Corso di Laurea in Ingegneria Aerospaziale
- Spazio -

Tesi di Laurea II livello

Study of innovative sensor configurations for evaluating the loads acting on the aircraft structure

Relatore: Prof. **Maggiore Paolo**
Co-relatori: Ing. **Dalla Vedova Matteo Davide Lorenzo**, Ing. **Berri Pier Carlo**, Ing. **Quattrocchi Gaetano**

Candidato: Lanciotti Edmondo

POLITECNICO DI TORINO
Aprile 2021

ABSTRACT

In sizing a flight control system, one of the main required parameter is the hinge moment. The hinge moment of a control surface is given by the forces acting on the surface itself multiplied by the mechanical transmissions arm. This moment is assumed to be evaluated on the hinge axis, which is generally the only rotating component of an aircraft wing. Due to its high rate of variation depending on flight attitude and conditions, hinge moment boundaries are often numerically evaluated during design phase.

The present study aims to introduce a methodology and draft guidelines to collect and analyze a great number of data entry, in order to fill a dataset to be used for building a simple Deep Learning Network (DLN) model. The model will actually represent a "virtual sensor" able to estimate wing hinge moments during some flight scenarios.
Even if it has been chosen a real case study in terms of geometry and structure, to limit the complexity not all data have been measured by own from beginning. In fact, every approach for the calculation of a wing hinge moment needs the engineer to be informed at least about the aerodynamic characteristics of the wing, not to mention flight conditions. Moreover, through the analysis of these characteristics it is also possible to emulate a particular sensor detection, then to consider it as a useful input for the aforementioned network.

Aerodynamic analyses have been performed through Xflr5, an analysis tool for airfoils, wings and planes operating at low Reynolds Numbers. MatLab has been used to capture and collect data about hinge moment value and parameters (aileron deflection, flight speed, angle of attack) which produced it. Several simulations have been run to prepare a significant dataset and reach a dimension for DLN model to be well trained. All about data exploratory analysis and DLN modelling (building, training, results evaluation) has been performed through "Jupyter", a non-profit, open-source project evolved to support interactive data science and scientific computing. After having trained the model on how to forecast hinge moment values starting from some inputs, obtained predictions have been evaluated to better understand performances and capabilities of the model itself.

The DLN has been optimized to maximize forecasting "goodness" without complicate too much the adopted structure. Indeed, only if prediction error is under a certain threshold, the model may be considered "ready to deploy". The design has been driven by a simple but fully modular approach, in which each component is open for future developments, providing at the same time a guide line about the expected output to move towards the DLN. It is also important to highlight how the choice of some parameters can affect hinge moment magnitude and a simple model makes this possible and easier.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The concept of "Information Revolution" has been introduced in 1974 by Donald M. Lamberton [Wik20c]. Since then, humanity explored and improved the potential of computer technology. Nowadays the dawn of the digital era is something passed: the world adapted to informatics changes and got deeper in domination of these technologies.
One of the most discussed topics in terms of feasibility (and ethic), is the application of artificial intelligence to different aspect of life. The AI has the power to lighten and improve lots of tasks in a different way comparing to the "simpler" calculation algorithms. One can imagine what a computer could be able to do if, over the fast calculation capabilities, it also could master decision making.

Actually, the AI field is pretty young, and just a rudimentary emulation of the human mind is possible. What is missing is the deduction/intuition skill which characterize the human being. Moreover, even if it's possible to realize a complex brain-like structure just with informatic algorithms, hardware availability and technology still represent substantial problems[Som19]. Therefore today these kind of technologies are not yet evolved enough to completely replace the human component (and maybe this will never happen), but they are already used in fields like engineering and economy to assist some of the tasks.

Machine Learning and Deep Learning are branches of artificial intelligence. They are useful to estimate, to forecast, and to analyze datasets and trends of functions. Generally speaking, they use statistical methods to increase performances of an algorithm which has to identify patterns within data.
The great potential of these methods founds application in many disciplines. For example, NASA uses algorithms based on decision trees to classify celestial objects for Palomar Sky Survey project. This sky mapping resulted in 3 terabytes of image data, most of them automatically classified via AI. An other kind of application is the automatic guide control for vehicles, for example the ALVINN system, which used his own strategies to learn how to drive without human assistance for 90 minutes at 70 miles per hours on public roads (among other cars) [Wik19]. This is pretty near the topic of the thesis because, with similiar techniques, most of the models based on sensor controlling can be realised. According to this, what follows is the implementation of a simple IA model in the aerospace field.

1.1 Purpose

The thesis will deal with the adoption of a cutting-edge technology to evaluate the response to a sensor input. Behind the analysis, there is the will to show strengths and weaknesses of the implementation of this kind of technology in the field of avionics.

1.2 Objective

The thesis will discuss the adoption of a deep learning network model based on an artificial sensor input to forecast moment value on the hinge axis of a wing. Basically, this will be the presentation of a procedure which starts with the determination of the dataset and all related data-acquisition criteria and ends with DLN model performances evaluation.

1.3 Case study

To reach the objective, it has been chosen a case study, "ANUBI" aircraft. It is a low winged monoplane with inverted T tail and tracting motor designed to take part to the "Air Cargo Challenge 2017". The Icarus Polito team, who made the aircraft,

"*was born in Politecnico di Torino thanks to the passion for aerospace of the students that decided to move a step in its creation in the year 2015. Among the missions that Icarus decided to undertake the day of its constitution, the Air Cargo Challenge represented a good starting point and a great opportunity to enrich the personal expertise and knowledge of the students involved.*"

(from ANUBI Project report for ACC [Tea17])

As it will be seen (sections 2.4.3 and 2.4.3), it has been necessary to make some simplifications/assumptions (structure, flight conditions...) to better adapt the objective to the case study.

2 PROJECT OVERVIEW

2.1 The hinge moment

The hinge moment on a wing/tail is given by the forces acting on the control surface multiplied by the arm of mechanical transmissions. Different kind of forces can be classified:

- Mass Forces: if the gravity centre of the control surface is not placed on the hinge axis. That means the presence of a further moment arm. At the moment, these forces will be neglected, because a statically balanced control surface is assumed.
- Aerodynamic Forces, which are:
  - Pressure Forces;
  - Friction Forces (neglected too);

A lift-like formulation may be considered both for wing and for tail hinge moment evaluation:

$$H = C_H S_e c_e \frac{1}{2} \rho V^2 \tag{2.1.1}$$

It is implied that H is applied on the hinge axis, the only rotative component of the wing. The hinge moment coefficient C_H primarily depends on:

- Angle between:
  - Control surface zero-lift axis $C_{L_t} = 0$
  - Horizon α_t
- Control surface deflection angle δ.

In the special case where control surface has a symmetric profile, the zero-lift axis is coincident to the chord, so it can be defined:

$$\alpha_s = (\alpha_t)_{\delta=\delta_{tab}=0^\circ} \tag{2.1.2}$$

Considering tail, there are three different kind of tabs, which vary depending on their own structure and function:

- Trim Tab;
- Servo Tab;
- Compensator Tab;

The hinge moment coefficient can be written as a linear combination of other factors (tab deflection included), as follows:

$$C_H = b_0 + b_1 \alpha_s + b_2 \delta + b_3 \delta_{tab} \tag{2.1.3}$$

Where α_s, δ and δ_{tab} have structural boundaries, so:

- The first parameter must be $b_0 = 0$ when profile is symmetric;

- As regards C_H variation on respect to the other angle parameters, a linear relationship can be assumed:

$$b_1 = \frac{\partial C_H}{\partial \alpha_s} = cost$$

$$b_2 = \frac{\partial C_H}{\partial \delta} = cost$$

$$b_3 = \frac{\partial C_H}{\partial \delta_{tab}} = cost$$

b_1, b_2, b_3 have a negative value because positive rotations of the element to which they refer correspond to a diving bent for the aircraft. Theoretical treatments for the exact calulation of these coefficients are complex. Nevertheless some geometric features have been found having a more noticeable influence in the calculus. Precisely:

- Profile type;
- Chord Ratio between elevator and whole tail surface $\frac{c_e}{c_t}$;
- Chord ratio $\frac{c_b}{c_e}$, where c_b is that chord portion which goes from hinge axis to leading edge of the control surface, called "beak";
- Elevator's beak shape;
- Chink between stabiliser and elevator. (if exists and depending on entity and shape);
- Angle defined by the trailing edge of the profile β;
- Trailing edge shape;
- Reynolds and Mach numbers: this last dependency is the reason why b_1, b_2 and b_3 couldn't always be taken as constant for the whole flight envelope;

The hinge moment can be written with an alternative formulation, considering that the incidence α_s is:

$$\alpha_s = \alpha_{wb}\left(1 - \frac{\partial \varepsilon}{\partial \alpha}\right) - i$$

Where α_{wb} is:

$$\alpha_{wb} = \alpha + \frac{\alpha_t}{a}\frac{S_t}{S}i$$

By substitution in C_H expression, following relationship can be obtained:

$$C_H = b_0 - b_1\frac{i}{1+F} + b_1(1 - \frac{\partial \varepsilon}{\partial \alpha})\alpha + b_2\delta + b_3\delta_{tab} \qquad (2.1.4)$$

Some terms of the previous formulation can be grouped as follow:

$$C_{H_0} = b_0 - b_1\frac{i}{1+F}$$
$$C_{H_\alpha} = b_1\left(1 - \frac{\partial \varepsilon}{\partial \alpha}\right)$$

That is the relationship between equations 2.1.3 and 2.1.3 coefficients.

However, for the present work, hinge moment is numerically evaluated, using pressure force distribution.

2.2 Procedure

The study has been divided in 3 main steps:

1. Aerodynamic Study: to evaluate the pressure coefficient on the whole wing area and the hinge moment of the interested part of the wing;
2. Dataset preparation: using Matlab as main code language to automatize the activities as possible;
3. Creation of the model: using Python as main code language to train the forecasting model.

Each phase uses previously acquired data, so what follows can be considered as a procedure (one of many) to build a specific model with a specific goal, that is to forecast the hinge moment of an aircraft's wing.
At the end it will be discussed the preparation for further developments of the project in terms of:

- Structural studies: realizing a wing FEM map using the obtained pressure distribution;
- Different input consideration: to start looking to the final goal of the whole project, that is to realize a forecasting model able to use actually installed (or to be installed) sensors, to perform the hinge moment prediction on ANUBI. This means a correlation between moment and other inputs, which may be different or cumulative with respect to the used ones.

2.3 Data

2.3.1 Geometry

As regard starting data, it was primarily necessary to know the geometry of the case study. These informations are contained in ICARUS databases. Here is reported just what it's strictly necessary to understand the project. The hinge moment problem has been studied considering only the wing of

Element	Value	Misure
Wingspan	3.8	m
Aspect Ratio	18.56	
Wing Surface	0.778	m^2
Max Chord	0.27	m
Min Chord	0.15	m
Wing Mass	0.62	kg

Table 2.1: Wing Geometry Data

the aircraft, so table 2.1 already contains most of what is needed to start

modelling.
By the way, during the first part of the design, other data have been used to produce CADs and draws, useful to better understand the situation. Unfortunately, these informations are confidential and not much can be showed here. Note that the half wing is partitioned in three portions. Each one

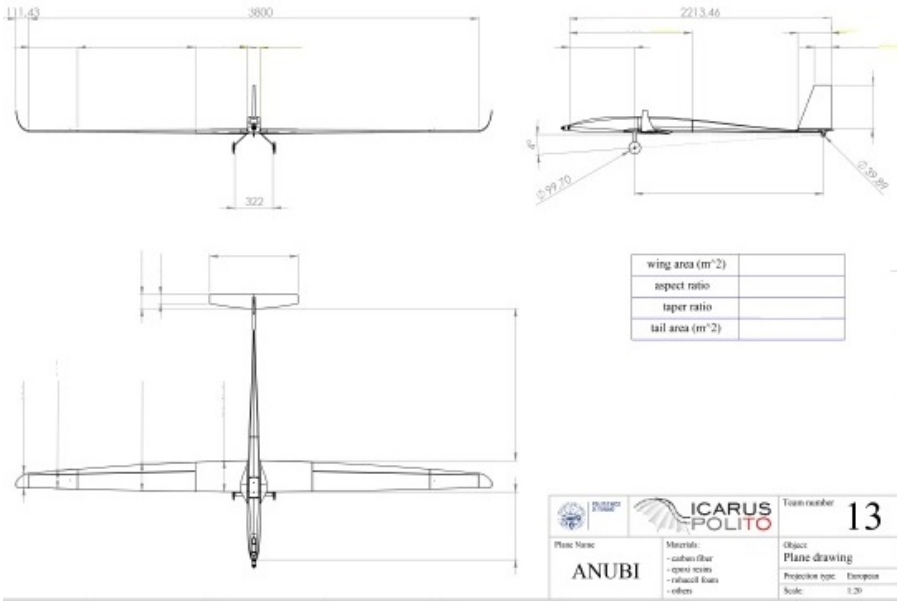


Figure 2.1: Anubi technical draw (from ICARUS team)

has its own control surface. From 3.4 it cannot be seen the hub control surface because of its exclusive nature. It has been drawn as part of the fixed surface, but it's actually a retractile flap (see 2.1).

