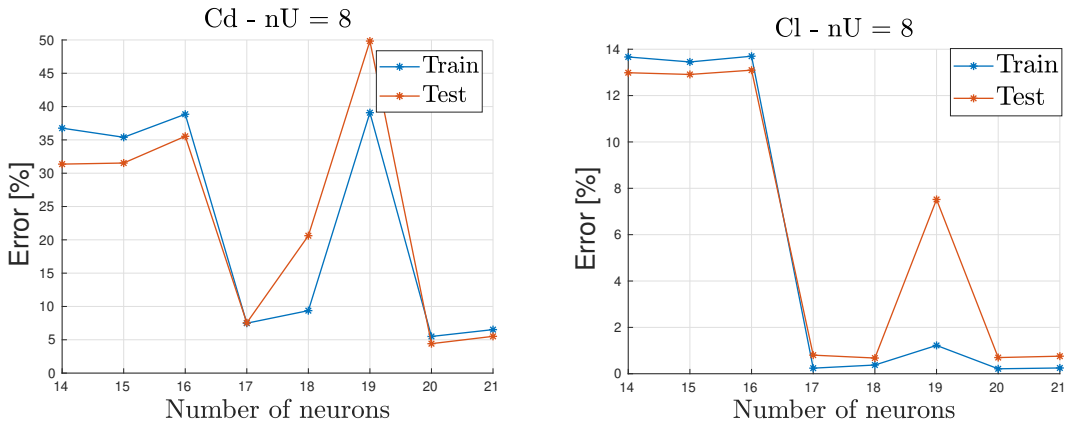


(g) Shape comparison - Airfoil 14



(h) Cp comparison - Airfoil 14

Figure 6.8: Airfoils shapes and pressure coefficients comparison - Test 3



(a) Drag coefficient RMSE

(b) Lift coefficient RMSE

Figure 6.9: Learning errors - Test 3

Table 6.3: Learning errors - 20 neurons

-	ε_{Train} [%]	ε_{Test} [%]	ε_{CV} [%]	ε_{G} [%]
C_l	0.2133	0.6779	0.7124	0.4991
C_d	4.416	5.498	5.507	1.091

The results discussed in this section have been obtained with 300 samples. The RMSE values related to the 20-neurons network are reported in Table 6.3. The errors made by this surrogate model are comparable with the ones made by the 15-neurons network presented in the Test 2 in Section 6.2.1.

Test 4

Since the learning results of the Test 3 did not improve significantly increasing the number of samples, to reduce further the number of samples required to learn a surrogate model, the maximum bump intensity has been reduced by the 60%. In this way, the parameters space has been significantly reduced and the learning task has been eased since, with the same number of samples, in this new space the points describing the parameters are denser. The number of samples generated in this case is 500. The computations required 135 computer hours.

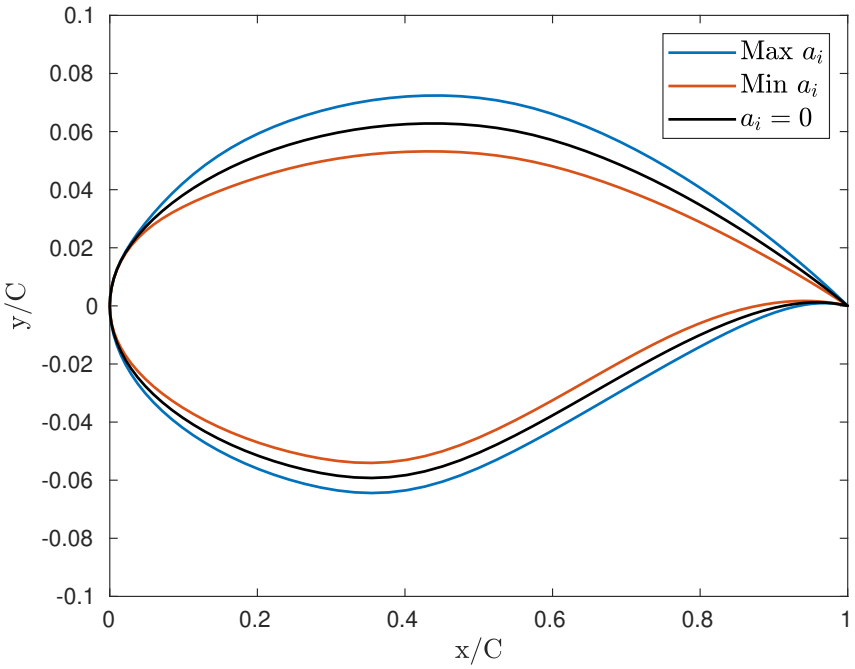
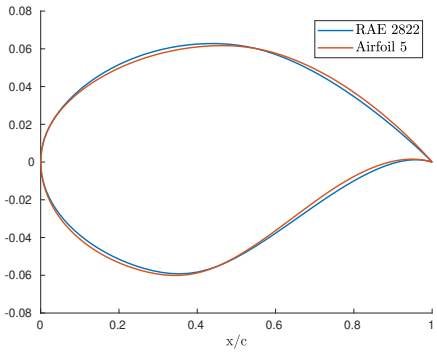


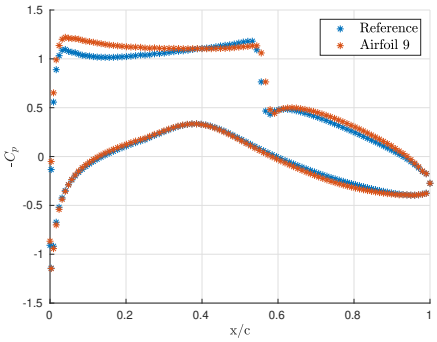
Figure 6.10: Design space - Test 4

Figure 6.10 shows the design space for this case, presenting the airfoils that correspond to the maximum and minimum values of the parameters a_i, together with the original airfoil, that corresponds to $a_i = 0$, for $i = 1, 2, ...8$.

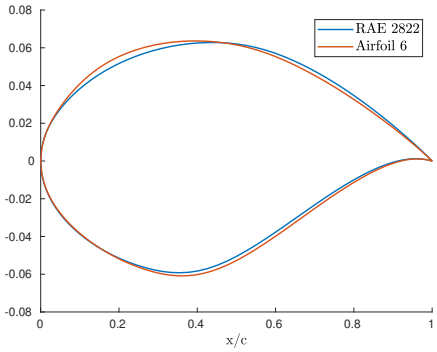
As it can be seen in Figure 6.11, with this parametrization the airfoil deformation is significantly reduced, and the shape of the pressure coefficient curve is more regular than all the ones shown previously.



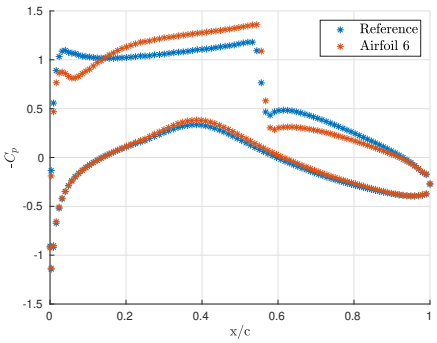
(a) Shape comparison - Airfoil 5



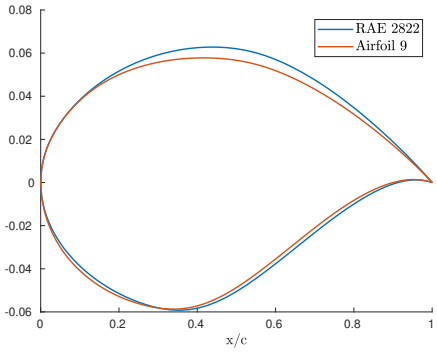
(b) Cp comparison - Airfoil 5



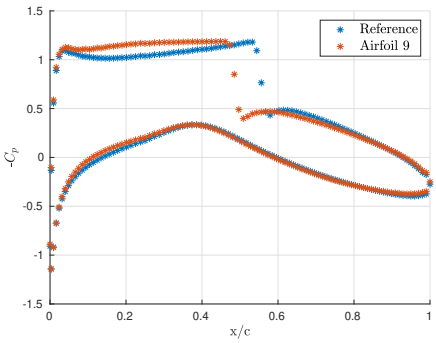
(c) Shape comparison - Airfoil 6



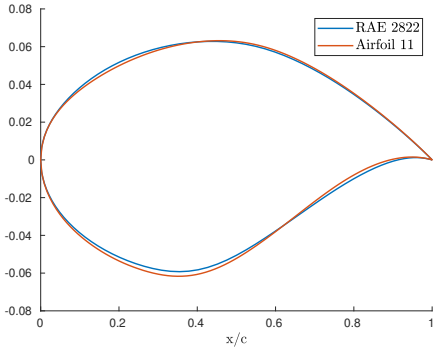
(d) Cp comparison - Airfoil 6



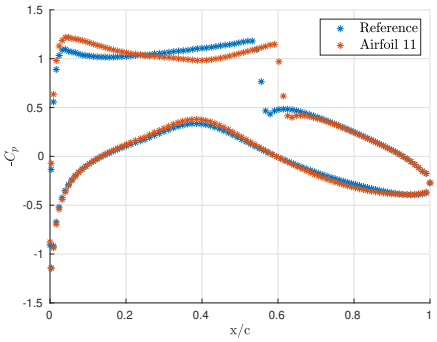
(e) Shape comparison - Airfoil 9



(f) Cp comparison - Airfoil 9



(g) Shape comparison - Airfoil 11

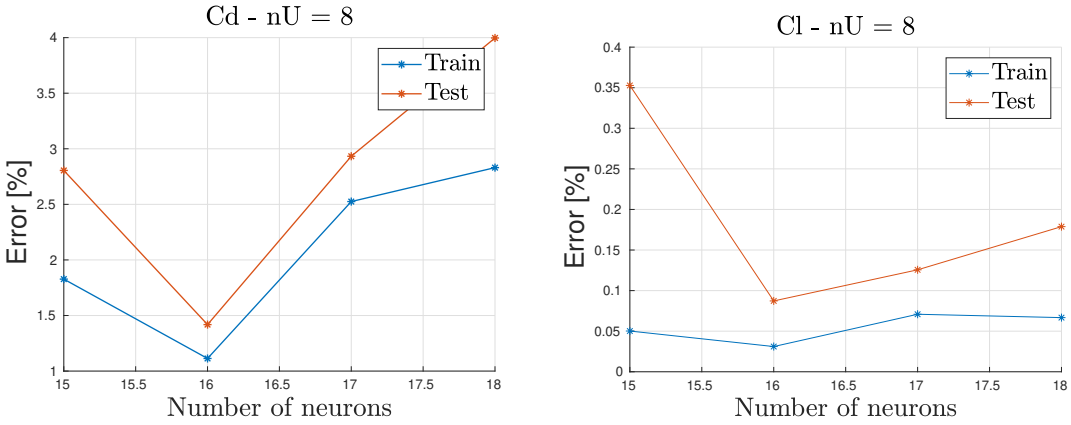


(h) Cp comparison - Airfoil 11

Figure 6.11: Airfoils shapes and pressure coefficients comparison - Test 4

Table 6.4: Learning errors - 16 neurons

-	ε_{Train} [%]	ε_{Test} [%]	ε_{CV} [%]	ε_{G} [%]
C_l	0.0310	0.0873	0.0879	0.0569
C_d	1.113	1.392	1.418	0.305



(a) Drag coefficient RMSE

(b) Lift coefficient RMSE

Figure 6.12: Learning errors - Test 4

Figure 6.12 shows the RMSE on the train and test with the number of hidden neurons.

