UNIVERSITÀ DEGLI STUDI DI GENOVA

SCUOLA POLITECNICA

DIME

Dipartimento di Ingegneria Meccanica, Energetica,
Gestionale e dei Trasporti

TESI DI LAUREA MAGISTRALE

IN INGEGNERIA MECCANICA - ENERGIA E AERONAUTICA

Implementation of a Surrogate Model in CEASIOMpy using a Multi-Fidelity strategy for aerodynamic coefficients prediction

Relatori:

Chiar.^{mo} Prof. Alessandro Bottaro

Dott. Ing. Jan B. Vos

Dott. Giacomo Benedetti

Allievo:

Giacomo Gronda

Marzo 2025

Sommario

Questa tesi esplora l'applicazione di algoritmi di Machine Learning alle simulazioni CFD, con particolare attenzione allo sviluppo, test e validazione di due nuovi moduli per il software CEASIOMpy, sviluppato e mantenuto da Airinnova e CFS Engineering. I moduli consentono l'addestramento e l'uso di modelli surrogati basati su una strategia Multi-Fedeltà per la previsione di coefficienti aerodinamici chiave, fondamentali nella valutazione delle prestazioni aerodinamiche degli aeromobili. Il lavoro si inserisce nella fase preliminare della progettazione aeronautica, in cui numerose configurazioni con molteplici parametri sconosciuti devono essere analizzate. Tradizionalmente, la valutazione di queste configurazioni richiede centinaia di test in galleria del vento e migliaia di simulazioni CFD ad alta e bassa fedeltà, con costi computazionali elevati. Il modello surrogato sviluppato in questa tesi mira a ridurre tale onere computazionale, fornendo approssimazioni accurate a basso costo dei modelli ad alta fedeltà in funzione delle variabili di progetto. Il nuovo modulo si integra facilmente con i tool già esistenti di CEASIOMpy, in particolare con PyAVL, GMSH e SU2Run, elementi chiave per l'impostazione e l'esecuzione delle simulazioni CFD. Un aspetto innovativo del lavoro è l'implementazione di un approccio Multi-Fedeltà che supporta più livelli di dati a bassa fedeltà, aumentando la flessibilità e l'adattabilità del processo di modellazione surrogata. La tesi inizia inquadrando il problema e presentando un'analisi approfondita dello stato dell'arte. Successivamente, descrive la progettazione e l'implementazione del modulo, illustrandone i principi e il funzionamento. L'approccio Multi-Fedeltà combina l'efficienza delle simulazioni a bassa fedeltà con la precisione di quelle ad alta fedeltà, sfruttando tecniche avanzate di campionamento basate sul Design of Experiments (DoE). Vengono analizzati e impiegati diversi metodi di campionamento per esplorare efficacemente lo spazio degli input, garantendo che le simulazioni a bassa fedeltà coprano un ampio dominio, mentre quelle ad alta fedeltà si concentrino nelle regioni critiche. L'efficacia del modulo è dimostrata dalla capacità di generare modelli surrogati in grado di prevedere con accuratezza i coefficienti aerodinamici (C_L,C_D, etc.) con un notevole risparmio di risorse computazionali. La tesi si conclude con una valutazione critica dei risultati ottenuti e con suggerimenti per sviluppi futuri, tra cui l'estensione della strategia Multi-Fedeltà a problemi di ottimizzazione multi-obiettivo nella progettazione aeronautica.

Abstract

This thesis explores the application of Machine Learning algorithms to CFD simulations. Specifically, it focuses on the development, testing, and validation of two new modules for the Python-based software CEASIOMpy, developed and maintained by Airinnova and CFS Engineering. The two modules enables training and usage of surrogate models based on a Multi-Fidelity strategy to predict key aerodynamic coefficients, which are essential for evaluating aircraft performance. The work is situated in the context of the preliminary design phase of aircraft, where numerous configurations with a large number of unknown parameters must be analyzed. Testing these configurations traditionally requires hundreds of wind tunnel experiments and thousands of high and low-fidelity CFD simulations, which are computationally expensive. The surrogate model developed in this thesis aims to reduce the computational burden by providing low-cost yet accurate approximations of full-order models across different values of design variables. The new module integrates easily with existing CEASIOMpy tools, particularly with pyAVL, GMSH and SU2Run, key modules for setting up and performing CFD simulations. These simulations supply the high and low-fidelity datasets required to train the surrogate model. Notably, the Multi-Fidelity approach implemented in this work allows for multiple levels of low-fidelity data, further enhancing the flexibility and adaptability of the surrogate modeling process. The thesis begins by framing the problem and presenting a detailed state-of-the-art analysis. It then delves into the design and implementation of the module, explaining its underlying principles and functionality. The Multi-Fidelity approach combines the efficiency of low-fidelity simulations with the accuracy of high-fidelity ones. A critical component of this strategy is an adequate sampling technique, based on a Design of Experiments (DoE) methodology. Some sampling methods are described and employed to efficiently explore the input space, ensuring that low-fidelity simulations cover a broad domain, while high-fidelity simulations are concentrated in regions requiring greater accuracy. The effectiveness of the module is demonstrated through its ability to generate surrogate models that accurately predict aerodynamic coefficients (C_L,C_D, etc.) with significantly reduced computational resources. The thesis concludes with a critical evaluation of the results and suggestions for future work, including possible extensions of the Multi-Fidelity strategy to multi-objective optimization problems in aircraft design.

Ringraziamenti

Durante questi cinque anni (e mezzo) di università sono cambiato tanto, forse del tutto. Se ciò è successo, lo devo principalmente alle persone che mi hanno accompagnato e che ho incontrato lungo il percorso. Non credo che in queste poche righe, scritte anche un po' per consuetudine, riuscirò a ringraziare tutti come vorrei... ma voglio almeno citare le persone più importanti.

Ringrazio innanzitutto il professor Bottaro, che mi ha permesso di vivere questa esperienza fantastica. Poi Jan, che mi ha accolto nella splendida realtà di CFS Engineering, dove Giacomo, Roman e Alan sono sempre stati disponibili ad aiutarmi e a condividere risate durante le pause caffè. Un ringraziamento speciale a Giacomo, che sin da subito mi ha guidato in questa nuova avventura, sopportando le mie mille domande e facendomi sentire parte della vita a Losanna.

Il gruppo dell'uni: Lollo, Vitto, Matte, Fra, Cri, Anna, Peppo... e tutti gli altri. Compagni di crisi, di cirulle e di risate. (P.S. Non me ne vogliano gli altri, ma una menzione d'onore va a Fra, senza il quale oggi sarei ancora davanti ad Ansys).

Ringrazio poi mamma, papà e Franci, che mi hanno sostenuto in tutto. La Flo, che con la sua infinita pazienza e capacità di comprendermi mi ha accompagnato fino a questo traguardo.

Infine, i miei amici (in ordine alfabetico così non faccio torti: Buba, Cevvu, Ciccio, Giò, Lollo, Mazzo, Nico, Pie, Saffio, Senny), che non vedevo l'ora di ritrovare ogni volta che tornavo, perché sapevo che sarebbero stati sempre lì, pronti a ridere insieme e a rendere ogni momento speciale

Acknowledgements

The work carried out in this thesis was made in the framework of the Colossus project, funded by the European Union under Grant Agreement no 101097120. The Swiss participation in the Colossus project was supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 22.00609.

Contents

1. Introduction

During the past decades, artificial intelligence (AI) has achieved remarkable success, transforming various aspects of our lives. This transformation is evident in our daily routines, as habits evolve under the relentless advancement of AI. Interestingly, the origins of artificial intelligence date back much earlier than many might imagine. On August 31, 1955, the term "artificial intelligence" was coined in a proposal for a "2-month, 10-man study of artificial intelligence." The workshop, held a year later in July and August 1956, is widely considered the official birthdate of this groundbreaking field [1].

After its inception, the development of AI went through periods of slow progress, as the world had to prepare for its full potential. However, at the dawn of the 21st century, AI made a glorious comeback, driven by emerging massive computing power, the collection of colossal datasets (the big data phenomenon), and advances in data analytics.

Artificial intelligence has since diversified into various fields, one of the most prominent being Machine Learning (ML). Machine learning is the field of study that enables computers to learn from data and make decisions without explicit programming. It plays a critical role in today's data-driven world, with applications ranging from weather forecasting to energy exploration and environmental monitoring. For instance, analyzing data collected from satellites and sensors using ML has become essential for these tasks.

Another domain where ML has proven to be invaluable is Computational Fluid Dynamics (CFD). CFD is a branch of fluid mechanics that uses numerical analysis and algorithms to solve and analyze problems involving fluid flows. These simulations are performed using computers to calculate the interaction of liquids and gases with surfaces defined by boundary conditions [2]. The precision and accuracy of CFD solutions are directly proportional to computational costs, making time efficiency a critical factor in their implementation.

Given the importance of achieving accurate results in the shortest possible time, CFD naturally aligns with machine learning algorithms as a powerful ally. Machine learning can enhance CFD by reducing computational costs while maintaining solution accuracy, enabling engineers to explore larger design spaces and optimize processes more effectively.

This synergy forms the foundation of the present work, which focuses on the development of a module that combines CFD results with the synthesis and predictive capabilities of machine learning.

The objective is to create a robust tool tailored to the conceptual design phase of aircraft, where rapid exploration of configurations and performance predictions are essential. Specifically, the module constructs a surrogate model by integrating results from CFD simulations at different fidelity levels, effectively capturing the relationship between input parameters and aerodynamic outputs. This approach allows for a more efficient use of computational resources while preserving accuracy in aerodynamic predictions.

The developed module is part of the CEASIOMpy software environment, a multidisciplinary tool designed for aircraft conceptual design. Using the combination of CFD and machine learning, this module enhances the ability to quickly explore design alternatives and predict aerodynamic performance, providing valuable support in the early stages of aircraft development.

1.1. Aircraft Conceptual Design

The process of aircraft design is a very complex engineering task. It involves a combination of disciplines that must be blended to yield the optimal configuration that meets the given requirements. This is inevitably a highly iterative procedure consisting of alternating phases of synthesis and analysis. The integration of advanced flying control systems with aerodynamic and structural design is a unique and vital process. Aerodynamic performance must be balanced across the operating speed range, blended with powerplant characteristics, ensuring satisfactory control and stability, while at the same time minimizing structural penalties and subsequent increases in mass [3]. At the same time, technical aspects must be taken in conjunction with other critical factors such as safety, financial constraints, market demands, and environmental impact.

The entire design process benefits greatly from the advanced tools available in computer-aided design (CAD) and computer-aided manufacturing (CAM). The design process itself, as well as the designed configuration, evolves over time: this can be visualized through the product fidelity curve in Figure 1.1, which highlights the key phases of the design.

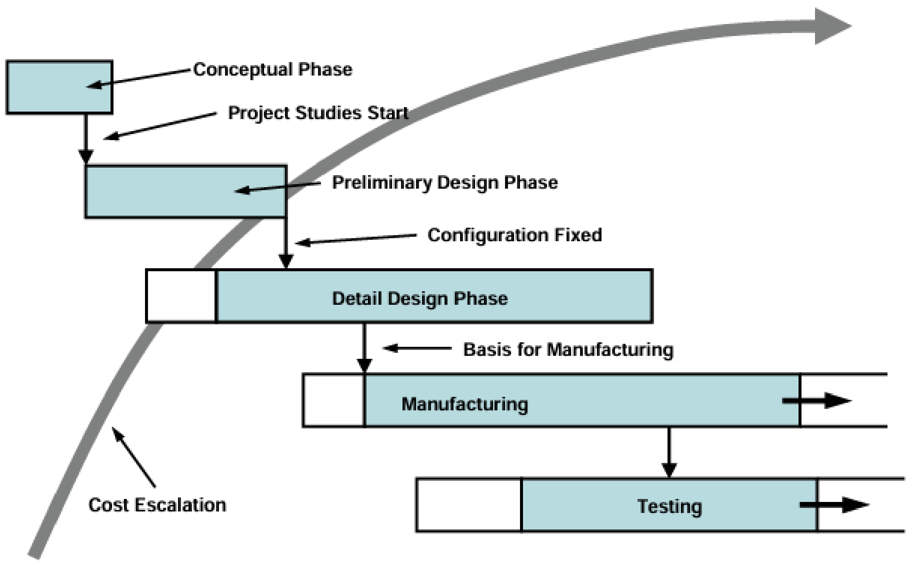


Figure 1.1. Impact of design phases on aircraft life-cycle cost

According to widely accepted descriptions, the aircraft design process is typically divided into three main phases:

- Conceptual Design
- Preliminary Design
- Detail Design

The goal of the *Conceptual Design* phase is to identify a feasible concept and optimize it as much as possible. During this phase, a large number of potential designs are generated and evaluated against the requirements. The focus is on exploring multiple solutions and narrowing them down to a few promising concepts for further analysis (e.g. fig. 1.2). This phase involves making key decisions based on limited information, which can greatly impact the outcome of the entire project. Computer simulations are used, and initial physical models can be built to validate ideas.

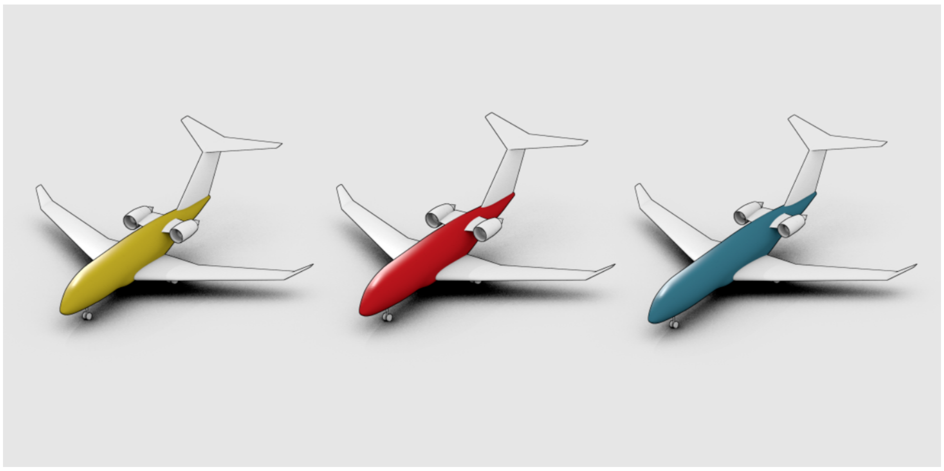


Figure 1.2. Conceptual design of family of three business jets from AGILE4.0 project [4]

The next step, known as the *Preliminary Design* phase, involves refining the selected concept to establish its feasibility. Detailed analyses and simulations are conducted to fine-tune the geometry while shaping all the subsystems. The aim is to finalize the design characteristics to a sufficient level of accuracy so that the aircraft can proceed to the manufacturing stage. At this point, *multidisciplinary design optimization (MDO)* becomes critical. Teams from distinct disciplines work in a distributed environment with individually defined variables, constraints, and computational tools. This phase plays a crucial role in verifying that the aircraft concept meets performance, safety, and cost requirements [5].

The final phase, the *Detail Design* phase, focuses on fully defining all components and systems of the aircraft in intricate detail. Manufacturing documentation is produced in this phase, and the tools used, while similar to earlier stages, must provide the highest possible accuracy. Only aircraft selected for production progress to this stage.

The earliest design process is a crucial stage, which commits up to 80 per cent of the life-cycle cost, even though these costs will not appear on financial records until much later. Any mistakes made during the conceptual or preliminary phases can have serious consequences—at best, resulting in design inefficiencies, and at worst, leading to project cancellation. Due to the high level of uncertainty at this stage, improvements to the conceptual design phase offer the greatest potential for innovation and cost savings. For this reason, the focus of this thesis is on enhancing the conceptual design stage by developing a surrogate model strategy. The objective is to improve the fidelity of simulation results while reducing computational time, thereby providing an effective tool for aircraft design optimization.

1.2. Aerodynamic Forces and Coefficients

In order to model and study the performance of an aircraft, it is necessary to know why it flies. The fundamental components of an aircraft are the wings, which are responsible for generating the lift that enables it to fly. The wings are bodies immersed in a fluid and, since the fluid can flow around the body, the "point of contact" is any point on the surface of the body, Thus, the transmission or application of mechanical forces between a solid body and a fluid occurs at every point on the surface. These forces are generated by two primary mechanisms: *pressure* and *shear stress* distribution over the body surface.

Regardless of the complexity of the body's shape, these two mechanisms are solely responsible for the resultant aerodynamic forces R and moments M acting on the body. The pressure p acts normal to the surface, while the shear stress τ acts tangentially [6]. Both p and τ have units of force per unit area (e.g., Newtons per square meter).

The net aerodynamic force R and moment M are obtained by integrating the distributions of p and τ over the entire body surface. These forces can be decomposed into components relative to the freestream velocity V_∞, as illustrated in Figure 1.3.

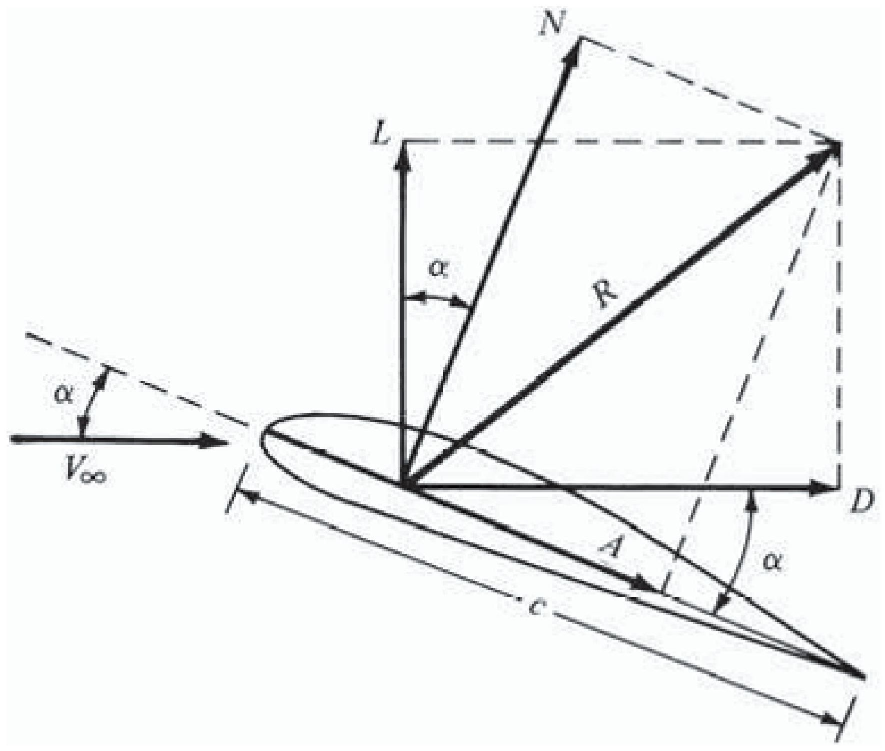


Figure 1.3. Resultant aerodynamic force and the components into which it splits

The chord c is the linear distance from the leading edge to the trailing edge of the body. The components are defined as:

- L: Lift, the component of R perpendicular to V_∞.
- D: Drag, the component of R parallel to V_∞.

Alternatively, R can be decomposed into components perpendicular and parallel to the chord line c of the body:

- N: Normal force, perpendicular to c.
- A: Axial force, parallel to c.

The angle of attack α is defined as the angle between V_∞ and c, resulting in the following relations between the two sets of force components:

$$L = N\cos\alpha - A\sin\alpha, \tag{1.1}$$

$$D = N\sin\alpha + A\cos\alpha. \tag{1.2}$$

The aerodynamic forces and moments on a body result from the integration of pressure and shear stress distributions over its surface.

Considering a two-dimensional body with an upper surface and a lower surface, as depicted in Figure 1.4.

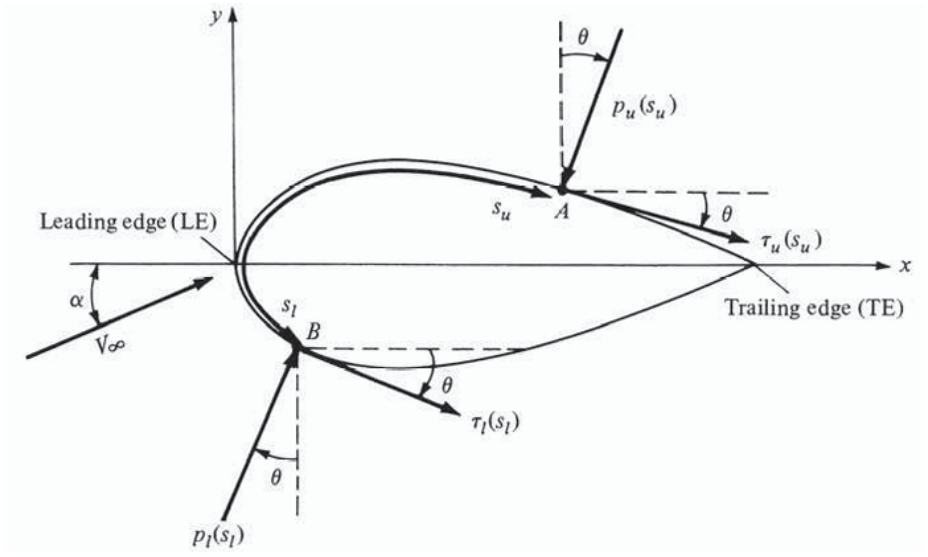


Figure 1.4. Nomenclature for the integration of pressure and shear stress distributions over a two-dimensional body surface

Let s_u and s_l be the distances along the upper and lower surfaces, respectively, measured from the leading edge. At a given point, the pressure is normal to the surface and is oriented at an angle θ relative to the perpendicular; shear stress is tangential to the surface and is oriented at the same angle θ relative to the horizontal. The primes on N and A denote force per unit span. The forces due to pressure p and shear stress τ on an elemental area dS can be expressed as follows:

On the upper surface:

$$dN'_u = -p_u \cos\theta\, ds_u - \tau_u \sin\theta\, ds_u, \tag{1.3}$$

$$dA'_u = -p_u \sin\theta\, ds_u + \tau_u \cos\theta\, ds_u. \tag{1.4}$$

On the lower surface:

$$dN'_l = p_l \cos\theta\, ds_l - \tau_l \sin\theta\, ds_l, \tag{1.5}$$

$$dA'_l = p_l \sin\theta\, ds_l + \tau_l \cos\theta\, ds_l. \tag{1.6}$$

The total normal force N and axial force A per unit span are obtained by integrating over the entire body surface from the leading edge (LE) to the trailing edge (TE):

$$N' = \int_{\text{LE}}^{\text{TE}} (dN_u + dN_l), \tag{1.7}$$

$$A' = \int_{\mathrm{LE}}^{\mathrm{TE}} (dA_u + dA_l)\,. \tag{1.8}$$

The aerodynamic moment about a reference point (e.g., leading edge) can also be derived by integrating the contributions of pressure and shear stress. By convention, moments that tend to increase α (pitch up) are positive, and moments that tend to decrease α (pitch down) are negative. For example, the moment per unit span about the leading edge is given by:

$$\begin{aligned} M'_{\mathrm{LE}} = &\int_{\mathrm{LE}}^{\mathrm{TE}} \Big[(p_u \cos\theta + \tau_u \sin\theta)x - (p_u \sin\theta - \tau_u \cos\theta)y\Big]\, ds_u \\ + &\int_{\mathrm{LE}}^{\mathrm{TE}} \Big[(-p_l \cos\theta + \tau_l \sin\theta)x + (p_l \sin\theta + \tau_l \cos\theta)y\Big]\, ds_l. \end{aligned} \tag{1.9}$$

From Equations 1.7 and 1.8, we see that the normal and axial forces on the body are due to the distributed loads imposed by the pressure and shear stress distributions. Moreover, these distributed loads generate a moment about the leading edge, as given by Equation 1.9. As we can see from Figure 1.3, the resultant force should be located on the body such that it produces the same effect as the distributed loads. The center of pressure is the point on the body where the resultant aerodynamic force acts, and it can be computed from the aerodynamic moment. For a given reference point, the center of pressure location x_{cp} relative to the chord length c is given by:

$$x_{cp} = -\frac{M}{L}. \tag{1.10}$$

The center of pressure is particularly useful in analyzing stability and control, as it represents the point at which the aerodynamic force can be assumed to act.