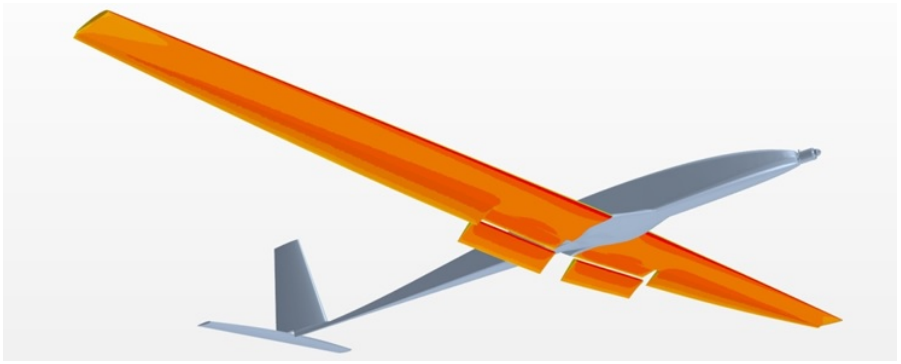


Figure 2.2: ANUBI model visualization ([Tea17])

2.3.2 DLN inputs

As it will be seen, a forecasting model works learning from a bunch of samples and using acquired skills to forecast something new. To clarify the nomenclature:

- Example: it is a input to which correspond a well known output. In other words, it is the set of data the designer give to the calculator, accompanied by the expected response which the calculator has to learn and interpret;
- To forecast: once the calculator had seen a right number of examples it becomes able to understand the correlation between inputs and outputs and, moreover, to predict a particular output even if it receives in input something it has never seen before, provided it is of the same type of what it learned from.

For the present project, it has been chosen to predict the hinge moment value (output), basing on three parameters (inputs):

- Control surface deflection δ;
- Flight velocity v;
- Angle of attack α;

As regard velocity, ICARUS maneuver diagram (fig. 2.2) shows that, in extended flap configuration (black line), ANUBI goes from 12 $[m/s]$ to 22 $[m/s]$.
Velocities:

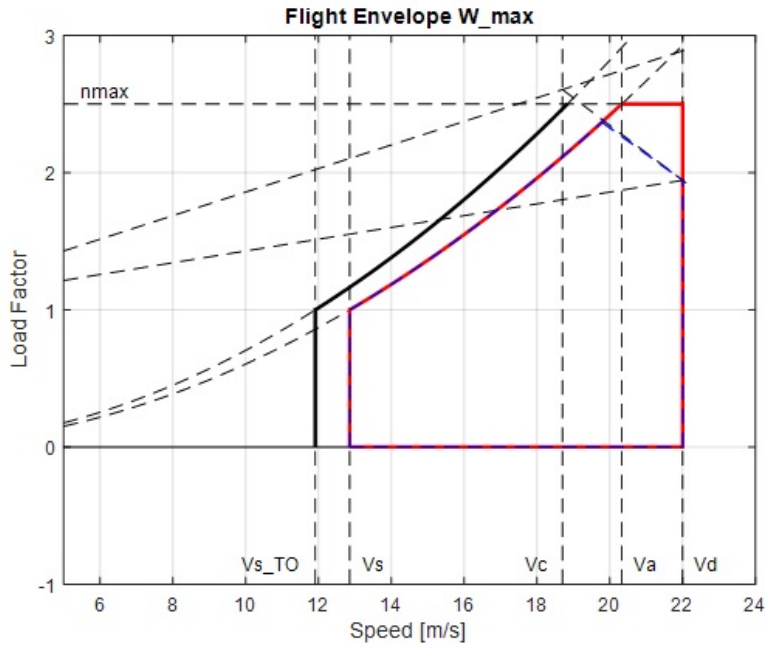


Figure 2.3: Maneuver Diagram ([Tea17])

- Vs_TO: stall speed with extended flap;
- Vs: stall speed with retracted flap;
- Vc: cruise speed;
- Va: maneuver speed;
- Vd: design speed (diving speed);

Note that the used range of velocities considers a mission profile that includes take off, climbing and cruise. By the way, the range of incidences considered does not take into account a particular mission profile phase. It can be seen as a series of incidence perturbation during a constant cruise flight at a certain speed.
Last parameter, δ, is defined by construction in a specific range of variation. Table 2.2 shows detailed informations about what has been said. These

Parameter	Min Value	Max Value	Mean Value	Step	Nodes	Misure
Control surface deflection	−4	4	0	1	9	[deg]
Velocity	12	22	17	0.5	21	[m/s]
Angle of attack	−5	10	2.5	1	16	[deg]

Table 2.2: Input ranges of variation

ranges of variation had been used to generate a certain number of complete examples. What was expected is a number of examples equal to the number of combinations that could have been obtained from the parameters themselves. Table 2.3 shows this, considering a first hypothesis of final dataset dimension and partition in:

- Total examples: total number of parameters combinations;
- Training examples: to train the model;
- Test examples: to test if the model is training well;
- Validation examples: to evaluate the finished model performances.

Dataset	Percentage of "Total"	Data Points
Total		3024
Training	70	2116.8
Test	20	604.8
Validation	10	302.8

Table 2.3: Input ranges of variation

2.4 Tools

To perform the job, different softwares and IDE have been used:

- Xflr5
- Excel

- Matlab
- Jupyter
- Hyperworks (future developments)

2.4.1 Xflr5 [Web21]

XFLR5 *is an analysis tool for airfoils, wings and planes operating at low Reynolds Numbers*. It includes:

- XFoil's Direct and Inverse analysis capabilities;
- Wing design and analysis capabilities based on the Lifting Line Theory, on the Vortex Lattice Method, and on a 3D Panel Method.

In every CFD problem, everything starts with the Navier-Stockes equations. They are for aerodynamics what Maxwell equations are for electrostatics. One of the millennium problem is to find an exact solution to these equations:

$$\begin{aligned} \frac{\partial \delta}{\partial t} u_i + \sum_{j=1}^{n} u_j \frac{\partial u_i}{\partial x_j} &= \nu \Delta u_i - \frac{\partial p}{\partial x_i} + f_i(x,t) \qquad (x \in R^n, t \geqslant 0) \\ \operatorname{div} u &= \sum_{i}^{n} \frac{\partial u_i}{\partial x_i} \qquad (x \in R^n, t \geqslant 0) \\ &\text{with initial conditions} \\ u(x,0) &= u(x) \qquad (x \in R^n, t \geqslant 0) \end{aligned} \tag{2.4.1}$$

By now, just a numeric solution can be evaluated. To deal with so complex problems, it is commonly used to make assumptions:

- Viscosity is neglected, so that Euler equations can be obtained;
- Flow is considered irrotational, so it can be assumed potential;
- Mach number is considered under a certain value (0.3/0.4), so the flow can be considered compressible and the velocity potential equation can be reduced to Laplace equation.

NOTES ON VIRTUAL SINGULARITIES [Coro9] This paragraph deals with elementary solutions for Laplace equation. As it will be seen, a body (lifting or not), can be modelled as a distribution (or a combination of distributions) of some "singularities" which depend on some boundary conditions. Even if their intensities are unknown at the beginning of the problem, it can be said that they consist in the elementary solutions of the Laplace equation and their meaning is in analogy with the electrostatics: they are for the aerodynamics, what "charge" and "dipole" are for electrostatics:

- **Source and Sink**

$$\begin{aligned} \phi(r,\theta) &= \frac{\dot{m}}{2\pi} \log(r) \\ \varphi(r,\theta) &= \frac{\dot{m}}{2\pi} \theta \end{aligned}$$

  Where:

- $\dot{m} \rightarrow$ mass flow, emitted by the source itself [1];
- $r \rightarrow$ distance from the source;
- $\theta \rightarrow$ angular distance from the source;

- **Vortex**

$$\phi(r,\theta) = \frac{\Gamma}{2\pi}\theta$$
$$\varphi(r,\theta) = \frac{\Gamma}{2\pi}\log(r)$$

Where:

- $\Gamma \rightarrow$ Vortex Intensity, or Circulation
- Vortex is the dual motion field with respect to the source.

- **Dipole, or Doublet**

$$\phi(r,\theta) = -\frac{\mu}{2\pi}\frac{\cos(\theta)}{r}$$
$$\varphi(r,\theta) = \frac{\mu}{2\pi}\frac{\sin(\theta)}{r}$$

Where:

- $\mu = qd \rightarrow$ a dipole is the effect combination between a source and a sink (source opposite) placed at a distance $d \rightarrow 0$ so that the product μ always keeps a finite value.

With the previously mentioned assumptions, Laplace equation can be solved with high precision, but one other problem raise: neglecting the viscosity term is a strong approximation which describes a different behaviour, if compared with real fluids. Here is where Xfoil (and all its related codes, as Xflr5) gains its merit.

The main effect of viscosity is to create a thin boundary layer on all surfaces (lifting and not), so that it is necessary to confront with a 3D boundary layer equation. This can be done in different ways, but in these pages, the focus will be on xflr5 method.

The Xflr5 job is divided in two main steps:

1. Xfoil's direct analysis combines theoretical and empiric methods of turbulence and transition, producing its results. It uses integral equations to solve the boundary layer.
2. Xflr5 interpolates the results obtained from the direct analysis and reintroduces them in a 3D inviscid model.

It can been already seen that this is an approximated solution, but it suites the objective of the thesis.

3D PROBLEM The models usually used to solve Laplace equation are:

- Lifting Line Theory
- Vortex Lattice Method
- Panel Method

and they are all implemented in Xflr5

1 The source is placed in the centre of the reference system

3D	Elementary solutions	Boundary Condition
Lifting Line Theory (LLT)	Vortices	Neumann
Vortex Lattice Method (VLM)	Vortices	Neumann
Panel	Source/sink and doublet sheets	Dirichlet and/or Neumann

Figure 2.4: 3D Methods

NOTES ABOUT PANEL METHODS [Coro9] [Caco6] Panel Methods is the name of some recent techniques able to solve the Laplace equation of a generic body. Mathematical formulation is made on the Green identity, while numeric formulation uses to discretize the whole geometry with simple mesh element.
Without getting too deep, it is possible to write down the potential of a generic point, not belonging to body's contour and identified by $\vec{r}_0$, by:

$$\phi(\vec{r}_0) = \frac{1}{4\pi}\left(\int_{\partial V} \phi \frac{\partial(\frac{1}{\vec{r}})}{\partial\vec{n}} dS - \int_{\partial V} \frac{1}{\vec{r}}\frac{\partial\phi}{\partial\vec{n}} dS\right)$$

According to what is reported in table 2.4, the first term of the previous formulation represents a doublet (intensity: ϕ) potential, while the second one is a source (intensity: $\frac{\partial\phi}{\partial\vec{n}}$) potential.
Considering a lifting body, an other contribute must be taken into account: the wake. It is modelled as a doublet distribution equivalent to a vortex layer.
The main purpose of the problem is to evaluate the intensity of the unknown aerodynamic singularities, in order to use these results to evaluate velocities and forces in every point of the system.
However, the numeric formulation consists in solving the problem in some specific control points, which are the "panels", i.e. mesh elements. Every panel is generally described by a polynomial. The great is the desired accuracy, the great should be the polynomial order. By the way, a first order polynomial is the most common choice. Every panel has a centroid, which is the control point itself, but is identified by its vertexes, named "grid points".
Xflr5 uses flat panels and applies the singularities according to the geometric mesh:

- Sinks and sources densities are uniform on every panel;
- Every doublet sheet is equivalent to a vortex ring.

No matter what is the method, the next step consists in solving the linear system:

$$\left[a_{ij}\right]\begin{bmatrix}\mu_0\\ \mu_1\\ \dots\\ \mu_{N-1}\end{bmatrix} = \begin{bmatrix}RHS_0\\ RHS_1\\ \dots RHS_{N-1}\end{bmatrix} \tag{2.4.2}$$

where a_{ij} coefficients represent the potential influence of a panel on an other.
Last thing to do is to calculate lift and drag starting from the density of the vortex singularities. Some years ago it was commonly used to sum forces acting on each panel. However, tests showed that this is a not so precise method. It's preferred to determine lift and drag on the Trefftz far field plane. Surfaces moments are evaluated with pressure forces on the wing[2].

2D PROBLEM Once it has been done, all that remains is to reintroduce viscosity, in order to consider something closer to a real problem.
Xfoil does this operation solving Laplace equation of inviscid field all around the profile in a first moment.
This because the viscous analysis needs the inviscid solution of this problem to continue. Note that the presence of a laminar or quasi-laminar regime is something that make sense only considering viscosity in the study.
The viscous solution shows that there are some favorable pressure gradients in those zones where there is a higher density of fluid fillets, as on the leading edge, where the flow accelerates from zero to its maximum value (then it decelerates going towards the trailing edge, where there is an adverse pressure gradient).

LOW REYNOLDS NUMBERS For low Reynolds numbers ($\leqslant 5 \cdot 10^5$) it can be said that the flow meet a profile in the stagnation point, then it splits on the two surfaces, upper and lower, in laminar regime. Nevertheless, sooner or later, surface velocity switches from a zero value to a negative one. That is what is called "separation bubble". It's nothing but a region where the flow separates from the surface and becomes progressively turbulent, and then, eventually, it joins again the surface, with positive velocity.
Before the end of the path, other bubbles can show up and, anyhow, a

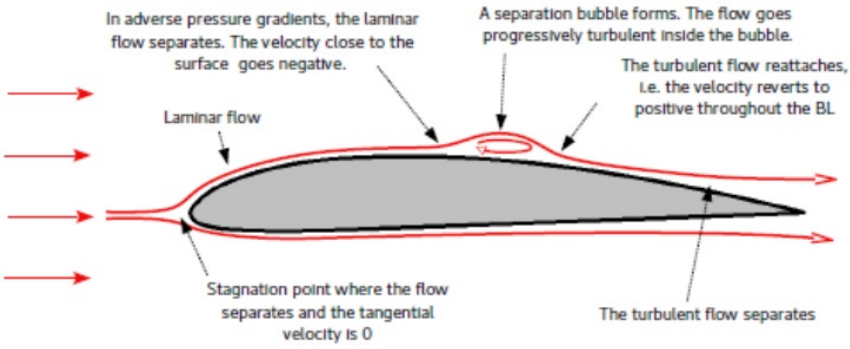


Figure 2.5: Separation bubble scheme

separation occurs at the trailing edge, where it forms the wake, but with a positive value of velocity because of the absence of the no-slip condition[3].
Xfoil analyses show the boundary layer limit speed, from the leading edge to the trailing one. Transition from laminar to turbulent flow is a very complex problem in 2D, and even more in 3D. In 2D analyses it is important to note that:

2 Velocities obtained from the solved potential field are used within Bernoulli equation to evaluate pressure distribution
3 Flow zero velocity condition at the contact point with the surface.