The mean squared errors are far smaller than the previous tests, this can be explained considering that the design space has been significantly reduced and the number of samples is higher than all the tests previously shown. The results for the best network are summarized in Table 6.4.

7 | Shape Optimization of Airfoils

Once the surrogate model has been found, an optimization algorithm is be used to find the combination of input parameters that gives the best results in terms of some objective function.

7.1. Objective Function

For the surrogate model-based optimization, once the ANN has been successfully trained, the computational cost of shape optimization task is minimal, the optimization algorithm converges to the optimal solution within one minute of computations. This feature allows to perform optimizations with different Objective Functions at a negligible additional cost.

7.1.1. Aerodynamic Efficiency

One of the most common objective functions that can be considered is the aerodynamic efficiency, or lift-to-drag ratio. With this choice, the objective is to minimize this functional $O_1(C_l, C_d)$ to find an airfoil that shows a higher efficiency.

$$\mathcal{O}_1(E(C_l, C_d)) \;=\; \frac{E^{ref}}{E}. \tag{7.1}$$

Where $E = C_l/C_d$ is the aerodynamic efficiency, C_l and C_d are the force coefficients of the optimized airfoil, C_l^{ref} and C_d^{ref} are the force coefficients of the reference airfoil (see Table 4.2).

With reference to this particular test case, it is expected to find an airfoil that has a lower drag, due to the reduction or elimination of the shock wave on the suction side, at the cost of a decreased lift coefficient.

7.1.2. Drag Minimization at Fixed Lift

The final objective of this work is to perform a shape optimization of the airfoil to reduce drag at constant lift, the objective function should be defined accordingly, as in equation (7.2), which has been used by Mishra et al. in [39].

$$\mathcal{O}_2(C_l, C_d) = \frac{C_d}{C_d^{ref}} + \mathcal{P} \max(0,\ 0.999 - \frac{C_l}{C_l^{ref}}). \tag{7.2}$$

Where C_l and C_d are the force coefficients of the optimized airfoil, C_l^{ref} and C_d^{ref} are the force coefficients of the reference airfoil (see Table 4.2) and $\mathcal{P}$ is a penalization factor equal to 100.

The first part of the left-hand side of the equation is less than one when the sample airfoil has a lower drag coefficient compared to the reference, whereas the last term penalizes the airfoils that show a lower lift coefficient with respect to the original one, using the max function that takes the maximum value between the two terms inside the round brackets. In this way, when the lift coefficient of the new sample is greater or equal to the reference, the second term contribution to the objective function is null, whereas when this is not satisfied the value of the objective function is high and dominated by this penalization term.

The variation of the objective function with the aerodynamic coefficient is shown in Figures 7.1 and 7.2.

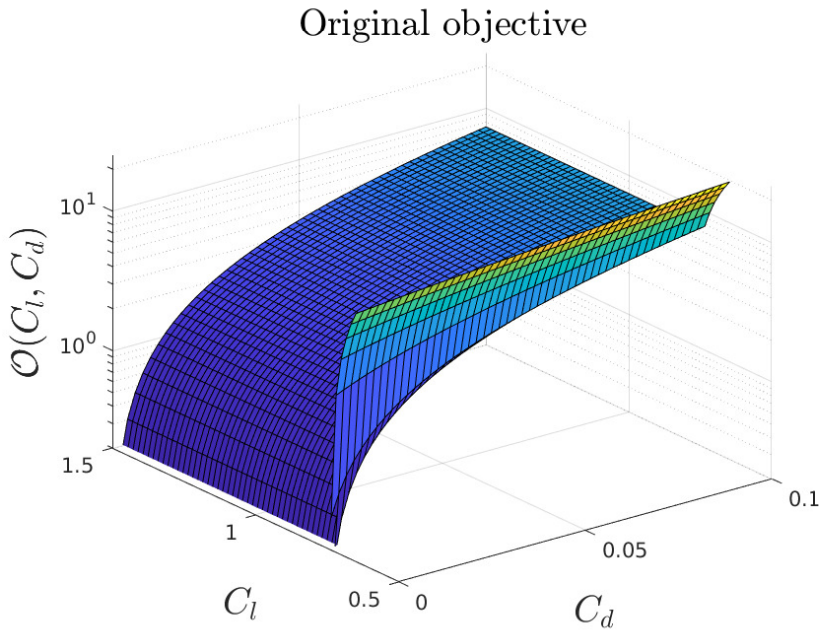


Figure 7.1: Objective Function

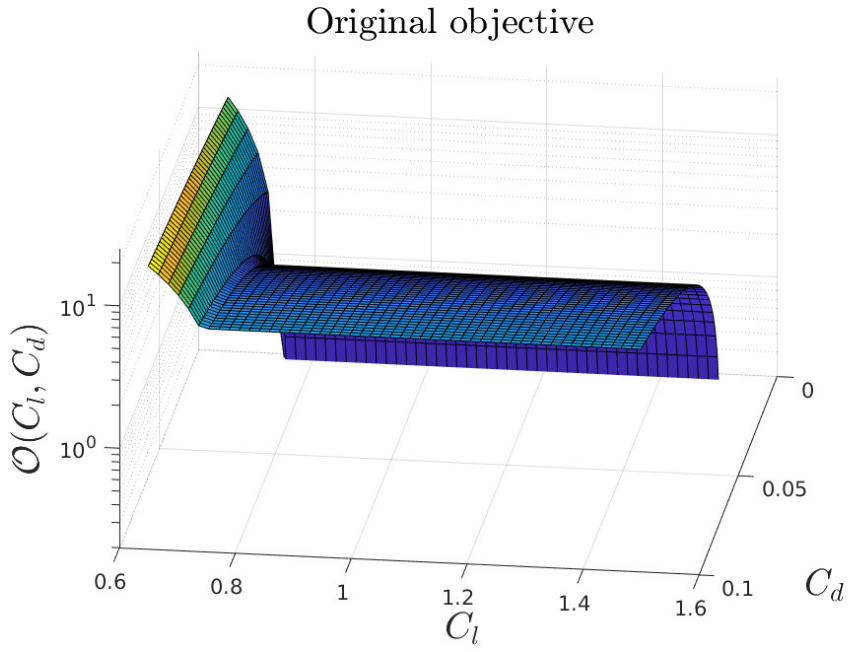


Figure 7.2: Objective Function

7.2. Optimization Strategy

The optimization is performed using the the interior-point algorithm, which solves a sequence of approximate minimization problems. It is described by Byrd et al. in

[8]. This algorithm is implemented in the Matlab [62] *fmincon* function, which is a gradient-based optimizer designed to work on problems where the objective function and its constraints are continuous and have continuous first derivatives.

Since the algorithm requires an objective function whose derivatives are continuous, and the function (7.2) has a discontinuous derivative along the line $C_l = C_l^{ref}$, the objective function has been modified to go along with *fmincon*. To obtain a smooth function, the penalization term has been defined using an exponential function, as follows:

$$\bar{\mathcal{O}}_2(C_l, C_d) \;=\; \frac{C_d}{C_d^{ref}} \;+\; \mathcal{P}\; e^{(-A\frac{C_l}{C_l^{ref}}+B)}. \tag{7.3}$$

The smoothed function $\bar{\mathcal{O}}_2$ obtained with $A = 50$, $B = 42$ is shown in Figures 7.3 and 7.4.

To provide the objective function to *fmincon*, it is necessary to write a Matlab function *obj_fun.m* that contains the selected artificial neural network. This function takes as inputs a vector of the network's input coordinates and returns as output the value of the objective function associated with those inputs. The aerodynamic force coefficients used inside this function are calculated by means of the ANN. The optimization has been run with the following parameters assigned to *fmincon*:

- Central finite differences for the gradient calculation;
- Initial point $U_0 = (0, ..., \ 0)^T$;
- $-1 \leq U_k \leq 1$ for every iteration k.

With U being the vector of the input variables, which are the rescaled values assigned to the design parameters.

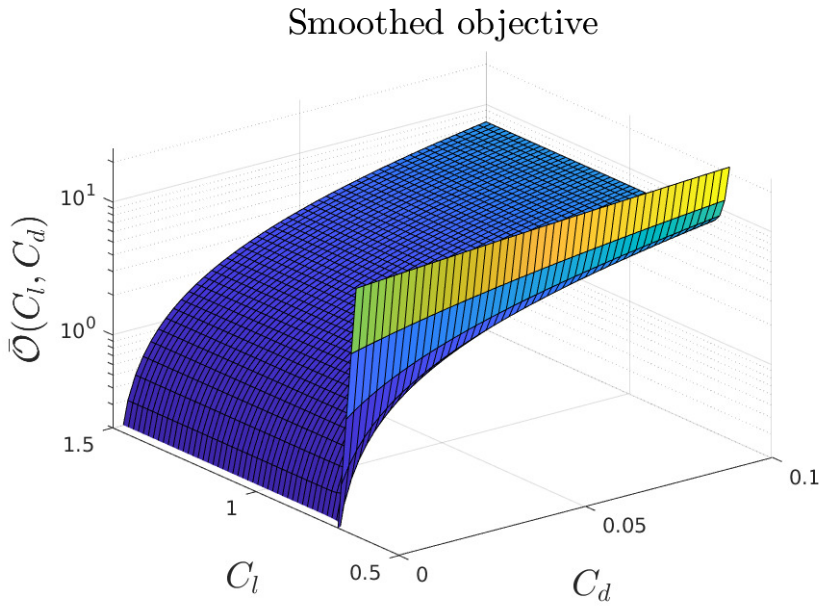


Figure 7.3: Smoothed objective function

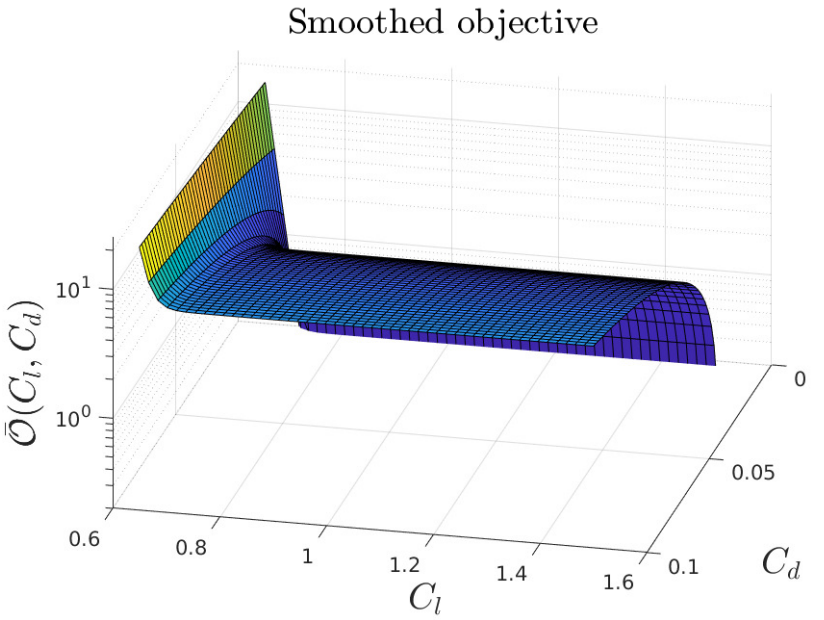


Figure 7.4: Smoothed objective function

Since the inputs have been re-scaled before the training phase, the ANN should be fed with values ranging into $[-1,\ 1]$ to provide accurate predictions of the force coefficients.