These forces can be expressed in dimensionless form using aerodynamic coefficients, which are functions of freestream conditions and reference quantities. The lift, drag, and moment coefficients, along with the center of pressure, play a fundamental role in the study and design of aerodynamic systems.

Dimensionless Aerodynamic Coefficients

The aerodynamic characteristics of a body are more fundamentally described by the force and moment coefficients than by the actual forces and moments themselves.

Lift, drag, and moment depend on the density of the air ρ_∞, the relative velocity V_∞, the air's viscosity and compressibility (μ_∞ and a_∞, where a_∞ is the speed of sound), the surface area over which the air flows S, the shape of the body, and the body's inclination to the flow

α.

$$L = L(\rho_\infty, V_\infty, S, \alpha, \mu_\infty, a_\infty), \tag{1.11}$$

$$D = D(\rho_\infty, V_\infty, S, \alpha, \mu_\infty, a_\infty), \tag{1.12}$$

$$M = M(\rho_\infty, V_\infty, S, \alpha, \mu_\infty, a_\infty). \tag{1.13}$$

The aerodynamic forces and moments are often expressed in dimensionless form using the freestream dynamic pressure q_∞ and appropriate reference quantities. The freestream dynamic pressure is defined as:

$$q_\infty = \frac{1}{2}\rho_\infty V_\infty^2, \tag{1.14}$$

where ρ_∞ is the freestream density and V_∞ is the freestream velocity. Using q_∞, the force and moment coefficients are defined as follows:

$$C_L = \frac{L}{q_\infty S}, \tag{1.15}$$

$$C_D = \frac{D}{q_\infty S}, \tag{1.16}$$

$$C_M = \frac{M}{q_\infty S l}, \tag{1.17}$$

where S is the reference area and l is the reference length.

The choice of S and l depends on the geometry of the body. For example, for an airfoil, S is typically the planform area, and l is the chord length. These coefficients allow for a consistent comparison of aerodynamic performance across different shapes and conditions.

Lift and drag coefficients play a strong role in the preliminary design and performance analysis of airplanes: they are much more than just the conveniently defined terms discussed so far, they are fundamental quantities, which make the difference between intelligent engineering and simply groping in the dark.

Let us define two fundamental dimensionless numbers for the aerodynamics context, namely the *Reynolds* and *Mach* numbers:

$$Re = \frac{\rho V L}{\mu}, \tag{1.18}$$

$$M = \frac{V}{a}, \tag{1.19}$$

The method of dimensional analysis (a very powerful and elegant approach used to identify governing nondimensional parameters in a physical problem) leads to the following result [6]. For a given body *shape*, we have:

$$C_L = \phi(\alpha, Re, M_\infty), \tag{1.20}$$

$$C_D = \psi(\alpha, Re, M_\infty), \tag{1.21}$$

$$C_M = \chi(\alpha, Re, M_\infty). \tag{1.22}$$

The variation of the aerodynamic coefficients C_L, C_D, and C_M with the angle of attack α provides valuable insights into the behavior of the aerodynamic forces and moments for a given body. These variations are critical for understanding and predicting the performance and stability of aircraft.

For the lift coefficient C_L, its variation with α follows a characteristic trend. In the linear region of the lift curve, C_L increases approximately linearly with α, as shown in Figure 1.5. The slope of this linear portion, known as the lift-curve slope, is denoted by a_0 and represents the rate of change of C_L per degree of α. Theoretical and experimental results agree closely in this region. Notably, there exists a specific angle of attack, $\alpha_{L=0}$, at which the lift coefficient becomes zero. For positively cambered bodies, $\alpha_{L=0}$ is negative, while for symmetric shapes, $\alpha_{L=0} = 0°$.

As α increases beyond the linear range, the lift coefficient reaches a maximum value, $(C_L)_{\max}$, before decreasing due to flow separation. This phenomenon, known as stall, marks the onset of nonlinear aerodynamic behavior and significantly impacts performance and control.

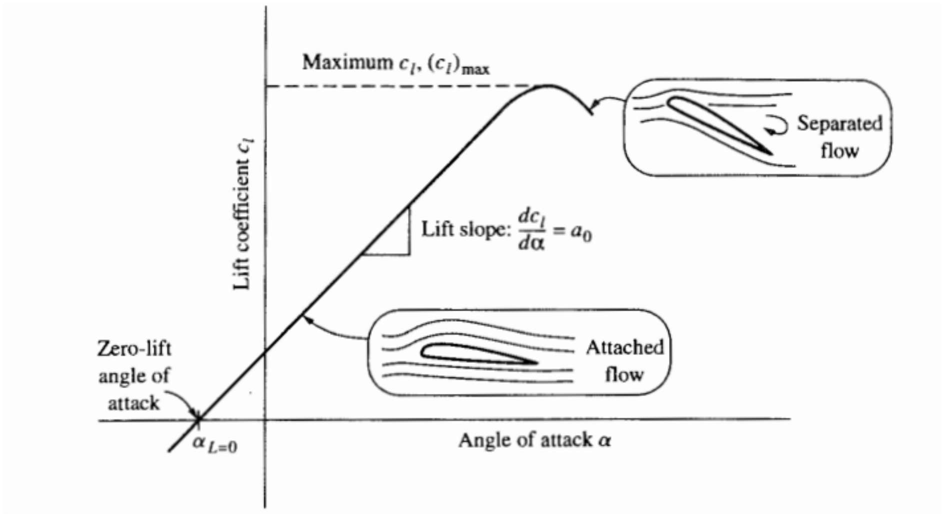

Maximum c_l, $(c_l)_{\max}$
Separated flow
Lift coefficient c_l
Lift slope: $\frac{dc_l}{d\alpha} = a_0$
Attached flow
Zero-lift angle of attack
$\alpha_{L=0}$
Angle of attack α

Figure 1.5. Sketch of a generic lift curve

The drag coefficient C_D also varies with α, as depicted in Figure 1.6. For small angles of attack, C_D remains relatively low and shows a nearly parabolic relationship with C_L. The minimum drag coefficient, $(C_D)_{\min}$, occurs at a specific angle of attack where the drag is minimized. As α increases, C_D rises sharply due to the growing influence of pressure drag, which results from flow separation over the body.

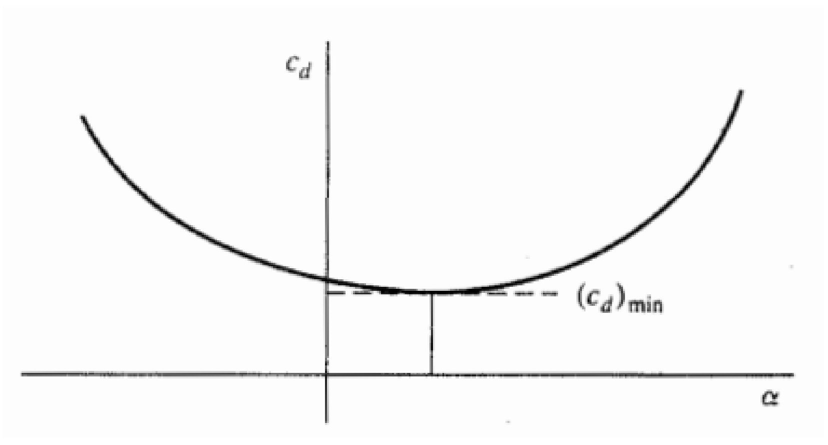
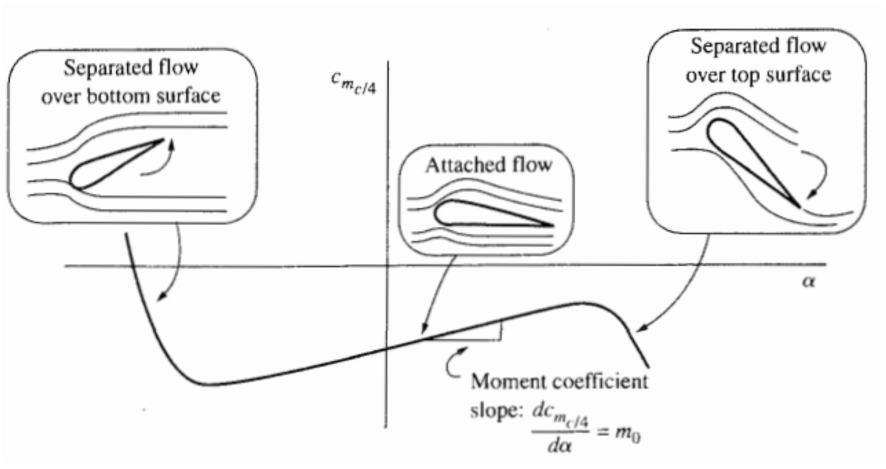


Figure 1.6. Sketch of a generic drag curve

The moment coefficient C_M around a specified reference point (e.g., the quarter-chord point) provides insight into the aerodynamic stability of the body. Figure 1.7 illustrates the variation of C_M with α. Within the linear range, C_M exhibits a nearly constant slope, $m_0 = dC_M/d\alpha$. However, as α increases and flow separation occurs, C_M becomes nonlinear, reflecting the destabilizing effects of separated flow.



Separated flow over bottom surface
$c_{m_{c/4}}$
Attached flow
Separated flow over top surface
α
Moment coefficient slope: $\frac{dc_{m_{c/4}}}{d\alpha} = m_0$

Figure 1.7. Sketch of a generic moment curve

1.3. Machine Learning for Aerodynamic Forces Prediction

The design and optimization of an aircraft require the evaluation of aerodynamic forces across a vast range of configurations. This analysis is fundamental for assessing performance limits, ensuring safety, and improving efficiency.

Computational Fluid Dynamics (CFD) has revolutionized this process by reducing dependence on wind tunnel testing and flight experiments. However, high-fidelity CFD simulations remain computationally expensive, especially when extensive parametric studies are needed. Machine Learning (ML) provides an effective solution by learning complex aerodynamic behaviors from existing datasets, significantly reducing the number of required simulations.

By integrating ML techniques, engineers can efficiently predict aerodynamic forces such as lift, drag, and moments, capturing the nonlinear dependencies on key flight parameters like angle of attack, Mach number, and altitude. Additionally, ML facilitates multi-fidelity modeling by combining data from high- and low-fidelity simulations, enhancing prediction accuracy and optimizing computational resources.

This capability enables rapid aerodynamic assessments, making real-time design optimization and flight control applications more feasible. As a result, ML is becoming an essential tool in modern aerospace engineering, complementing traditional simulation-based approaches.

2. Machine Learning applied to CFD

The potential of Machine Learning applied to CFD has already been briefly introduced in the previous chapter. This chapter will explore the state of the art, the theory and the main descriptive equations of both technologies. Particular attention will be paid to the software used to run the simulations and the algorithms implemented in the python codes behind the modules developed.

2.1. CFD Fundamentals

Computational Fluid Dynamics, usually abbreviated as CFD, is a branch of fluid mechanics that uses numerical analysis and algorithms to solve and analyse problems involving fluid flows. Computers are used to perform the calculations necessary to simulate the interaction of liquids and gases with surfaces defined by boundary conditions.

CFD has evolved from a mathematical curiosity to an indispensable tool in almost every branch of fluid dynamics, and it is fair to say that this technology has had an enormous impact on both commercial and military aircraft design.

CFD makes it possible to model the airflow around aircraft to predict lift and drag, known as external aerodynamics; for that and many other reasons it's increasingly being used by industries in multi-disciplinary design and analysis of aerospace products. CFD can also simulate complex systems within the aircraft's interior, such as cabin air circulation, to predict air quality.

The complete CFD process is divided into many steps, for instance we can refer to Figure 2.1 and take a brief look at the five showed steps.

To be correct, before the five steps shown, a *preliminary step* is necessary in which the flow problem is formulated. During this preliminary step, several questions must be answered: What is the objective of the analysis? What is the easiest way to obtain those objectives? What dimensionality of the spatial model is required? What temporal modeling is appropriate? (steady or unsteady), what is the nature of the viscous flow? (inviscid, laminar, turbulent), etc... Then, the choice of approach depends on the desired fidelity of the solution. For example, simpler methods, such as *potential flow models*, may be sufficient to capture basic aerodynamic characteristics in inviscid and irrotational flows. In contrast, higher-fidelity methods, such as the *Euler equations* or *Reynolds-Averaged Navier-Stokes (RANS) equations*, are necessary to account for compressibility effects or the influence of viscous forces and turbulence.

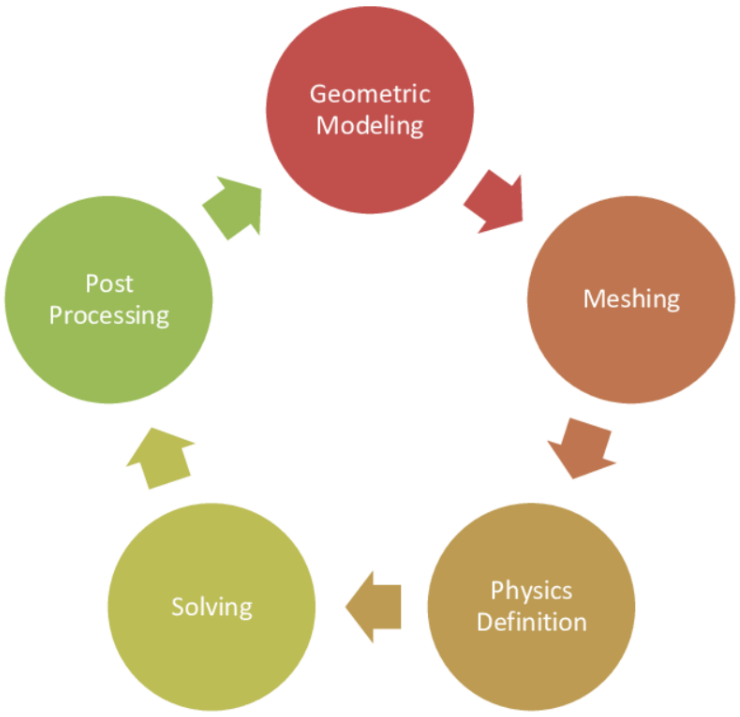


Figure 2.1. General Steps to perform CFD Analysis

After formulating the flow problem, the first step is to *model the geometry*, usually using CAD software. Simplifications balance accuracy with computational effort, while the flow domain is defined with boundaries aligning to the body or open for flow entry and exit. The design must match the generation of the grid and its topology.

The second step is *meshing*, which consists of discretizing the domain into a grid, where the governing fluid equations are solved within each cell. To improve computational efficiency, the meshing process can be performed in parallel by distributing different cell groups across multiple processors, whether on a high-performance computing (HPC) system or a standard multi-core architecture [7].

The third fundamental step in CFD, *physics definition*, involves specifying the physical conditions and parameters for the simulation. This includes defining boundary and initial conditions for the flow domain, selecting appropriate models (e.g., turbulence or chemistry), and determining the simulation strategy, such as time-marching or space-marching approaches. Additionally, input files containing grid and flow data, along with boundary condition information, must be prepared to ensure consistency with the simulation setup.

Then, during the *solving* step, the simulation runs using different possible settings, such as interactive or batch processing, and can also be distributed across multiple processors. As the simulation progresses, the results are monitored to check if a "converged" solution has been reached, meaning the iterative process has stabilized.

At the end, once a converged solution is reached, the *post-processing* phase involves extracting key flow properties such as thrust, lift, and drag from the computed flowfield. These results can then be compared with analytical, computational, or experimental data to assess their accuracy and validity, though validation is not always feasible in every case.

2.1.1. Overview of CFD Approaches and Principal Equations

Every CFD tool, whether commercial or open source, uses a mathematical model and numerical method to predict the desired flow physics. The most common CFD tools are based on the Navier-Stokes (N-S) equations. These equations are a set of coupled partial differential equations that describe the relationships between velocity, pressure, temperature, density, and the forces acting on a moving fluid:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \quad \text{(Continuity Equation)} \tag{2.1}$$

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) = -\nabla p + \nabla \cdot \tau + \mathbf{S}_u \quad \text{(Momentum Equation)} \tag{2.2}$$

$$\frac{\partial (\rho e_t)}{\partial t} + \nabla \cdot (\rho e_t \mathbf{u}) = -\nabla \cdot \mathbf{q} - \nabla \cdot (p\mathbf{u}) + \tau : \nabla \mathbf{u} + \mathbf{S}_{e_t} \quad \text{(Energy Equation)} \tag{2.3}$$

where ρ is the density, $\mathbf{u}$ is the velocity vector, p is the pressure, τ is the viscous stress tensor, e_t is the total energy per unit volume, $\mathbf{q}$ is the heat flux, and $\mathbf{S}_u$ and $\mathbf{S}_{e_t}$ are the source terms for momentum and energy, respectively.

The complexity of solving the N-S equations depends on the assumptions and approximations applied to the flow problem. In this study, three levels of fidelity have been considered for simulation: the *panel method*, the *Euler equations*, and the *Reynolds-Averaged Navier-Stokes (RANS) equations*.

Panel Methods

This approach simplifies the governing equations by assuming potential flow, where the fluid is inviscid and irrotational. The velocity field can be expressed as the gradient of a scalar potential ϕ, which satisfies Laplace's equation:

$$\nabla^2 \phi = 0 \tag{2.4}$$

Panel methods are computationally efficient for predicting aerodynamic properties, such as lift and drag, but do not capture viscous effects or rotational flows.

Euler Equations

The Euler equations are derived by neglecting viscous terms in the N-S equations, assuming inviscid flow. These equations consist of the conservation of mass, momentum, and energy:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0 \quad \text{(Continuity Equation)} \tag{2.5}$$

$$\frac{\partial (\rho \vec{u})}{\partial t} + \nabla \cdot (\rho \vec{u}\vec{u}) + \nabla p = 0 \quad \text{(Momentum Equation)} \tag{2.6}$$

$$\frac{\partial (\rho e_t)}{\partial t} + \nabla \cdot [(\rho e_t + p)\vec{u}] = 0 \quad \text{(Energy Equation)} \tag{2.7}$$

Euler equations are suitable for capturing shock waves and flow features in high-speed regimes but cannot model boundary layers or viscous effects.

Reynolds-Averaged Navier-Stokes (RANS) Equations

To model turbulent flows, the N-S equations are averaged over time, introducing additional terms known as Reynolds stresses. The resulting RANS equations are:

$$\frac{\partial \bar{\rho}}{\partial t} + \nabla \cdot (\bar{\rho} \bar{\vec{u}}) = 0 \quad \text{(Continuity Equation)} \tag{2.8}$$

$$\frac{\partial (\bar{\rho} \bar{\vec{u}})}{\partial t} + \nabla \cdot (\bar{\rho} \bar{\vec{u}}\bar{\vec{u}}) + \nabla \bar{p} = \nabla \cdot \left(\bar{\tau} - \overline{\rho \vec{u}' \vec{u}'}\right) + \bar{\rho}\vec{g} \quad \text{(Momentum Equation)} \tag{2.9}$$

Turbulence models, such as *Spalart-Allmaras* $k-\varepsilon$ or $k-\omega$, are introduced to close the equations by relating Reynolds stresses to mean flow quantities. RANS provides a balance between computational cost and accuracy, making it a standard approach for solving complex engineering problems involving viscous flows.

2.1.2. Meshing

Mesh generation is a pre-processing step in computational fluid simulations that involves discretizing the domain of interest into smaller elements [8]. In this process, the geometry is divided into a collection of simple shapes, such as triangles or quadrilaterals in 2D, and tetrahedra or hexahedra in 3D. These elements are arranged so that they share faces, edges, or nodes at their intersections, ensuring a continuous representation of the domain. The quality and resolution of the mesh significantly impact the accuracy and efficiency of the simulation.

The process of mesh generation can be broadly classified into two categories based on the topology of the elements that fill the domain. These two basic categories are known as *structured* and *unstructured* meshes. A *structured mesh* is defined as a set of quad or hexahedral elements with an implicit connectivity of the points in the mesh. The structured mesh generation for complex geometries is a time-consuming task due to the possible need of breaking the domain manually into several blocks depending on the nature of the geometry. An *unstructured mesh* is defined as a set of elements, commonly triangles or tetrahedra, with explicitly defined connectivity. The unstructured mesh generation process involves two main steps: point creation and the definition of connectivity between these points.

Flexibility and automation make unstructured meshes a favorable choice. In fact, it is well known that they have an advantage over structured meshes when handling complex geometries.

However, the accuracy of the solution may be lower compared to structured meshes due to the presence of *skewed elements* in sensitive regions like boundary layers.

In particular, this type of mesh is not recommended for RANS simulations, as it struggles to capture viscous effects, which involve the interaction between the fluid and walls in the boundary layer region.

To address this issue and combine the advantages of both structured and unstructured meshes, a common approach is *hybrid mesh* generation. In a hybrid mesh, the viscous region is filled with prismatic or hexahedral cells, while the rest of the domain is discretized with tetrahedral cells. This approach not only reduces the total number of elements compared to a fully unstructured mesh with similar resolution but also improves the accuracy in viscous regions due to the rectangular shape of the elements.

Since numerical simulations have proven critical for aircraft design and manufacture, mesh quality evaluation is a very important part of the process. Unfortunately, defining whether a mesh is good or poor quality is a very difficult task. The many meshing programs that have been developed in recent years allow the user to vary an infinite number of properties, thickening elements, changing the shape of elements or adapting elements to the flow. Thereby CFD professionals with experience are required to frequently check and evaluate the current mesh quality and make timely adjustments.

What is certain is that computational time is an important factor to be taken into account. The use of very dense meshes does not always lead to better solutions. In fact, once a certain number of elements is reached into which the domain is divided, the solution of the calculation does not improve significantly and the only thing that is achieved by further increasing the number of mesh elements is to increase the time required for the calculation. In order to find the right number of mesh elements, it is necessary to carry out what is known as *mesh independence*, i.e. the study of the evolution of the performance parameters characteristic of the case being studied as the number of elements increases.

Mesh quality is crucial in computational simulations. Several metrics help evaluate and optimize mesh properties:

- *Aspect Ratio (AR):* Measures the shape quality of elements, particularly tetrahedra. It is defined as:

$$\mathrm{AR} = \frac{l_{\max}}{h_{\min}}.$$

A lower AR improves numerical stability.