- Transition happens when a space wave amplification factor reaches a critic value, i.e. the ”Ncrit” factor;
- Turbulent flow starts with small sparks that can extends along the profile till a complete turbulence condition.

For low Reynolds numbers, the transition is induced by the separation, and that is what Xfoil tries to predict, evaluating what happens inside the bubble.

The 2D problem is solved in two steps:

- Numerically solving the inviscid potential problem (Laplace equation for velocity field);
- Using the solution as an input for the boundary layer problem.

But this is not the end. Until now, the problem didn't face the ”Goldstein singularity”:
the boundary layer significantly disturb the inviscid flow, because the profile acts as it has a local additional thickness. In other words, it happens that the inviscid flow tries to ”handle” the boundary layer, but the boundary layer doesn't reply correctly, and viceversa.
This additional thickness is called ”displacement thickness, δ^*”.
To solve the issue it is necessary to introduce a constant updating of the boundary layer condition in the process, to inform every time the inviscid flow about δ^*. This iterative form of calculus is called ”Interactive Boundary Layer-IBL”, and it can be done in different ways:

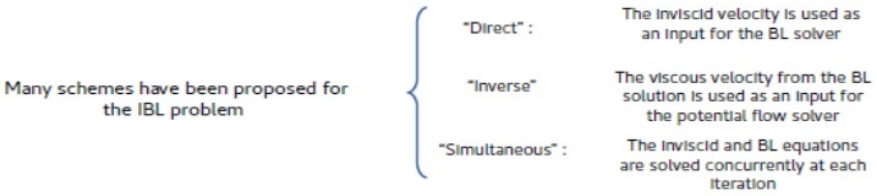


Figure 2.6: IBL strategies

”Simultaneus” scheme is the one developed by Mark Drela and H.Youngren in 90's with Xfoil.

2.4.2 Jupyter and Python related tools

Project Jupyter [Jup] exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages. For the present work it has been used the Jupyter Notebook IDE. The Jupyter Notebook is an open-source web application that allows to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modelling, data visualization, machine learning, and much more. All of the project has been performed according to the skills acquired from "Python Crash Course for Data Science and Machine Learning" by Pierian Data [Jos].

2.4.3 Limitations

Aerodynamics (Xflr5) [Web21]

- Viscosity:
  - Lack of interaction loop between boundary layer and inviscid flow for VLM and 3D Panel method;
  - Estimation of viscous drag;
- Panel Modelling:
  - Flat quad panels;
  - Uniform strength singularities;
  - Incomplete modelling of body-wing interactions.

However, limiting the analysis to the evaluation of the hinge moment, all of this does not affect the problem so much, or at least, numerical errors are in agreement with the abstraction level of the project phase.

Xflr5 has been used from lots of users to analyze different problems. During these tests, the software resulted in reasonable behaviour, in terms of C_p and C_l evaluation, if compared with a more complex CFD software [Cas19], [Ami08].
Two main issues have been observed:

- The analysis fails around stall configurations;
- Drag is underestimated (probably in its viscous term) when the software interpolates the 2D polars.

During the present work, the aforementioned stall issue has been experimented and caused the loss of some data, and it has been decided not to push the analysis to its limits (i.e. $\alpha > 10°$), in order to get reliable results. The second one is something which can be neglected, due to the very small contribution of the viscous drag to the hinge moment evaluation, because of the short force arm with respect to the hinge.

General

- Computational cost (mesh quality);
- Time cost (number of analyses).

2.4.4 Assumptions

- Flight altitude: maximum value allowed during "Air Cargo Challenge 2017", i.e. 100 [m];
- All wing analysis: Body and tail are neglected, according to the abstraction level of the present work;

3 AERODYNAMIC MODEL

3.1 Introduction

In order to generate a great number of configurations, model precision had to meet computational time requirements. Different kind of strategies have been taken into account in order to gain hinge moment data, but at the end it was decided to use Xflr5, the same software used by ICARUS for preliminary evaluations on Anubi. The trade off was about:

1. model precision;
2. computational time required;
3. software ability to handle batches of data;
4. process automation capabilities.

The best choice would have been to get real data, then to conduct experiments. Even a CFD simulation would have been a good compromise. It has been decided to adopt Xflr5 because it met three trade off elements out of four, at the expense of a slightly less precise analysis. However, the objective is about creating a complete procedure, to be improved in further analyses.

3.2 2D Analysis

For the direct foil analysis, it was necessary first of all to identify the profile type. For the wing, ICARUS chooses the Selig Donovan 7037, whose coordinates can be obtained online. After having adjusted the coordinates file format, it has been imported in Xflr5, obtaining the visualization in fig. 3.1 for a 0 flap configuration.

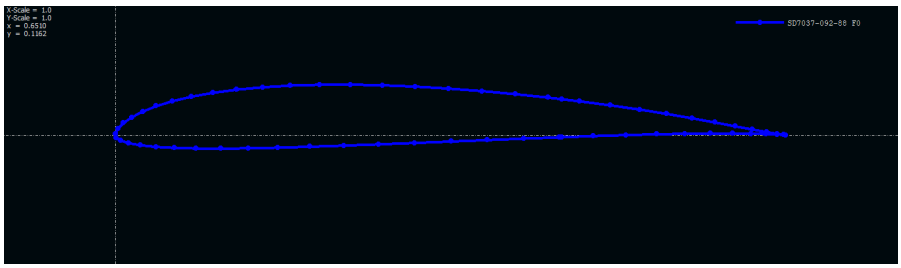
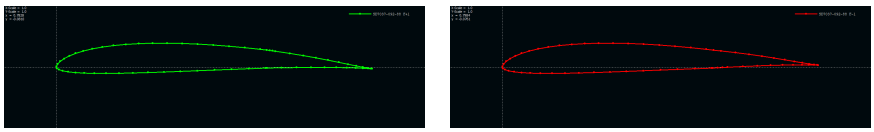


Figure 3.1: $\delta = 0°$

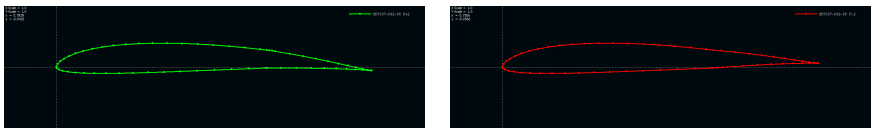
During this phase, Xflr5 allows to modify the profile according to what is possible to do in Xfoil too. So the user is able to scale camber and thickness, to edit coordinates, to normalize and refine the profile and even to to set flaps, trailing edge gap and leading edge radius.
All of the 9 flap configurations could have been created in a single window,

but it was decided to realize different models using 9 different ".xfl" files.
Once the profile is ready, the program needs to calculate a family of polars



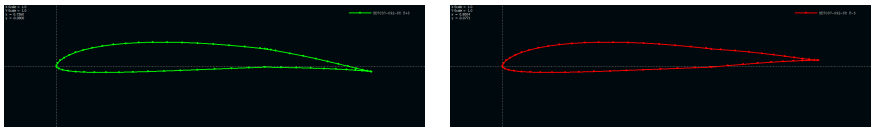
(a) $\delta = 1$ (b) $\delta = -1$

Figure 3.2: $\delta = \pm 1°$



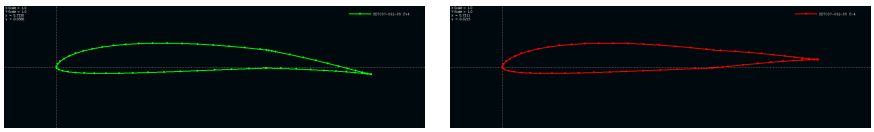
(a) $\delta = 2$ (b) $\delta = -2$

Figure 3.3: $\delta = \pm 2°$



(a) $\delta = 3$ (b) $\delta = -3$

Figure 3.4: $\delta = \pm 3°$



(a) $\delta = 4$ (b) $\delta = -4$

Figure 3.5: $\delta = \pm 4°$

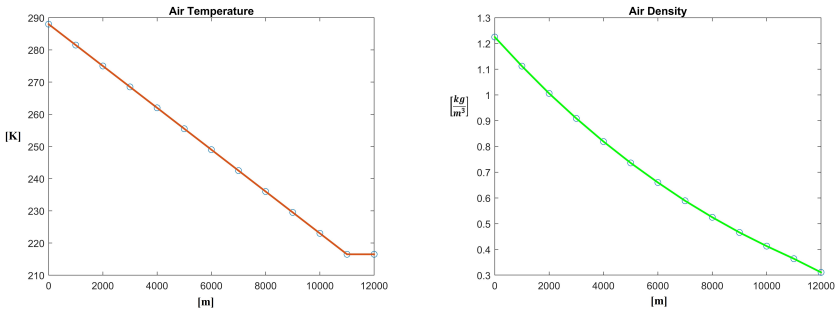
to be interpolated in order to estimate the viscous contribute. This family of polars can be obtained considering the following elements:

- Flight Reynolds Conditions;
- Variation range of the angle of attack.

The second parameter is an input of the analysis, as it can be seen in section 2.3.2. On the other hand, flight Reynolds conditions have to be evaluated.

One of the hypothesis is to assume an altitude of 100 [m] (sec. 2.4.4), that is to consider air temperature and density values not so different from the standard ones. By the way, interpolating some online reference data ([Met]), figures 3.6a and 3.6b have been obtained, accompanied by the values:

- $T = 287,35$ [K]
- $\rho = 1.2137$ [kg/m^3]



(a) Temperature vs Altitude

(b) Density vs Altitude

Figure 3.6: Atmospheric parameters interpolations

Now, Reynolds number is a measure of the ratio of inertia forces to viscous forces: the greater the speed, the lower the impact of viscous forces. It can be evaluated as follows:

$$Re = \frac{u * c}{\nu} \tag{3.2.1}$$

where:

- u: local speed value;
- c: local chord length;
- ν: kinematic viscosity.

ν is in turn a function of dynamic viscosity μ and density ρ:

$$\nu = \frac{\mu}{\rho} \tag{3.2.2}$$

so, knowing speed and chord length, all that remains is to evaluate the dynamic viscosity and insert the result in the equation system. For this purpose, it can be used Sutherland equation:

$$\mu = S \cdot \frac{T^{3/2}}{T + \chi} \tag{3.2.3}$$

where:

- $S = 1.46 \cdot 10^{-6} \rightarrow$ Air Sutherland constant $\left[\frac{kg}{m \cdot s \cdot K^{0.5}}\right]$
- $\chi = 110 \rightarrow$ Air reference temperature [K]

Note that the purpose is to evaluate a range of Reynolds values which can be used to describe viscosity conditions during all the flight. It can be done observing that, at fixed altitude (and so ν), a maximum and a minimum value for equation 3.2.1 are linked to the maximum and minimum values of c and u:

- $u = [12\,,\,22][m]$
- $c = [0.15\,,\,0.27][m/s]$

Substituting, it can be obtained:

$$Re = [1.2206\,,\,4.0281] \cdot 10^5$$

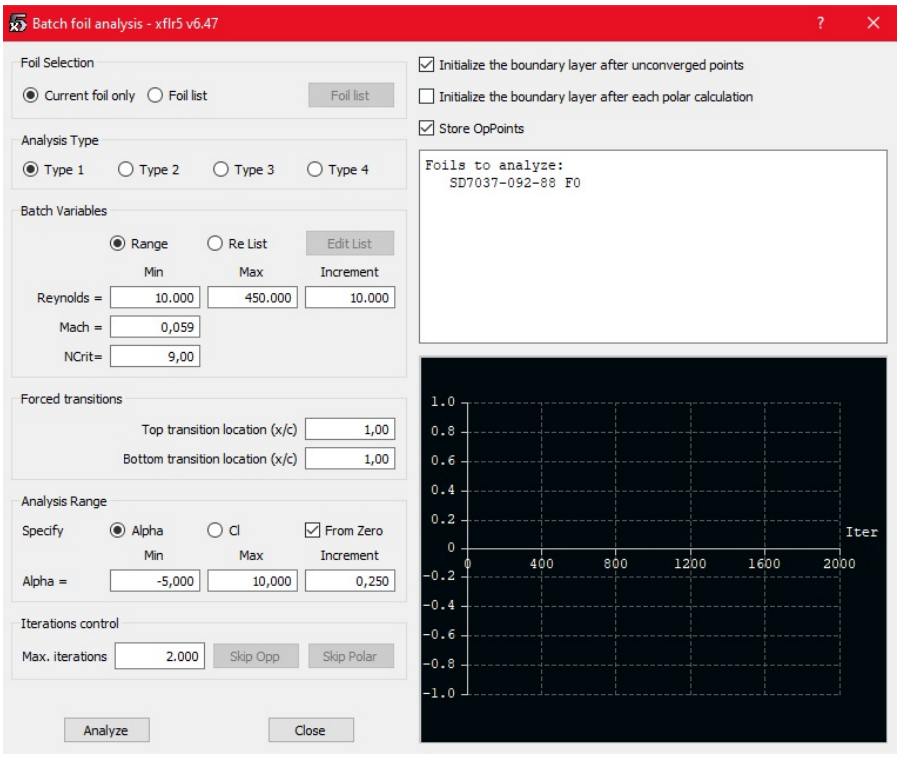


Figure 3.7: Batch analysis interface

All the polars in this range are eligible for Xfoil interpolation.