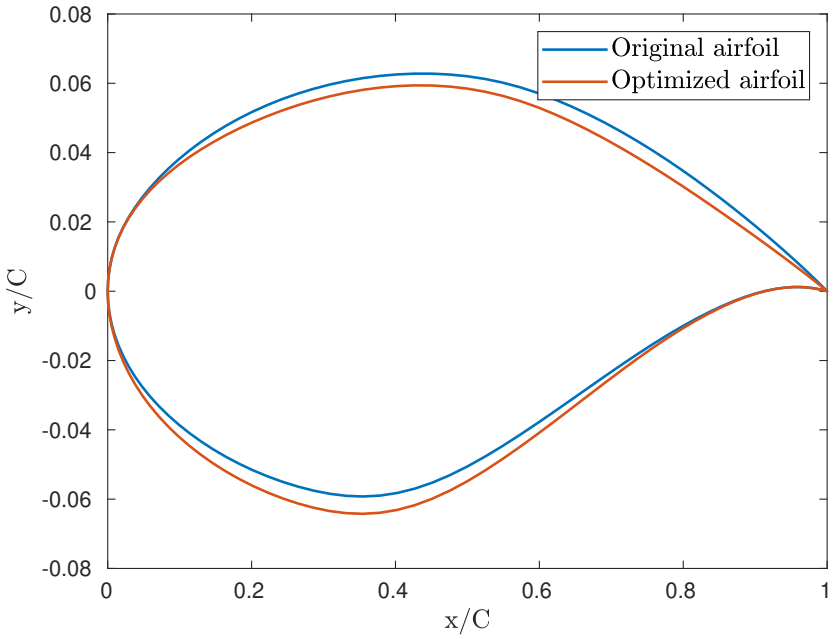


Figure 7.5: Optimized airfoil

7.3. Standard Surrogate Model Optimization

The optimizations presented in this section have been performed using the 16 neurons surrogate model presented in Section 6.2.2, the procedure associated with these results has been called Standard Surrogate Model Optimization and it consists in the usage of a unique ANNs-based surrogate model for each optimization.

7.3.1. Aerodynamic Efficiency

With the objective of increasing the aerodynamic efficiency, the optimization algorithm makes 642 calls to the surrogate model, iterating for 74 seconds. The optimized airfoil is shown in Figure 7.5. The predicted coefficients, together with the real coefficients computed with OpenFoam are reported in Table 7.1.

The airfoil result of the optimization shows a 15.44% decrease of the objective function, with a 24.32% decrease in C_d, which on the baseline airfoil was equal to 0.7524 and a 12.87% decrease in C_d, which on the baseline airfoil was 0.01355.

The pressure coefficient plot, compared to the reference solution, is shown in Figure 7.6, whereas the Mach number and pressure contours comparison with the original airfoil is shown in Figures 7.8 and 7.7.

The flow on the optimized airfoil accelerates more at the beginning of the suction

Table 7.1: Predicted and computed coefficients

-	C_l	C_d
Airfoil 1 - Predicted	0.6574	0.009744
Airfoil 1 - Simulated	0.6558	0.009986
Prediction Error [%]	0.243	2.48

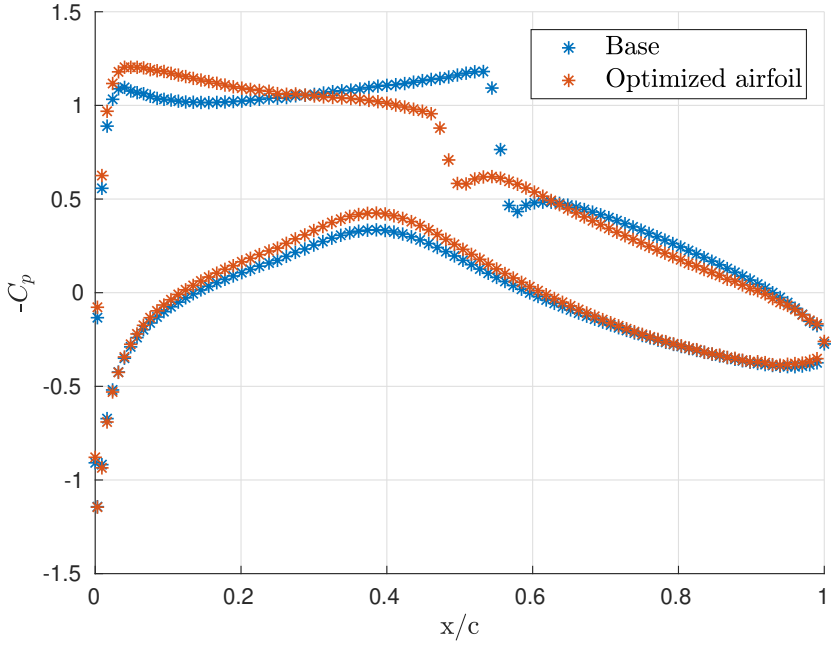
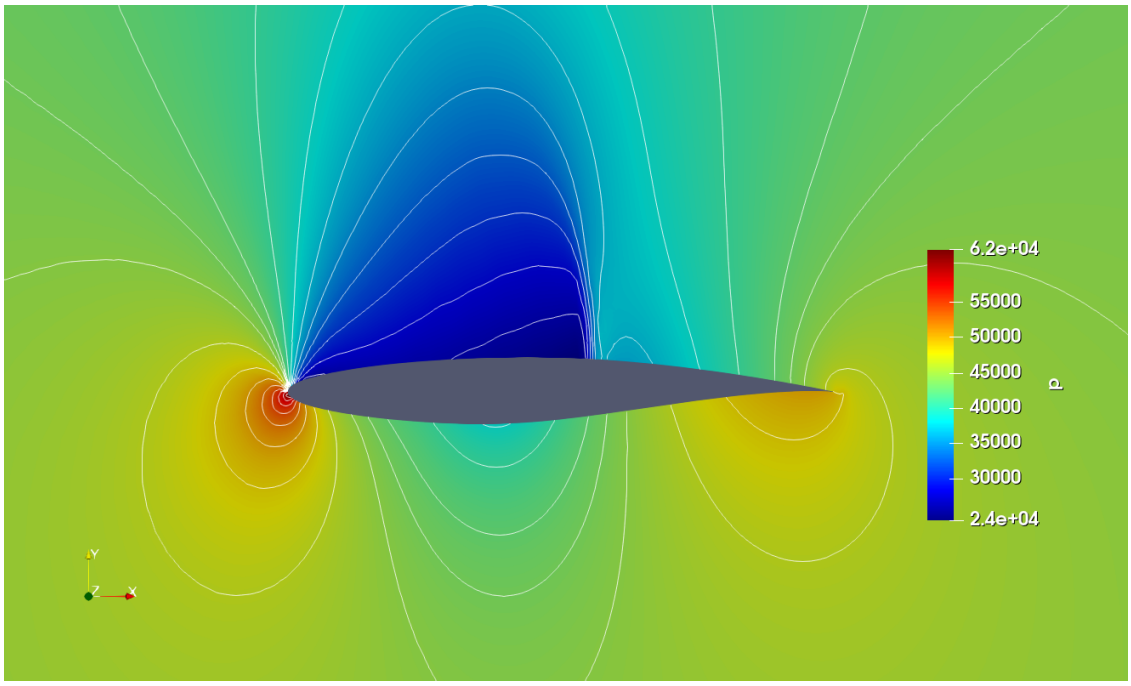
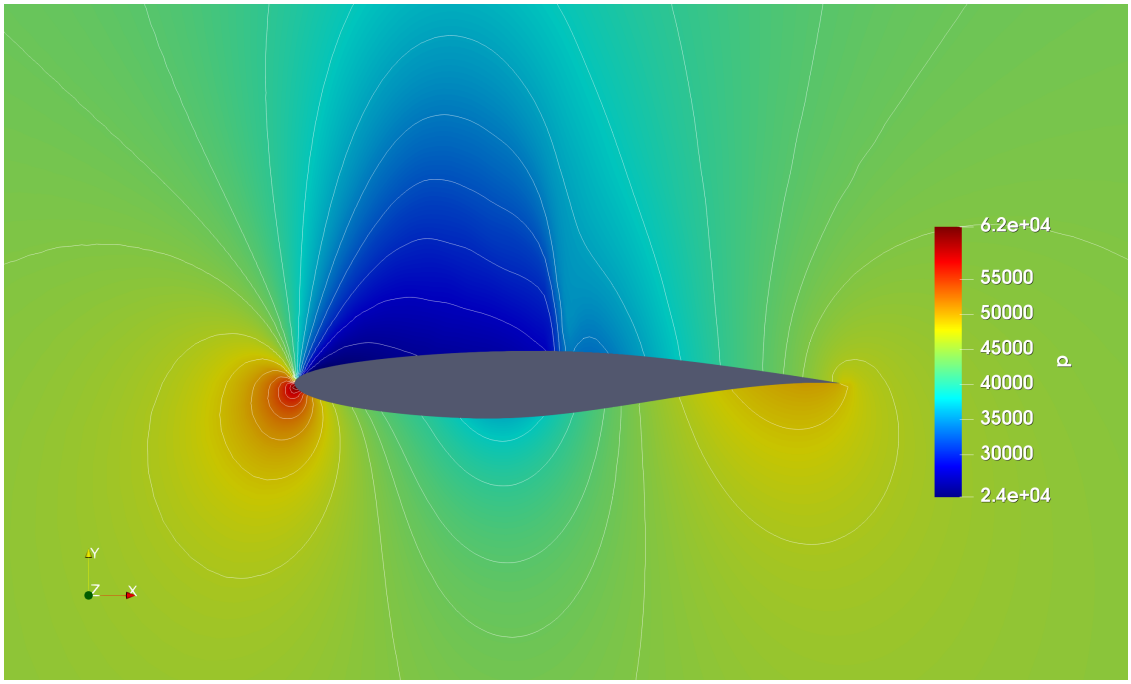


Figure 7.6: Optimized airfoil - Pressure coefficient

side and then it slowly decreases until the shock occurs. The latter is anticipated with respect to the reference solution, an its intensity is strongly decreased, this justifies the reduction in drag.