- *Orthogonality:* Evaluates the angle between adjacent elements. It is given by:

  $$\cos\theta = \frac{\vec{s}\cdot\vec{n}}{|\vec{s}||\vec{n}|}.$$

  Higher orthogonality enhances solver convergence.

- *Skewness:* Measures deviation from ideal shapes. Defined as:

  $$\text{Skewness} = 90^\circ - \text{Minimum Angle}.$$

  Values above 60°-70° can cause solver issues.

- *Additional Metrics:* Include volume and Jacobian determinant, ensuring correct element mapping.

These criteria help improve mesh reliability in simulations.

The process of grid generation is in general extremely complex and requires dedicated software tools to help in defining grids that follow the solid surfaces. For this thesis work, the open-source software GMSH© was used, which is automatically installed together with the CEASIOMpy packages. GMSH© is an automatic 3D finite element mesh generator with build-in pre- and post-processing facilities [9]. Developed by Christophe Geuzaine and Jean-François Remacle since 1997, GMSH© contains 4 modules: for geometry description, meshing, solving and post-processing.

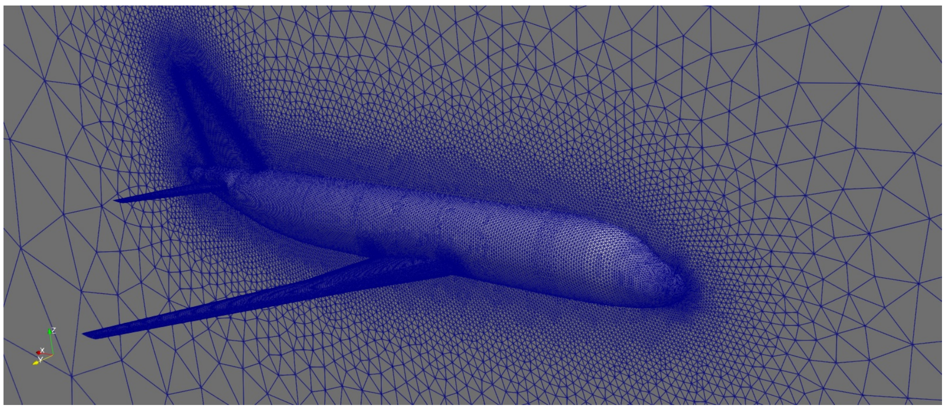


Figure 2.2. Example of an euleran mesh generated with GMSH software

In combination with the meshing module, Pentagrow© from SUMO© was used to generate the *prism layer* for RANS simulations. Pentagrow is a command line tool that generates hybrid pentahedral-tetrahedral meshes around existing surface meshes. Prism layer is so called because of the way in which the faces of the core mesh are projected onto the solid boundary, resulting in prism-shaped cells. The resolution of the boundary layer requires the grid to be clustered in a direction normal to the surface, with the distance of the first grid point from the wall being well within the laminar sublayer of the boundary layer. For turbulent flows, if a *wall-resolving* approach is used, the first point on the wall should have a y^+ value of less than 1.0. However, for a *wall-function* approach, a higher y^+ (typically > 30) value is typically required.

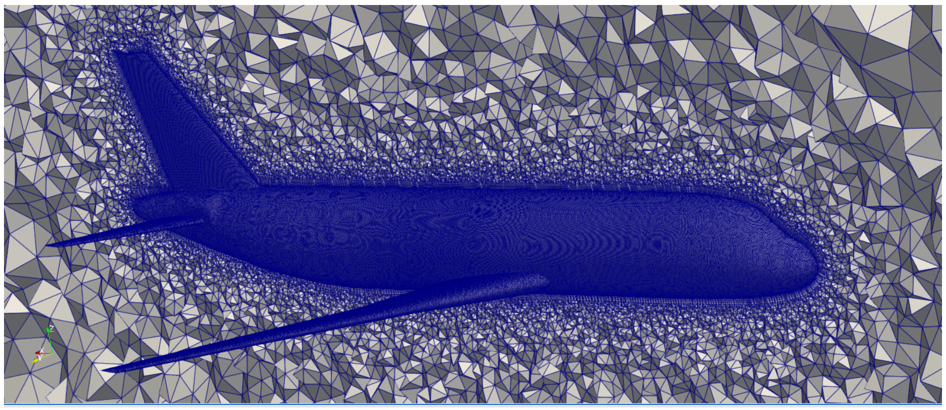


Figure 2.3. Example of a RANS mesh generated with GMSH and Pentagrow software

2.1.3. Simulations

Once the mesh is generated, several important choices must be made before proceeding with numerical simulations. The selection of the governing equations, turbulence model, and numerical schemes significantly influences the accuracy and stability of the results. One key decision is whether to solve the inviscid Euler equations or the viscous Navier-Stokes equations. For high-Reynolds-number flows, turbulence modeling is essential, and different approaches such as RANS (Reynolds-Averaged Navier-Stokes), LES (Large Eddy Simulation), or hybrid methods can be employed. Additionally, boundary conditions must be carefully defined to ensure a realistic representation of the physical problem, while grid refinement studies help assess the sensitivity of the solution to mesh resolution.

To perform the simulations, we used the SU2© software suite from Stanford University, which is automatically installed with the CEASIOMpy packages.

SU2© is a suite of tools written in C++ for the numerical solution of partial differential equations (PDE) and performing PDE constrained optimization. The core of the suite is a Reynolds-averaged Navier–Stokes (RANS) solver capable of simulating the compressible, turbulent flows that are representative of many problems in aerospace and mechanical engineering [10].

The SU2 software package includes many tools:

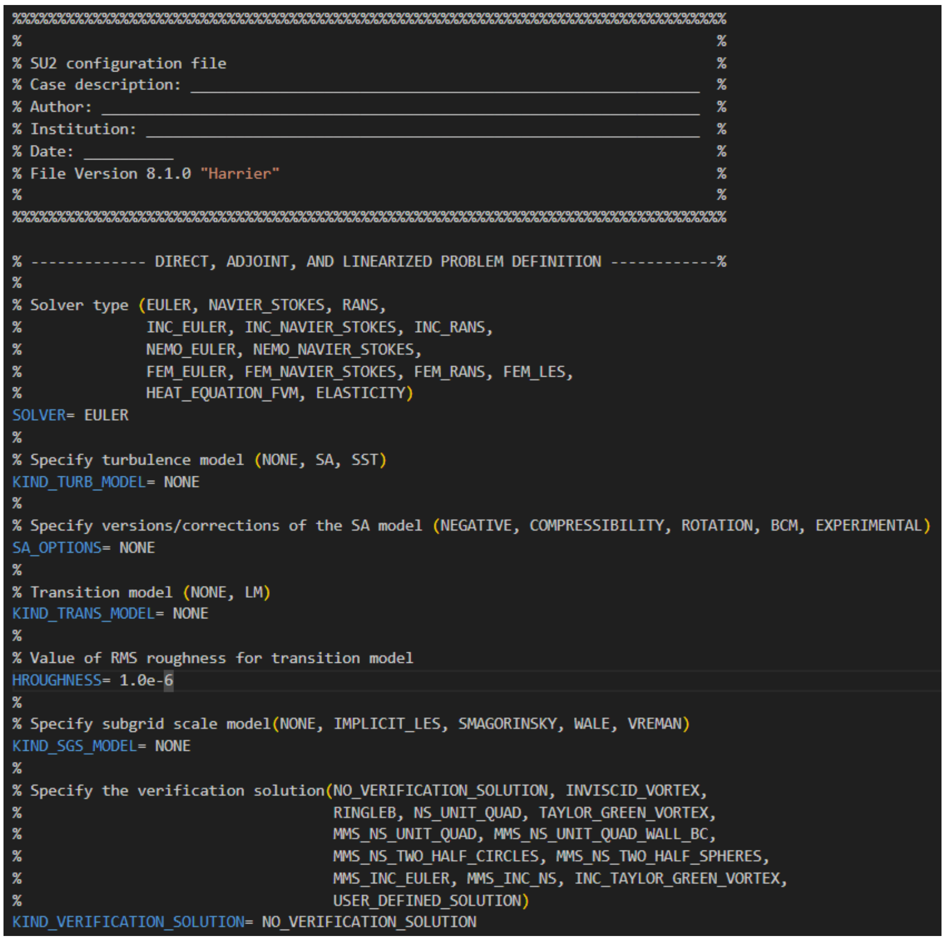
- *SU2 MSH*: the tool for grid adaptation based on different techniques.
- *SU2 CFD*: the module to solve direct, adjoint, and linearized problems for the Euler, Navier-Stokes, and Reynolds-Averaged Navier-Stokes (RANS) equation sets.
- *SU2 DOT*: this module calculates the partial derivative of a functional, taking into account the variation of the aerodynamic surface.
- *SU2 DEF*: calculates the geometrical deformation of surfaces.
- *SU2 GEO*: the tool to preprocess geometrical information; it is used to compute the geometric constraints.
- *SU2 SOL*: the tool that generates output files with volume and surface solutions.

One of the main benefits of SU2 is that it is an open-source software package. The software is also well-documented and well-maintained, with a strong community of users and developers who are constantly working to improve the software and add new features.

Using SU2, both Euler and RANS simulations were conducted. The turbulence model (Spalart-Allmaras), CFL number, boundary conditions, and number of iterations were defined in the configuration file.

Euler simulations were performed on an unstructured mesh, while RANS simulations used a hybrid mesh with prism layers to better capture boundary layer effects. This setup allowed for an assessment of the impact on numerical accuracy and computational cost.

Convergence criteria were carefully monitored by tracking residuals, lift and drag coefficients, and other aerodynamic parameters. Mesh refinement studies were also conducted to ensure grid independence of the results, reducing numerical errors. Additionally, post-processing tools within SU2 were used to analyze the aerodynamic coefficients, visualize flow fields using pressure and velocity contours, and evaluate convergence behavior through residual plots. This ensured a comprehensive assessment of the simulation results, allowing for iterative improvements in mesh quality, turbulence modelling, and solver settings to enhance the reliability and accuracy of the numerical predictions.