Figure 3.7 shows Xflr5 batch analysis interface. As it can be seen there are two ways to pass Reynolds number values to the software: from a list and from a range. In both cases it is necessary to pass at least one Mach value. The difference between these data entry methods is that the list type allows to proceed with a variable Mach analysis, which means a different speed condition. As regard range data entry, just one value of Mach number is needed. The choice between this two criteria is strictly subjective: the program will always evaluate a local Mach value as a function of a fraction of unit length chord (true chord value will be considered during 3D modelling) and will use it to perform its calculations. Usually, for these analyses the standard Mach value is zero for every Re, because under a value of $M = 0.3 - 0.4$ compressibility effects can be neglected. By the way, for the present study, sound speed can be calculated as follows:

$$a = \sqrt{\gamma \cdot \frac{R}{\mathcal{M}} T}$$

With:

- $R = 8314.4626 \left[\frac{J}{kg \cdot K}\right] \rightarrow$ Gas universal constant
- $\mathcal{M} = 28.96 \left[\frac{g}{mol}\right] \rightarrow$ Air molar mass

So it results:

$$M = \frac{u}{a} = [0.0353, 0.0647]$$

The mean Mach value had been chosen, in combination with a range data entry for Reynolds number (with an overestimated value in both directions,

to be sure to cover all the flight situations).
Before starting the analysis, it has been considered a proper range of incidences, with a variation step small enough to guarantee a dense family of polars (fig. 3.8). These polars will be interpolated to estimate the viscosity effects for the 3D wing, as said in section 2.4.1.

Note, from the top left graph, that there's a sort of dependency between

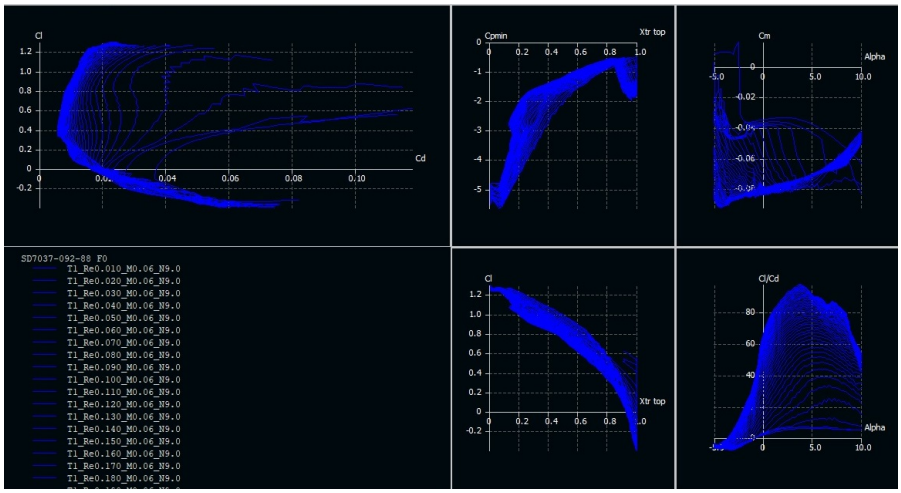


Figure 3.8: Polars for $\delta = 0°$

drag coefficient C_D and Reynolds number. Assuming C_D as the sum of two contributes:

$$C_D = C_{D_i} + C_{D_v}$$

Where:

- $C_{D_i} \rightarrow$ Induced drag coefficient. Speed independent;
- $C_{D_v} \rightarrow$ Viscous (or Profile) drag coefficient. Speed dependent.

It can be said that in 2D analyses, the induced contribution does not exist, so $C_D = C_{D_v}$. In this analysis, drag coefficient depends on speed, so on Reynolds number.
Moreover, remembering the meaning of Reynolds number, figure 3.8 shows that in high Re zones, C_D is low because inertial forces are stronger than viscous ones. When Re falls, C_D raises.

3.3 3D Model

Xflr5 provides a smart 3D interface that allows to model in terms of CAD and mesh a lot of plane architectures. Small aircrafts like ANUBI can be virtually built knowing geometric characteristics and disposition of masses and one of the four available type of analyses can be run knowing a few data about flight conditions.
At the beginning of the study, a full ANUBI configuration was realized, as it can be seen in fig. 3.9.

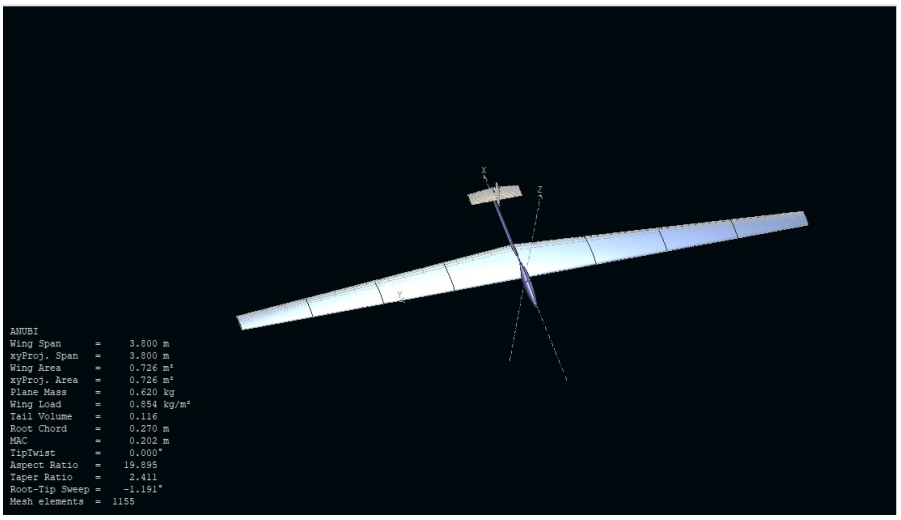


Figure 3.9: ANUBI 3D model with Xflr5

However, it was decided to dispose this strategy for many reasons:

- First of all, the target is to evaluate the hinge moment on a particular portion of the half wing;
- Tail geometry is still in updating and data are difficult to find;
- It was demonstrated that body interaction with lifting surfaces represent a noise, more than a real information [Web]. Morover, 3D Panel analysis usually results in errors since it needs closed and non intersecting volumes: this hits limitations of Xflr5, which has some non-intersection issues between volumes (wing-body, tail-rudder, tail-body).

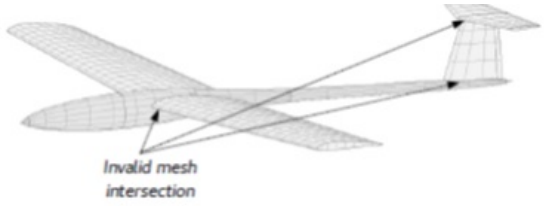


Figure 3.10: 3D Panels invalid mesh visualization example [Web21]

At the end, an all-wing model has been considered enough for the project objective.

From the "Wing Edition" panel of the "Wing and Plane design section" of Xflr5 it is possible to model a wing plane by inserting its dimensions in terms of local chords, and wingspan. Note that inputs must be related to the half wing. The program will automatically generate a specular portion.

It has been decided to divide the half wing in three portions with four profile sections. Sections are useful to generate a right mesh and allows to use different profile configurations for each wing portion. Each section has been placed in a particular position along wingspan, to recreate the partitioning previously shown in figure 2.1. Knowing these wingspan values, it has been easy to interpolate local chords values via Matlab, considering a linear taper.
Last things to do, assuming no dihedral and no twist for the wing, were to set the offset parameters to each profile and to assign the foil previously analysed in Xfoil section. All of the operations have generated the sizing

	y (m)	ord (m	fset (m	edral(	wist(°	foil	-panel	X-dist	-panel	Y-dist
1	0,000	0,270	0,000	0,0	0,00	SD7037-092-88 F0	11	Cosine	4	-Sine
2	0,409	0,244	0,009	0,0	0,00	SD7037-092-88 F0	11	Cosine	11	-Sine
3	1,404	0,181	0,028	0,0	0,00	SD7037-092-88 F0	11	Cosine	5	-Sine
4	1,900	0,150	0,037		0,00	SD7037-092-88 F0				

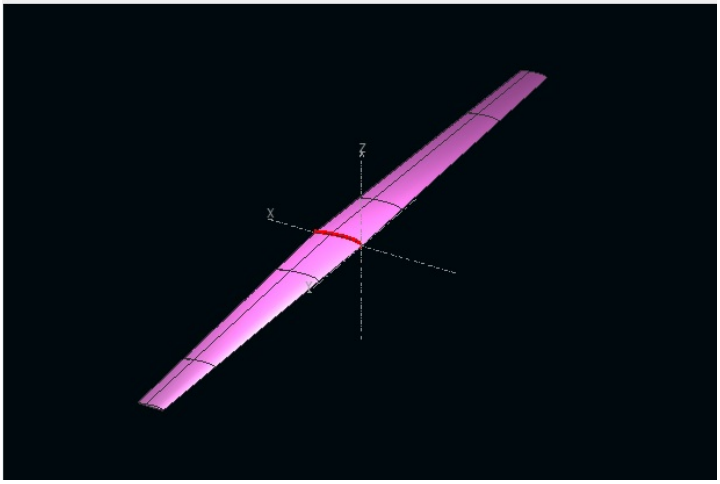


Figure 3.11: Wing modelling

reported in tab. 3.1.
"Wing Edition" panel is also the place where the mesh has to be prepared. Xflr5 usually suggests a superficial mesh it finds reasonable to connect elements from a wing portion to the adjacent one. For this analysis it has been chosen the suggested mesh, whose characteristics are reported in tab. 3.2.

MESH STRATEGY Xflr5 suggests a mesh it finds reasonable. By the way it has been considered to study some different path. For example it has been observed the effect on the hinge moment adopting twice as many suggested panels. Resulting comparison can be seen in figure 3.12.
As shown, the double size[1] mesh produced greater values for hinge moment. Generally speaking, the larger the number of panels, the better the

1 in terms of "number of panels"

Parameter	Value	Misure
Wing Span	3.80	[m]
Wing Area	0.8	[m^2]
Mean Geometry Chord	0.21	[m]
Mean Aero Chord	0.22	[m]
Aspect Ratio	18.11	
Taper Ratio	1.8	

Table 3.1: Wing Characteristics

y [m]	**x-panels**	**x-dist**	**y-panels**	**y dist**
0	11	Cosine	4	Sine
0.409	11	Cosine	11	Sine
1.404	11	Cosine	5	Sine
1.9				

Table 3.2: Mesh Characteristics

analysis accuracy. In any case, real experiments should be conducted to assess accuracy and this is something that transcends the objectives of the thesis. Moreover, the maximum difference between the two mesh hinge output is 15% with respect to the suggested mesh (blue line), and it decreases with α reaching the value of 3% as minimum difference.

Once the plane is ready, Xflr5 wants the user to define the analysis. From "Define an Analysis" panel it can be chosen one of four analysis type.

As regard the project, final database needs pressures and hinge moments informations in terms of a velocity v input[2], so "type 1" analysis (Fixed speed) has been chosen. Same panel, but in "Analysis" it can be selected the solver method, referring to one of the following:

- Lifting Line Theory;
- VLM1 - Horseshoe Vortex;
- VLM2 - Ring Vortex;
- 3D Panel.

VLM and 3D Panel were both good for the analysis, but it was chosen the second, because it allows to obtain pressure distribution on both top and bottom surface (VLM method plots pressure distribution just on the camber line).
Inertia and air temperature and density have been set with the values early

2 Control surface deflection δ has been fixed during foil design and angle of attack will be defined later

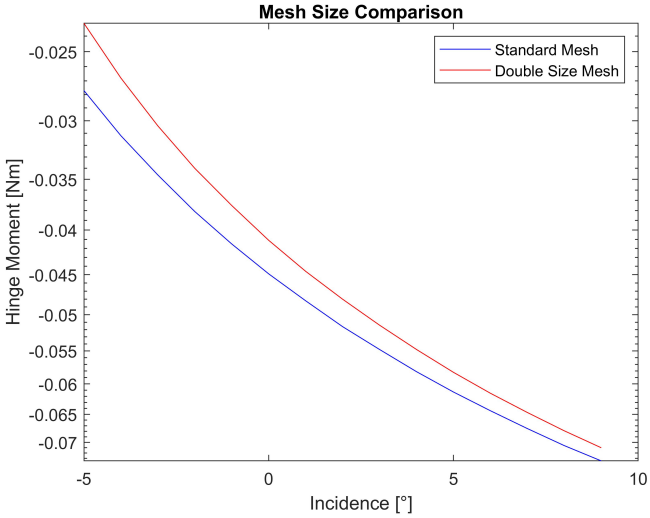


Figure 3.12: Hinge moment difference

calculated (pag. 18). Note that Xflr5, to make up for the underestimation of drag, allows to add different source of extra drag. But this function has been neglected for the present study.
Last thing to do is to define the range of incidence to perform the analysis within. Range and variation step have to be the same chosen during DLN input definition (tab. 2.2).

Starting the analysis, Xflr5 solves Laplace equation with the chosen method (3D panel), then tries to interpolate Xfoil viscous results with the one obtained in the 3D analysis.
Lots of errors can be obtained in this phase, because of the limitations of Xflr5, causing dangerous loss of precious data. By the way, it has been possible to avoid some of this errors because of the moderate flight conditions in terms of speed, incidence and control surface deflection.