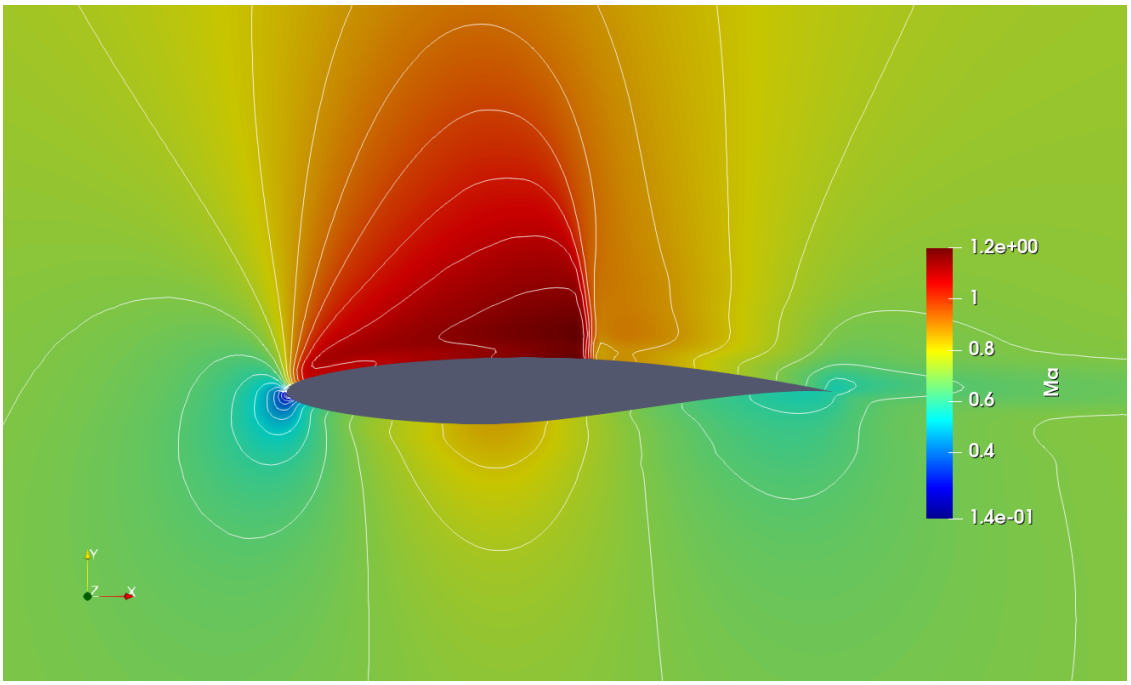


(a) Pressure contour - Original Airfoil

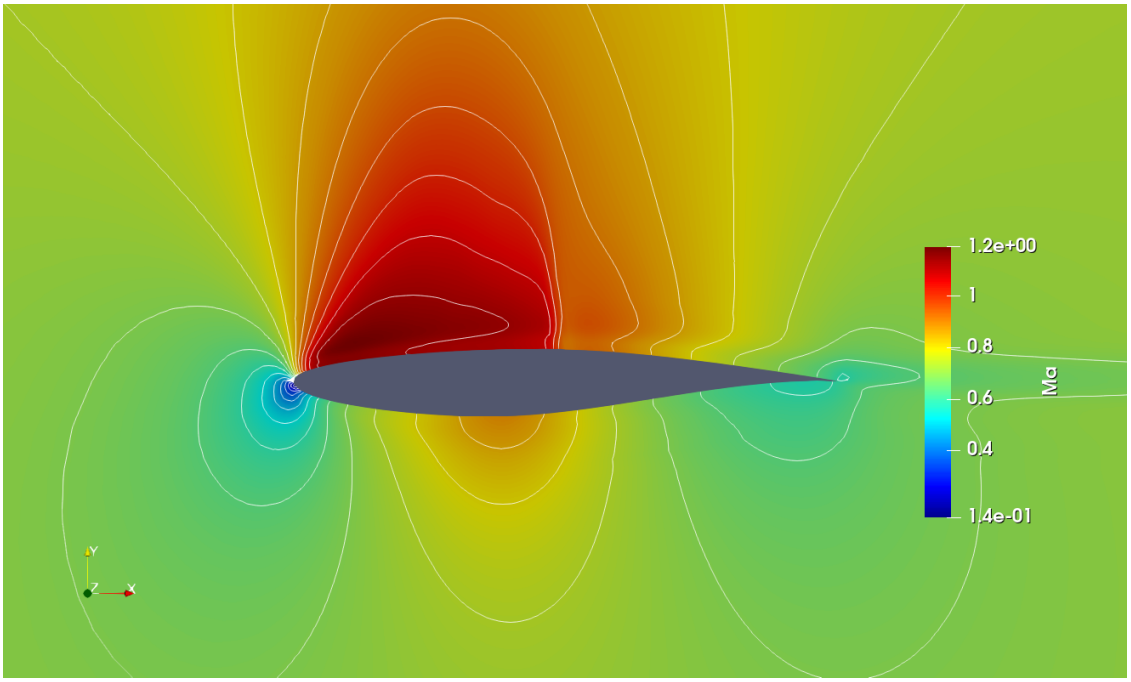


(b) Pressure contour - Optimized Airfoil

Figure 7.7: Pressure contours - Comparison



(a) Mach number contour - Original Airfoil



(b) Mach number contour - Optimized Airfoil

Figure 7.8: Mach number contours - Comparison

7.3.2. Drag Minimization at Fixed Lift

With the goal of reducing drag at fixed lift, the optimization algorithm makes 397 calls to the ANN surrogate model, finding the minimum in 46 seconds.

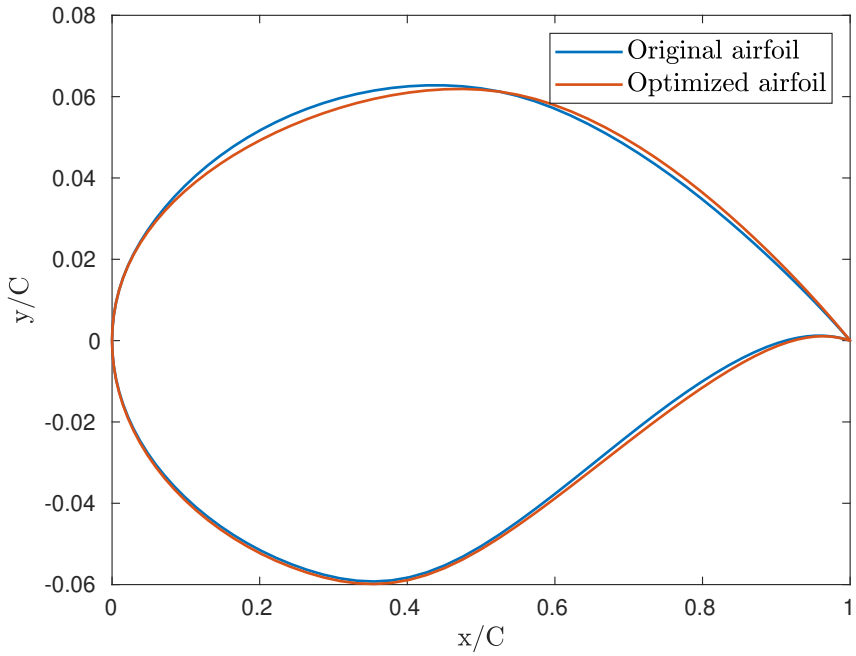


Figure 7.9: Optimized airfoil

Table 7.2: Predicted and computed coefficients

-	C_l	C_d
Optimized Airfoil - Predicted	0.7596	0.01154
Optimized Airfoil - Simulated	0.7588	0.01182
Prediction Error [%]	0.106	2.37

The optimized airfoil is shown in Figure 7.9, and the predicted coefficients, together with the real coefficients computed with OpenFoam are reported in Table 7.2. On the optimized geometry, the ANN provides an accurate prevision of both coefficients, with only a 0.1% error on the lift coefficient and a 2.37% error on the drag coefficient, which has been the most difficult to predict during the whole work. The procedure allows to successfully find a geometry that decreases drag without decreasing lift. As it can be seen in Table 7.2, using this ANN it is possible to obtain a significant reduction in drag (12.79%), together with a 2.34% increase in lift.

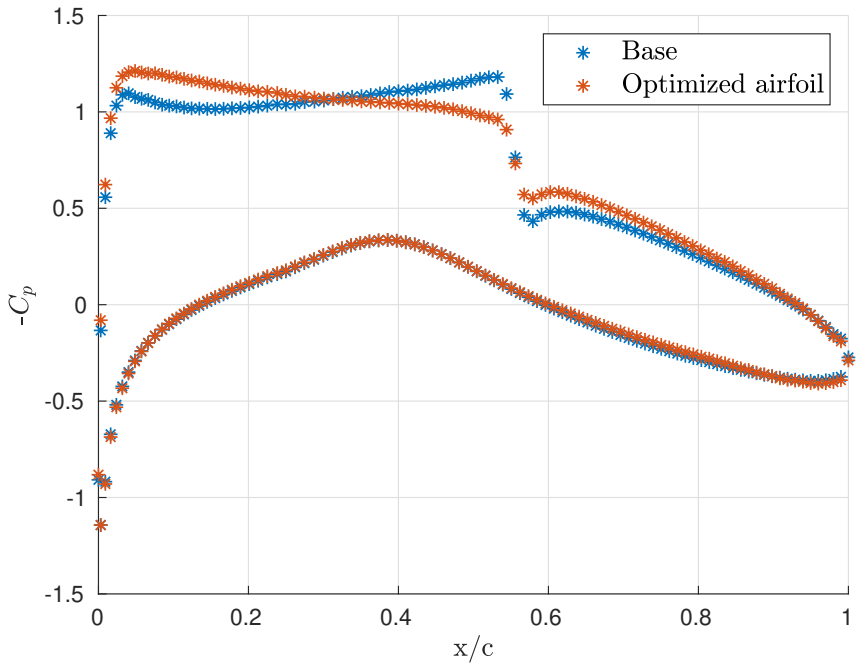


Figure 7.10: Optimized airfoil - Pressure coefficient

Figure 7.10 shows the behaviour of the pressure coefficient compared to the reference solution. On the pressure side it shows moderate changes, whereas there are significant changes on the suction side. The flow accelerates more in the first part, getting to a lower pressure coefficient near the leading edge. Then, thanks to a gentle deceleration with a low pressure gradient, the shock wave intensity is strongly reduced, allowing a significant reduction in terms of total drag.

Figures 7.11 and 7.12 show the the Mach number and pressure contours around the optimized airfoil. The maximum Mach number is equal to 1.18 and it is located near the leading edge on the suction side, while thanks to the smooth deceleration, the Mach number before the shock is equal to 1.08, this allows to reduce the shock intensity and hence the shock wave drag. The maximum Mach number on the original airfoil, that is shown in Figure 7.8 (a), is equal to 1.19 and it is obtained just before the shock wave.