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
% SU2 configuration file                                                       %
% Case description: _________________________________________________________  %
% Author: ___________________________________________________________________  %
% Institution: ______________________________________________________________  %
% Date: __________                                                             %
% File Version 8.1.0 "Harrier"                                                 %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ------------ DIRECT, ADJOINT, AND LINEARIZED PROBLEM DEFINITION ------------%
%
% Solver type (EULER, NAVIER_STOKES, RANS,
%              INC_EULER, INC_NAVIER_STOKES, INC_RANS,
%              NEMO_EULER, NEMO_NAVIER_STOKES,
%              FEM_EULER, FEM_NAVIER_STOKES, FEM_RANS, FEM_LES,
%              HEAT_EQUATION_FVM, ELASTICITY)
SOLVER= EULER
%
% Specify turbulence model (NONE, SA, SST)
KIND_TURB_MODEL= NONE
%
% Specify versions/corrections of the SA model (NEGATIVE, COMPRESSIBILITY, ROTATION, BCM, EXPERIMENTAL)
SA_OPTIONS= NONE
%
% Transition model (NONE, LM)
KIND_TRANS_MODEL= NONE
%
% Value of RMS roughness for transition model
HROUGHNESS= 1.0e-6
%
% Specify subgrid scale model(NONE, IMPLICIT_LES, SMAGORINSKY, WALE, VREMAN)
KIND_SGS_MODEL= NONE
%
% Specify the verification solution(NO_VERIFICATION_SOLUTION, INVISCID_VORTEX,
%                                   RINGLEB, NS_UNIT_QUAD, TAYLOR_GREEN_VORTEX,
%                                   MMS_NS_UNIT_QUAD, MMS_NS_UNIT_QUAD_WALL_BC,
%                                   MMS_NS_TWO_HALF_CIRCLES, MMS_NS_TWO_HALF_SPHERES,
%                                   MMS_INC_EULER, MMS_INC_NS, INC_TAYLOR_GREEN_VORTEX,
%                                   USER_DEFINED_SOLUTION)
KIND_VERIFICATION_SOLUTION= NO_VERIFICATION_SOLUTION
```

Figure 2.4. Example of a Config File where all SU2 settings are written

2.2. Machine Learning Fundamentals

Machine learning (ML) is a specialized field within artificial intelligence (AI). While both disciplines are closely related, they have distinct objectives: ML aims at improving systems performance using self-learning algorithms, while AI tries to mimic natural intelligence solving complex problems and enabling decision making (although not maximizing the system efficiency) [11]. In essence, ML focuses on enabling systems to improve autonomously through experience, whereas AI seeks to emulate human-like cognitive processes. Arthur Samuel, famous for developing a checkers-playing program, provided a foundational definition: *"Machine learning is defined as the field of study that gives computers the ability to learn without being explicitly programmed."* Complementing this, another perspective describes ML as *"the technique that improves system performance by learning from experience via computational methods [12]."*

In the context of computer systems, experience is typically represented as data. The primary task of machine learning is to design algorithms that can process this data to build predictive models. These models, trained on existing data, enable the system to make accurate predictions on new, unseen observations. Thus, while computer science as a whole studies algorithms in general, machine learning specifically focuses on the development and refinement of algorithms capable of learning autonomously from data.

Currently, machine learning capabilities are advancing at an incredible rate, and fluid mechanics is beginning to tap into the full potential of these powerful methods. Many tasks in fluid mechanics, such as reduced-order modelling, shape optimization, uncertainty quantification, and feedback control, may be posed as optimization and regression tasks. Machine learning can dramatically improve optimization performance and reduce convergence time. Moreover, it is also employed for dimensionality reduction, identifying low-dimensional manifolds and discrete flow regimes, which significantly enhance understanding.

Artificial intelligence (AI) and machine learning (ML) have been introduced in the aerospace industry for various applications connected to the reduction of aircraft's environmental impact, including data interpretation, system management, customer service or aircraft modelling and to generate new high-fidelity databases at a reduced (economic and CPU) cost , solving problems of optimization, flow control, or even providing optimal sensors distributions for solid mechanics or aeroelasticity applications [11]

Machine learning models can also be viewed as an evolution of statistical models. Both statistical and ML models start with a dataset, which may vary in size and sparsity, impacting the model's accuracy. However, the key difference lies in their objectives: statistics draws population inferences from a sample, and machine learning finds generalizable predictive patterns [13].

Machine learning models differ from simpler statistical models in that they do not necessarily assume a specific form for the data. Instead, they are data-driven, learning relationships directly from the data with greater flexibility. Additionally, these models can be significantly more complex, often at the expense of interpretability. For instance, neural networks can have multiple layers, each with varying numbers of nodes, and utilize different activation functions [14].

These models are needed when, starting from a dataset of inputs, we aim to predict outputs without necessarily knowing the underlying relationship. This approach is often described as "black-box," meaning that while we do not understand how the model relates inputs to outputs, we achieve satisfactory results, which is sufficient for our purposes. A *surrogate model* is a clear example of a machine learning model, and it will be discussed in greater detail in the following chapter.

2.2.1. Machine Learning Categories

Machine learning relies on a variety of algorithms to solve data-related problems. Data scientists often highlight that there is no universal, one-size-fits-all algorithm, and the choice depends on the task and the data involved. Generally, machine learning algorithms can be categorized into three main types: *supervised learning*, *unsupervised learning*, and *reinforcement learning*.

Supervised learning involves training a model to map inputs to outputs using labeled data, where the correct answers are provided. This approach enables the algorithm to generalize from examples to make predictions on new data. Common supervised learning algorithms include regression, which predicts continuous numerical values (e.g., house prices), and classification, which predicts discrete categories (e.g., identifying spam emails).

Unsupervised learning, in contrast, works with unlabeled data. The algorithm identifies patterns or relationships without explicit guidance. Clustering is a common unsupervised method used to group similar data points, such as segmenting customers for marketing purposes or analyzing biological data like DNA sequences. Another example is anomaly detection, which identifies deviations from normal patterns, often used in fraud detection and quality control.

Reinforcement learning is a less common category that focuses on enabling an agent to learn by interacting with its environment and maximizing cumulative rewards. It is used in applications like robotics and video games, where systems learn from their actions to improve performance.

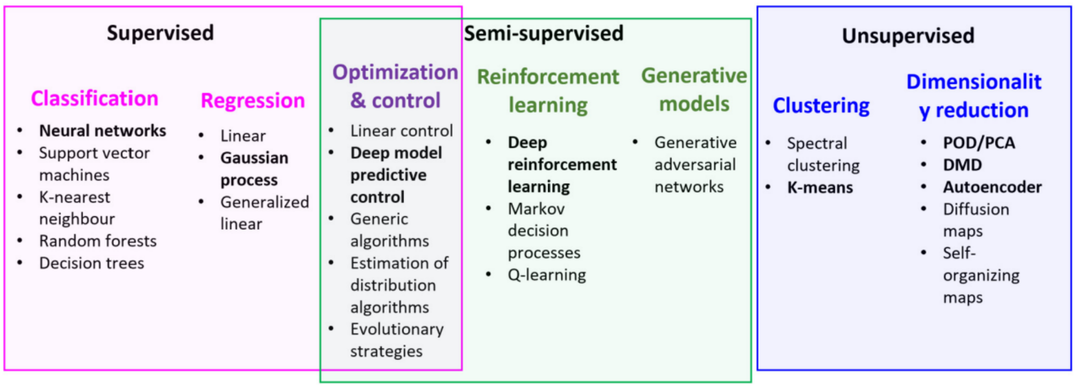


Figure 2.5. Machine learning methods: a general overview. Classification extracted from [15]. In bold, the most popular techniques in the field of aerospace engineering

2.2.2. Machine Learning General Framework

A machine learning pipeline refers to the sequence of steps or processes involved in developing and applying a machine learning model. This pipeline is not universal, for this report we follow a general framework illustrated in 2.6.

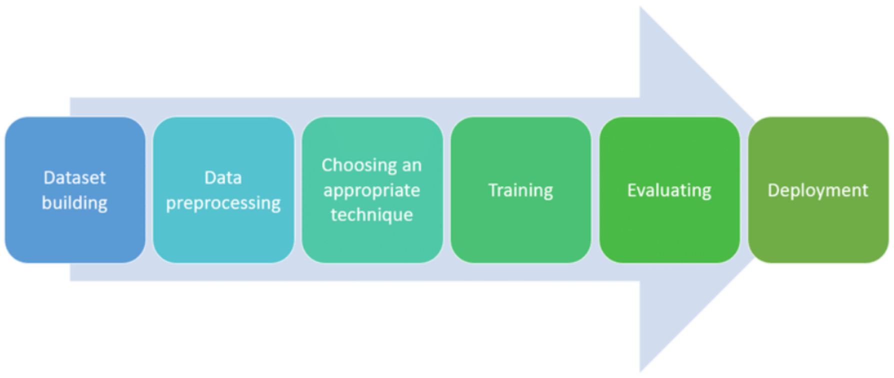


Figure 2.6. A general set of steps to define a machine learning pipeline

Problem Definition

This can be considered "step zero". Defining the problem in a meaningful way is critical, as poorly defined problems lead to useless outcomes. It is essential to identify appropriate independent variables and define the valid range for each parameter, understanding how these choices influence the solution. This requires knowledge of the physics and mathematics underlying the problem, as well as the origin of the data being used.

Data Building

Machine learning requires data as a foundation. Many machine learning models are well-suited for tabular data, which is among the most common and popular data formats. Tabular data is structured into rows and columns, where each column represents a feature or variable, and each row corresponds to an individual sample or data point. Columns are typically categorized into inputs and outputs, and the quality of the relationships between them directly affects the performance of the resulting model.

Modern machine learning models can handle diverse data sources simultaneously, such as high- and low-fidelity simulations or experimental data. This capability is particularly valuable in fields like mechanical and aerospace engineering, where large volumes of heterogeneous data are common.

Another crucial concept in handling large datasets is *sampling*, which involves selecting a subset of data or design points from a larger dataset or design space. One of the main strategies to achieve an optimal sampling is *Design of Experiments (DoE)*, a structured approach that ensures a well-distributed and representative selection of points to maximize the information gained from simulations or experiments. Popular sampling methods include *Random Sampling*, *Latin Hypercube Sampling*, and *Adaptive Sampling*, each offering different advantages depending on the problem complexity and the desired accuracy of the surrogate model.

Additionally, *bias* can play a significant role in machine learning. Data bias can lead to models that are not generalizable, resulting in poor performance on unseen data. It is crucial to monitor data sources and ensure the training set represents the real-world variability accurately. Bias can come from several sources, including sampling bias, measurement bias, or model bias. Ensuring that the data are diverse and representative helps mitigate these issues and improves the robustness of the model.

Data Preprocessing

Data pre-processing is a crucial phase in machine learning that involves transforming raw data into a form suitable for modeling. The complexity of this process depends on the type of data available, but it is essential for ensuring that machine learning and deep learning algorithms perform optimally. The first step in data pre-processing is *Exploratory Data Analysis (EDA)*, where patterns, correlations, and anomalies in the dataset are identified. This stage is vital because understanding the relationships in the data allows for informed decisions on how to manipulate it for better model performance.

Following EDA, the next step is *feature engineering*. The goal of feature engineering is to enhance or manipulate the dataset in ways that improve the predictive power of the model. During this phase, it is important to focus on *correlation* and *variance*:

Correlation refers to the relationship between input and output variables. Highly correlated features can provide valuable insights into which variables influence the output. If many inputs are correlated, it may be better to eliminate redundant variables to avoid *collinearity*, which can distort the model's interpretation.

Variance measures the dispersion of data points. Features that capture the majority of the data's variance are typically more significant. High-variance features are considered more important because they carry more information about the data's overall behavior.

Additionally, data cleaning, reduction, transformation, and enrichment are essential tasks in feature engineering. These manipulations are necessary to handle missing values, imbalanced datasets, small sample sizes, or converting categorical data into numerical form or vice versa. Such transformations are critical for improving model accuracy and making the dataset more suitable for machine learning tasks.

After preprocessing, a crucial step is *data splitting*, which divides the data into three subsets: *training data*, *validation data*, and *test data*. Typically, this is done using random selection, with splits like 70% for training, 20% for validation, and 10% for testing. Training data is used to adjust the internal parameters of the model. The validation data helps in evaluating the model's error during training, guiding the optimization process. The test data, kept separate from the training and validation data, is used to assess the model's performance on unseen data, providing an unbiased evaluation. This approach ensures that the model is not overfitting the training data and can generalize well to new data.

Choosing an Appropriate Technique

Choosing the right machine learning algorithm is crucial and depends on the nature of the data and the problem at hand. Algorithms for regression, classification, clustering, and dimensionality reduction offer various tools for tackling different tasks. For this report, the focus will be on regression models which will be explored in detail in the next chapters. When selecting an algorithm, considerations include the size and type of the dataset, interpretability requirements, computational resources, and the desired balance between bias and variance.

Training and Evaluating

Training a machine learning model involves teaching it to learn from the input data and adjusting its internal parameters to minimize the error. The training phase consists of feeding the model with a large set of training data and updating its parameters using a specific algorithm, such as *Stochastic Gradient Descent (SGD)*. The outcome of this phase is a *trained model*, which can then be deployed for inference or prediction tasks.

In addition to selecting the appropriate algorithm, one of the critical steps in training is finding an optimal set of *hyperparameters*. Hyperparameters are values that guide the learning process, such as the learning rate, number of layers in a neural network, or batch size. The process of hyperparameter optimization is essential because these values significantly impact the performance of the model. Common optimization techniques include *grid search*, *random search*, and *Bayesian optimization*.

During training, a typical procedure involves the following steps:

1. Presenting a batch of randomly sampled training data.
2. Calculating the loss function, which quantifies the error between the model's predictions and the actual outcomes.
3. Computing the gradient of the loss function with respect to the model's parameters.
4. Adjusting the parameters in the direction of the negative gradient, scaled by a chosen learning rate.
5. Repeating this process until the loss function converges or reaches an acceptable threshold.

One of the most widely used methods in ML training is *Stochastic Gradient Descent (SGD)*. SGD minimizes the loss function by updating model parameters based on the gradient of the loss with respect to the model's parameters. The key feature of SGD is that the gradient is calculated from a randomly selected subset of the data, which introduces randomness into the training process, helping the model avoid getting stuck in local minima and improving its ability to generalize.

It's also important to discuss *evaluation metrics*, which are used to assess how well a model is performing. These metrics provide quantitative measures of the model's accuracy, robustness, and generalizability. Common metrics for regression models include:

- *Mean Squared Error (MSE)*: Measures the average squared difference between the predicted and actual values. It is commonly used as the loss function for gradient descent algorithms.
- *Root Mean Squared Error (RMSE)*: Provides an estimate of the error in the same units as the target variable, offering a practical understanding of how much error to expect from the model.
- *Mean Absolute Error (MAE)*: Computes the average absolute difference between the predicted and actual values, providing a robust measure of error.

These evaluation metrics, along with the results from testing and validation, help refine the model by identifying areas where it underperforms, allowing practitioners to make necessary adjustments. Regular evaluation and fine-tuning lead to better generalization, ensuring the model works effectively across various real-world scenarios.

Deployment

The deployment phase applies the trained model to new data for predictions. While training is computationally intensive and requires large datasets, inference is typically faster and less resource-intensive. In some cases, incremental learning combines training and inference by continuously updating the model with new data during its deployment. This approach is especially useful in dynamic environments where data evolves over time.

2.3. Surrogate Model

Surrogate models are simplified approximations of more complex, higher-order models. They are used to map input data to outputs when the actual relationship between the two is unknown or computationally expensive to evaluate [16]. Surrogate models can thus be seen as simple representations of complex systems with a trade-off between accuracy and computational efficiency. This trade-off is a critical consideration during their construction.

In general, surrogate modeling refers to a set of techniques that utilize previously sampled data to build predictive models. Usually they are constructed using computer experiments, where the design parameters span a carefully chosen range of values within the design space. These values are selected following specific criteria or patterns, often employing a technique known as the *Design of Experiments (DoE)*.

The computationally expensive simulations are conducted at these selected points, and the corresponding responses are recorded. Based on this input/output data, the behavior of the complex simulation is approximated using simplified functional relationships, referred to as surrogate models. Consequently, the detailed nature of the original simulation becomes secondary, with the surrogate model serving as a computationally efficient, and above all very fast, approximation of the system [17].

In the aeronautical sector, surrogate modeling techniques have been extensively applied to aerodynamic analysis and optimization. For instance, in the initial stages of the design process, computational fluid dynamics (CFD) simulations can be aided by surrogate models capable of predicting aerodynamic performance with reasonable precision [18]. This reduces reliance on expensive simulations or experiments, minimizing the number of required tests.

In the machine learning domain, surrogate models based on artificial *neural networks (ANNs*) and *support vector machines (SVMs)* have also been employed for aerodynamic coefficient prediction, aerodynamic design, and robust design or uncertainty quantification.

One of the most widely used methods for building surrogate models is *Kriging*, which has proven to be particularly effective in applications such as airfoil design [19].

The implementation of surrogate models in *CEASIOMpy* leverages the *Surrogate Modelling Toolbox (SMT)*, an open-source *Python* package that provides a suite of surrogate modelling methods [20].

2.3.1. Sampling

Sampling is critical to the resulting accuracy and usefulness of a machine learning model. It refers to the process of selecting a subset of data or cases from a larger dataset or design space, akin to design exploration studies conducted with simulations.

Sampling methods are numerous and can be broadly divided into two main categories:

- *Probability Sampling*: Every element of the population has an equal chance of being selected. Probability sampling provides the best opportunity to create a sample that is truly representative of the population.

- *Non-Probability Sampling*: Not all elements have an equal chance of being selected. As a result, there is a higher risk of obtaining a non-representative sample, which may lead to non-generalizable results.

If the objective is to explore the entire domain effectively to train the surrogate model, the choice will naturally lean towards *probability sampling*. As discussed in the *Data Building* section of Chapter 2.2.2, popular sampling methods include *Random Sampling*, *Latin Hypercube Sampling* (LHS), and *Adaptive Sampling*. A closer examination of these methods helps in understanding the newly implemented algorithm, which incorporates all of them.

- *Random Sampling*: Every individual sample or item in the population has an equal chance of being selected. Simple random sampling provides an impartial, representative dataset while offsetting confounding effects from known and unknown factors. This method is the most straightforward of all probability sampling techniques, requiring only a single random selection and little prior knowledge of the population. Because it relies on randomization, research performed on such a sample generally has high internal and external validity and a reduced risk of biases, such as sampling bias or selection bias.

- *Latin Hypercube Sampling*: This method involves dividing the range of each input parameter into non-overlapping intervals based on the desired sample size. LHS is rooted in the Latin square design, where each row and column contains a single sample. A *hypercube* extends this principle to multiple dimensions, allowing for sampling across several dimensions and hyperplanes. One-dimensional LHS divides the domain into n equal partitions, with a random data point chosen in each partition. LHS is particularly effective when it is essential to efficiently explore and fill the entire parameter space while minimizing the number of samples required. It roughly corresponds to random sampling with guaranteed space filling properties even for small sampling size.

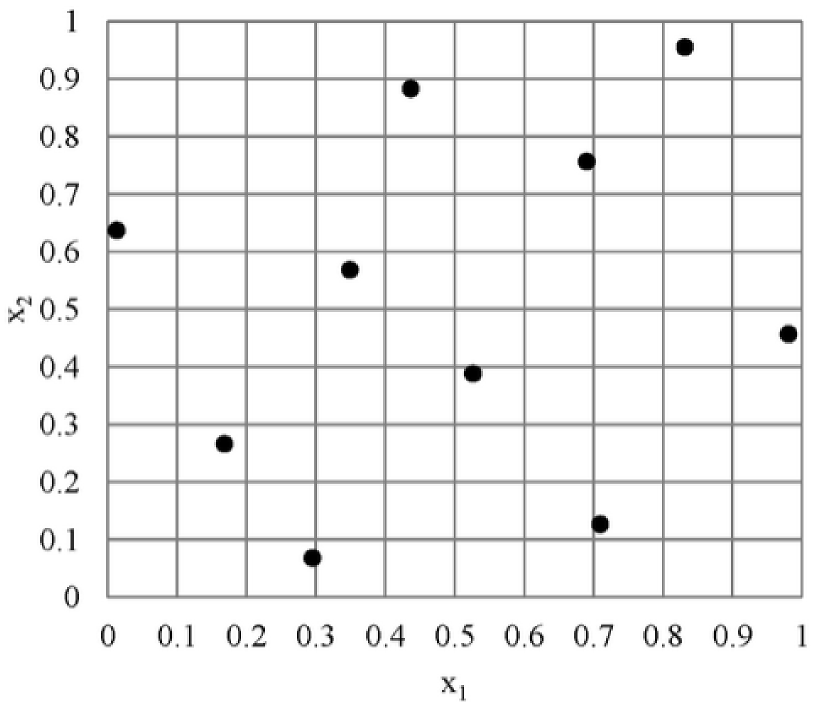


Figure 2.7. An example of a Latin Hypercube Sampling of a general domain

- *Adaptive Sampling*: Rather than being a specific method, adaptive sampling is a general approach to sampling problems. It is particularly suitable for cases where the sampled characteristic is rare or spatially clustered. Adaptive sampling allows the selection of observations to depend on prior observations. For instance, initial samples can be used to build an initial model. Regions of interest, such as areas with high sensitivity or variance, are then identified, and additional samples are adaptively added to these regions to improve model accuracy. An adaptive sampling strategy is implemented in the SMTrain module, which uses the extreme values of the domain as initial samples and then refines the model with high variance points.

2.3.2. Regression Models

To implement a surrogate model, various algorithms can be chosen. This section details the implementation of *Kriging*, utilizing the *SMT Toolbox*, as it is a fundamental component of the multi-fidelity strategy.

Kriging Model

The Kriging model was initially developed in the field of geostatistics by Danie G. Krige. Later, Sacks et al. (1989) extended it to computer simulation experiments. The Kriging model, also known as *Gaussian process regression*, is a method of interpolation based on Gaussian process governed by prior covariances. It is an unbiased estimation method (which means that, on average, it introduces no systematic error) that can not only predict the response values of unknown samples but also quantify the uncertainty of these predictions [21]. This method uses a limited set of sampled data points to estimate the value of a variable over a continuous spatial field. The Kriging model was chosen to approximate the objective function because of its accuracy and robustness with small data sets [22].

Unlike simpler methods such as Inverse Distance Weighted Interpolation, linear regression, or Gaussian decays, Kriging leverages the spatial correlation between sampled points to interpolate values in the spatial field. The interpolation is based on the spatial arrangement of the empirical observations rather than on a presumed model of spatial distribution.

The Kriging model approximates a target function $\hat{y}$ based on a set of m design points for q dependent variables and n independent variables. These design points are represented as:

$$X = \left[x_j^{(i)}\right], \quad i = 1, \ldots, m, \ j = 1, \ldots, n \in \mathbb{R}^{m\times n}, \tag{2.10}$$

$$Y = \left[y_j^{(i)}\right], \quad i = 1, \ldots, m, \ j = 1, \ldots, q \in \mathbb{R}^{m\times q}. \tag{2.11}$$

At an unsampled location x^*, the Kriging approximation for the target function is given by:

$$\hat{y}_k(x^*) = F(\beta_{:,k}, x^*) + e_k, \quad k = 1, \ldots, q, \tag{2.12}$$

where F is the deterministic trend modeled by a regression function, and e is a stochastic term representing local deviations. The regression component F can be constant, linear, or quadratic:

$$F(\beta, x^*) = \sum_{h=1}^{n} \beta_{h,k} x_h^* + \beta_{0,k}. \tag{2.13}$$