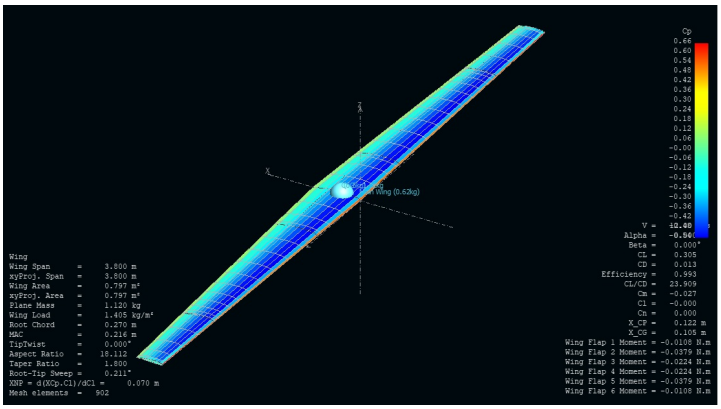


Figure 3.13: C_P contour map for $\delta = 0[deg]$,$v = 12\,[m/s]$,$\alpha = 0[deg]$

For the end of the analysis, Xflr5 features lots of visualization options, going from C_P contour maps to pressure vectors display and others (Lift, Induced drag, Downwash speed...). For each operation point has been possible to

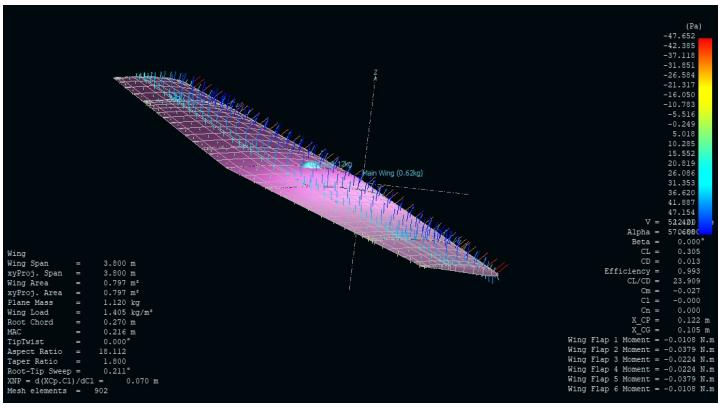


Figure 3.14: Pressure vectors distribution for $\delta = 0[deg]$,$v = 12\,[m/s]$,$\alpha = 0[deg]$

extract a .tex file, containing all of the calculations, to be used in the next project phase.

3.4 Output file

Xflr5 works considers "Operating Points". After wing analysis, which uses the Xfoil direct analysis results too, an OP si created.
"An operating point of a given foil is defined by its angle of attack and its Re number. Always associated to a foil and to a Polar object, the OpPoint stores the inviscid and viscous results of the analysis.
Any number of OpPoints may be stored in the runtime database, the only limitation being computer memory" (Xflr5 Guidelines [Uns09]). OpPoints may use significant memory resources, but this was not the case (after all Xflr5 was chosen to minimize computational requirements and to produce a lot of simulations).

For every OP, a .txt file can be extracted. It contains every information about calculated values. Figure shows the heading of one of these files. As shown, almost all required data for the DLN model can be acquired from the heading of the Xflr5 output file. Actually, the program is able to evaluate a flap moment for every wing partition considering the hinge axis position given as input during foil designing. It stands clearly up to the user to create a proper partitioning on the wing, and for the present project it has been decided to realize a "ready to use" sizing even before discover the flap moment feature.
In any case, the output file also provides the pressure coefficients all over the mesh, in a way that can be seen in figure 3.16. This could be useful to evaluate hinge moment with a different solver, as a FEM software.
C_p values are displayed as a function of the x, y, z coordinates of the plane, accompanied by an "Area" value of the single mesh panel. As said, they can be used to evaluate pressure forces considering that:

$$P = q \cdot S \cdot C_p$$

Where:

- $S = Area \rightarrow$ Panel Surface;

```
xflr5 v6.47

Plane Name
T1-12.0 m/s-Panel                    <-- Analysis Type
QInf  =   12.000000 m/s              <-- Speed
Alpha =    0.000000                  <-- Incidence
Beta  =    0.000°
Phi   =    0.000°
Ctrl  =    0.000
CL    =    0.308959
Cy    =   -0.000000
Cd    =    0.012743     ICd   =    0.001685     PCd   =    0.011058
Cl    = 3.98803e-16
Cm    =  -0.0287203
ICn   =    0.000000     PCn   =    0.000000
XCP   =    0.124470     YCP   =   -0.000000     ZCP   =    0.013828
XNP   =    0.000000
Bending =    8.247676

Main WingFlap    1 moment =   -0.0128 N.m
Flap     2 moment =   -0.0449 N.m
Flap     3 moment =   -0.0225 N.m
Flap     4 moment =   -0.0225 N.m
Flap     5 moment =   -0.0449 N.m
Flap     6 moment =   -0.0128 N.m
```

Figure 3.15: .txt output sample for $\delta = 0[\mathrm{deg}]$,$v = 12\,[\mathrm{m/s}]$,$\alpha = 0[\mathrm{deg}]$

- $q = \frac{1}{2}\rho v^2 \rightarrow$ Dynamic Pressure.

Also reference system coordinates of each centre of pressure could be used for a different purpose: they are eligible to generate a map to import pressures on a different solver. After having explored the output file, the analysis proceed towards data handling and process automation to operate with lots of files.

Main Wing Cp Coefficients

Panel	CtrlPt.x	CtrlPt.y	CtrlPt.z	Nx	Ny	Nz	Area	Cp
Main WingCp Coefficients								
Strip 1								
11	1.807e-01	-1.850e+00	1.683e-04	-0.020	0.001	-1.000	1.688e-03	0.1663
12	1.637e-01	-1.850e+00	1.892e-04	0.017	-0.001	-1.000	1.688e-03	0.1208
13	1.466e-01	-1.850e+00	-2.333e-04	0.032	-0.001	-0.999	1.688e-03	0.0792
14	1.318e-01	-1.850e+00	-7.517e-04	0.038	-0.001	-0.999	1.266e-03	0.0505
15	1.190e-01	-1.850e+00	-1.247e-03	0.039	-0.001	-0.999	1.266e-03	0.0266
16	1.063e-01	-1.850e+00	-1.748e-03	0.039	-0.001	-0.999	1.266e-03	-0.0009
17	9.350e-02	-1.850e+00	-2.231e-03	0.037	-0.001	-0.999	1.266e-03	-0.0315
18	8.075e-02	-1.850e+00	-2.664e-03	0.031	-0.001	-1.000	1.266e-03	-0.0712
19	6.799e-02	-1.850e+00	-2.980e-03	0.018	-0.001	-1.000	1.265e-03	-0.1313
20	5.524e-02	-1.850e+00	-3.007e-03	-0.014	-0.001	-1.000	1.265e-03	-0.2331
21	4.248e-02	-1.850e+00	-1.318e-03	-0.243	-0.004	-0.970	1.304e-03	-0.2688
22	4.248e-02	-1.850e+00	3.952e-03	-0.499	-0.009	0.867	1.460e-03	0.1998
23	5.524e-02	-1.850e+00	8.833e-03	-0.187	-0.006	0.982	1.288e-03	-0.4555
24	6.799e-02	-1.850e+00	1.060e-02	-0.087	-0.005	0.996	1.270e-03	-0.4596
25	8.075e-02	-1.850e+00	1.132e-02	-0.025	-0.005	1.000	1.266e-03	-0.4248
26	9.350e-02	-1.850e+00	1.136e-02	0.019	-0.005	1.000	1.265e-03	-0.3809
27	1.063e-01	-1.850e+00	1.089e-02	0.053	-0.005	0.999	1.267e-03	-0.3375
28	1.190e-01	-1.850e+00	1.003e-02	0.082	-0.005	0.997	1.270e-03	-0.2930
29	1.318e-01	-1.850e+00	8.823e-03	0.106	-0.006	0.994	1.272e-03	-0.2451
30	1.466e-01	-1.850e+00	7.019e-03	0.131	-0.006	0.991	1.702e-03	-0.1856
31	1.637e-01	-1.850e+00	4.449e-03	0.167	-0.007	0.986	1.712e-03	-0.0845
32	1.807e-01	-1.850e+00	1.503e-03	0.174	-0.008	0.985	1.714e-03	0.0308

Figure 3.16: Pressure coefficients for $\delta = 0[\mathrm{deg}]$,$v = 12\,[\mathrm{m/s}]$,$\alpha = 0[\mathrm{deg}]$

4 DATA HANDLING AND AUTOMATION

4.1 Automation Strategy

Since Xflr5 has a closed interface, it was not possible to automatize the generation of all output files. Moreover, each analysis had to be supervised in order not to operate with wrong data. So, the first part of the project has been to realize all of the DLN examples manually.
On the other side, for data handling phase it has been possible to a single Matlab code able to select each Xflr5 output file and explore it taking out desired data.

4.1.1 String updating and file location

STRING UPDATING This code is a triple "for" loop that passes through each value of δ, v and α vectors to update the different strings' key words contained in the name of each file. This is necessary to identify the target output file.
All that follows could have been possible because of the way Xflr5 names its output, using the characteristics of the operating point. For example, the string "MainWing_a = 0.00_v = 12.00ms" refers to a particular simulation. It can be seen that there are some indication (a = α, v) which can be update every time in a "for" loop to identify different simulations. The only not showed parameter is δ, because it doesn't identify an OP point itself, but just a profile edition, so Xflr5 doesn't use it to classify its results.

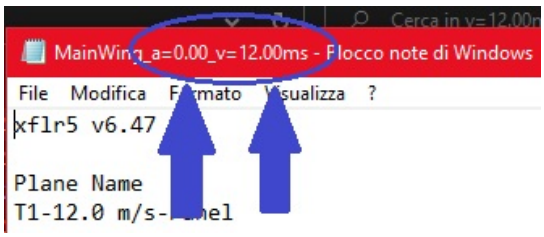


Figure 4.1: File string name

The problem can be solved easily including Xflr5 outputs in different folders, depending on the δ value used for the analysis itself.
At the and, available information are enough to build an updating algorithm using a triple loop a the three vectors of the main variables.

FILE LOCATION Since Xflr5 outputs must be included in proper folders (related to δ), every iteration of the previous loop has to locate each file by string comparison and move it in the work folder, in order to operate on it. To generalize the process as possible, a Matlab "isfile" function has been used to check if the file is present in the specified folder. This allows to add

or remove simulation outputs, to make the code eligible for future analyses, just modifying main inputs, as the $\delta/\nu/\alpha$ range (and obviously producing aforementioned simulations with Xflr5). Basically, the "location" part of the code is about generating updated strings which identify a particular system path. At the end of the external loop, the same path string is used to bring back the input file to its initial position.

4.1.2 Data extraction

All the process has the main purpose to extract a set of values that identifies a complete DLN example. This means that the code must provides four values per iteration (full 3 loops):

- $\delta \rightarrow$ Taken from external loop present index;
- $\nu \rightarrow$ Taken from middle loop present index;
- $\alpha \rightarrow$ Taken from inner loop present index.
- $H \rightarrow$ To be extracted from .txt file.

The problem is just to extract hinge moment. The fastest way is to extract the "Flap Moment" of the right partition from the .txt file. This can be done creating a function able to "explore" the the .txt in some way. It was chosen to declare the exact position (in terms of rows and columns) of the desired flap moment value. That was possible thanks to Matlab "textscan" function. This function store its results in a cell array, and it's easy to take them out and evaluate them as needed.

Including before the loop an Excel file generation command it has been possible, at this point of the process, to add elements to this ".xls" file, having care not to overwrite previous data.

EXCEL FILE ADJUSTING In order to import the full dataset in DLN environment it has been necessary to convert the final database from ".xls" tabulated (fig. 4.2a) to ".csv" comma separated (fig. 4.2b). It has been possible thanks to Excel basic functions.

	A	B	C	D
1	delta	v	alpha	H
2	-4	12	-4	-0.0035
3	-4	12	-3	-0.0065
4	-4	12	-2	-0.0094
5	-4	12	-1	-0.0124
6	-4	12	0	-0.0154
7	-4	12	1	-0.0183
8	-4	12	2	-0.0212
9	-4	12	3	-0.0242
10	-4	12	4	-0.0271
11	-4	12	5	-0.0299
12	-4	12	6	-0.0328
13	-4	12	7	-0.0356
14	-4	12	8	-0.0384
15	-4	12	9	-0.0411
16	-4	12.5	-4	-0.0038
17	-4	12.5	-3	-0.007
18	-4	12.5	-2	-0.0102
19	-4	12.5	-1	-0.0135
20	-4	12.5	0	-0.0167

(a) ".xls" t.

	A
1	delta,v,alpha,H
2	-4,12,-4,-0.0035
3	-4,12,-3,-0.0065
4	-4,12,-2,-0.0094
5	-4,12,-1,-0.0124
6	-4,12,0,-0.0154
7	-4,12,1,-0.0183
8	-4,12,2,-0.0212
9	-4,12,3,-0.0242
10	-4,12,4,-0.0271
11	-4,12,5,-0.0299
12	-4,12,6,-0.0328
13	-4,12,7,-0.0356
14	-4,12,8,-0.0384
15	-4,12,9,-0.0411
16	-4,12.5,-4,-0.0038
17	-4,12.5,-3,-0.007
18	-4,12.5,-2,-0.0102
19	-4,12.5,-1,-0.0135
20	-4,12.5,0,-0.0167

(b) ".csv" c.s.

Figure 4.2: Format conversion

4.2 GUI

In order to simplify future developments, a simple graphical user interface has been built. This GUI allows the user to start the data handling by simply entering some known data.