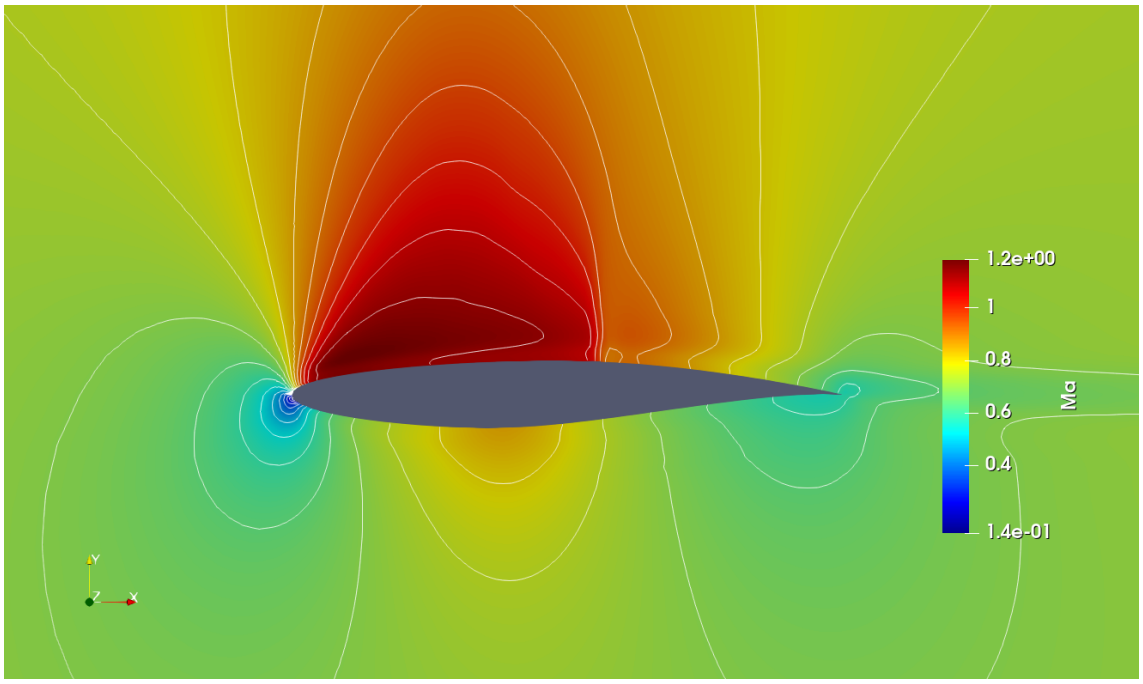


Figure 7.11: Optimized airfoil - Mach number contour

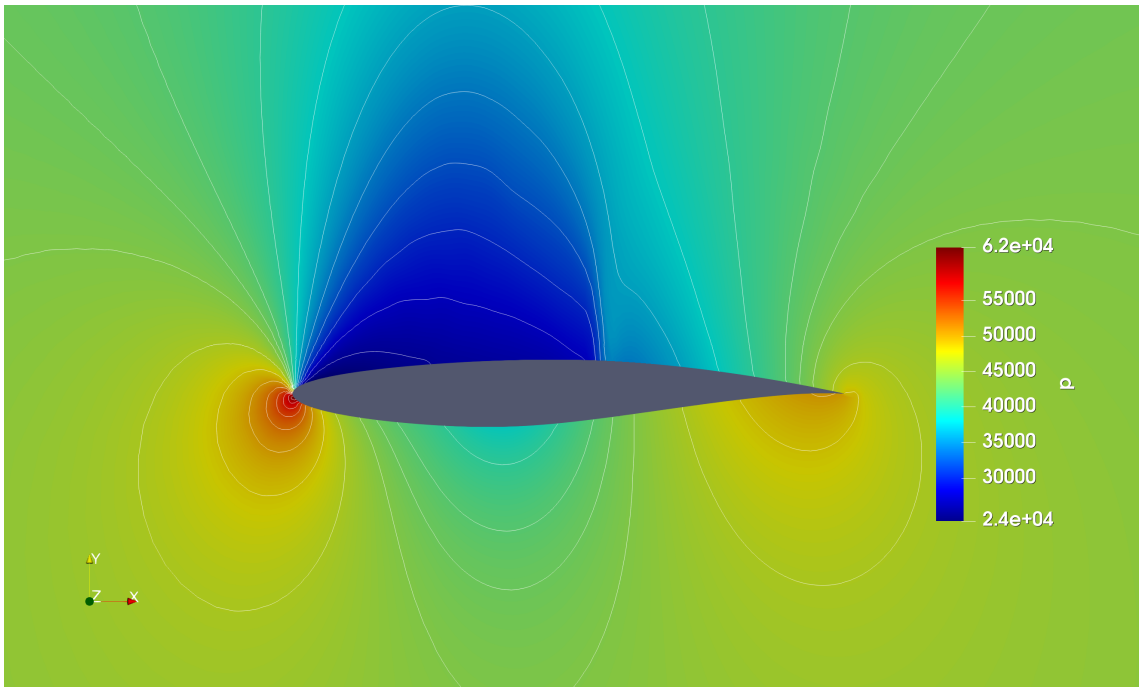


Figure 7.12: Optimized airfoil - Pressure contour

Figure 7.13 shows the comparison between the shocks on the suction side of the two airfoils. The Mach jump across the shock is reduced by 46%, the numerical values

Table 7.3: Suction side shock wave comparison - Numerical values

-	M_1	M_2	Δ M
Base	1.19	0.848	0.342
Optimized	1.08	0.898	0.182

are reported in Table 7.3, where M_1 and M_2 are the Mach number values before and after the shock wave.

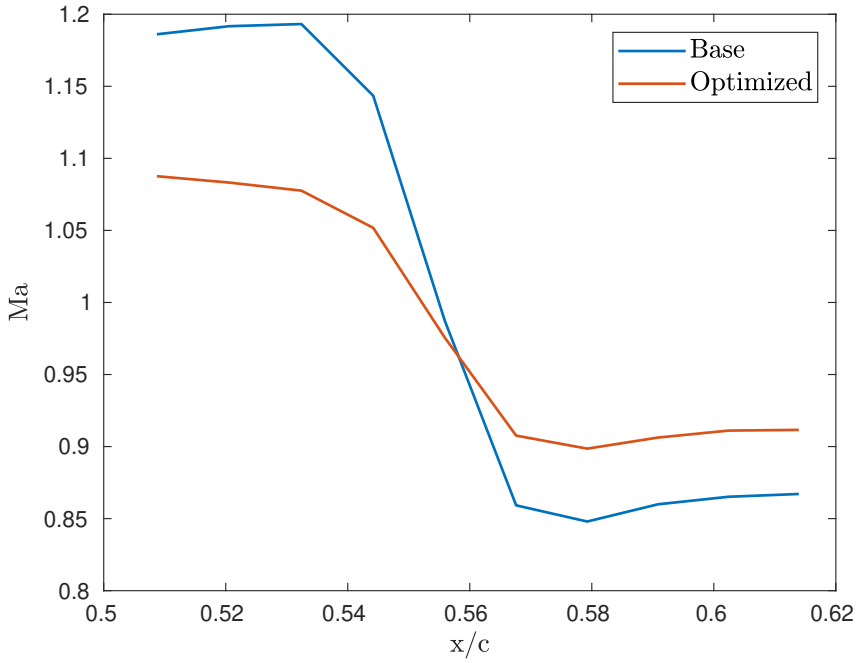


Figure 7.13: Suction side shock wave comparison

7.4. Iterative Optimization with Surrogate Model

In this section we present an alternative procedure to perform the optimization using ANNs-based surrogate models. This is intended to investigate different possibilities aimed to reduce further the computational cost, which is mainly associated to the CFD solution of the airfoil samples needed to train the ANNs.

Starting from the same 500 samples generated in the Test 4 (in Section 6.2.2), an iterative procedure is set up.

The iterative procedure is explained in algorithm 1, where D_k is the percentage dimension of the space S_k compared to the original parameters space, d^k is the distance between the minimums found in the last two iterations, $\hat{d}$ is a tolerance set

Algorithm 1 Iterative Optimization with Surrogate Model algorithm

while $d^k > \hat{d}$ **do**
  Select the space dimension D_k, generating the new design space $\mathbf{S_k}$;
  Select the samples already generated within $\mathbf{S_k}$;
  Train the ANNs-based surrogate model M_k;
  if $\mathcal{E}_{cv} < tol$ **then**
    Add 25 new samples according to the Sobol sequence;
  end if
  Perform the optimization with the model M_k, finding the minimum U_k;
  Compute the relative norm of the step $d^k = \frac{||U_k - U_{k-1}||}{||U_k||}$;
end while

Table 7.4: Iterative Optimization with Surrogate Model steps

Step	C_l	C_d	ΔC_d^k [%]	ΔC_d^{tot} [%]
1	0.7537	0.01216	-	10.28
2	0.7608	0.01175	3.372	13.31
3	0.7584	0.01170	0.4274	13.67

to 0.5%, *tol* is the maximum accepted generalization error, measured with the error on the cross-validation set, set to 1.5%.

Selecting the first two spaces so that their dimension is a quarter of the design space of the fourth test, leads to an optimum solution that allows a 13% reduction in the drag coefficient. This result is an improvement compared to the one obtained in Section 7.3.2. The important advantage that is associated with this procedure is in the computational time required for the generation of an appropriate training set.

For the first two iterations, the number of samples required is indeed 234, which is significantly lower compared to the previous result. This allows a 54% reduction in the computational time required, that drops from 135 to 65 computer hours.

For the third iteration it was needed to generate 50 new samples in order to obtain a sufficiently accurate surrogate model, this has increased the total computational time to 78 hours. In this case, the space dimension was set to one tenth of the fourth experiment design space, since the norm of the second step was significantly lower than the first, indicating the approach to a minimum in the objective function.

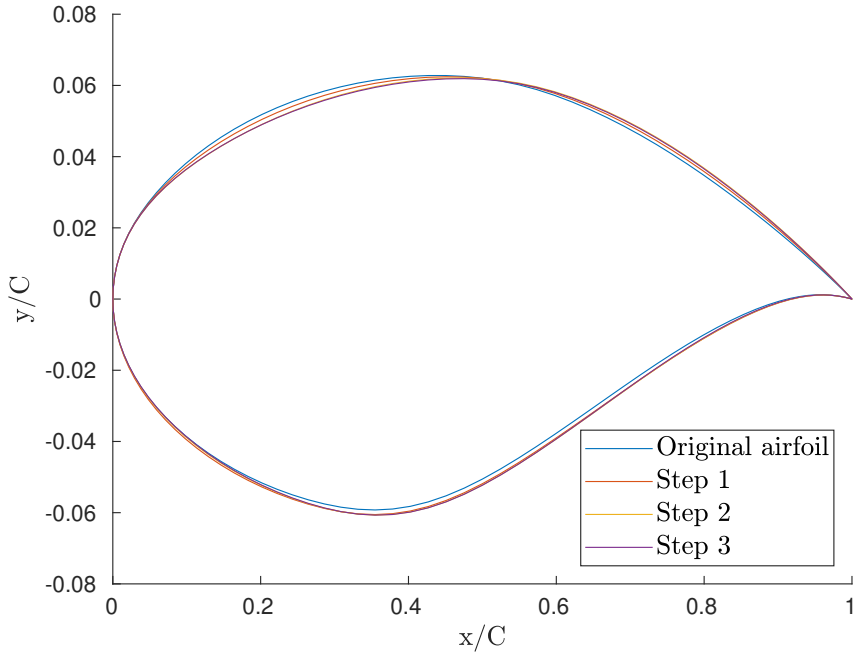


Figure 7.14: Optimized airfoils sequence

Table 7.4 presents the force coefficients together with the iterative optimization step k, with ΔC_d^k being the drag coefficient improvements compared to the previous iteration and with ΔC_d^{tot} being the drag coefficient improvements compared to the original airfoil. The last two iterations give similar results, confirming that the procedure is converged at a local minimum. The airfoil obtained with the third iteration gives the best results within the whole analysis.