The stochastic term e is modeled as a realization of a Gaussian process with zero mean and covariance:

$$\text{cov}[Z(x^{(i)}), Z(x^{(j)})] = \sigma^2 R(x^{(i)}, x^{(j)}), \tag{2.14}$$

where R is the correlation function.

The correlation between points $x^{(i)}$ and $x^{(j)}$ is defined as:

$$R(x^{(i)}, x^{(j)}) = \prod_{l=1}^{n} \exp(-\theta_l |x_l^{(i)} - x_l^{(j)}|^{p_l}), \tag{2.15}$$

where $\theta_l \geq 0$ are the hyperparameters, and $p_l \in [1, 2]$. Common correlation functions include:

- *Exponential*: $p = 1$
- *Gaussian*: $p = 2$
- *Matérn 5/2*: A more flexible correlation suitable for differentiable functions.

In the *SMT Toolbox* library, a Python tool that will be breefly discussed in Section 4.1, these are denoted as `abs_exp`, `squar_exp`, `matern52`, among others.

The $m \times m$ correlation matrix for the observed data points is:

$$\Psi = [\psi_{i,j}], \quad \psi_{i,j} = \exp(-d(x^{(i)}, x^{(j)})), \quad i, j = 1, \dots, m. \tag{2.16}$$

The correlation vector between the unsampled location x^* and observed data is:

$$\psi = [\psi_k], \quad \psi_k = \exp(-d(x^*, x_k)), \quad k = 1, \dots, m. \tag{2.17}$$

The final Kriging prediction at x^* is given by:

$$\hat{y}(x^*) = F(\beta, x^*) + \psi^T \Psi^{-1} (Y - F). \tag{2.18}$$

Here, $Y - F$ represents the discrepancies between the observed data and the regression model, centered around the m sample points. These discrepancies are weighted by Ψ^{-1} and added to the regression predictor $F(x^*)$.

The confidence in the Kriging prediction is quantified by the mean squared error:

$$\hat{s}^2(x^*) = \sigma^2 \left[1 - \psi^T \Psi^{-1} \psi + \frac{1 - \mathbf{1}^T \Psi^{-1} \psi}{\mathbf{1}^T \Psi^{-1} \mathbf{1}} \right]. \tag{2.19}$$

If x^* is close to the sample points, the confidence in the prediction is higher, reflected by a smaller value of $\hat{s}^2(x^*)$.

2.3.3. Optimization

In machine learning, it is crucial to distinguish between *parameters* and *hyperparameters*. Hyperparameters are set prior to training and govern the learning process, whereas parameters are learned from the model during the training phase.

To optimize these hyperparameters, different optimization strategies could be used. Among the most common approaches, we find *grid search*, which systematically explores a predefined set of values, and *random search*, which selects values randomly within a given range. While these methods can be effective, they often require a large number of evaluations, making them computationally expensive, especially when dealing with complex models.

For this reason, *Bayesian optimization* has been chosen for this work. Unlike grid or random search, Bayesian optimization builds a probabilistic model of the objective function and strategically selects the most promising hyperparameter values. This allows it to find good solutions with fewer evaluations, reducing computational cost while maintaining high performance.

Bayesian optimization is based on a surrogate probabilistic model, typically a *Gaussian Process (GP)*, which approximates the true objective function by considering both the predicted value and the uncertainty associated with it. This model is iteratively refined as new evaluations are performed. The next point to evaluate is chosen using an *acquisition function*, which balances *exploration* (testing less certain regions) and *exploitation* (refining known good regions).

Among the most common acquisition functions, the *Expected Improvement (EI)* is used in this work. It is defined as:

$$EI(x) = \mathbb{E}[\max(0, f(x) - f^*)]$$

where $f(x)$ is the predicted function value at point x, and f^* is the best observed value so far. The EI function selects points where the expected improvement over the current best solution is maximized, allowing the optimization process to efficiently explore the hyperparameter space.

The main advantage of Bayesian optimization lies in its ability to reduce the number of function evaluations needed to reach a good solution, making it particularly suitable for expensive-to-train models.

3. CEASIOMpy

CEASIOM (Computerised Environment for Aircraft Synthesis and Integrated Optimisation Methods) is a powerful framework tool that integrates discipline-specific tools for conceptual design [23]. It was developed within the SimSAC project, funded by the European Commission's 6th Framework Program on Research, Technological Development and Demonstration and is mainly based on Matlab code. Then, in 2015, began the AGILE project, granted by the European Commission and coordinated by the DLR (German Aerospace Centre), whose target was multidisciplinary optimization using distributed analysis frameworks. The project aimed to reduce the time-to-market and development costs of new aircraft, which are two critical objectives for the aeronautical industry. In this context that CFS Engineering, in collaboration with Airinnova, started to develop the CEASIOMpy version entirely based on Python environment. CEASIOMpy is freely available to the public as an open-source software downloadable from GitHub. The fact that the software programming language is Python makes it even simpler and easier to use, as Python is a very popular language in the scientific and engineering communities due to its simplicity and versatility.

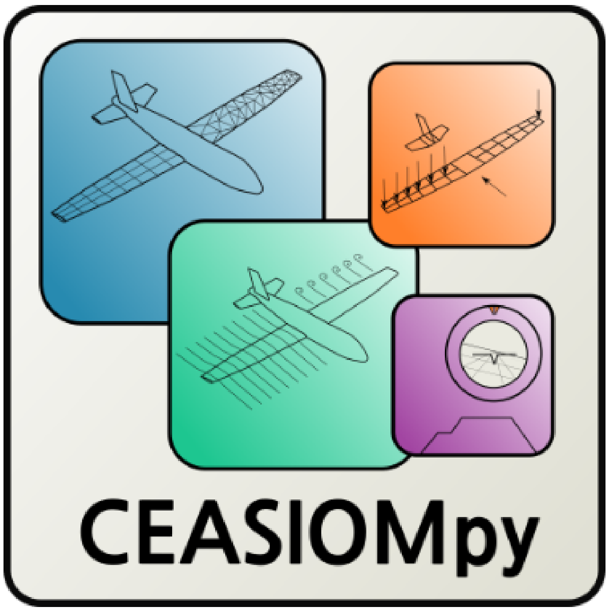


Figure 3.1. CEASIOMpy logo

CEASIOMpy offers a suite of tools tailored to various aircraft design disciplines, enabling the creation of complex design and optimization workflows for both conventional and unconventional aircraft configurations. The software is composed of domain-specific modules that can be interconnected in different sequences, depending on the specific application. A typical workflow to perform valuable CFD simulations looks like:

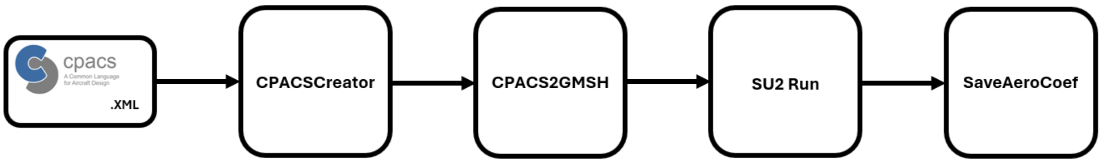


Figure 3.2. Example of CEASIOMpy workflow

3.1. Modules and Architecture

CEASIOMpy integrates into one application the main design disciplines: aerodynamics, structures and flight dynamics, impacting on the aircraft's performance. The modules that make up the frameworks are numerous and the programme is regularly updated with new ones. As can be read on the GuitHub page, the main modules are as follows:

- *General Modules*: This module provides tools of different types, which are very useful for the conceptual design of the aircraft. Among these tools are the two described in this report, which allows a surrogate model to be trained and used to predict the values of the aerodynamic coefficients.
- *Geometry and Mesh Module*: This module is useful for handling the geometry definition and meshing process. Most of the tools that make up this module process a CPACS file and allow the user to generate a 3D mesh, that can then be used with other tools according to a user-defined workflow.
- *Aerodynamics Module*: This module offers the most advanced tools of the software, which are for aerodynamic analysis. These tools allow numerical simulations to be carried out at different levels of fidelity:
  - *CLCalculator*: Determines the lift coefficient *CL* of an aircraft to sustain a cruise flight, for a given Mach number, altitude, and mass.
  - *PyAVL*: A *vortex lattice method (VLM)* solver for low-fidelity aerodynamic computations.
  - *SU2Run*: Prepares and runs calculations with the CFD code *SU2*, allowing Euler and RANS simulations.
- *Weight and Balance Module*: This module provides tools to estimate the weight and balance of an aircraft, a critical step in the design process that directly affects the performance and safety of the aircraft.

- *Mission Analysis Module*: This module will allow users to simulate the flight of an aircraft, considering factors such as fuel consumption, altitude, and weather conditions.

- *Structure Module*: This module is currently under developing, it will provide structural analysis tools. These tools will enable users to evaluate the strength and durability of the aircraft's components, ensuring safety and reliability. Through the *AeroFrame* module, it is possible to perform an aeroelastic analysis of the aircraft.

3.2. CPACS

CEASIOMpy is based on the open-standard format CPACS, a Common Parametric Aircraft Configuration Schema. It is a data definition for the air transportation system which is developed by the German Aerospace Center DLR. CPACS enables engineers to exchange information between their tools. It is therefore a driver for multi-disciplinary and multi-fidelity design in distributed environments. The CPACS format is structured in a hierarchical manner, intended to encompass all aspects of aircraft design from more general considerations to the most detailed definition. The components of the aircraft model are sorted by type (fuselages, wings, engines, etc.) like in Figure 3.3.

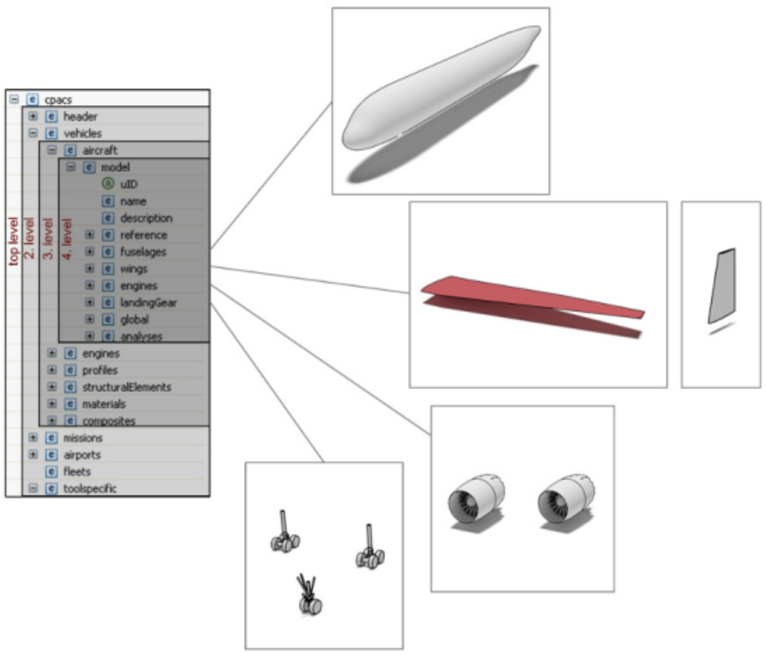


Figure 3.3. Example of CPACS hierarchical structure

Of course, all the tools are harmonised to use CPACS format files for input and output. This greatly simplifies communication between the different tools, but also requires the use of Python-specific libraries. The library in question is TiXI, which is a fast and simple

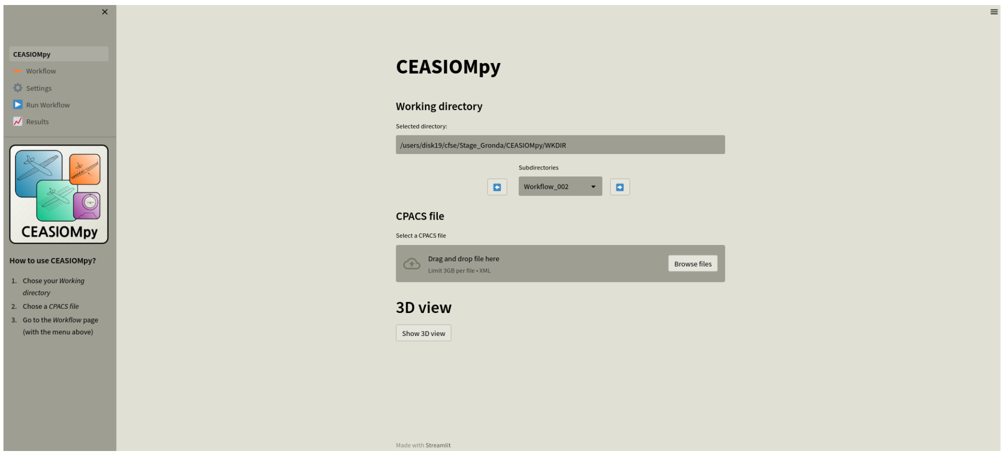
XML interface library that can be used by applications written in C, C++, Fortran, JAVA and Python.

3.3. Using of CEASIOMpy

As mentioned above, CEASIOMpy is open source software, so one can easily download it from its page on GitHub by cloning the [24] repository. Once cloned in the computer, the user will need to follow the installation steps, after which it can be opened from the command prompt after activating the *ceasiompy* environment.

To improve accessibility, CEASIOMpy features an intuitive Graphical User Interface (GUI) that streamlines user interaction. This interface allows for efficient management of inputs, visualization of results, and customization of workflows, making CEASIOMpy a valuable resource for both novice and experienced designers alike.

The configuration of CEASIOpy starts by asking the user for the folder where the results will be stored and the geometry to be studied, with the possibility of 3D visualization.



How to use CEASIOMpy?

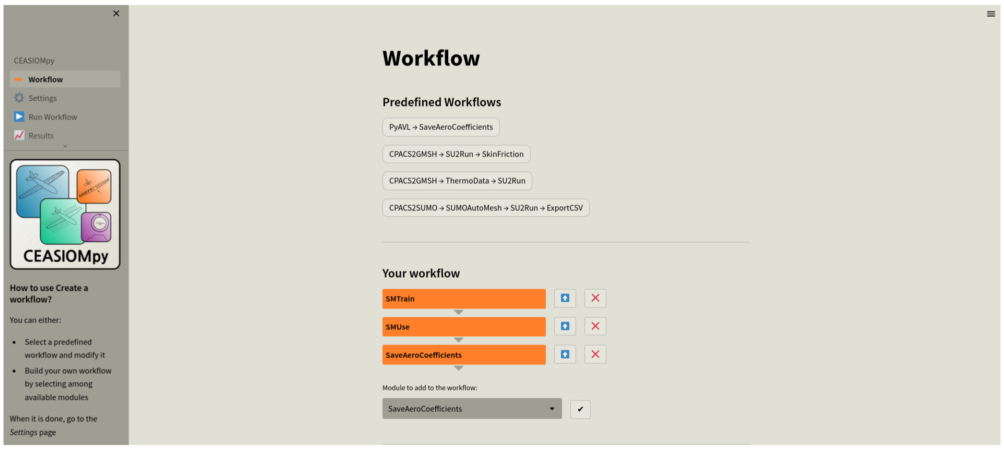
1. Chose your *Working directory*
2. Chose a *CPACS file*
3. Go to the *Workflow* page (with the menu above)

Selected directory: /users/disk19/cfse/Stage_Gronda/CEASIOMpy/WKDIR

Subdirectories: Workflow_002

Figure 3.4. First page of CEASIOMpy configuration

The next step is to configure the *Workflow* by selecting the modules of interest for the study to be performed. An example of a workflow that can be constructed for the training and the use of a surrogate model sees the selection in series of *SMTrain* for the training, *SMUse* for the use and *SaveAeroCoefficients* for the visualization of the results.


CEASIOMpy
Workflow
Settings
Run Workflow
Results
CEASIOMpy
Predefined Workflows
PyAVL → SaveAeroCoefficients
CPACS2GMSH → SU2Run → SkinFriction
CPACS2GMSH → ThermoData → SU2Run
CPACS2SUMO → SUMOAutoMesh → SU2Run → ExportCSV

Workflow

How to use Create a workflow?

You can either:

- Select a predefined workflow and modify it
- Build your own workflow by selecting among available modules

When it is done, go to the *Settings* page

Your workflow

SMTrain

SMUse

SaveAeroCoefficients

Module to add to the workflow: SaveAeroCoefficients

Figure 3.5. CEASIOMpy Workflow with the three modules selected

After that, we move on to the individual module *Settings*, which will be discussed in details in sections 4.3 and 4.4.

4. Multi-Fidelity Surrogate Model in CEASIOMpy

The aim of the thesis was to develop a new module, written in the Python language, involving the implementation of a surrogate model based on a machine learning strategy that exploits multi-fidelity (M-F) CFD simulations. The use of this surrogate model makes it possible to predict key aerodynamic coefficients such as C_L, C_D, and C_M, essential for the evaluation of aerodynamic performance.

What distinguishes and characterises the algorithm is the multi-fidelity strategy, which is particularly suitable for predicting and modelling the aerodynamic data of aircraft throughout their entire flight envelope. M-F is essentially a model management methodology. It uses a set of CFD methods with varying degrees of fidelity and computational expense or a single physical model evaluated on meshes of varying resolutions. A low-fidelity CFD method is used to automatically compute hundreds or thousands of solutions at points in the parameter space selected with a Design of Experiments (DoE) tool. A few points in the parameter space are computed using a medium and then a high-fidelity CFD method; these points are selected following an adaptive sampling logic described in the following sections. That algorithm is the base of two modules developed for CEASIOMpy. These modules, written entirely in the Python language, are necessary to train the model and then use the trained model to make predictions.

4.1. SMT Library

The Surrogate Modeling Toolbox (SMT) [25] is an open-source project originally developed through a collaboration between ONERA, NASA, ISAE-SUPAERO/ICA, and the University of Michigan. Today, Polytechnique Montréal and the University of California San Diego also contribute to its development.

SMT is a Python package designed to provide an easy-to-use library of surrogate models while facilitating the implementation of additional methods. Several other packages for surrogate modeling exist across different programming languages, such as Scikit-learn [26] in Python and SUMO [27] in MATLAB.

In addition to surrogate modeling techniques, SMT includes sampling methods, which are essential for constructing surrogate models. The library is organized into three main modules: sampling methods, benchmarking problems, and surrogate models. The sampling methods module implements various sampling techniques, the benchmarking problems module provides test functions for evaluation, and the surrogate models module offers different modeling techniques. Each module follows a unified interface, ensuring consistency across methods. Every implemented method adheres to this interface by defining the required functions, making SMT a flexible and extensible tool for surrogate modeling.

Table 1 lists the surrogate modelling methods currently available in SMT and summarizes the advantages and disadvantages of each method.

Table 1. Surrogate modeling methods provided by SMT

Method	**Advantages (+) and disadvantages (-)**
Kriging	+ Prediction variance, flexible - Costly if number of inputs or training points is large - Numerical issues when points are too close to each other
KPLS	+ Prediction variance, fast construction + Suitable for high-dimensional problems - Numerical issues when points are too close to each other
KPLSK	+ Prediction variance, fast construction + Suitable for high-dimensional problems - Numerical issues when points are too close to each other
GE-KPLS	+ Prediction variance, fast construction + Suitable for high-dimensional problems + Control of the correlation matrix size - Numerical issues when points are too close to each other - Choice of step parameter is not intuitive
RMTS	+ Fast prediction + Training scales well up to 10^5 training points + No issues with points that are too close to each other - Poor scaling with number of inputs above 4 - Slow training overall
RBF	+ Simple, only a single tuning parameter + Fast training for small number of training points - Susceptible to oscillations - Numerical issues when points are too close to each other
IDW	+ Simple, no training required - Derivatives are zero at training points - Poor overall accuracy
LS	+ Simple, fast construction - Accurate only for linear problems
QP	+ Simple, fast construction - Large number of points required for large number of inputs

Kriging models (also known as Gaussian processes) are one of SMT's key features.

SMT Toolbox Features

The *SMT Toolbox* provides several functionalities for implementing Kriging models. Specifically, it automatically normalizes input and output data before training the model.

The main available hyperparameters in the toolbox are:

- *Poly*: Defines a global trend in the model. It determines the type of regression function used to approximate the underlying data structure. Depending on the selected option, the model can assume a constant mean, a linear trend, or even a quadratic variation in the response. Available options are: `constant`, `linear`, `quadratic`.

- *Corr*: Controls the spatial correlation structure of the model. The choice of correlation function affects how the model interpolates between points and the smoothness of the predicted surface. Different correlation functions can capture varying levels of smoothness and periodicity in the data. Available correlation functions are: `pow_exp`, `abs_exp`, `squar_exp`, `squar_sin_exp`, `matern52`, `matern32`.

- *Theta0*: This parameter specifies the initial value of the spatial correlation length between points. It serves as a starting point for the optimization process that fine-tunes the model's correlation behavior. Proper tuning of this parameter impacts the sensitivity of the model to changes in input values.

- *Nugget*: Introduces a regularization component to improve numerical stability. It is particularly useful when working with noisy data, as it allows the model to accommodate small fluctuations instead of forcing an exact interpolation.

- *Hyper_opt*: This setting determines the optimization algorithm used to estimate the hyperparameters of the model. A well-chosen optimization method ensures that the model maximizes its likelihood and provides the best possible fit to the data. Supported methods are two: `Cobyla` and `TNC`.

- *Rho regressor*: This hyperparameter is relevant in multi-fidelity modeling, where it captures the relationship between low- and high-fidelity data. It helps the model learn from coarse approximations while refining predictions based on high-accuracy data.

Additionally, the toolbox requires the output training data to be provided as a one-dimensional vector.

4.2. Multi-Fidelity Strategy

Multi-Fidelity modelling is based on the assumption that in addition to an HF model that is sufficiently accurate but has a high computational cost, an LF model is used that is less accurate but also considerably less computationally demanding [28].

Building the multi-fidelity model has two main steps:

1. Populate the aerodynamic database over the whole flight envelope by the dense low-fidelity data samples.

2. Correct the data using the sparse high-fidelity samples.

The most popular method currently used is a correction-based method. The correction is called bridge function, scaling function or calibration and it can be divided into three distinct types. First, in the multiplicative scaling approach, a scaling function is constructed to represent the ratio between the HF and LF models. Second, in the additive scaling approach, a scaling function is constructed to capture the differences between the HF and LF models. The additive bridge function should also be of low order but of higher order than the multiplicative one. In general, additive functions are more accurate and robust than the multiplicative ones. Finally, in the hybrid scaling approach, scaling functions are constructed to utilize the advantages of both the multiplicative and additive scaling approaches.

For the present work, an additive bridge function approach introduced by Kennedy and O'Hagan was adopted [29], as it is the approach proposed by SMT Library for the formulation of Multi-Fidelity Kriging:

$$y_{\text{high}}(x) = \rho(x)\hat{y}_{\text{low}}(x) + \hat{\delta}(x), \tag{3.1}$$

where:

- $y_{\text{high}}(x)$ represents the high-fidelity model output,
- $y_{\text{low}}(x)$ is the low-fidelity model output,
- $\rho(x)$ is a scaling function that adjusts the low-fidelity prediction (constant, linear or quadratic),
- $\hat{\delta}(x)$ is the discrepancy function that captures the difference between the high- and low-fidelity models.

The implementation here follows the one proposed by Le Gratiet [30]. It offers the advantage of being recursive, it can easily be extended to various levels of fidelity and offers better scaling for high numbers of samples. An important assumption in using this recursive formulation is the fact that this method only uses nested sampling training points. So, if we have two fidelity levels (HF and LF):

$$X_{HF} \subset X_{LF}. \tag{3.2}$$

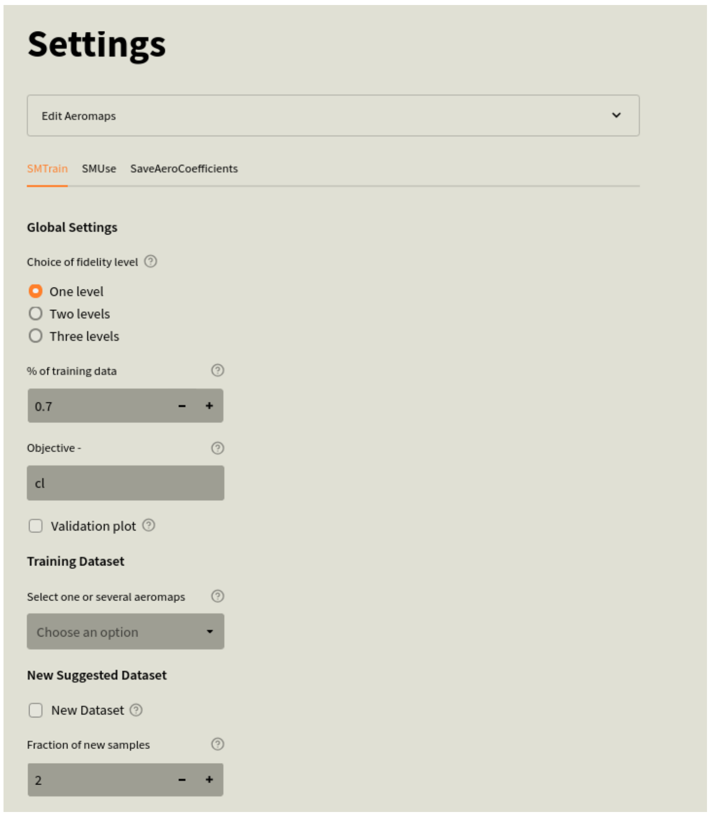
This means that every high-fidelity sample must correspond to a low-fidelity one, ensuring consistency across levels. This structure allows for better information transfer between fidelities and improves the efficiency of the surrogate model.

This must therefore be taken into account in the implementation of the algorithm, which includes a data search method to be explored that is shown in the next section.

4.3. SMTrain Module

The first module, SMTrain, is designed to train the surrogate model. The training process can be performed at three different fidelity levels, corresponding to the approximation methods used in simulations: the panel method (VLM), the Eulerian method, and the RANS method. The choice of these fidelity levels is aligned with the existing modules in CEASIOMpy, namely PyAVL and SU2. Consequently, an integration strategy was devised to ensure compatibility between the new module and the existing framework.

The module has been designed to allow users to customize their workflow. Firstly, users can choose to train the surrogate model with one, two, or three fidelity levels. Increasing the number of fidelity levels enhances the accuracy of the predictions but also increases the training time. Additionally, users can either provide their own datasets, containing numerical results to train the model, or define the domain ranges and integrate the module into a workflow with PyAVL or SU2.


Settings

Edit Aeromaps

SMTrain | SMUse | SaveAeroCoefficients

Global Settings

Choice of fidelity level

[✓ One level]
Two levels
Three levels

% of training data: 0.7

Objective -: cl

☐ Validation plot

Training Dataset

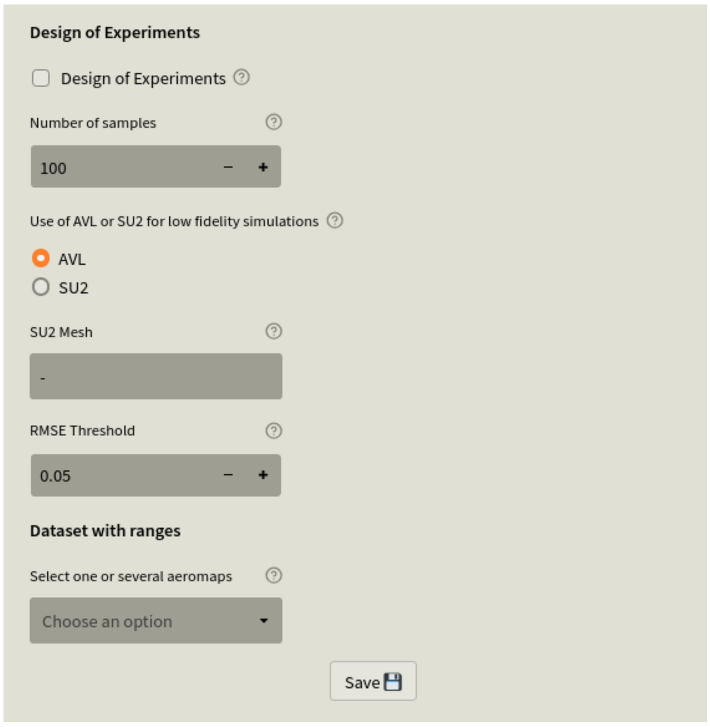
Select one or several aeromaps: Choose an option

New Suggested Dataset

☐ New Dataset

Fraction of new samples: 2

(a) First part of the SMTrain module interface


Design of Experiments

☐ Design of Experiments

Number of samples: 100

Use of AVL or SU2 for low fidelity simulations

[✓ AVL]
SU2

SU2 Mesh: -

RMSE Threshold: 0.05

Dataset with ranges

Select one or several aeromaps: Choose an option

Save

(b) Second part of the SMTrain module interface

Figure 4.1. GUI for the settings of SMTrain module

The datasets provided by the user must be in tabular form (CSV format). The first four columns should contain the values of altitude, Mach number, angle of attack, and sideslip angle for each data point, while the last columns should contain the aerodynamic coefficients.

If the user chooses to generate data from scratch by defining the domain ranges, the implemented sampling algorithm is the *Latin Hypercube Sampling* (LHS). This method ensures an intelligent selection of sample points, providing a comprehensive representation of the flight domain while minimizing the number of required points.

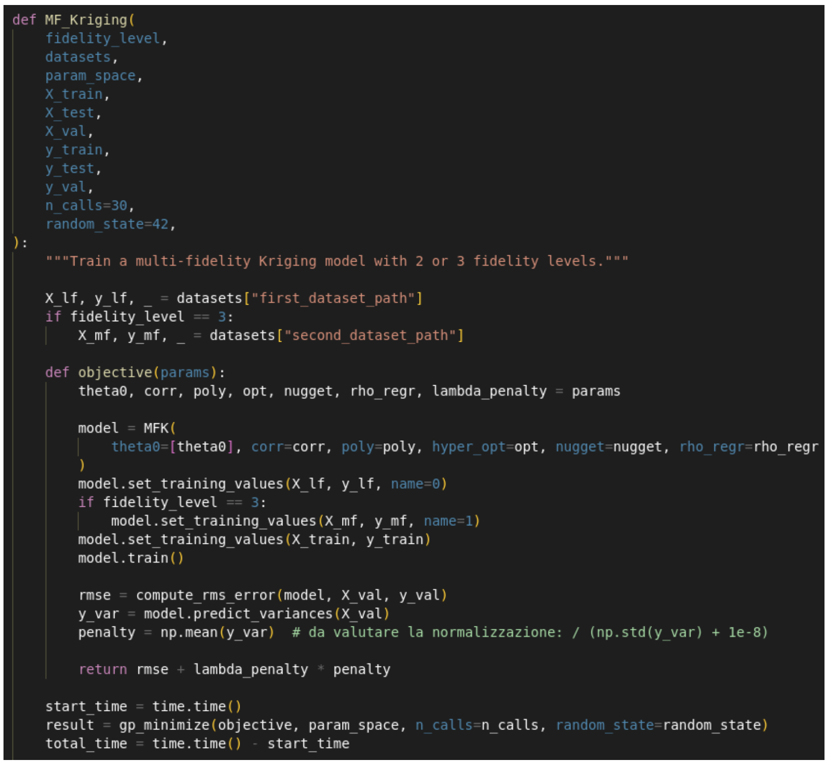
The sampled domain points must be consistent with the physical flight envelope. However, the *Design of Experiments* (DOE) techniques typically operate on rectangular (or cubic, in a 3D space) domains. This presents a challenge because the flight envelope of an aircraft is not a simple rectangular domain; maneuverability varies with altitude and speed. Therefore, users need to have prior knowledge of the aircraft's flight envelope to ensure proper sampling.

After sampling, the generated dataset is written into the CPACS file using the Tixi library, creating the so-called aeromap.

Once the dataset is obtained—either from user input or from simulations—it is used to train a surrogate model and make predictions. To ensure proper training, the data is split into three subsets, based on user specifications. By default, the dataset is divided as follows: 70% for training, 20% for validation, and 10% for testing.

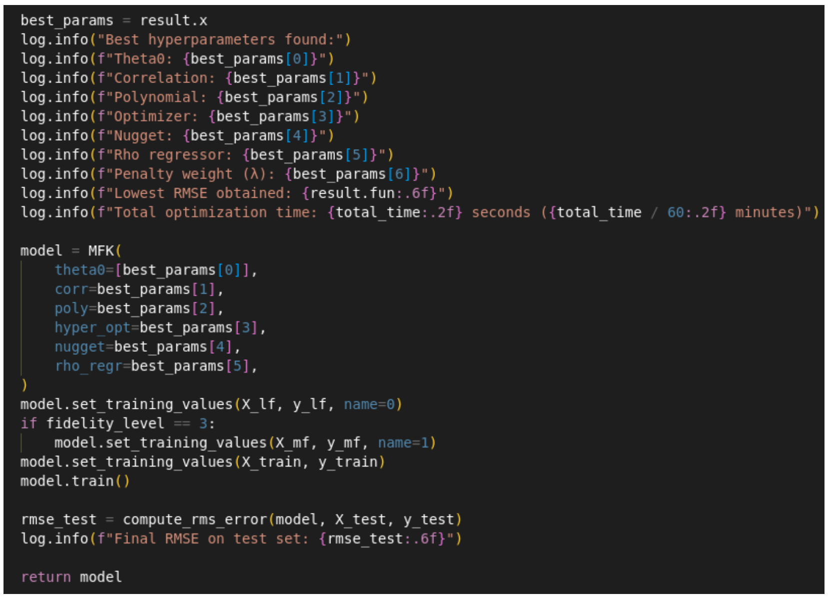
- The *training set* is used to adjust the internal parameters of the model so that it can best capture the relationship between inputs and outputs.
- The *validation set* is used for hyperparameter tuning, ensuring the best model configuration.
- The *test set* is used to evaluate the predictive performance of the model by computing its error on unseen data.

Hyperparameter tuning is performed, as discussed in Section 2.3.3, using Bayesian optimization. The hyperparameters are described in detail at the end of Section 2.3.2, but here we provide a brief overview of the optimization strategy. Initially, models optimized solely based on the objective function often produced unrealistic, highly oscillatory curves. To address this issue, an additional term was introduced in the objective function: the variance, penalized by a factor λ, which is also included in the optimization process. This modification led to a significant improvement in the smoothness of the generated curves while maintaining a low RMSE in the results.