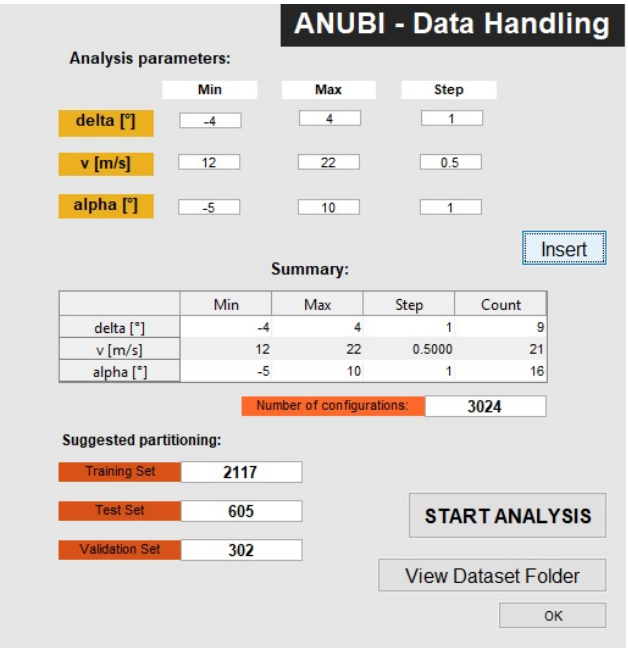


Figure 4.3: GUI

As shown in figure 4.3, one must only fill some fields with a description of the three analysis input ranges (minimum, maximum and step value). These values can be obtained from table 2.2,
Pressing "Insert" button, a summary of tables 2.2 and 2.3 will be displayed, then all that remains is to start the analysis. Pressing "Start Analysis", the background data handling code described so far will start and will look for Xflr5 output files considering user input. This means that after each epoch, the code updates the string to look for using defined ranges: every time it finds a string match inside a particular folder, it opens the file, reads and stores desired informations. It does not matter if, after a complete epoch, results no match: if a particular file does not exist, the code will simply skip to the next search. This allows not to worry about data loss during data handling.

5 DLN BACKGROUND

5.1 Supervised Learning

As previously said (sec. 2.3.2), DLN algorithm adopted in the present project needs a bunch of examples to be trained on. In the field of machine learning this paradigm is known as "supervised learning". In short, what happens is that the network receives a set of "labelled" inputs, i.e. the set of examples, then it trains on these inputs comparing each time its own output with the right one (the label), looking for errors and adjusting its own parameters to meet correct output requirements. This is exactly what is shown in fig 5.1.

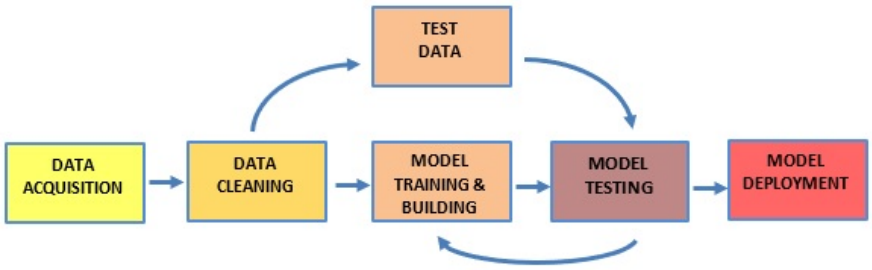


Figure 5.1: Machine learning flow chart

1. **Data acquisition**: first context analysis. Data origin (customers, information collections, sensors...);

2. **Data cleaning**: data should be "cleaned" (or elaborated), before the network is able to process them. A sort of cleaning is what has been done within Excel (page 30). In the present context, data cleaning is something more complex, like dealing with missing or non numeric values. This step has been avoided through Matlab, directly creating a ready to use database (chapter 4.2).

3. **Test Data and Model Training/Building**: this phase starts as an extension of the previous one: dataset still has to be managed, in terms of partition creation.
   At least two main dataset partition should be created:

   - Training data: usually 70% of whole dataset;
   - Test data: usually the remaining 30% of whole dataset.

   However, this is a simplified approach: to be sure that the model will work, it has to be tested on data it has never seen. By including just one test set, the algorithm will use it not only to determine the hyperparameters to be changed, but also to validate its results, and this procedure does not constitutes a strong limitation [Jos].

For the present project, a triple partition has been chosen by including a third set of about 150 examples to validate the model on brand new values. Note that in literature [Jos] the meaning of the words "validation set" and "test set" is exchanged with respect to what each of this nomenclature means in the present context. It was decided to adopt this terminology to better comprehend some python functions which call "test" the "validation" set.
Once the datase is partitioned, the first iteration proceeds building the model.

4. **Model testing**: the model built with the training data is ready to be tested so, by including test set input in the model, results can be compared with the true labels. If error is too large, the algorithm will step back and adjust hyperparameters.

5. **Model deployment**: assuming good results not only in the test set, but also in the validation set, the model can be considered ready to deploy. After this moment, it is not possible to get back and improve performances of the same model.

A complete model is described by accuracy indices that show its ability to work in real world situations. Those are the results of the application of some evaluation metrics to the last validation analyses.

5.2 Statistical performances evaluation

Performance is typically measured by four parameters scale:

- Accuracy;
- Recall;
- Precision;
- F1-Score.

A generic prediction will always result in one of two possible scenarios: correct or incorrect prediction[1].

- **Accuracy**

  It is the ratio between the number of correct predictions (CP) and the whole set of predictions (CP + IP):

  $$Accuracy = \frac{CP}{CP + IP} \tag{5.2.1}$$

  Accuracy, is a useful parameter for all kind of problems but those categorical problems[2] in which classes are not balanced.

- **Recall** and **Precision**

1 For sets containing continuous values set of values, a correct prediction can be assumed as a prediction whose error lies below a certain threshold

2 "In statistics, a categorical variable is a variable which can take one of a limited, and usually fixed, number of possible values, assigning each individual or other unit of observation to a particular group or nominal category on the basis of some qualitative property". [Wik21a]

Recall is the ability of the model to detect (or recall) all relevant cases within a dataset. According to this definition, recall formulation is:

$$Recall = \frac{TP}{TP + FN} \tag{5.2.2}$$

The ratio between the number of true positive cases (TP) and the sum of true positive and false negative cases (TP + FN). The meaning of these terms is presented in figure 5.1. Precision is the amount of data

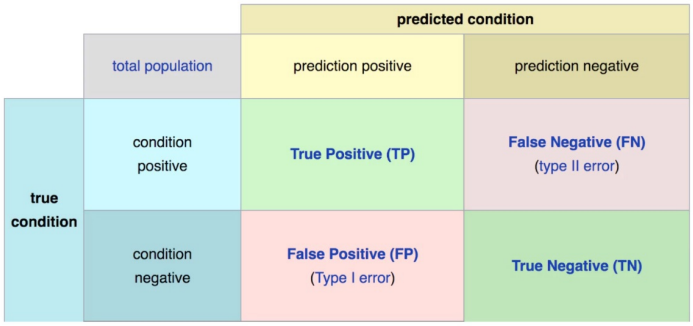


Figure 5.2: TP,FN,FP,TN meaning

which the model consider relevant, and which actually turns out to be such.

It is often established a trade off between Recall and Precision parameters, to get more consistent informations.

- **F1-Score**

  F1-Score is a method to get an optimal balance of recall and precision parameters. It consists in the harmonic mean of the two parameters:

$$F1 - Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{5.2.3}$$

However, these metrics were not meant to be used in regression problems, which is faced in the present context, and they could lead into bad evaluations. It has been preferred to switch towards a different set of metrics.

5.3 Error metrics for regression models

Facing a regression problem means trying to forecast continuous values having continuous inputs. These problems are different from categorical ones, like classification (where the results can be just ”A” or ”B”). For regression problems it is preferred to use following metrics:

- Mean Absolute Error;
- Mean Squared Error;
- Root-Mean Squared Error.

- **Mean Absolute Error - MAE**

  The mean of the errors absolute value:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_1 - \hat{y}_i| \tag{5.3.1}$$

  Unfortunately, this metric does not ”punish” great errors, as it can be seen in fig. 5.3.

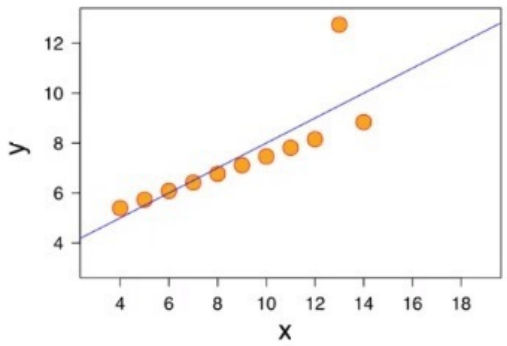


Figure 5.3: Low impact of the greatest error

- **Mean Squared Error - MSE**

  Born to overcome MAE limitations on great errors, the MSE is defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_1 - \hat{y}_i)^2 \tag{5.3.2}$$

  It is one of the most useful metric parameter but it has the issue to be (sometimes) hard to be interpreted because of the square power, that acts on values but also on measure units.

- **Root-Mean Squared Error - RMSE**

  As in the name:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_1 - \hat{y}_i)^2} \tag{5.3.3}$$

  RMSE in the most common metric parameter because it easily takes the merit of the MSE and overcome the interpretation problem by adopting the square root.

At the end, each metric has its limits. Error needs to be studied in dependency with the context: sometimes, even a small error cannot be neglected.

5.4 Deep Learning Neural Networks

The historical inspiration of ”deep learning” models is the will to build an artificial imitation of a biological, natural intelligence. As previously said,

this is not completely possible for now, but some sort of useful neuron-based algorithms can be realized nowadays. To understand the meaning of "neuron-based" algorithms, and so the code used in this project, it is necessary to clarify some theoretical aspects.

5.4.1 Perceptron Model

A human brain is characterized by neurons. A neuron has a central body, the nucleus, and some highly specialized ramifications. To adapt its functions to what is necessary for the analysis, neuron architecture can be simplified as shown (fig. 5.4). Figure reminds a sort of analogy with informatics:

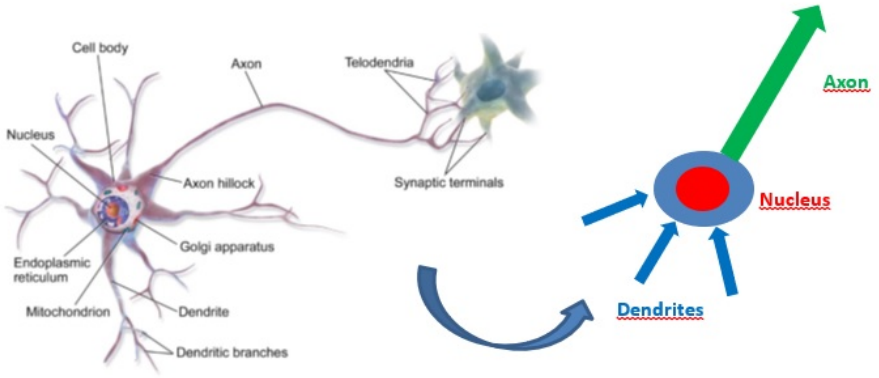


Figure 5.4: Neuron scheme

- Dendrites → Inputs;
- Nucleus → Generic couplig function;
- Axon → Output.

A sufficiently simple mathematical model based on a biological architecture is the **Perceptron**.

The perceptron is an elementary neural network form and it is depicted in figure 5.5. Characteristics:

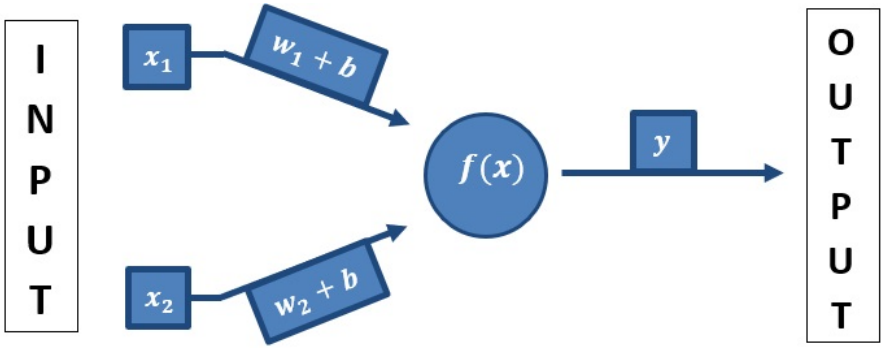


Figure 5.5: Perceptron Model

- $x_i \rightarrow$ Inputs;
- $w_i + b \rightarrow$ Hyperparameters term;
- $f(x) \rightarrow$ Coupling function;
- $y \rightarrow$ Output.

In other words, a perceptron is a structure in which a particular function associates a number of inputs in order to produce a single output. By neglecting the hyperparameters term, and considering a "sum" function, the behaviour of the perceptron in fig. 5.5 may be associated, in a first instance, to the formulation:

$$y = x_1 + x_2$$

But the real potential of a DLN can be expressed as the ability to "perceive" the input in a different and user-defined way. Here lies the meaning of the "**weight**" terms w_i:

$$y = x_1 w_1 + x_2 w_2$$

A particular situation may occur when one of the input is $x_i = 0$. In that case, the relative weight does not modify the interpretation of x_i. This is the reason why a further term is needed. This term is the "**bias**" b, which instructs the code over a simple statement: "the product $x_i w_i$ must be larger than b to generate some effect on the output":

$$y = (x_1 w_1 + b_1) + (x_2 w_2 + b_2)$$

Final formulation should consider the presence of more than two inputs, weights and biases:

$$\hat{y} = \sum_{i=1}^{n} (x_i w_i + b_i) \tag{5.4.1}$$

Note that the formulation above refers to just one neuron, but a network must be provided with a multitude of them for handling complex systems. Fortunately, single neurons can be coupled in order to produce a multilayer model, as shown in figure 5.6. Referring to previous picture:

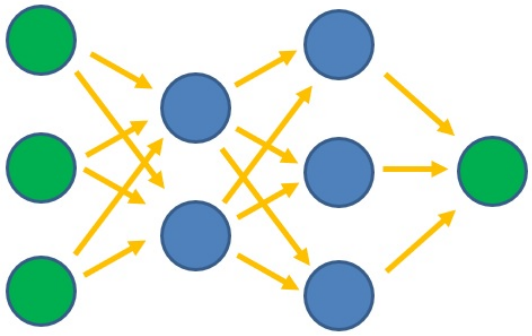


Figure 5.6: Multilayer network

- Characteristics:
  - Every neuron is connected to all of the ones belonging to the next layer ("Fully Connected System");

  - Each neuron has a single output value which is passed to multiple neurons.