Figure 7.14 shows the different airfoil shapes generated during the iterative procedure, while Figure 7.15 shows the pressure coefficient plots of those airfoils. From both figures it can be seen that there is a small difference between the last two iterations, that are significantly different from the previous ones. The reduction of the drag coefficient presented in Table 7.4 can be explained once again by the reduction of the shock wave intensity.

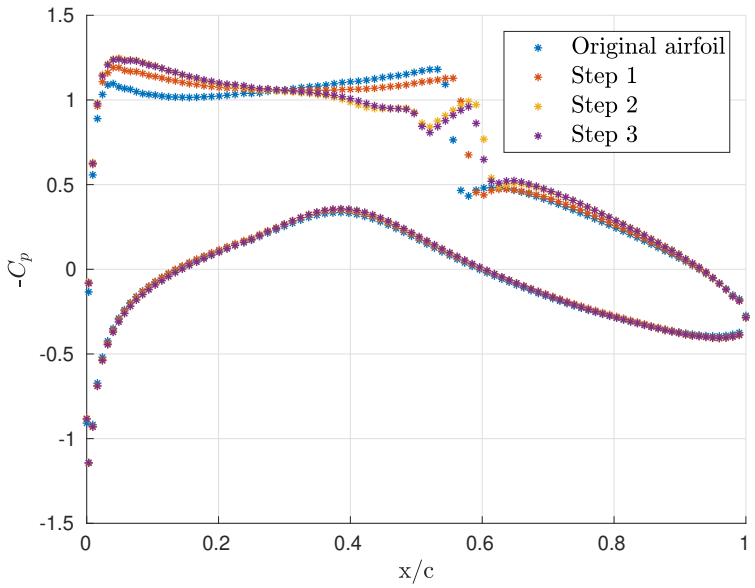


Figure 7.15: Pressure coefficients

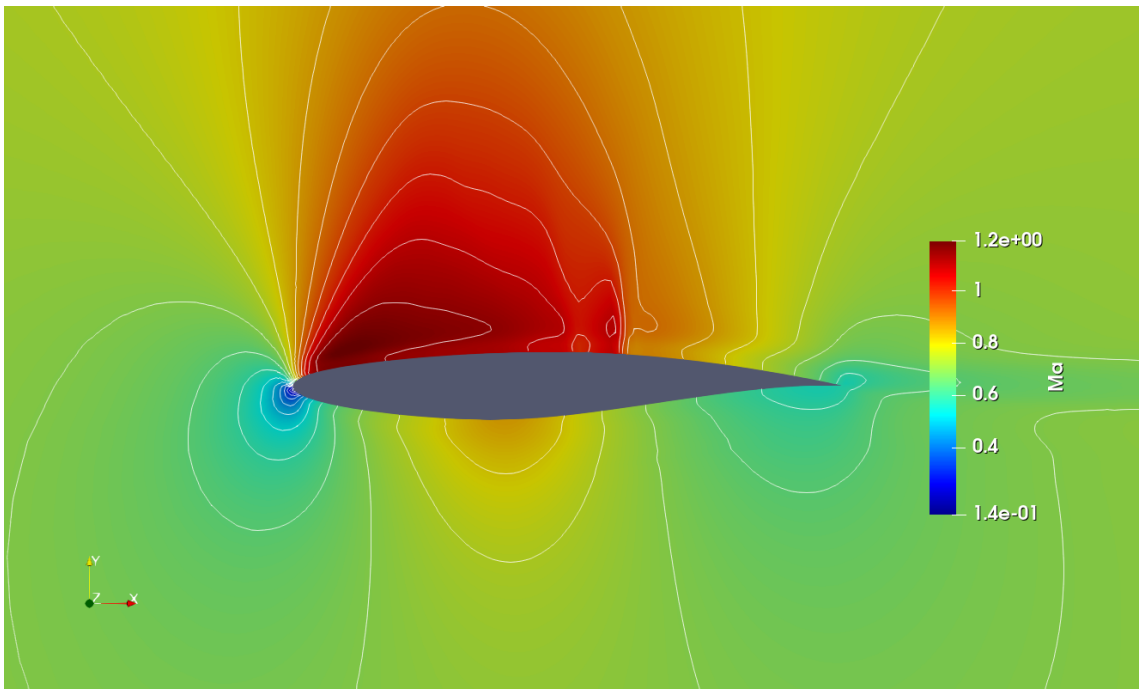


Figure 7.16: Optimized airfoil - Mach number contour

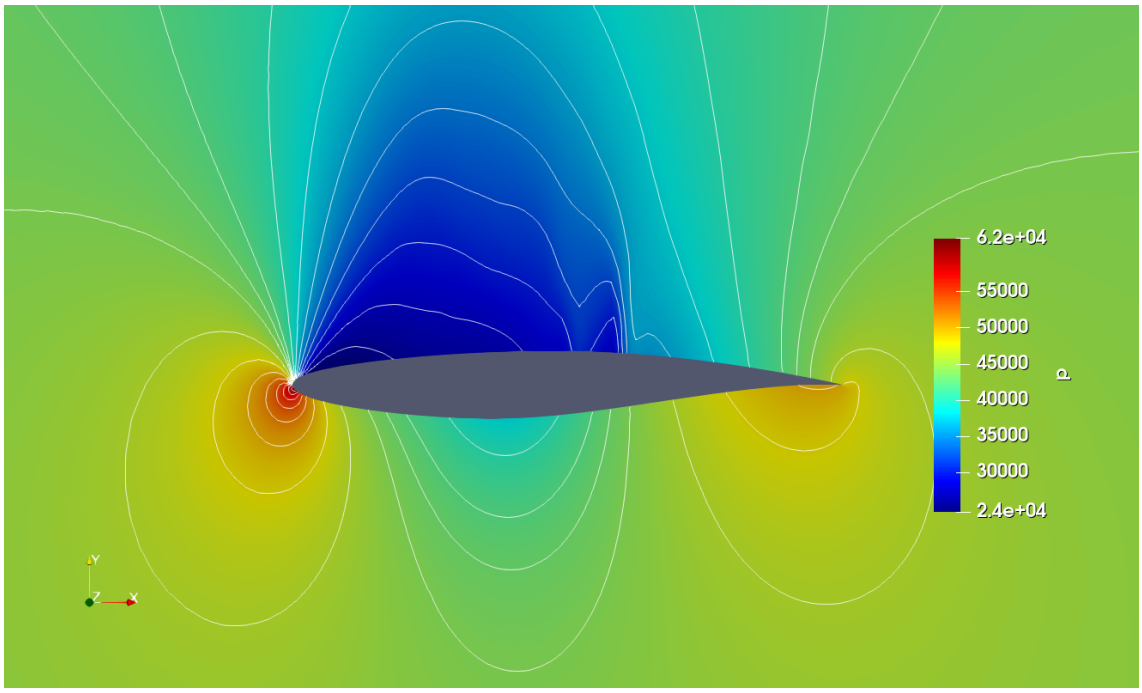


Figure 7.17: Optimized airfoil - Pressure contour

7.5. Shape Optimization with High Fidelity CFD Model

To investigate the advantages related to the optimization performed with the ANN surrogate model, an optimization using the High Fidelity CFD model has been performed. This procedure has been done using the parametrization explained in Section 6.2.2 for the Test 4. The only difference with the previous case is that, during the optimization, the force coefficients have been evaluated with the CFD solver. This has allowed us to obtain an estimate of the advantage in computational time associated with the construction of an ANNs-based surrogate model. Since this procedure requires to launch OpenFOAM from Matlab, the simulation time of a single case has increases of the 50%, requiring for a single simulation up to 30 minutes.

To reduce the computational time required for this comparison, the maximum number of iterations of the interior-point optimization algorithm has been set to 10, the final point will not be a minimum but the result can be used to estimate the differences both in terms of results and computational time between the direct simulation and the surrogate model. With these parameters, the algorithm makes 148 calls to the CFD solver, requiring 72 computer hours. The value of the objective function at

the final step is moderately reduced, assuming a value of 0.95, with a drag coefficient equal to 0.01256 and a lift coefficient equal to 0.7534.

With the same parameters, a surrogate model optimization has been performed, as in Section 7.3.2. The algorithm makes 99 calls to the surrogate model and its final point is characterized by a value of the objective function equal to 0.89, with a drag coefficient equal to 0.01196 and a lift coefficient equal to 0.7578.

These results show that, even though the number of samples required to train a ANNs-based surrogate model is high, a direct optimization using the CFD solver would reasonably require a larger time, without ensuring a better performance in terms of objective function value. Furthermore, the generation of a unique surrogate model that maps the same parameters space allows to perform optimizations with different objective functions at a negligible computational cost.

Considering instead the comparison of the HF Model Optimization with the Iterative Surrogate Model Optimization, the latter outperforms by far the former, both in terms of computational time and of the objective function value.

Table 7.5: HF Model Optimization vs Standard Surrogate Model Optimization

-	Number of calls	C_l	C_d
HF Model Optimization	148	0.7534	0.01256
Standard Surrogate Model Optimization	99	0.7578	0.01196

8 | Conclusions and Future Developments

The airfoil shape optimization requires multiple calls to expensive numerical CFD solvers, the usage of surrogate models in the context of constrained optimization is an attractive proposition. This work shows that, as long as the surrogate model provides an accurate approximation of the PDEs while being computationally cheap to evaluate, it can be used within standard optimization algorithms with good results, leading to computational advantages.

In this work, we carefully studied a geometry parametrization and mesh deformation procedure, aimed to the generation of the training data-set required by the ANNs-based surrogate model. Findings show that, with a limited number of design variables, RBFs cannot provide a smooth deformation as long as the airfoils' angle of attack is fixed. Hicks-Henne functions, instead, provide sufficiently smooth geometries, together with sufficient freedom in the shape deformation.

The generation of the ANNs-based surrogate model was performed using single hidden layer Feed Forward Neural Networks, implemented by F. Regazzoni in the *model-learning* library. It the context of surrogate model generation, it has been observed that the prediction of the Drag coefficient is more critical compared to the Lift coefficient. The prediction error on the drag coefficient determined, for the whole work, the increase in the number of samples required to obtain a sufficiently accurate surrogate model. Furthermore, the results show that when the geometry parametrization complexity increases, i.e. the design space is larger, the finding of a model with the required accuracy can be difficult.