```python
def MF_Kriging(
    fidelity_level,
    datasets,
    param_space,
    X_train,
    X_test,
    X_val,
    y_train,
    y_test,
    y_val,
    n_calls=30,
    random_state=42,
):
    """Train a multi-fidelity Kriging model with 2 or 3 fidelity levels."""

    X_lf, y_lf, _ = datasets["first_dataset_path"]
    if fidelity_level == 3:
        X_mf, y_mf, _ = datasets["second_dataset_path"]

    def objective(params):
        theta0, corr, poly, opt, nugget, rho_regr, lambda_penalty = params

        model = MFK(
            theta0=[theta0], corr=corr, poly=poly, hyper_opt=opt, nugget=nugget, rho_regr=rho_regr
        )
        model.set_training_values(X_lf, y_lf, name=0)
        if fidelity_level == 3:
            model.set_training_values(X_mf, y_mf, name=1)
        model.set_training_values(X_train, y_train)
        model.train()

        rmse = compute_rms_error(model, X_val, y_val)
        y_var = model.predict_variances(X_val)
        penalty = np.mean(y_var)  # da valutare la normalizzazione: / (np.std(y_var) + 1e-8)

        return rmse + lambda_penalty * penalty

    start_time = time.time()
    result = gp_minimize(objective, param_space, n_calls=n_calls, random_state=random_state)
    total_time = time.time() - start_time
```

(a) First part of the Bayesian Optimization process



```python
    best_params = result.x
    log.info("Best hyperparameters found:")
    log.info(f"Theta0: {best_params[0]}")
    log.info(f"Correlation: {best_params[1]}")
    log.info(f"Polynomial: {best_params[2]}")
    log.info(f"Optimizer: {best_params[3]}")
    log.info(f"Nugget: {best_params[4]}")
    log.info(f"Rho regressor: {best_params[5]}")
    log.info(f"Penalty weight (λ): {best_params[6]}")
    log.info(f"Lowest RMSE obtained: {result.fun:.6f}")
    log.info(f"Total optimization time: {total_time:.2f} seconds ({total_time / 60:.2f} minutes)")

    model = MFK(
        theta0=[best_params[0]],
        corr=best_params[1],
        poly=best_params[2],
        hyper_opt=best_params[3],
        nugget=best_params[4],
        rho_regr=best_params[5],
    )
    model.set_training_values(X_lf, y_lf, name=0)
    if fidelity_level == 3:
        model.set_training_values(X_mf, y_mf, name=1)
    model.set_training_values(X_train, y_train)
    model.train()

    rmse_test = compute_rms_error(model, X_test, y_test)
    log.info(f"Final RMSE on test set: {rmse_test:.6f}")