  Basically, what allows the network to learn a complex task is the repeated composition of affine and simple non linear functions.

- Nomenclature:
  - Input layer: first input acceptor layer (left green layer);
  - Output layer: last output layer (right green layer). For some problems, this layer may host more than one neuron.
  - Hidden Layers: intermediate layers. This is the core of the model, where most of the calculation is processed. These levels are usually hardly interpreted during debug sessions because of information "distance" from both the input and the related label.

  Hidden levels are what distinguishes a Simple Neural Network from a Deep Neural Network: DLN means the presence of at least two hidden levels.

ACTIVATION FUNCTION The weighted sum is just a part of the whole core function. For instance, recalling formulation 5.4.1, consider to assign the hyperparameterized term to a generic variable z_i:

$$z_i = w_i x_i + b_i$$

One other step must be done to produce the input: there is the need to "interpret" z_i.
For this reason, the "**Activation Function - AF**" will be the specific $f(z_i)$ which applies some sort of interpretation to z_i. There are many of them in literature and each one is used for a particular DLN job, depending on the context.
As concerns the present project, it has been adopted the "**Rectified Linear Unit**" function, **ReLu**. This is the most used function since 2011 for regression problems and it is basically easy to understand, because it operates:

$$f(z_i) = max(0, z)$$

In other words, if the output is less than zero, it will be treated as zero.

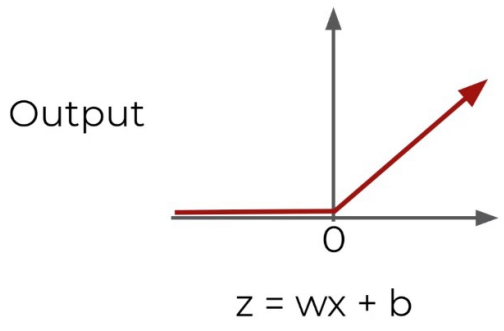


Figure 5.7: ReLu

If the output is higher, it will be treated as the maximum acquired value. "Rectifying activation functions were used to separate specific excitation and unspecific inhibition" (Wikipedia [Wik21c]).

[Bri20] The output of the activation function is used to build the "Cost Function" (next paragraph) and, as it will be seen, its importance lies in its derivative: during the backpropagation (paragraph 5.4.1) process the AF derivative is used and if it results in some near zero value, it will cause a gradual error vanishing. This vanishing is not a good news, because the learning process, without the guide represented by the error itself will slow down epoch after epoch. In order to obtain a proper output interpretation, in terms of backpropagation utility, a different kind of activation function should be used, depending on problem type. One example is the "step function", the first activation function adopted in the machine learning field. It was directly involved in the definition of "Perceptron[3]" itself. It applies:

$$f(z_i) = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases} \tag{5.4.2}$$

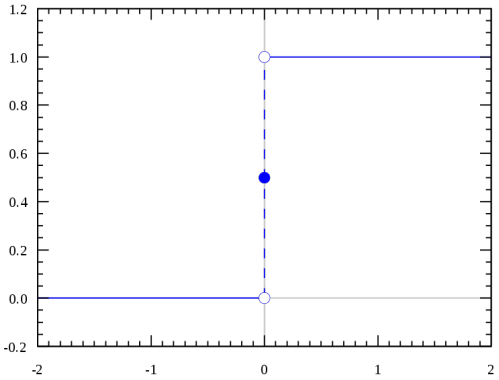


Figure 5.8: Step Function [Wik21d]

An other example is the "*sigmoid, or logistic function*" (fig. 5.9).

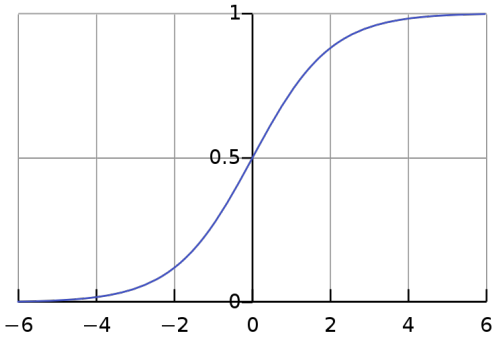


Figure 5.9: Sigmoid Function

$$f(z) = \frac{1}{1 + e^{-z}} \tag{5.4.3}$$

3 To be rigorous, all perceptron formulation but the one implementing the 5.4.2 should be referred as some sort of "modified" perceptron.

This was the most common AF before 2011 but it turns out to have a "small" derivative with a maximum value of 0.25, which soon tends to the zero value:

$$f'(z) = \frac{-e^{-z}}{(1+e^{-z})^2} \rightarrow \max|f'(z)| = \frac{1}{4}$$

This function can be observed in the picture (horizontal tangent $y = [01]$). For this reason the "ReLu" function is generally preferred. It has a derivative equal to 1 on the right side and to 0 on the left side. This means that, when the first network iteration will be started (generally with random hyperparameters) about half of the neurons will result in an AF output $f(z_i) \leqslant 0$, which means in turn that their derivative AF forms will be null and they can be considered "turned off" during cost evaluation. This operation is called "*Dropout*" and is a technique to avoid overfitting, reducing the number of cost evaluation to do during the most critical phase of training.

The ReLu function, summarizing, brings two beneficial effects:

- It helps to reduce overfitting;
- It is easy to implement, by an easy sign confront, without calculations:

if $(z <= 0)$
return 0;
else
return z;

The main ReLu limitation is the impossibility to be derived in $x = 0$ and this may yield some bad consequences during backpropagation. However, an in-depth study on the subject transcends the objectives of the thesis.

COST FUNCTIONS Each time the output layer produces its result, this has to be evaluated by comparison with true labels, and eventually adjusted. Adjusting the output means to step back and change hyperparameters. Note that in this phase, the algorithm is using the train dataset.
Performed comparison is tracked by a **Cost Function** which averages the results and produces a single output value i.e. the general "loss" of the process.
Considering:

- $y \rightarrow$ true value;
- $a \rightarrow$ predicted value;
- $z \rightarrow$ input definition;
- $\sigma \rightarrow$ activation function;

training iteration can be formulated as:

$$\sigma(z) = a$$

That is: input z is passed to the activation function σ and the result is assigned to the variable a.
The prediction in turn may be used in relation to y to evaluate a cost function. For instance, one of the most common one:

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2 \tag{5.4.4}$$

which is the **Squared Cost Function**.
"L" apex indicates the last level after which the cost function is applied. It allows the user to track the number of levels needed to reach a reasonable result.
As written, cost function appears to include many terms:

$$C(W, B, S^T, E^T)$$

Where:

- $W \rightarrow$ Network weight;
- $B \rightarrow$ Network biases;
- $S^T \rightarrow$ Single training sample input;
- $E^T \rightarrow$ Single training sample desired output.

If the problem contains vector inputs, C may become difficult to interpret, due to the number of different incorporated terms. This is the reason why applicative cost functions are designed to depend just on weight multiplicity:

$$C(w_1, w_2, w_3, ..., w_n)$$

Assuming this form, it is interesting to observe that it could exist a weight value which leads to a minimum in the cost function. This doesn't necessary mean that evaluating the point with null derivative is the right path to follow. As shown in the picture, weight vectors may be n-dimensional and

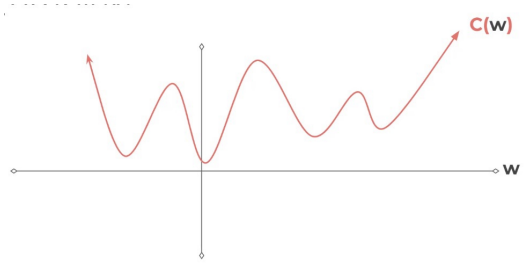
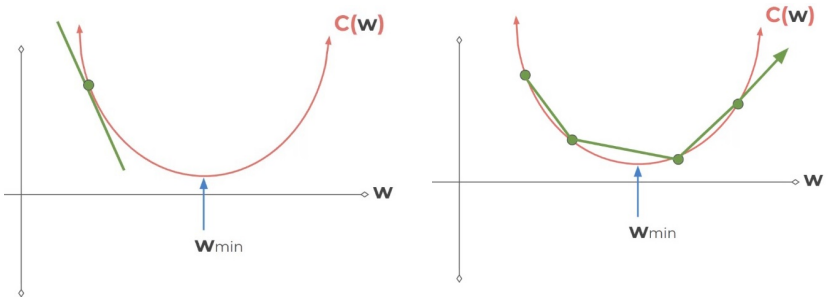


Figure 5.10: C function with different minimum values

associated cost functions may be a lot complex.
It is commonly used to adopt the strategy of the **"gradient descent"**, which consists in a series of iterations with the purpose of evaluating the slope of the tangent to the function in different points at x growing. The convergence of the tangent slope to the null value implies the presence of a minimum in that point. A small iteration step guarantees the convergence at the expanse of the computational time. On the other hand, if iteration step is too large, convergence point could be missed.The optimal step value is called "**Learning Rate**". Gradient descent with constant step is not the only or best method to use. It can be used a different version of GD which assumes a variable step value. These values will be larger when tangent slope is high and smaller when tangent slope approaches the null value. This new version has been named "**Stochastic Gradient Descent**" and it is a standard tool in machine learning.
For this project, a stochastic optimization method called Adam has been implemented in order to minimize the cost function. With respect to other known methods it has been shown having great performances (fig: 5.12).



(a) Gradient descent meaning **(b)** Large step failure

Figure 5.11: Gradient descent features

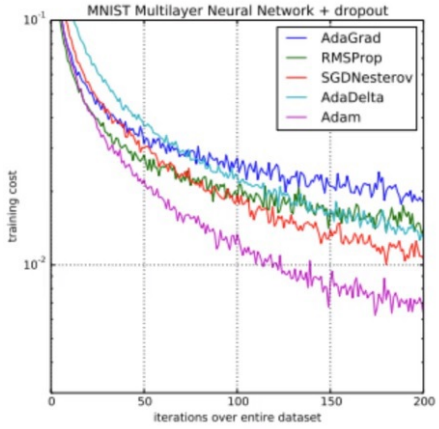


Figure 5.12: Adam performances[Jos]

BACKPROPAGATION In order to step back and adjust hyperparameters, the algorithm implements the "**backpropagation - BP**". Basically, the idea behind this operation is to establish, after each iteration, a criterion which introduces a variation in the weights as a function of the C values.
Considering a simple network as the one shown (fig. 5.13), the process described so far may be called "*forepropagation - FP*". As for backpropagation, a different kind of notation should be used.

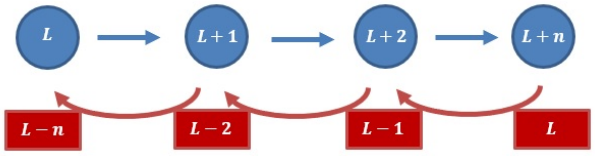


Figure 5.13: Notation in BP

Where:

$$n = L - 1$$

The first neuron on the left is indicated as $L - n = L - L + 1 = 1$ for the red notation (backward). This may appear counter-intuitive but it is made to consider that, as regard BP, the process starts from the right.

Adapting this to previous formulations (page 5.4.1), right neuron input expression may be written as:

$$z^L = w^L x^{L-1} + b^L \tag{5.4.5}$$

where $x^{L-1} = a^{L-1}$ is the output of the second neuron from the right. The expression above yields:

$$a^L = \sigma(z^L) \tag{5.4.6}$$

To which corresponds the cost function:

$$C_0(...) = (a^L - y)^2 \tag{5.4.7}$$

("0" subscript means that this is the first iteration across the network.)

To estimate the cost function sensitivity with respect to the weight vector, some already known relationships may be used within the chain rule[4]:

$$\frac{\partial C_0}{\partial w^L} = \frac{\partial z^L}{\partial w^L} \frac{\partial a^L}{\partial z^L} \frac{\partial C_0}{\partial a^L} \tag{5.4.8}$$

Note that it is possible to obtain each derivative form from previous relationships (eqs.5.4.5, 5.4.6, 5.4.7).

To evaluate the minimum value for the cost function it is necessary to operate with a gradient method. Then, this can be used to evaluate the error vector δ, using "Hadamard product", as follows:

$$\delta = \nabla_a C \odot \sigma'(z^L) \tag{5.4.9}$$

where $\nabla_a C = (a^L - y)$. "a" subscript means that the speed variation of C must be expressed as a function of the output "a".
From now on, "l" will be meant to identify the generic level but the last, which will be named L. The formulation above can be generalized and expressed as a function of the backward direction, as follows:

$$\delta^l = (w^{l+1})^T \delta^{l+1} \odot \sigma'(z^l) \tag{5.4.10}$$

Transposed weight vector is what "pushes" the error backward to level "l", along with Hadamard product for $\sigma^l(z^l)$, which pushes the error even through the activation function, in order to evaluate a sort of weighted error δ^l for the "l^{th}" input.
At the end, partial derivatives of the cost function can be evaluated for each level using the following relationships:

$$\begin{cases} \frac{\partial C}{\partial w^l_{jk}} & = a^{l-1}_k \delta^l_j \\ \frac{\partial C}{\partial b^l_j} & = \delta^l_j \end{cases}$$

In conclusion, all that has been said so far in this chapter, may be considered enough to understand the building of ANUBI DLN project in terms of theoretical background.