We proposed two main procedures to perform the airfoil optimization, a Standard Surrogate Model Optimization (SSMO) and an Iterative Surrogate Model Optimization (ISMO). The SSMO procedure involves the generation of a single surrogate model, mapping the whole parameters space. This model is used to perform the optimization. The ISMO procedure is intended to reduce the dimension of the

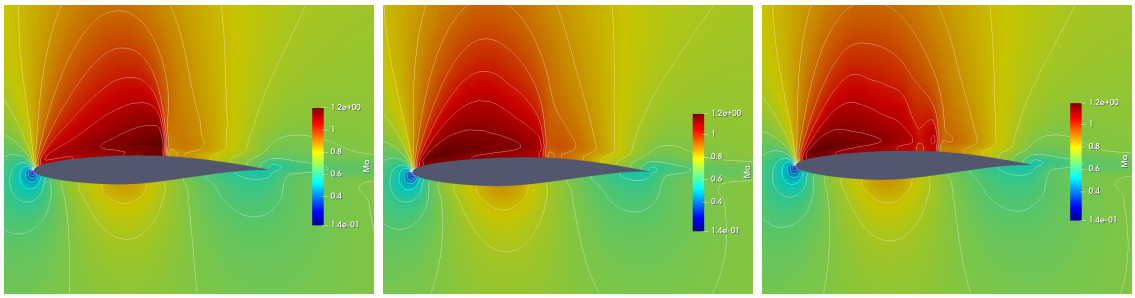


Figure 8.1: Mach number contours comparison. Original airfoil (left), optimized airfoil - SSMO (center), optimized airfoil - ISMO (right).

required data-set and hence the computational burden associated with the CFD simulation of the samples. It consists in finding a sequence of surrogate models, where at each step the parameters space is shifted to the local minimum found by the previous iteration. This method provides the most promising results, finding the lowest value of the objective function in the drag-minimization case and reducing the computational time required for the generation of the training set of up to the 43% compared to the Standard Surrogate Model Optimization. The SSMO represents instead a good choice when it is required to investigate the optimization results with different objective functions.

The Standard Surrogate Model Optimization allowed us to obtain a 15.4% improvement with the goal of increasing the aerodynamic efficiency, and a 12.8% improvement in the drag-reduction problem, requiring a total computational time of 135 hours. The Iterative Surrogate Model Optimization, instead, allowed us to obtain a 13.7% reduction in the drag coefficient, reducing the computational time required for the generation of the training set to 78 hours. Figure 8.1 shows the comparison between the original Mach number flow field (left), and the optimized airfoils with the drag-reduction objective, obtained with the Standard Surrogate Model Optimization (center) and with the Iterative Surrogate Model Optimization (right).

With the same parameters used in the Standard Surrogate Model Optimization, the optimization performed using the HF model instead of the surrogate model has shown to significantly increase the required computational time and, with the same number of iterations, it gave worse optimization results.

Further developments of this work could include some flow parameters, such as Reynolds Number or Angle of Attack within the design parameters, to investigate the usage of SSMO and ISMO in a multi-objective optimization. Moreover, to exploit the full potential of the *model-learning* library, the generation of a surrogate

model to perform shape optimization of an unsteady airfoil simulation could be examined.

Bibliography

[1] M. Y. Aghdam, S. R. K. Tabbakh, S. J. M. Chabok, and M. Kheyrabadi. Optimization of air traffic management efficiency based on deep learning enriched by the long short-term memory (lstm) and extreme learning machine (elm). *Journal of Big Data*, 8(1):1–26, 2021.

[2] D. Anderson. Fuel conservation - airframe maintenance for environmental performance, 2006. URL `https://www.icao.int/Meetings/EnvironmentalWorkshops/Documents/ICAO-TransportCanada-2006/Anderson.pdf`. Last Accessed 11/11/2021.

[3] R. Anyoha. The history of artificial intelligence, Aug 2017. URL `https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence`. Last Accessed 07/09/2021.

[4] Baidu, Feb 2021. URL `https://www.technologyreview.com/2021/01/14/1016122/these-five-ai-developments-will-shape-2021-and-beyond/`. MIT Technology Review. Last Accessed 9/09/2021.

[5] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[6] S. L. Brunton, B. R. Noack, and P. Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020.

[7] J. Burkardt, M. Gunzburger, and H.-C. Lee. Pod and cvt-based reduced-order modeling of navier–stokes flows. *Computer methods in applied mechanics and engineering*, 196(1-3):337–355, 2006.

[8] R. H. Byrd, J. C. Gilbert, and J. Nocedal. A trust region method based on interior point techniques for nonlinear programming. *Mathematical programming*, 89(1):149–185, 2000.

[9] P. Castonguay and S. Nadarajah. Effect of shape parameterization on

aerodynamic shape optimization. In *45th AIAA Aerospace Sciences Meeting and Exhibit*, page 59, 2007.

[10] P. Cook, M. McDonald, and M. Firmin. Aerofoil rae 2822-pressure distributions, and boundary layer and wake measurements. experimental data base for computer program assessment. *AGARD Report AR*, 138, 1979.

[11] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[12] A. De Boer, M. Van der Schoot, and H. Bijl. Mesh deformation based on radial basis function interpolation. *Computers & structures*, 85(11-14): 784–795, 2007.

[13] Directorate-General for Climate Action. Updated analysis of the non-co2 effects of aviation, 2020. URL `https://ec.europa.eu/clima/news-your-voice/news/updated-analysis-non-co2-effects-aviation-2020-11-24_en`. Last accessed 15/11/2021.

[14] X. Dou. Big data and smart aviation information management system. *Cogent Business & Management*, 7(1):1766736, 2020.

[15] K. Duraisamy, G. Iaccarino, and H. Xiao. Turbulence modeling in the age of data. *Annual Review of Fluid Mechanics*, 51:357–377, 2019.

[16] T. Duriez, S. L. Brunton, and B. R. Noack. *Machine learning control-taming nonlinear dynamics and turbulence*. Springer, 2017.

[17] EU Action. Reducing emissions from aviation, 2021. URL `https://ec.europa.eu/clima/eu-action/transport-emissions/reducing-emissions-aviation_en`. Last accessed 15/11/2021.

[18] R. Eyjolfsson. *Design and manufacture of pharmaceutical tablets*. Academic Press, 2014.

[19] H. Gagnon and D. W. Zingg. Two-level free-form and axial deformation for exploratory aerodynamic shape optimization. *Aiaa Journal*, 53(7):2015–2026, 2015.

[20] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[21] J. Goss and K. Subbarao. Inlet shape optimization based on pod model

reduction of the euler equations. In *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, page 5809, 2008.

[22] C. J. Greenshields, H. G. Weller, L. Gasparini, and J. M. Reese. Implementation of semi-discrete, non-staggered central schemes in a colocated, polyhedral, finite volume framework, for high-speed viscous flows. *International journal for numerical methods in fluids*, 63(1):1–21, 2010.

[23] C. J. Greenshields et al. Openfoam user guide. *OpenFOAM Foundation Ltd, version*, 3(1):47, 2015.

[24] M. Haenlein and A. Kaplan. A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. *California management review*, 61(4):5–14, 2019.

[25] N. Hall. Euler equations, 2021. URL `https://www.grc.nasa.gov/www/k-12/airplane/eulereqs.html`. Last Accessed 15/11/2021.

[26] J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2(1):84–90, 1960.

[27] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1):100–108, 1979.

[28] J. S. Hesthaven, G. Rozza, B. Stamm, et al. *Certified reduced basis methods for parametrized partial differential equations*, volume 590. Springer, 2016.

[29] R. M. Hicks and P. A. Henne. Wing design by numerical optimization. *Journal of Aircraft*, 15(7):407–412, 1978.

[30] W. Jank. Quasi-monte carlo sampling to improve the efficiency of monte carlo em. *Computational statistics & data analysis*, 48(4):685–701, 2005.

[31] S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

[32] R. Kruse, C. Borgelt, C. Braune, S. Mostaghim, and M. Steinbrecher. Computational intelligence-a methodological introduction, 2016.

[33] S. Kucherenko, D. Albrecht, and A. Saltelli. Exploring multi-dimensional

spaces: A comparison of latin hypercube and quasi monte carlo sampling techniques. *arXiv preprint arXiv:1505.02350*, 2015.

[34] R. Lapuh. Mesh morphing technique used with open-source cfd toolbox in multidisciplinary design optimisation. Master's thesis, Uppsala Universitet, 2019.

[35] R. J. LeVeque. *Numerical methods for conservation laws (2. ed.)*. Lectures in mathematics. Birkhäuser, 1992. ISBN 978-3-7643-2723-1.

[36] R. J. LeVeque et al. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge university press, 2002.

[37] C.-N. Li, Q. Wei, C.-L. Gong, and L.-X. Gu. An efficient multiple point selection study for mesh deformation using radial basis functions. *Aerospace Science and Technology*, 71:580–591, 2017.

[38] J. P. Lourenço. Supersonic and transonic adjoint-based optimization of airfoils. Master's thesis, Instituto Superior Técnico Lisboa, 2018.

[39] K. O. Lye, S. Mishra, D. Ray, and P. Chandrashekar. Iterative surrogate model optimization (ismo): An active learning algorithm for pde constrained optimization with deep neural networks. *Computer Methods in Applied Mechanics and Engineering*, 374:113575, 2021.

[40] M. J. Martin, E. Andres, C. Lozano, and E. Valero. Volumetric b-splines shape parametrization for aerodynamic shape design. *Aerospace Science and Technology*, 37:26–36, 2014.

[41] R. Maulik, O. San, A. Rasheed, and P. Vedula. Subgrid modelling for two-dimensional turbulence using neural networks. *Journal of Fluid Mechanics*, 858:122–144, 2019.

[42] S. Mishra and T. K. Rusch. Enhancing accuracy of deep learning algorithms by training with low-discrepancy sequences. *SIAM Journal on Numerical Analysis*, 59(3):1811–1834, 2021.

[43] S. Mishra and T. K. Rusch. Enhancing accuracy of deep learning algorithms by training with low-discrepancy sequences. *SIAM Journal on Numerical Analysis*, 59(3):1811–1834, 2021.

[44] B. Mohammadi and O. Pironneau. *Applied shape optimization for fluids*. Oxford university press, 2010.

[45] F. Moukalled, L. Mangani, M. Darwish, et al. *The finite volume method in computational fluid dynamics*, volume 113. Springer, 2016.

[46] S. Painchaud-Ouellet, C. Tribes, J.-Y. Trépanier, and D. Pelletier. Airfoil shape optimization using a nonuniform rational b-splines parametrization under thickness constraint. *AIAA journal*, 44(10):2170–2178, 2006.

[47] A. Quarteroni and S. Quarteroni. *Numerical models for differential problems*, volume 2. Springer, 2009.

[48] A. Quarteroni, A. Manzoni, and F. Negri. *Reduced basis methods for partial differential equations: an introduction*, volume 92. Springer, 2015.

[49] N. V. Queipo, R. T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, and P. K. Tucker. Surrogate-based analysis and optimization. *Progress in aerospace sciences*, 41(1):1–28, 2005.

[50] D. P. Raymer. Aircraft design: a conceptual approach (aiaa education series). *Reston, Virginia*, 2012.

[51] F. Regazzoni. Model-learning repository, 2019. URL `https://github.com/FrancescoRegazzoni/model-learning`. Last accessed 18/10/2021.

[52] F. Regazzoni, L. Dede, and A. Quarteroni. Machine learning for fast and reliable solution of time-dependent differential equations. *Journal of Computational physics*, 397:108852, 2019.