    return model
```

(b) Second part of the Bayesian Optimization process.

Figure 4.2. Python function that performs the Bayesian Optimization and trains the final module

After completing the optimization procedure, the final model is saved, and a validation plot is generated. This plot compares the actual test values ($ytest$) with the predicted values obtained from $Xtest$, an example is shown in Figure4.3

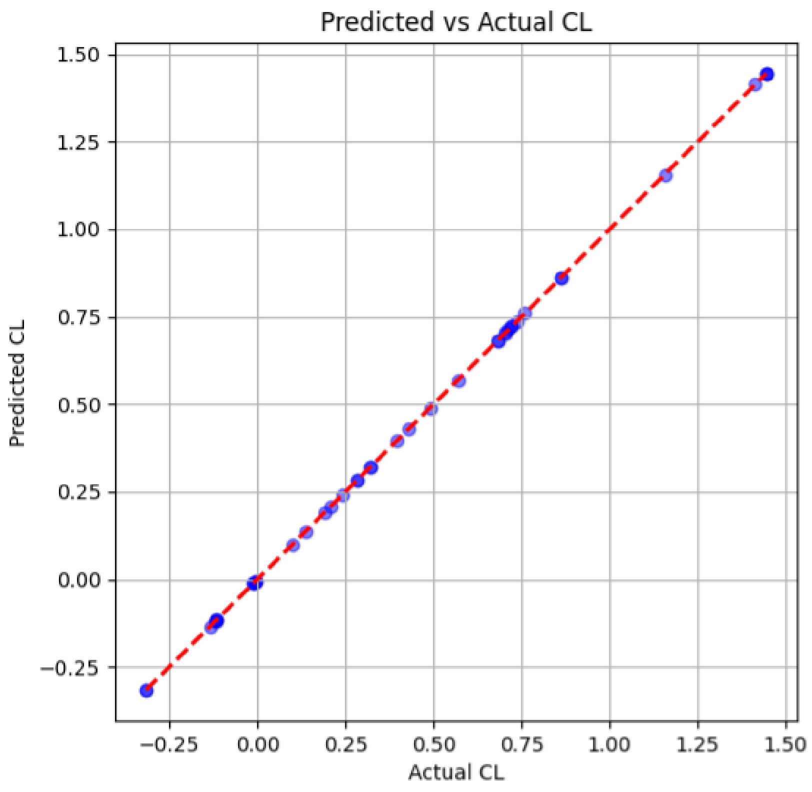


Figure 4.3. Example of predicted VS actual coefficients

At this stage, if the user has enabled the option "Suggest New Points," the algorithm identifies points where the model's predictions significantly deviate from the simulation results. These points are selected based on their variance, and the user can specify the percentage of points to be further refined or use the module's default setting. Additionally, the algorithm automatically adds points at the domain boundaries, where prediction accuracy is typically lower, requiring higher-fidelity refinement.

This strategy can be iterated for additional fidelity levels, progressively improving the accuracy of the predictions. The multi-fidelity fusion method employed is described in Section 4.2.

If the user has selected the "Design of Experiments" option along with "Two Levels" of fidelity, the code will automatically execute an iterative loop based on an adaptive sampling strategy. Similar to the previous approach, adaptive sampling selects points with the highest variance; however, in this case, points are not specified as a fraction by the user but are instead added one at a time during each iteration. The results for these high-variance points are refined using Euler or RANS simulations. From the GUI, users must specify an RMSE threshold, which is compared to the RMSE of the model at each iteration. Once the model's RMSE falls below the specified threshold, the loop stops, and the trained model is saved. While this approach may seem computationally expensive, the number of nested points can be significantly lower, potentially leading to an overall reduction in computational time.

4.4. SMUse Module

The second module, SMUse, is responsible for making predictions using the trained surrogate model. The user can utilize this module in two different ways:

- By providing a dataset containing the input parameters for which predictions are required, along with a previously trained surrogate model.
- By placing this module directly after SMTrain in the workflow, in which case it automatically retrieves the trained model without requiring user input.

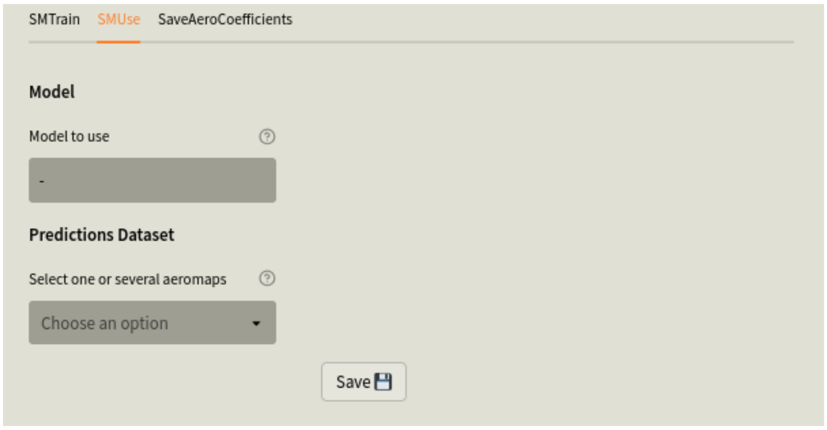


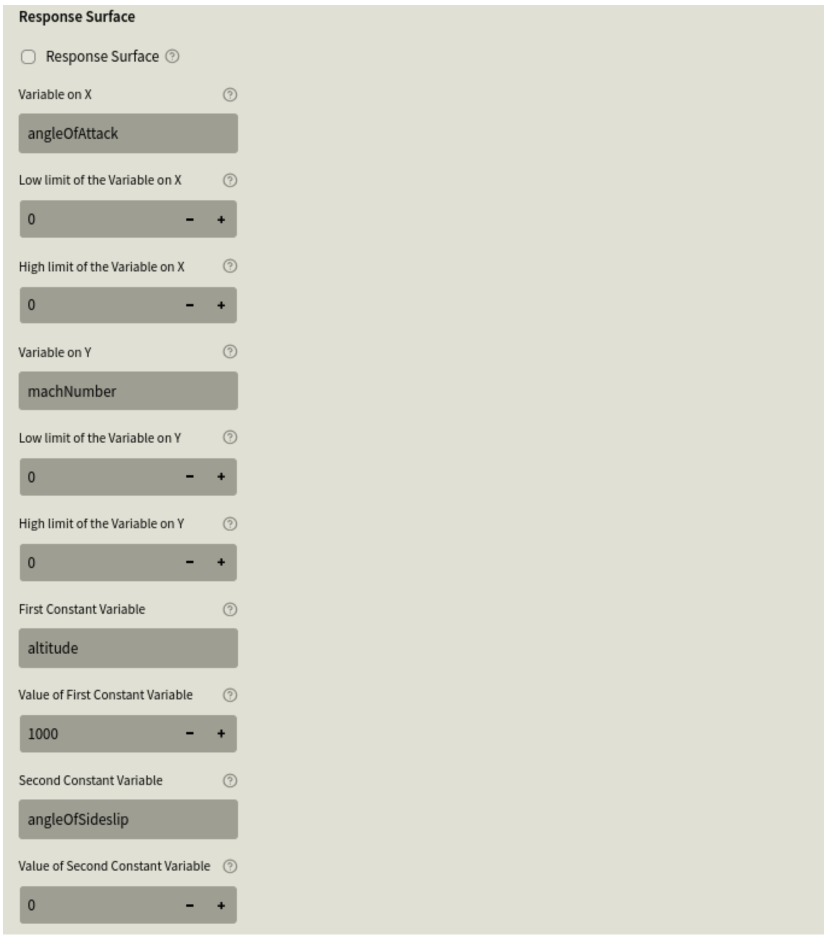
Figure 4.4. GUI for the settings of SMUse module

Once the input dataset and surrogate model are available, the module performs the necessary predictions. The results are then saved in a CSV file and visualized through plots to help analyze the model's behavior.

Following the SMUse module, users can optionally integrate the *SaveAeroCoefficents*.

The module has been expanded from previous possibilities to display the surrogate model's results.

This module is then designed to generate response surfaces and graphical representations of aerodynamic coefficients with respect to selected variables.


Response Surface

☐ Response Surface

Variable on X: angleOfAttack

Low limit of the Variable on X: 0

High limit of the Variable on X: 0

Variable on Y: machNumber

Low limit of the Variable on Y: 0

High limit of the Variable on Y: 0

First Constant Variable: altitude

Value of First Constant Variable: 1000

Second Constant Variable: angleOfSideslip

Value of Second Constant Variable: 0

Figure 4.5. GUI for the settings of a Response Surface in SaveAereoCoefficients

Users can specify which parameters should remain constant while visualizing the variation of others. This feature allows for an intuitive analysis of how aerodynamic coefficients change within the defined domain, providing a valuable tool for model interpretation and validation.

These graphs are visible in the CEASIOMpy *Results* section.

5. Application Case Study

To validate the implemented model and assess the effectiveness of it, the results obtained in this study are compared with those presented in [31] and [32]. The goal is to replicate the methodologies described in these works and evaluate the performance of the surrogate models in different test scenarios.

The first two test cases follow the procedure outlined in [31], where two simple Kriging models are trained and their predictions are evaluated. To replicate the results, the SMTrain module was used for model training, and SMUse was employed for generating predictions. The accuracy of these models is then assessed by comparing their predictions against reference aerodynamic data, demonstrating their validity in interpolating aerodynamic coefficients.

The third test case presents a more complex and realistic application of SMTrain. In this case, the methodology described in [32] is followed, utilizing an updated dataset to train a multi-fidelity Kriging model. The response surfaces and aerodynamic coefficient distributions obtained with this method are then compared with those generated using a simple Kriging model trained without Bayesian optimization. This comparison, conducted through the *SaveAeroCoefficients* module, provides insight into the advantages and limitations of multi-fidelity modeling in aerodynamic analysis.

This study aims to provide a structured validation of the implemented methodology, demonstrating its reliability in surrogate-based aerodynamic modeling and highlighting potential improvements for future applications.

5.1. Variation of Angle of Attack

The first test with the surrogate model aims to predict aerodynamic coefficients for Angles of Attack (AoA) that were not explicitly calculated. To assess the model's performance, different numbers of training points are used. This test is conducted using aerodynamic data from PyAVL, following the workflow shown in Figure 5.1.

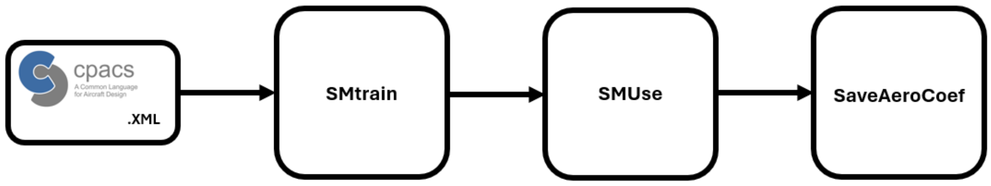


Figure 5.1. CEASIOMpy workflow for training and using the surrogate model for AoA prediction

The test is performed using the unmanned aerial vehicle OPTIMALE, a Medium Altitude Long Endurance (MALE) aircraft with a conventional low-wing configuration and a T-tail. The OPTIMALE configuration was developed as part of the German AeroStruct research project [33] and has been used in several European projects, including AGILE [4]. Using CEASIOMpy's CPACS visualization and modification tool (*CPACSCreator*), the aircraft geometry can be visualized:

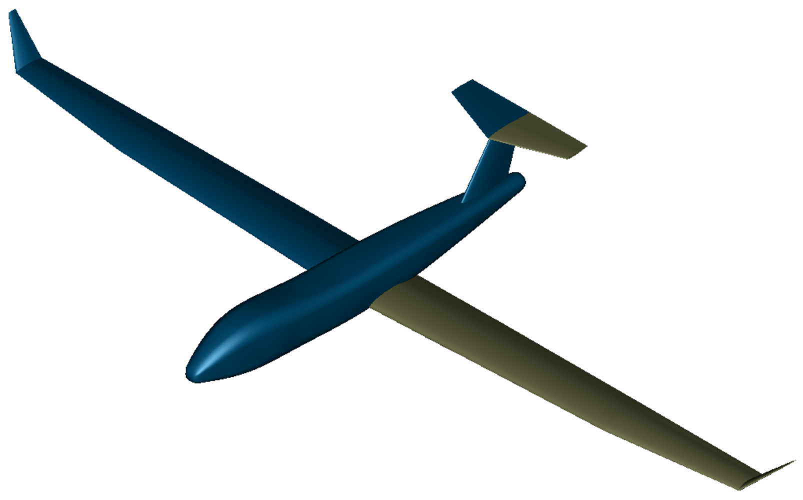


Figure 5.2. OPTIMALE CPACS configuration in CPACSCreator

To determine the minimum number of training points needed for an accurate surrogate model, a simple Kriging model is trained with different dataset sizes, ranging from 17 down to only 3 AoA values, while keeping altitude, Mach number, and sideslip angle constant:

- Altitude: 10,000 m
- Mach Number: 0.5
- Angle of Attack: from -3° to 14°
- Angle of Sideslip: 0°

The predicted C_D values from the surrogate model are then compared at two intermediate AoA values (1.5° and 6.5°) against those calculated with PyAVL. It is important to note that the SMTrain module requires at least 4 points to function correctly. Therefore, to train a model with only 3 points, a simple Kriging model without Bayesian optimization (i.e., without a validation set) is used.

The error between the predicted and actual results is evaluated using the *absolute percentage error*. Figure 5.3 shows how the error decreases as the number of training points increases. For this simple case, a model trained with 6 points achieves a prediction error below 0.1% (specifically, 0.07%), which is well within acceptable limits.

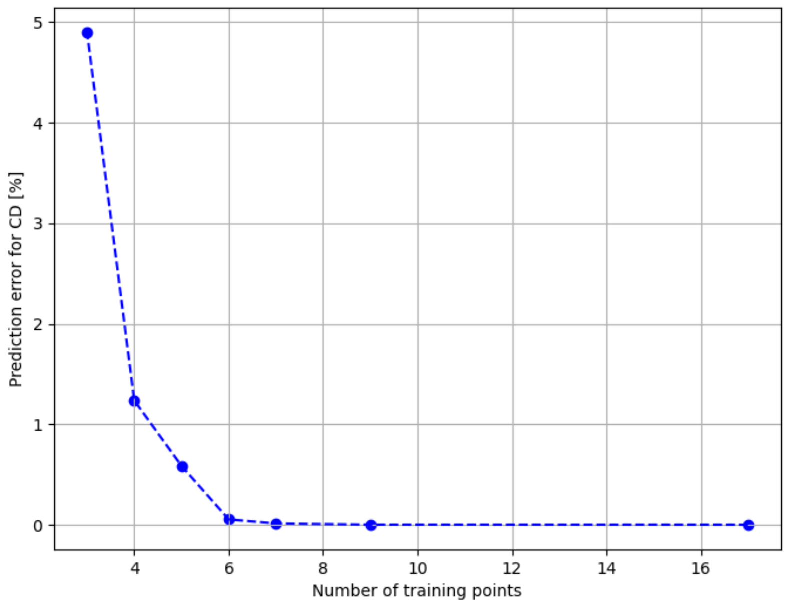


Figure 5.3. Prediction error of C_D at AoA $= 1.5^\circ$

This first test confirms the predictive capability of the module. In particular, it highlights how the Kriging algorithm is well-suited for working with small datasets. This justifies the choice of the model, which proves to be highly useful in aeronautical applications where obtaining new simulation points is often time-consuming.

5.2. Variation of Angle of Attack and Mach Number

In this second test, an additional variable is introduced: a simple Kriging model is trained using a set of points at different AoA and Mach numbers while keeping altitude and AoS constant. The objective is to assess the accuracy of the predictions by directly comparing them with the actual results. Furthermore, to evaluate the impact of Latin Hypercube Sampling (LHS) on prediction accuracy, two different input distributions are analyzed.

Aerodynamic data for this test are obtained from Euler simulations, following a workflow similar to the previous test, but using the SU2 module instead of PyAVL.

The simulations are performed on a simple geometry known as labAR, which serves as a baseline test case in the CEASIOMpy environment implementation process. This model is shown in Figure 5.4.

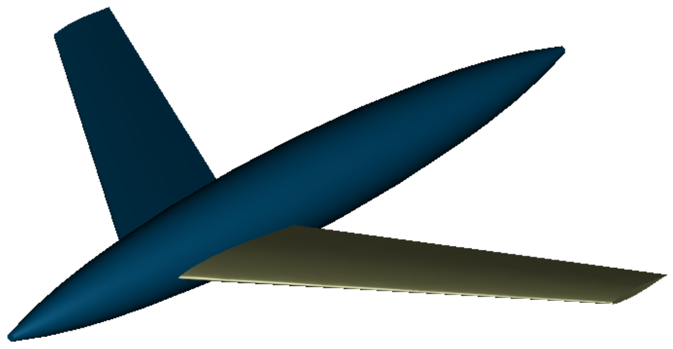


Figure 5.4. LabAR CPACS configuration in CPACSCreator

To use the SU2 solver, a computational mesh is required. This mesh was generated using the CPACS2GMSH module in CEASIOMpy. It is a 3D unstructured mesh with *3.4 million*s elements, obtained after a mesh independence analysis, as shown in Figure 5.5. Several meshes with increasing element counts were tested, and the lift and drag coefficients were monitored to identify the point at which further refinement produced negligible changes in aerodynamic results. The selected mesh size represents the optimal trade-off where the aerodynamic coefficients converge, ensuring reliable predictions without excessive computational cost.

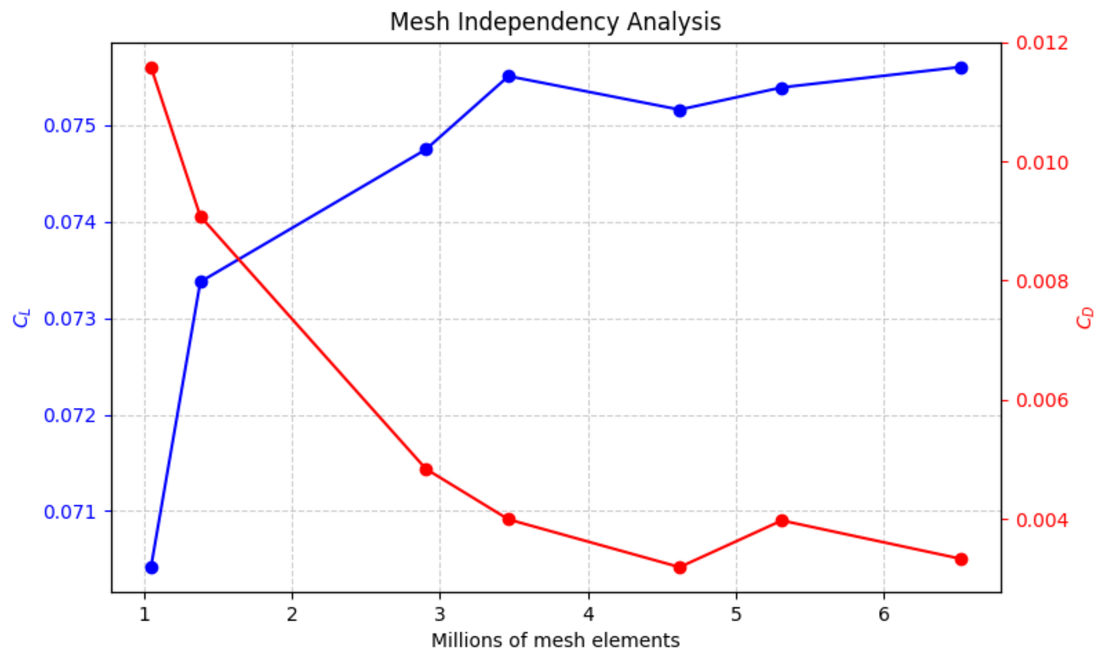


Figure 5.5. Mesh independence study with Lift and Drag Coefficients against the number of elements of the mesh

Two different training sets, each containing 25 points, are used and are visualized in Figure 5.6 and Figure 5.7.

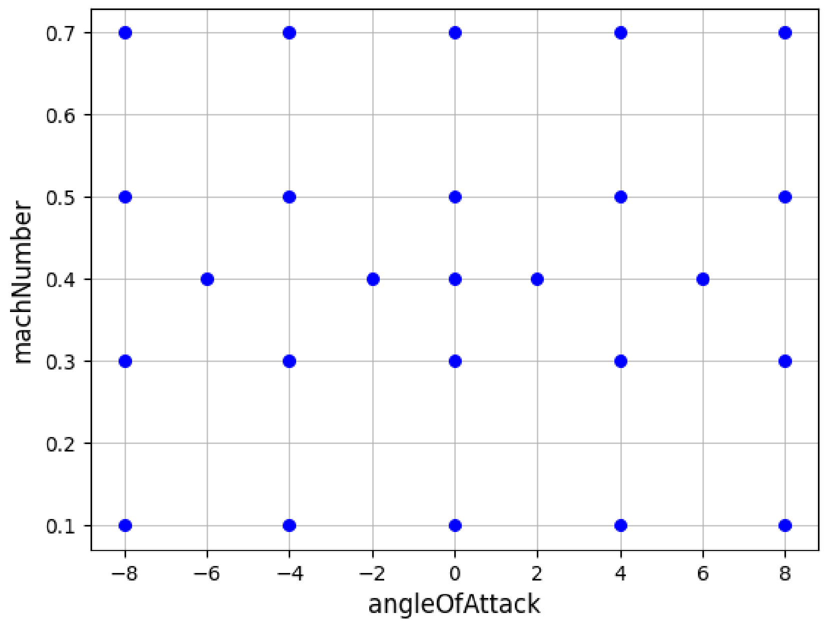


Figure 5.6. Distribution of AoA and Mach number values for the training set, same of [31]

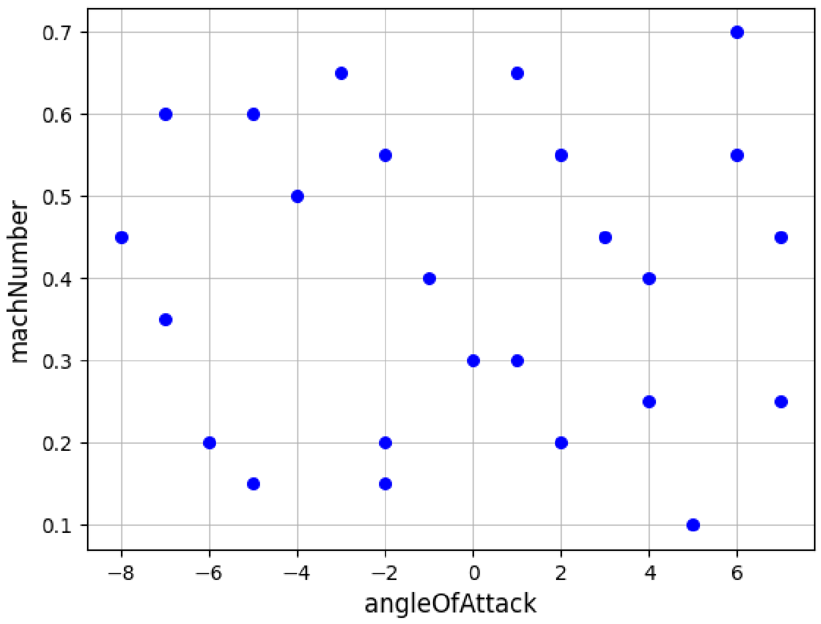


Figure 5.7. Distribution of AoA and Mach number values for the training set, using a Latin Hypercube Sampling

For each dataset, three models were trained, one for each aerodynamic coefficient. The SMT library also allows the training of multi-output models, which can predict multiple coefficients simultaneously. However, this approach increases training complexity, requiring more data points and often resulting in slightly less accurate predictions. Since training a single-output surrogate model, even with a few dozen optimization iterations, takes only a few seconds, the decision was made to train three separate models instead.

Table 2 presents a comparison between the aerodynamic coefficients obtained from SU2 and those predicted by the surrogate model. The test points vary in Mach number and AoA. It is worth noting that the percentage errors are consistently low (generally below 0.25%), confirming the effectiveness of hyperparameter optimization in reducing errors while keeping the number of training iterations limited.

The table includes predictions from models trained on both a grid search sampled dataset and one generated using Latin Hypercube Sampling. In general, the LHS-trained model tends to yield lower errors (highlighted in bold), although for such small errors, the improvement is not always significant.

Altitude	Mach	AoA	AoS	Source	C_L	C_D	C_m
10000	0.35	3	0	SU2	0.1615	0.0014	-0.3133
				Predicted	0.1616	-0.0008	-0.3139
				Error %	**0.0104**	0.2230	0.0577
				Predicted with LHS	0.1617	0.0009	-0.3132
				Error %	0.0256	**0.0534**	**0.0112**
10000	0.45	-3	0	SU2	-0.1662	0.0018	0.3227
				Predicted	-0.1668	0.0027	0.3229
				Error %	0.0556	0.0917	**0.0137**
				Predicted with LHS	-0.1662	0.0015	0.3219
				Error %	**0.0036**	**0.0281**	0.0839
10000	0.55	-3	0	SU2	-0.1728	0.0022	0.3363
				Predicted	-0.1721	0.0044	0.3309
				Error %	0.0783	**0.2162**	0.5344
				Predicted with LHS	-0.1733	0.0025	0.3380
				Error %	**0.0476**	0.0272	**0.1778**

Table 2. Comparison between aerodynamic coefficients predicted by the surrogate model and calculated using SU2, lower error for predictions are highlighted in bold

This model training strategy could be applied to real-world cases where users need reliable aerodynamic predictions. For example, it could be used to generate aerodynamic databases for stability and mission analysis.

5.3. Complete Multi-Fidelity Model Training

This test case considers the D150 conceptual aircraft configuration, a regional jetliner that does not correspond to an existing aircraft, but is similar to an Airbus A320 or a Boeing 737. The aircraft geometry is provided through a CPACS file and imported into the module; it is shown in Figure 5.8.

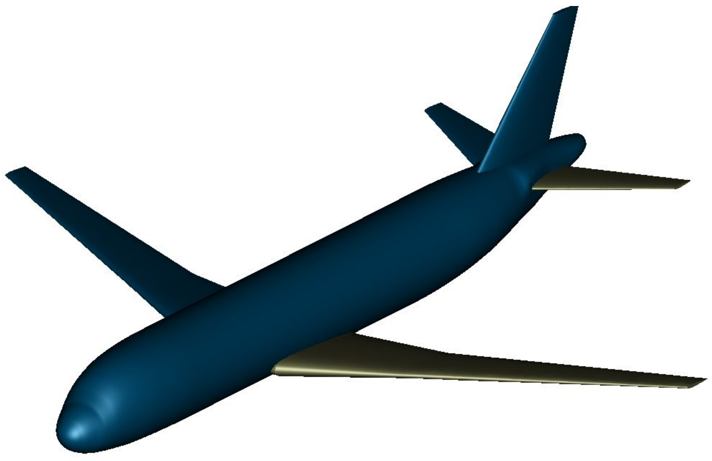


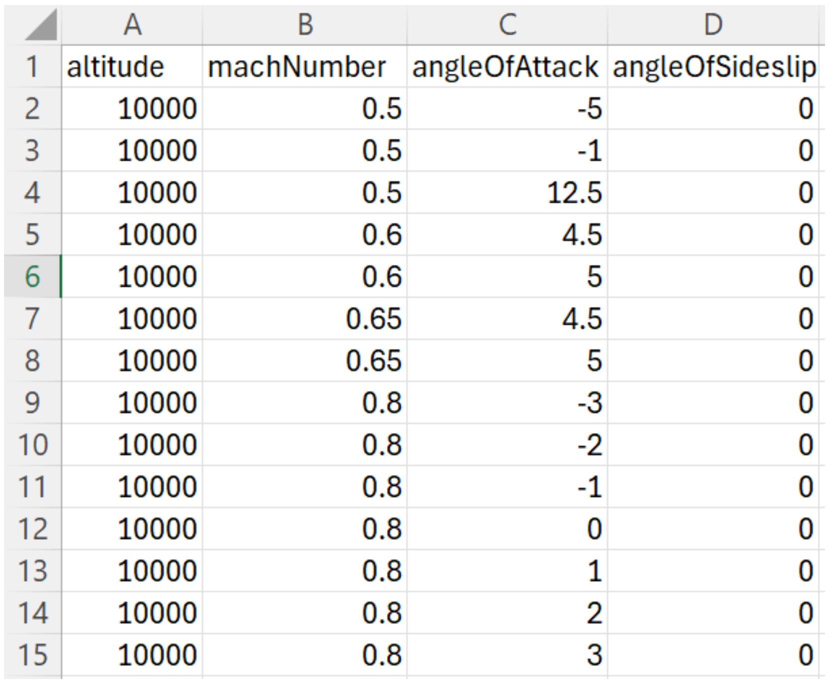
Figure 5.8. Visualization of the D150 aircraft using CPACSCreator module.

This test case uses the module with two fidelity levels: low-fidelity results are obtained from the PyAVL module, while medium-fidelity results are computed using the SU2Run module with the Euler solver.

Three surrogate models are trained for the three coefficients C_L, C_D and C_M, obtaining for each a low RMSE value below 5%. Additionally, the final visualization will include response surfaces for all three models, along with graphs showing variations with respect to the angle of attack at a specific altitude, keeping Mach number as a parameter at three different values. Furthermore, a comparison will be made with models trained without Bayesian optimization, using a simple Kriging algorithm trained solely on the dataset of Eulerian values.

5.3.1. Definition of the Computational Domain

The first step involves defining the computational domain by varying three parameters: flight altitude, Mach number, and angle of attack. The domain is provided to the code in the form of a CSV file, an example of which is shown in Figure 5.9.



	A	B	C	D
1	altitude	machNumber	angleOfAttack	angleOfSideslip
2	10000	0.5	-5	0
3	10000	0.5	-1	0
4	10000	0.5	12.5	0
5	10000	0.6	4.5	0
6	10000	0.6	5	0
7	10000	0.65	4.5	0
8	10000	0.65	5	0
9	10000	0.8	-3	0
10	10000	0.8	-2	0
11	10000	0.8	-1	0
12	10000	0.8	0	0
13	10000	0.8	1	0
14	10000	0.8	2	0
15	10000	0.8	3	0

Figure 5.9. Aeromap example in a CSV file

The three-dimensional domain is defined by the following ranges:

- Altitude: between 9000 m and 11000 m
- Mach Number: from 0.5 to 0.9
- Angle of Attack: from -5° to 15°

To optimize the selection of input data, the domain is sampled using the Latin Hypercube Sampling (LHS) method, ensuring an efficient Design of Experiments (DoE). However, this domain initially represents only a set of triplets (altitude, Mach number, angle of attack) without considering the aircraft's actual flight envelope.

A commercial airliner operates within the constraints of a flight maneuver diagram, which defines operational limits based on load factor, maneuvering capabilities, and airspeed. Aerodynamic forces such as lift and drag vary significantly with two of the three parameters defining the domain: velocity and angle of attack. As a result, the domain is further refined by limiting the angle of attack as the Mach number increases, according to structural and aerodynamic constraints. The updated domain is shown in Figure 5.10 and Figure 5.11.

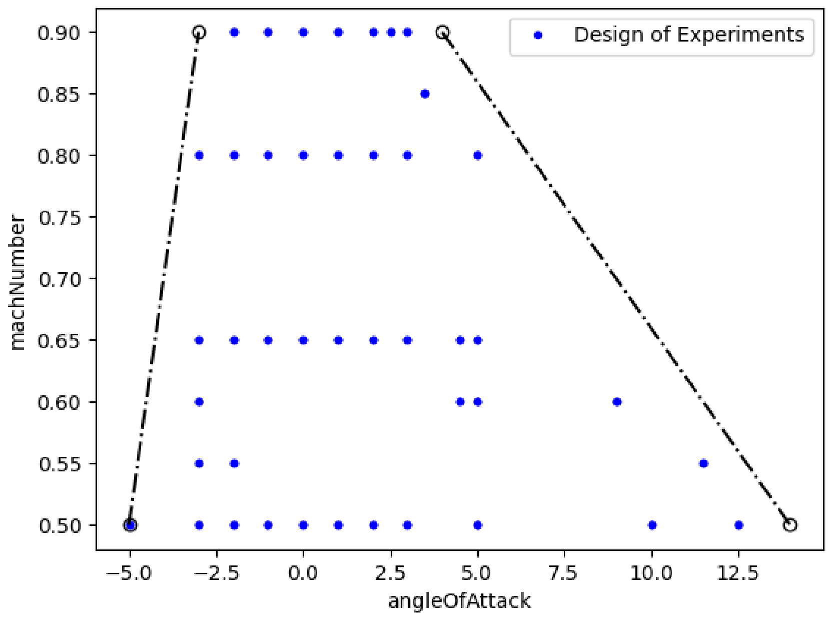


Figure 5.10. Two dimensional visualization of the update domain

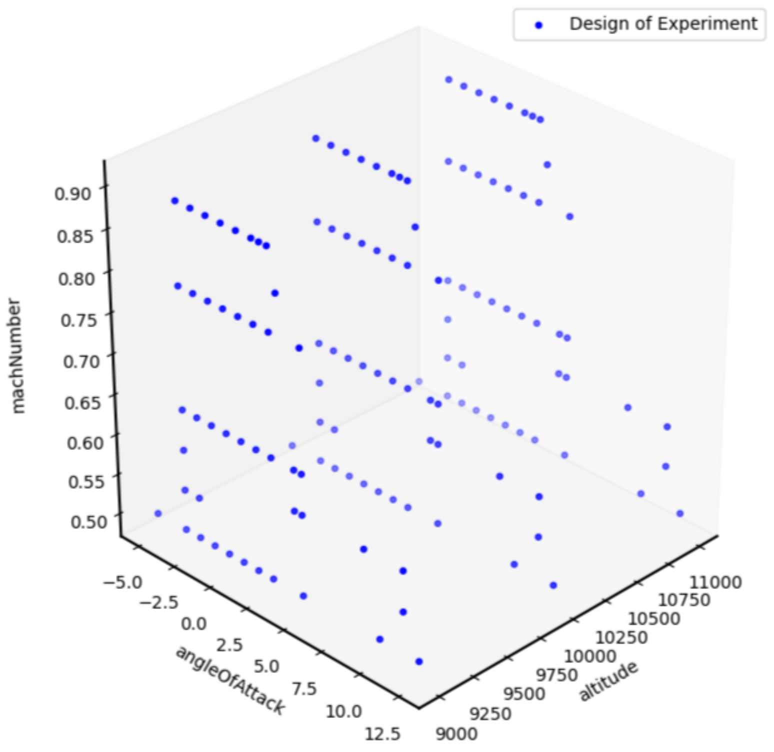


Figure 5.11. Three dimensional visualization of the update domain

5.3.2. Low-Fidelity Simulations

The next step is to set up and run the PyAVL module, which uses the panel method to perform very fast simulations. This Vortex Lattice Method (VLM) represents lifting surfaces, such as wings, as vortex surfaces, neglecting profile thickness and fluid viscosity. Consequently, the aerodynamic coefficients, particularly the drag coefficient (C_D), are approximate, but they still provide a useful initial estimate.

These points form an aerodynamic database (aeromap) that is later used to refine the predictions with higher-fidelity simulations. The obtained input-output pairs (denoted as X and y) are split into three datasets for training, validation, and testing of the surrogate model. Following the standard approach, the split is 70% for training, 20% for validation, and 10% for testing. These datasets serve distinct purposes: training is used to fit the model, validation is employed for hyperparameter tuning, and the test set is used to evaluate model accuracy and prevent overfitting.

Hyperparameter tuning is performed via Bayesian optimization, as described in Section 2.3.3. Once training is completed, the code suggests additional sampling points based on three criteria:

- Selection of points with the highest variance in predictions
- Inclusion of the maximum and minimum angle of attack values
- Addition of points where Mach number is greater than 0.7, since VLM methods become increasingly inaccurate at high Mach numbers

The new dataset, shown in Figure 5.12, ensures the Multi-Fidelity Kriging algorithm has a *nested* structure, meaning the refined solutions share the same input values as the initial dataset.

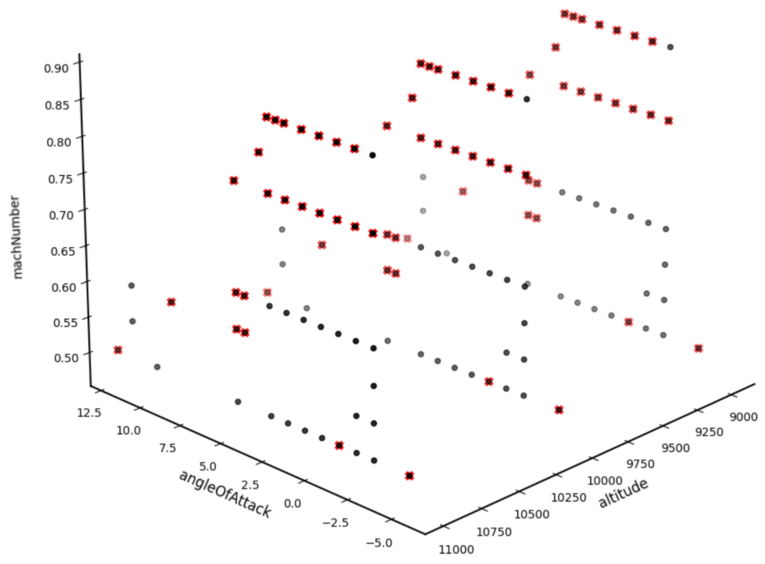


Figure 5.12. Three dimensional visualization of the update domain with new points

5.3.3. High-Fidelity Simulations and Multi-Fidelity Model Training

With the updated dataset, Eulerian simulations are performed using the SU2 module. A 3D Eulerian mesh is generated with the GMSH software, containing *4.2 millions* tetrahedral elements (Figures 5.13 and 5.14) and using as reference the mesh indipendence analysis performed in [34]. Eulerian simulations provide more accurate results than VLM methods, but they are significantly more computationally expensive. While Eulerian simulations capture compressibility effects, they still neglect viscous effects, leading to an underestimation of the drag coefficient.

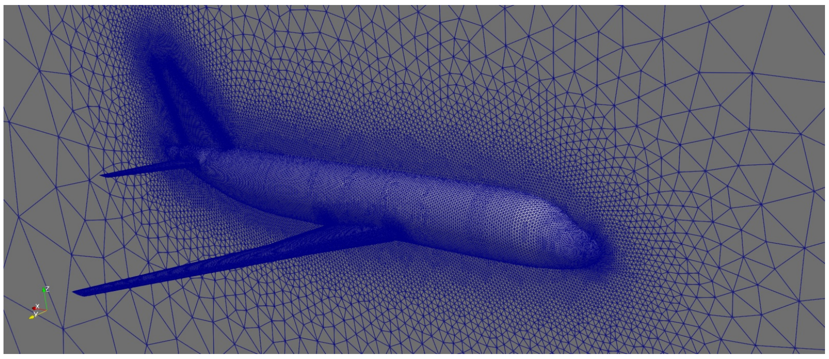


Figure 5.13. Complete D150 mesh created with GMSH

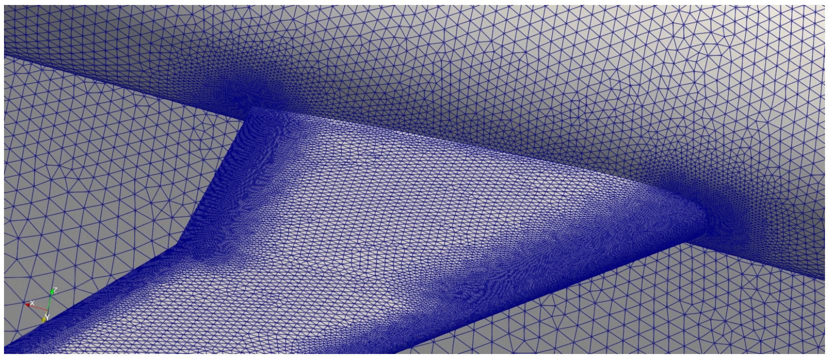


Figure 5.14. Detail of the D150 wing mesh created with GMSH

Once the results from SU2 are collected, the Multi-Fidelity Kriging models are trained. As described in the previous chapter, the algorithm merges the low- and high-fidelity data to enhance prediction accuracy. The hyperparameter tuning process is repeated, and the model is selected based on the lowest combined RMSE and variance values. The final RMSE is below 0.01 for all three models, achieving the target error threshold of less than 5%.

5.3.4. Visualization and Model Comparison