[53] F. Regazzoni, D. Chapelle, and P. Moireau. Combining data assimilation and machine learning to build data-driven models for unknown long time dynamics—applications in cardiovascular modeling. *International Journal for Numerical Methods in Biomedical Engineering*, page e3471, 2021.

[54] S. Rowland. Maximize model performance without maximizing effort, 2020. URL `https://blogs.sas.com/content/subconsciousmusings/2020/04/08/maximize-model-performance/`. Last accessed 26/10/2021.

[55] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[56] J. Samareh. Aerodynamic shape optimization based on free-form deformation. In *10th AIAA/ISSMO multidisciplinary analysis and optimization conference*, page 4630, 2004.

[57] M. A. Samel. Numerical investigation of gas-particle supersonic flow. Master's thesis, University of Massachusetts Amherst, 2011.

[58] M. Sharbatdar and C. Ollivier Gooch. Anisotropic mesh adaptation: recovering quasi-structured meshes. In *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, page 149, 2012.

[59] I. M. Sobol′. On the distribution of points in a cube and the approximate evaluation of integrals. *Zhurnal Vychislitelńoi Matematiki i Matematicheskoi Fiziki*, 7(4):784–802, 1967.

[60] STF solutions. Automatic airfoil c-grid generation for openfoam. Available at: `https://curiosityfluids.com/2019/04/22/automatic-airfoil-cmesh-generation-for-openfoam-rev-1/`, 2019. Last accessed 18/10/2021.

[61] B. Tang. Orthogonal array-based latin hypercubes. *Journal of the American statistical association*, 88(424):1392–1397, 1993.

[62] The MathWorks Inc. *MATLAB - version R2019a*. MathWorks, Natick, Massachusetts, 2019.

[63] J. Tu, G.-H. Yeoh, and C. Liu. Chapter 4 - cfd techniques—the basics. In J. Tu, G.-H. Yeoh, and C. Liu, editors, *Computational Fluid Dynamics (Second edition)*, pages 123–175. Butterworth-Heinemann, 2013. ISBN 978-0-08-098243-4. doi: https://doi.org/10.1016/B978-0-08-098243-4.00004-4. URL `https://www.sciencedirect.com/science/article/pii/B9780080982434000044`.

[64] A. M. Turing. Computing machinery and intelligence. In *Parsing the turing test*, pages 23–65. Springer, 2009.

[65] J. Viquerat, J. Rabault, A. Kuhnle, H. Ghraieb, A. Larcher, and E. Hachem. Direct shape optimization through deep reinforcement learning. *Journal of Computational Physics*, 428:110080, 2021.

[66] A. Viviani, A. Aprovitola, G. Pezzella, and C. Rainone. Cfd design capabilities for next generation high-speed aircraft. *Acta Astronautica*, 178:143–158, 2021.

[67] H.-Y. Wu, S. Yang, F. Liu, and H.-M. Tsai. Comparisons of three geometric representations of airfoils for aerodynamic optimization. In *16th AIAA computational fluid dynamics conference*, page 4095, 2003.

[68] F. Yang, Z. Yue, L. Li, and W. Yang. Aerodynamic optimization method

based on bezier curve and radial basis function. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 232(3): 459–471, 2018.

[69] Y. Yu, X. Si, C. Hu, and J. Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7):1235–1270, 2019.

A | Appendix A

A.1. Numerical schemes

In this section the numerical schemes set in the *fvSolution* file are reported:

```
/*--------------------------------* C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

fluxScheme      Kurganov;

ddtSchemes
{
    default         CrankNicolson 0;
}

gradSchemes
{
    default         cellLimited Gauss linear 1;
    grad(U)         cellLimited Gauss linear 1;
}

divSchemes
{
    default                     none;
    div(phi,U)                  Gauss linearUpwindV grad(U);
}

laplacianSchemes
{
    default         Gauss linear limited 1;
}

interpolationSchemes
{
    default         linear;
    reconstruct(rho) vanLeer;
    reconstruct(U)  vanLeerV;
    reconstruct(T)  vanLeer;
}

snGradSchemes
{
    default         limited 1;
}
```

```
// ************************************************************************* //
```

A.2. fvSolution

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.2                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    "(rho|rhoU|rhoE)"
    {
        solver          diagonal;
    }

    U
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        nSweeps         2;
        tolerance       1e-09;
        relTol          0.01;
    }

    h
    {
        $U;
        tolerance       1e-10;
        relTol          0;
    }
}


// ************************************************************************* //
```

A.3. controlDict

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

```
application     rhoCentralFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         0.108;

deltaT          1e-6;

adjustTimeStep  yes;

maxCo           0.1;

writeControl    adjustableRunTime;

writeInterval   0.01;

purgeWrite      0;

writeFormat     ascii;

writePrecision   8;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

functions
{
    #includeFunc MachNo
    #includeFunc residuals
    forces
    {
        type            forceCoeffs;
        libs            ("libforces.so");
        writeControl   timeStep;
        writeControl   timeStep;
        writeInterval  1;


        pName       p;
        UName       U;
        log         true;
        patches
        (
            airfoil
        );

        rhoInf      0.5966;

        CofR        (0 0 0);
        liftDir     (-0.0403 0.9992 0);
        dragDir     (0.9992 0.0403 0);
        pitchAxis   (0 0 1);
        magUInf     233.6216;
        lRef        1;
        Aref        1;
    }
}
runTimeControl1
{
    type            runTimeControl;
    libs            ("libutilityFunctionObjects.so");
    conditions
    {
        condition0
        {
            type            average;
            functionObject  forceCoeffs;
```

```
            fields          (Cd);
            tolerance       1e-1;
            window          10;
            groupID         1;
        }
    }
}

// ************************************************************************* //
```

A.4. Constant folder

Momentum Transport

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      momentumTransport;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

simulationType laminar;


// ************************************************************************* //
```

Thermophysical properties

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      thermophysicalProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

thermoType
{
    type            hePsiThermo;
    mixture         pureMixture;
    transport       const;
    thermo          hConst;
    equationOfState perfectGas;
    specie          specie;
    energy          sensibleInternalEnergy;
}

mixture
```

```
{
    specie
    {
        molWeight    28.9;
    }
    thermodynamics
    {
        Cp           1005;
        Hf           0;
    }
    transport
    {
        mu           0;
        Pr           0.71;
    }
}

// ************************************************************************* //
```

A.5. Boundary conditions - 0 folder

Temperature

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      T;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

Tinlet          255.6;

dimensions      [0 0 0 1 0 0 0];

internalField   uniform $Tinlet;

boundaryField
{
    farfield
    {
        type            inletOutlet;
        inletValue      uniform $Tinlet;
        value           $inletValue;
    }

    airfoil
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            empty;
    }

    #includeEtc "caseDicts/setConstraintTypes"
}

// ************************************************************************* //
```

Mach number

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      Ma;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0.72783363;

boundaryField
{
    frontAndBack
    {
        type            empty;
    }
    farfield
    {
        type            calculated;
        value           uniform 0.72783363;
    }
    airfoil
    {
        type            calculated;
        value           uniform 0.72783363;
    }
}


// ************************************************************************* //
```

Velocity

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

Uinlet          (233.4317 9.4164  0);

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform $Uinlet;

boundaryField
{
    farfield
    {
        type            freestreamVelocity;
```

```
        freestreamValue uniform $Uinlet;
        value           uniform $Uinlet;
    }
    airfoil
    {
        type            slip;
    }

    frontAndBack
    {
        type            empty;
    }
    #includeEtc "caseDicts/setConstraintTypes"
}

// ************************************************************************* //
```

Pressure

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

pOut            43765;

dimensions      [1 -1 -2 0 0 0 0];

internalField   uniform $pOut;

boundaryField
{
    farfield
    {
        type            fixedValue;
        value           uniform  43765;
    }

    airfoil
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            empty;
    }

    #includeEtc "caseDicts/setConstraintTypes"
}

// ************************************************************************* //
```

List of Figures

List of Tables

List of Symbols

Variable	Description	SI unit
α	angle of attack	rad
β	linear polynomial coefficient	[-]
γ	RBFs coefficient	[-]
ε_{cv}	generalization error	[-]
ε_G	generalization gap	[-]
ε_{Train}	train set error	[-]
ε_{Test}	test set error	[-]
Θ	neuron bias	[-]
μ	dynamic viscosity	[kg/(ms)]
ρ	density	[kg/m^3]
σ	RBFs smoothing parameter	[-]
ϕ	compressibility factor	[-]
Φ	Radial Basis Function	[-]
Ω	control volume	[m^3]
a	HH bump intensity	[-]
C	aerofoil chord	[m]
C_d	drag coefficient	[-]
C_l	lift coefficient	[-]
$\hat{d}$	reference norm of the ISMO step	[-]
d^k	norm of the ISMO step k	[-]
E	aerodynamic efficiency	[-]
E	energy	[kg m^2/s^2]
$\mathcal{J}$	ANNs loss function	[-]
M	Mach number	[-]
$\mathcal{O}$	optimization objective function	

p	pressure	[kg/(m s^2)]
P	penalization factor	[-]
Re	Reynolds number	[-]
$\mathbf{s}$	RBF displacement	[m]
t	time	[s]
$\hat{t}$	fictitious time	[s]
T	static temperature	[K]
u	conservative variable	[-]
U	design variable	[-]
v	velocity	[m/s]
w	artificial neuron weight	[-]
w	HH bump width	[-]
x_i^c	control point	[-]
x_i^M	HH bump center position	[-]
y_0	original airfoil y-coordinates	[-]
y_{mod}	deformed airfoil y-coordinates	[-]
z	neuron inner state	[-]

Ringraziamenti

Desidero ringraziare il Professor Luca Dede' per avermi seguito con pazienza e per i suoi preziosi consigli.

Ringrazio di cuore i miei genitori e mia sorella Francesca, per avermi sempre sostenuto e spronato, e per avermi supportato nei momenti più complicati.

Ringrazio i miei Nonni Aronna, Anna e Romano per i loro insegnamenti e per l'entusiasmo con il quale hanno condiviso i miei successi. Devo un ringraziamento particolare a mia nonna Anna, per tutti i suoi "*Studia!*" che mi hanno sempre strappato un sorriso.

Ringrazio Giacomo, Alberto, Julian, Tonio e Bart per aver reso più leggeri questi anni di studio: dalle partite a briscola e alla condivisione dello stress pre-esame.

Ringrazio Nicoló, Andrea, Paolo, Francesco A., Francesco T., Federico, Luca e Marco per non avermi mai fatto sentire la distanza. Nonostante sia mancato in molte occasioni, ad ogni ritrovo mi avete fatto sentire come se non me ne fossi mai andato.

Ringrazio infine Carmen, con la quale ho condiviso tutto questo percorso, per aver aver affrontato ogni sfida insieme, rendendo possibile il raggiungimento di questo risultato; per aver condiviso i momenti di difficoltà rendendoli tollerabili e per aver reso i momenti di felicità indimenticabili; per avermi sostenuto, supportato e sopportato, per essere diventata imprescindibile nella mia vita e con la quale spero di poter condividere ancora tantissime gioie.