To evaluate the performance of the trained models, response surfaces for the aerodynamic coefficients (C_L, C_D, and C_M) are plotted as functions of the angle of attack and Mach number at a fixed altitude. Additionally, 2D plots of the aerodynamic coefficients as functions of the angle of attack are provided for three different Mach numbers (0.5, 0.65, and 0.8). The scatter points from numerical simulations are overlaid on these plots to visually assess model accuracy.

The images on the left show the response surfaces with scattered data points overlaid. The color scale represents the increasing values of the aerodynamic coefficient plotted on the z-axis, making the response surface easier to interpret. The black scatter points correspond to results from PyAVL simulations, while the red points represent data from SU2 simulations. The same simulation points are shown in the images on the right, this time as dots and crosses, representing slices of the response surface at the three specified Mach numbers.

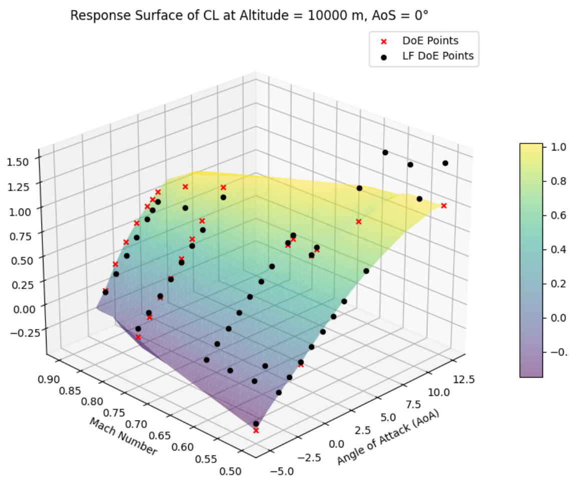
The C_L plot shows that the Euler and VLM results initially agree at low angles of attack. However, as the angle of attack increases, the panel method (VLM) fails to capture flow separation effects because it relies on the potential flow approximation. In contrast, the Euler simulations are able to account for compressibility effects, leading to a more realistic trend that aligns with typical aerodynamic behavior. It is important to note that the degree to which the surrogate model follows the high-fidelity simulations is itself an optimization parameter.

The C_D plot also reveals a significant difference between the VLM and Euler results. As mentioned earlier, neither method accounts for viscous effects. The surrogate model predicts higher drag than the low-fidelity samples because they do not capture wave drag. This is a promising indication that the surrogate model successfully incorporates compressibility effects from the high-fidelity data.

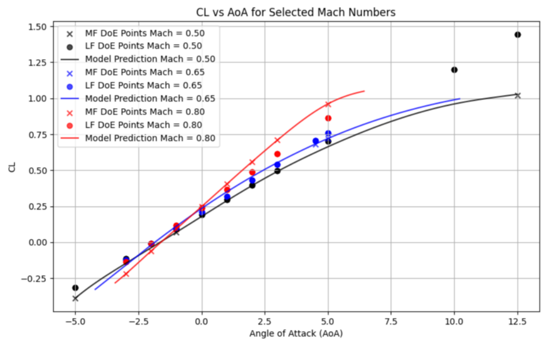
Moreover, the impact of compressibility effects becomes even more evident at higher flow velocities, as the discrepancy between the VLM and Euler results increases with Mach number. This trend is expected, as compressibility corrections become increasingly important in transonic conditions, where shock waves and nonlinear aerodynamic phenomena significantly influence drag.

The C_M plots appear to be less accurately approximated compared to the other coefficients. One possible reason is that moment coefficients are more sensitive to small variations in flow conditions and require more refined modeling to achieve high accuracy. Additionally, discrepancies between the Euler and VLM results may stem from the fact that the VLM method does not fully capture aerodynamic center shifts at higher angles of attack.

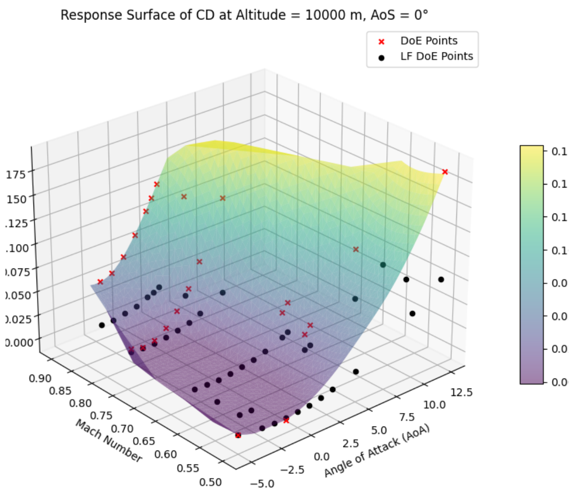
Figures 5.15 illustrate these results, highlighting the influence of Mach number on aerodynamic coefficients.



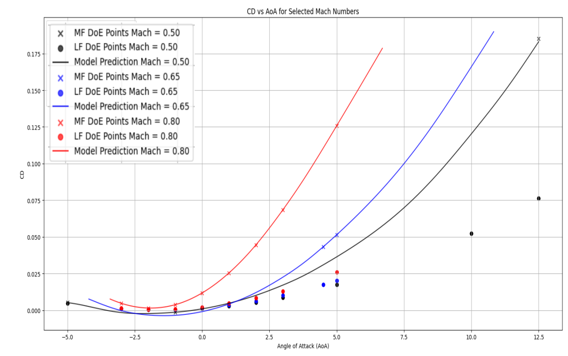
(a) Response surface for C_L



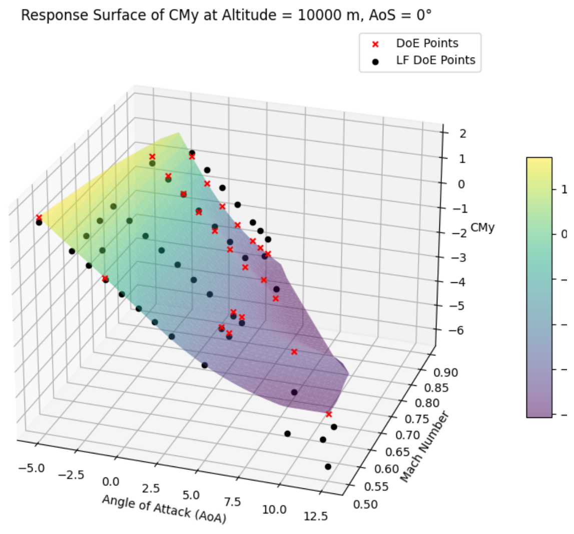
(b) Curves of C_L for different Mach numbers at a fixed altitude



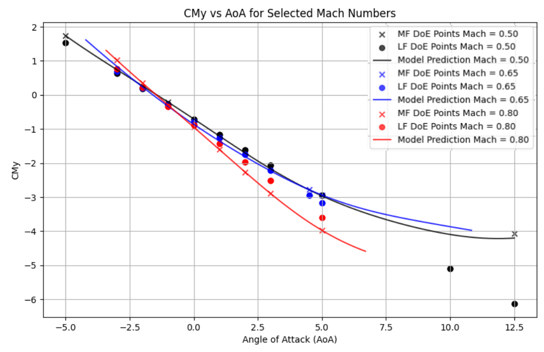
(c) Response surface for C_D



(d) Curves of C_D for different Mach numbers at a fixed altitude



(e) Response surface for C_M



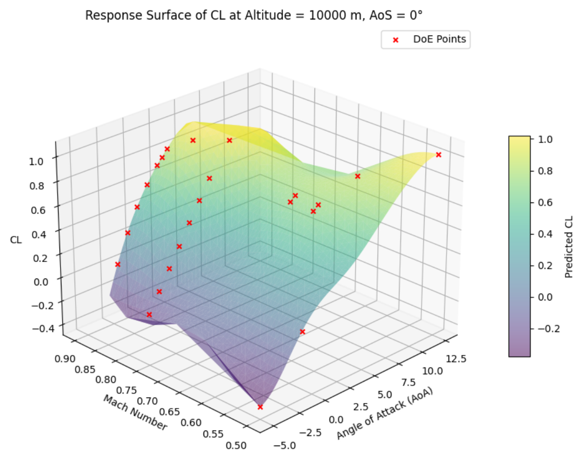
(f) Curves of C_M for different Mach numbers at a fixed altitude

Figure 5.15. Response surfaces (left) and aerodynamic coefficient variations (right) for C_L, C_D, and C_M

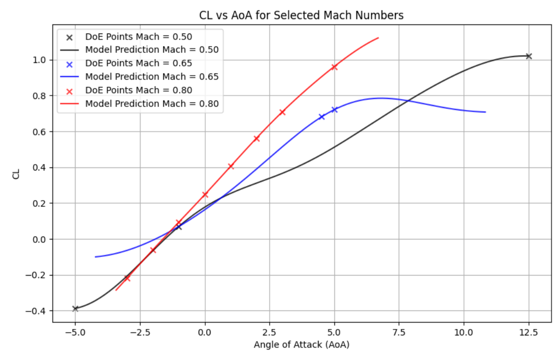
Finally, a comparison is made with models trained using only the Kriging algorithm on Eulerian data, without Bayesian optimization. Figure 5.16 presents the response surfaces and aerodynamic coefficient variations predicted by this simpler model.

Compared to the Bayesian-optimized model (Figure 5.15), the Kriging-only approach shows larger errors, particularly in regions where the Eulerian dataset is sparse. This is evident in the response surfaces, which appear less smooth and more prone to oscillations. Additionally, the aerodynamic coefficient curves exhibit greater discrepancies at higher Mach numbers and angles of attack.

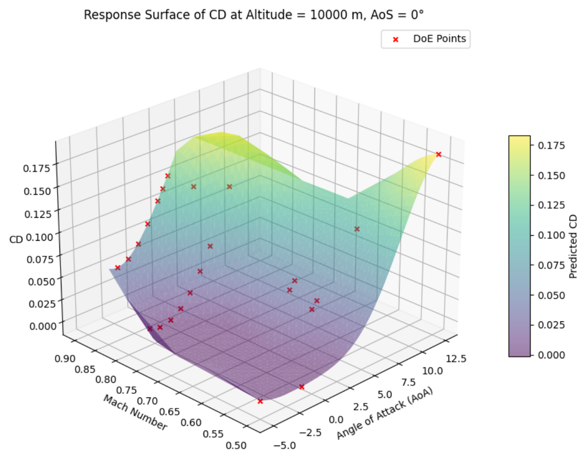
Despite these differences, the root mean square error (RMSE) for the Kriging-only model remains below 5% for C_L (0.029) and C_D (0.046), but exceeds this threshold for C_M (0.089). This suggests that while the Kriging model alone can provide reasonably accurate predictions for lift and drag, its performance in capturing the moment coefficient is less reliable. Further tuning or the use of additional data could help reduce this error.



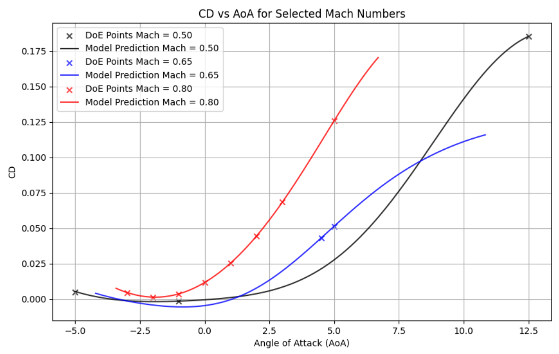
(a) Response surface for C_L (Kriging only)



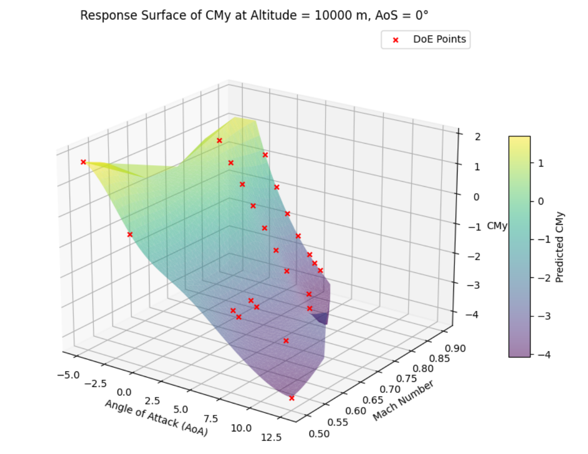
(b) Curves of C_L for different Mach numbers (Kriging only)



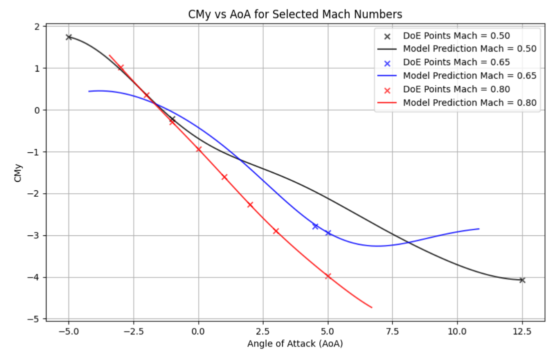
(c) Response surface for C_D (Kriging only)



(d) Curves of C_D for different Mach numbers (Kriging only)



(e) Response surface for C_M (Kriging only)



(f) Curves of C_M for different Mach numbers (Kriging only)

Figure 5.16. Response surfaces (left) and aerodynamic coefficient variations (right) for the Kriging-only model. The predictions show greater errors compared to the Bayesian-optimized approach.

6. Conclusions and Future Developments

This thesis presented the development and implementation of two Python-based modules that enable the training and subsequent use of a surrogate model based on a multi-fidelity strategy. These modules can be used independently or integrated into a CEASIOMpy workflow, providing a valuable tool for the preliminary design phase of an aircraft. Their functionalities have been validated through an application case study, initially using two simpler test cases to assess the accuracy of the surrogate models in controlled conditions. The methodology was then applied to a more complex and comprehensive case to evaluate the performance of the approach in a realistic scenario.

However, the addition of a third fidelity level was not achieved due to convergence issues encountered in SU2 when performing RANS simulations on complex geometries and demanding configurations. While Eulerian simulations did not present difficulties, RANS cases were affected by the onset of shock waves on surfaces, such as the upper wing, and significant flow separation at high angles of attack, particularly in transonic regimes. These challenges prevented the successful integration of higher-fidelity data into the multi-fidelity framework.

The results obtained confirm that surrogate modeling can significantly reduce computational costs while maintaining high accuracy in predicting aerodynamic coefficients. The ability to merge different fidelity levels in a single predictive framework has proven to be a promising approach, offering a balance between computational efficiency and precision.

Based on the work carried out, several possible future developments can be identified:

Firstly, it would be interesting to explore the Co-Kriging algorithm, also available in the SMT library. Unlike the standard multi-fidelity Kriging approach, Co-Kriging allows the use of non-nested data points. This would enable the combination of different datasets, providing greater flexibility. However, implementing this method might introduce additional complexity, requiring careful parameter tuning to achieve optimal results. While non-nested data provide more flexibility, they may lead to inconsistencies between fidelity levels, making the modeling process more challenging. In contrast, using nested data ensures a structured relationship between fidelities, simplifying the information transfer and improving model stability. Furthermore, Co-Kriging would necessitate updating the entire CEASIOMpy environment to a newer Python version, which could be a considerable effort for developers.

Another possible improvement is the implementation of a more advanced adaptive sampling strategy, such as the one suggested by Mengmeng et al. [32]. This strategy incorporates additional criteria, such as maximum curvature (*MaxHessian*) and the Expected Improvement Function, to determine the most relevant points for high-fidelity simulations. Introducing such a technique could enhance model accuracy by focusing computational resources on the most critical regions of the design space.

A natural extension of this work would be the development of an optimization module for aircraft geometry. This module could leverage the surrogate model and response surfaces to perform shape optimization, identifying optimal values for aerodynamic coefficients while adjusting parameters such as wing position, chord length, or taper ratio. The optimization process could be carried out using gradient-based or gradient-free methods, depending on the nature of the objective function and constraints. A potential approach would be to employ genetic algorithms or particle swarm optimization for exploring a broad range of configurations efficiently. Additionally, incorporating constraints related to structural integrity and manufacturability would ensure the feasibility of the optimized design.

Furthermore, an adjoint-based optimization approach could be considered, where the best configuration is determined with respect to a specific coefficient defined as the variable of interest. The adjoint method is particularly advantageous for high-dimensional design spaces, as it provides sensitivity information with respect to multiple design variables at a computational cost independent of their number. This approach could be applied in a framework where the surrogate model accelerates the evaluation of aerodynamic performance, while the adjoint method refines the design by providing precise gradient information. Integrating these techniques into a hybrid optimization strategy could significantly enhance the efficiency and accuracy of the design process.

In summary, the work presented in this thesis provides a foundation for integrating machine learning techniques into aircraft design workflows. Future advancements will further refine aerodynamic prediction and design optimization, making them increasingly practical and effective for real-world aerospace engineering challenges.

References

[1] Gil Press. “A Very Short History of Artificial Intelligence (AI)”. In: *Forbes* (2016). URL: https://www.forbes.com/sites/gilpress/2016/12/30/a-very-short-history-of-artificial-intelligence-ai/.

[2] Ideen Sadrehaghighi. *Essentials of CFD*. CFD Open Series, 2023.

[3] Denis Howe. *Aircraft Conceptual Design Synthesis*. London: Professional Engineering Publishing Ltd, 2000.

[4] J. H. Bussemaker et al. “Collaborative Design of a Business Jet Family Using the AGILE 4.0 MBSE Environment”. In: *AIAA AVIATION 2022 Forum*. Chicago, USA, 2022. DOI: 10.2514/6.2022-3446.

[5] Mengmeng Zhang. “Contributions to Variable Fidelity MDO Framework for Collaborative and Integrated Aircraft Design”. PhD thesis. Stockholm, Sweden: KTH Royal Institute of Technology, 2015. URL: https://www.diva-portal.org/smash/get/diva2:793740/FULLTEXT01.pdf.

[6] Jr. John D. Anderson. *Fundamentals of Aerodynamics*. 6th ed. New York: McGraw-Hill Education, 2017.

[7] Ansys Inc. *What is Computational Fluid Dynamics?* 2025. URL: https://www.ansys.com/simulation-topics/what-is-computational-fluid-dynamics.

[8] Ideen Sadrehaghighi. *Mesh Generation in CFD*. CFD Open Series, 2020. ISBN: 979-8-6876-1234-5.

[9] Christophe Geuzaine and Jean-François Remacle. *Gmsh Reference Manual*. Gmsh Development Team. 2025. URL: https://gmsh.info.

[10] Thomas D. Economon et al. “SU2: An Open-Source Suite for Multiphysics Simulation and Design”. In: *AIAA Journal* 54.3 (2016), pp. 828–846. DOI: 10.2514/1.J053813.

[11] Soledad Le Clainche et al. “Improving aircraft performance using machine learning: A review”. In: *Aerospace Science and Technology* 130 (2023), p. 107792. DOI: 10.1016/j.ast.2022.107792.

[12] Zhi-Hua Zhou and Shaowu Liu. *Machine Learning*. Singapore: Springer Nature Singapore, 2021. DOI: 10.1007/978-981-15-2282-6.

[13] Danilo Bzdok, Naomi Altman, and Martin Krzywinski. “Statistics versus machine learning”. In: *Nature Methods* 15.4 (2018), pp. 233–234. DOI: 10.1038/nmeth.4642. URL: https://www.nature.com/articles/nmeth.4642.

[14] Justin Hodges. *Approaching Machine Learning Problems in Computational Fluid Dynamics and Computer Aided Engineering Applications: A Monograph for Beginners*. Independently published, 2024. ISBN: 979-8-87870-231-7. URL: `https://www.amazon.com/dp/B0CZF4YN31`.

[15] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. "Machine Learning for Fluid Mechanics". In: *Annual Review of Fluid Mechanics* 52 (2020), pp. 477–508. DOI: `10.1146/annurev-fluid-010719-060214`.

[16] Bianca Williams and Selen Cremaschi. "Novel Tool for Selecting Surrogate Modeling Techniques for Surface Approximation". In: *31st European Symposium on Computer Aided Process Engineering*. Ed. by Metin Türkay and Rafiqul Gani. Vol. 50. Elsevier, 2021, pp. 127–132. DOI: `10.1016/B978-0-12-823377-1.50022-0`.

[17] Mandar N. Thombre, Heinz A. Preisig, and Misganaw B. Addis. "Developing Surrogate Models via Computer Based Experiments". In: *12th International Symposium on Process Systems Engineering and 25th European Symposium on Computer Aided Process Engineering*. Ed. by Krist V. Gernaey, Jakob K. Huusom, and Rafiqul Gani. Elsevier, 2015, pp. 157–162. DOI: `10.1016/B978-0-444-63578-5.50032-5`.

[18] E. Andrés-Pérez and C. Paulete-Periáñez. "On the Application of Surrogate Regression Models for Aerodynamic Coefficient Prediction". In: *Complex & Intelligent Systems* 7 (2021), pp. 2025–2038. DOI: `10.1007/s40747-021-00352-9`.

[19] Jichao Li, Mohamed A. Bouhlel, and Joaquim R. R. A. Martins. "Data-Based Approach for Fast Airfoil Analysis and Optimization". In: *AIAA Journal* 56.6 (2018), pp. 2177–2190. DOI: `10.2514/1.J056602`.

[20] M. A. Bouhlel et al. "A Python Surrogate Modeling Framework with Derivatives". In: *Advances in Engineering Software* 135 (2019), p. 102662. DOI: `10.1016/j.advengsoft.2019.03.005`.

[21] Chunlin He et al. "A Review of Surrogate-Assisted Evolutionary Algorithms for Expensive Optimization Problems". In: *Expert Systems with Applications* 202 (2023), p. 117193. DOI: `10.1016/j.eswa.2022.117193`.

[22] Timothy W. Simpson et al. "Metamodels for Computer-Based Engineering Design: Survey and Recommendations". In: *Engineering with Computers* 17 (2001), pp. 129–150. DOI: `10.1007/PL00007198`.

[23] Arthur Rizzi. "Modeling and Simulating Aircraft Stability and Control—The SimSAC Project". In: *Progress in Aerospace Sciences* 47 (6 2011), pp. 318–368. DOI: `10.1016/j.paerosci.2011.05.003`.

[24] CFS Engineering. *CEASIOMpy: Open Source Conceptual Aircraft Design Environment*. Accessed: 2025-03-21. 2024. URL: https://github.com/cfsengineering/CEASIOMpy.

[25] Mohamed Amine Bouhlel et al. *SMT: Surrogate Modeling Toolbox*. Accessed: 2025-03-21. 2024. URL: https://smt.readthedocs.io/en/latest/.

[26] Fabian Pedregosa et al. *Scikit-learn: Machine Learning in Python*. Accessed: 2025-03-21. 2024. URL: https://scikit-learn.org/stable/.

[27] DLR - German Aerospace Center. *SUMO: Simulation of Urban MObility*. Accessed: 2025-03-21. 2024. URL: https://sumo.dlr.de/docs/index.html.

[28] F. A. C. Viana et al. "Special Section on Multidisciplinary Design Optimization: Metamodeling in Multidisciplinary Design Optimization: How Far Have We Really Come?" In: *AIAA Journal* 52.4 (2014), pp. 670–690. DOI: 10.2514/1.J052940.

[29] M. C. Kennedy and A. O'Hagan. "Bayesian Calibration of Computer Models". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63.3 (2001), pp. 425–464. DOI: 10.1111/1467-9868.00294.

[30] L. Le Gratiet. "Multi-Fidelity Gaussian Process Regression for Computer Experiments". PhD thesis. Université Paris-Diderot - Paris VII, 2013. URL: https://tel.archives-ouvertes.fr/tel-00822156.

[31] Vivien Riolo Aidan Jungo and Jan B. Vos. "Using Surrogate Models to Speed up the Creation of Aerodynamic Databases in CEASIOMpy". In: *EWADE-READ 2020* (2020).

[32] Mengmeng Zhang et al. "Data Fusion and Aerodynamic Surrogate Modeling for Handling Qualities Analysis". In: *Aerospace Science and Technology* 98 (2020), p. 105703. DOI: 10.1016/j.ast.2019.105703.

[33] Ralf Heinrich, ed. *AeroStruct: Enable and Learn How to Integrate Flexibility in Design*. Vol. 138. Notes on Numerical Fluid Mechanics and Multidisciplinary Design. Springer International Publishing, 2018. ISBN: 978-3-319-72019-7. DOI: 10.1007/978-3-319-72020-3. URL: https://link.springer.com/book/10.1007/978-3-319-72020-3.

[34] Aidan Jungo et al. "Benchmarking New CEASIOM with CPACS Adoption for Aerodynamic Analysis and Flight Simulation". In: *Aircraft Engineering and Aerospace Technology* 90.4 (2018), pp. 613–626.

Nomenclature

Symbol	Definition
α	Angle of attack (deg)
β	Sideslip angle (deg)
p	Pressure (Pa)
T	Temperature (K)
η	Efficiency
μ	Dynamic viscosity (Pa·s)
ν	Kinematic viscosity (m^2/s)
τ	Shear stress (Pa)
θ	Angle relative to the perpendicular (deg)
δ	Discrepancy function
q_∞	Freestream dynamic pressure (Pa)
Re	Reynolds number
g	Gravity acceleration (m/s^2)
t	Time (s)
M	Mach number (–)
ρ	Air density (kg/m^3)
V_∞	Freestream velocity (m/s)
a	Speed of sound (m/s)
e	Energy (J)
u	Velocity (m/s)
y^+	Non-dimensional wall distance
x_{cp}	Center of pressure location (m)
c	Chord (m)
l	Reference length (m)
S	Reference area (m^2)
D	Drag force (N)
L	Lift force (N)
M_{moment}	Moment (Nm)
N	Normal force (N)
A	Axial force (N)
C_D	Drag coefficient
C_L	Lift coefficient
C_M	Moment coefficient
R	Resultant aerodynamic forces (N)

∇	Gradient
∂	Partial derivative
d	Derivative
λ	Variance penalty factor
m	Design points
n	Independent variables
$X \in \mathbb{R}^{m\times n}$	Design matrix
$Y \in \mathbb{R}^{m\times q}$	Response matrix
x^*	Unsampled location
$\hat{y}_k(x^*)$	Kriging prediction
σ^2	Process variance
$R(x^{(i)},x^{(j)})$	Correlation function
θ_l	Correlation hyperparameter
$\hat{s}^2(x^*)$	Prediction MSE
$\mathbf{1}$	Ones vector
ψ	Correlation vector
Superscripts and Subscripts	
∞	Freestream condition
eq	Equivalent
m	Mean
max	Maximum
min	Minimum
LE	Leading edge
TE	Trailing edge
u	Upper surface
l	Lower surface
x	x-direction
y	y-direction
z	z-direction
Abbreviations and Acronyms	
AI	Artificial Intelligence
ANN	Artificial Neural Networks
AR	Aspect Ratio
CAD	Computer-Aided Design
CAM	Computer-Aided Manufacturing
CFD	Computational Fluid Dynamics
CFL	Courant-Friedrichs-Lewy Number

CPACS	Common Parametric Aircraft Configuration Schema
CPU	Central Processing Unit
CSV	Comma-separated values
DoE	Design of Experiments
DLR	German Aerospace Centre
EDA	Exploratory Data Analysis
EI	Expected improvement
HPC	High-Performance Computing
GP	Gaussian process
GUI	Graphical User Interface
LES	Large Eddy Simulation
LHS	Latin Hypercube Sampling
MAE	Mean Absolute Error
MDO	Multidisciplinary Design Optimization
ML	Machine Learning
MSE	Mean Squared Error
LF	Low Fidelity
HF	High Fidelity
MF	Multi-Fidelity
PDE	Partial Differential Equations
RANS	Reynolds-Averaged Navier-Stokes
RMSE	Root Mean Squared Error
SGD	Stochastic Gradient Descent
SMT	Surrogate Modeling Toolbox
SVM	Support Vector Machines
VLM	Vortex Lattice Method