5.5 Overfitting

Before dealing with overfitting, it is strongly recommended to introduce the concept of "bias and variance".

4 mathematical expression which allows to evaluate a derivative relationship using some others.

5.5.1 Bias and Variance

One of the main feature behind the machine learning idea is the "**generalization**" capability, which is the ability of a model to apply information learned during training to brand new input.

"The difference between optimization and machine learning arises from the goal of generalization: while optimization algorithms can minimize the loss on a training set, machine learning is concerned with minimizing the loss on unseen samples. Characterizing the generalization of various learning algorithms is an active topic especially for deep learning algorithms." [Wik21b]

A good generalization needs the model to be able to recognize the difference between "bias" and "variance":

- "**Bias**" *is how a statistic mean is different from the actual value*. For instance, it is know that 100 is the mean global intelligence quotient (IQ). Anyway, taking as samples just the physics Nobel prizes (IQ $>$ 130 [Bru]), evaluated mean would be different from the actual world-based one.
- "**Variance**" *is how samples differ from the mean value* or, equivalently, *how data population is distributed around "true" value*. For instance, if the mean global IQ is 100, there will be more people within the $[90, 110]$ range and less outside. Besides, people outside this range can be seen as the "outlayers" of a hypothetical IQ estimation problem.

These concepts can be applied to machine learning, where they represents [Gov]:

- Bias $\rightarrow$ systematic error. Error between mean prediction and mean true value. Great bias means a too simple model, able to produce relevant errors during both training and testing phase;
- Variance $\rightarrow$ sensitivity to training data randomness. It is the prediction variability, considering to train the model multiple times on different datasets. Great variance means the model pays attention to training data but is not able to generalize new data, producing relevant errors on testing phase.

Some sort of visualization of what has been said can be found in figure 5.14, which has to be interpreted as follows:

- Every point is a mean prediction in the training set of the model after a training;
- Points differ each other because a variation in the training set is assumed for each training;
- The centre of the target means the perfect prediction.

Previous figure may be associated to table 5.1. Table meaning is:

- LB-LV $\rightarrow$ Training dataset is able to produce very good predictions because it describes a particular standard situation (low bias) and does not contain outlayers (low variance). Referring to previous example, this is like considering a bunch of standard (between 90 and 100) and not so different people IQs to estimate mean world IQ. Definitely a non realistic problem, unable to operate in case of outlayer estimation.

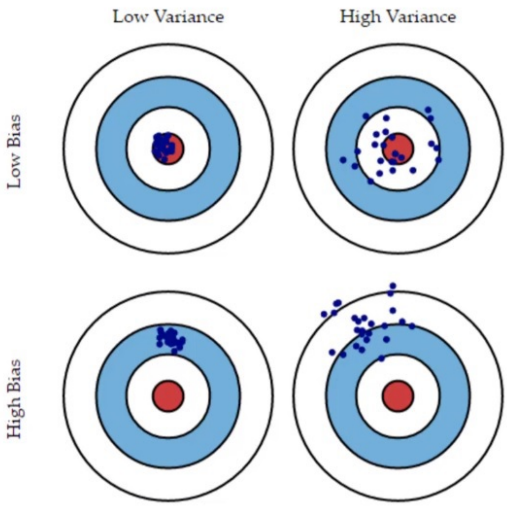


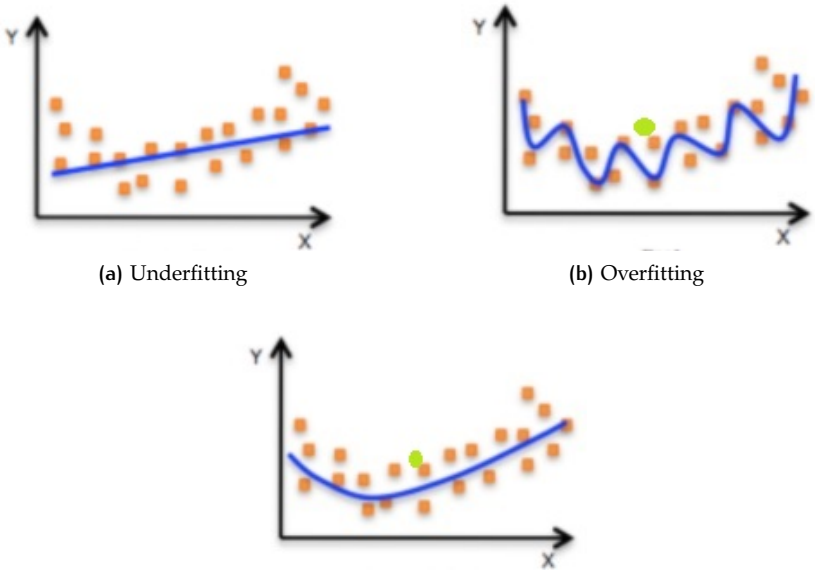
Figure 5.14: Bias and Variance [Jos]

	Low Variance - LV	**High Variance - HV**
Low Bias - LB	Simple Dataset - Correct prediction	Overfitting
High Bias - HB	Underfitting	Tough Dataset - Uncorrect predictions

Table 5.1: Bias Variance trade off

- LB-HV → Training dataset produces lots of different results (high variance) in what is assumed as a standard situation (low bias). Referring to previous example, this is like considering a bunch of standard but very different IQ values, to describe mean world IQ. Definitely a situation in which some data could behave as ”noise” for the training. When it happens, the system is referred to as ”**overfitted**”.
- HB-LV → Training dataset produces a localized (lo variance) and noticeable error (high bias) because it is not able to recognize data pattern. It can be observed that the error is systematic and it depends on data quality and quantity. As regards quantity, this is the main problem behind ”**underfitted**” models.
- HB-HV → Training dataset produces different and significant errors because data are not descriptive of the actual problem and they also lead to a noisy training. Referring to previous example, this is like considering a bunch of very different outlayered IQ values to describe mean world IQ.

Concluding, underfitting is surely a problem, but it is easy to recognize, while overfitting is the main issue which can be found while operating with DLN. One of the main overfitting alert is given by the comparison between training predictions and test predictions. If training set prediction are too better than test set prediction, system may be overfitted with noise. A perfect visualization of the phenomena discussed so far is figure 5.15. Overfitted systems are also called ”flexibles”, because of their appearance,



(a) Underfitting

(b) Overfitting

(c) Correct fitting

Figure 5.15: Fitting problems [Gov]

referring to the same figure, green dot represents a generic new test label to be forecast. One can immediately see that the error is greater in the overfitted dataset than in the correctly fitted one and this well summarize the consequence of the problem.

6 ANUBI HINGE MOMENT FORECASTING MODEL

6.1 Introduction

After having performed data cleaning (chapter 4.2), a couple of dataset have been produced in order to use them in this way:

- Dataset "**df**1": contains data regarding all δ, ν, α configurations but the $\delta = 0$ ones;
- Dataset "**df**2": contains data about $\delta = 0$ configurations.

Each dataset provides labelled inputs needed to train, test and validate the model. In particular:

- "df1": train/test set;
- "df2": validation set.

The model has been created and validated in a Jupyter IDE:

THE JUPYTER NOTEBOOK is an open-source web application that allows to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modelling, data visualization, machine learning, and much more [Jup]

6.2 Data Preparation and Exploration

First of all, to perform data preparation and exploration it is necessary to import some libraries:

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import cm
from matplotlib.ticker import LinearLocator
from scipy.spatial import ConvexHull
from mpl˙toolkits import mplot3d
```

To understand:

- "*panda*" library may be associated to a powerful Excel version, and it is useful tu handle series, dataframes and so on. It is a sort of high-level building block for data analysis;
- "*numpy*" library provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment

of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more [Num];

- "*matplotlib*" is the native python library for data visualization methods. It was created by John Hunter to emulate Matlab plotting capabilities;
- "*seaborn*" library is useful for statistic plotting and it is well made for working with panda objects.
- Other imported tools are for graphical design.

Once having imported df1 and df2 as "*pd.dataframes*", through a simple command it is easy to get a description about each set:

[6]:
```
df1.describe().transpose()
```

[6]:

	count	mean	std	min	25%	50%	75%	max
delta	2401.0	-0.088297	2.705032	-4.0000	-3.0000	-1.0000	2.0000	4.0000
v	2401.0	16.963140	3.033321	12.0000	14.5000	17.0000	19.5000	22.0000
alpha	2401.0	2.102041	4.142463	-5.0000	-1.0000	2.0000	6.0000	9.0000
H	2401.0	-0.089814	0.051890	-0.2692	-0.1217	-0.0803	-0.0505	-0.0035

[7]:
```
df2.describe().transpose()
```

[7]:

	count	mean	std	min	25%	50%	75%	max
delta	314.0	0.000000	0.000000	0.0000	0.000000	0.00000	0.00000	0.0000
v	314.0	16.984076	3.024089	12.0000	14.500000	17.00000	19.50000	22.0000
alpha	314.0	2.022293	4.316120	-5.0000	-2.000000	2.00000	6.00000	9.0000
H	314.0	-0.090132	0.040610	-0.2096	-0.116175	-0.08165	-0.05915	-0.0253

At the beginning of the project, it had been estimated to get 3024 examples, to be split in three main dataset, as described in table 2.3.
However, python count (second column) shows that the sum of all available datapoints is:

$$\texttt{Total} = 2401 + 314 = 2715$$

That means a loss of 309 configurations.
Introducing a new dataframe, such as "df=df1+df2", it can be plotted the total data loss as a function of α:

[8]:
```
plt.figure(figsize=(12,8))
sns.countplot(df1['alpha'])
```

[8]: ¡matplotlib.axes._subplots.AxesSubplot at 0x1a4a5296250¿

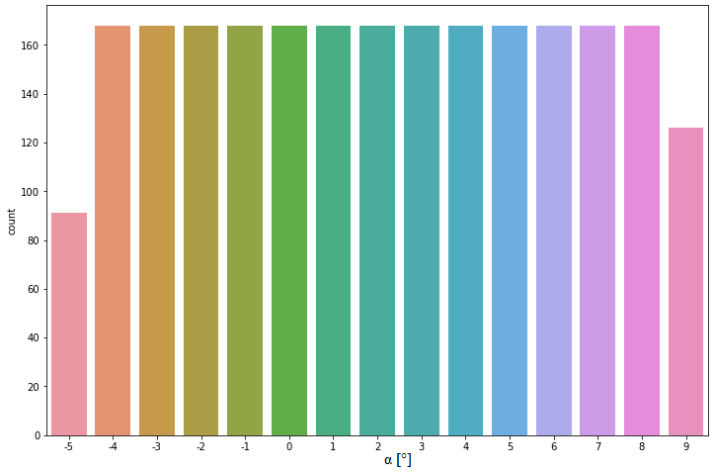


Figure 6.1: Data loss

Figure 6.1 clearly shows that all missing values are related to two different incidence values, which means that the problem is not in the data handling code or python implementation, but it is due to the aerodynamic analysis itself.
The reason behind this loss is easy, Xflr5 solver limitations have been hit: during 3D analyses with some configuration of δ and ν, the solver has not managed to interpolate boundary values of the α range.
This loss is not a problem for the project, since it does not affect model performances so much, considering that df2 input values (i.e. validation elements) lie inside δ, ν, α ranges.

Continuing with discussion, it is very useful to plot the distribution of H values, to estimate both:

- The range of variation;
- Some form of density of the distribution.

This can be done using the distribution plot provided by *seaborn*:

[8]:
```
plt.figure(figsize=(12,8))
sns.distplot(df1['H'])
```

Figure 6.2 shows an approximated distribution of hinge moment values. y-axis represents the probability density function and also a kernel density estimation (blue line) is plotted. Anyway, what really matters is that from the figure it can be observed that most of results are $-0.12, -0.11 < H < -0.02, -0.03$. For further convenience, it will be assumed the notation R for the range $H = [-0.12, -0.02]$. This graph gives some important information about what the model will be. A higher density of the output in a particular region means that the model could perform better predictions in that range. But it is not necessarily true and sometimes a model can even reach its training limit (i.e. overfitting) in less time (therefore training less), due to this high local density. This eventually results in worse predictions for some values.

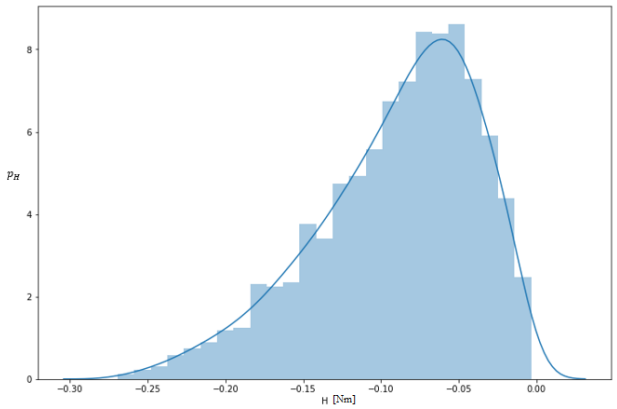


Figure 6.2: H distribution

However, this is a common issue in DLN models, and the reason why DLN is still to be face with a trial and error approach.
An other aerodynamic feature can be observed from 6.2. All hinge moments have a negative value, meaning the control surface is always trying to rotate around its axis in the same direction, no matter the magnitude and the combination of the inputs. This can be easily referred to two reasons:

- There is no combination among considered parameters which is able to generate a pressure resultant, on the upper side of the control surface, bigger than the lower side one. So the control surface always tries to rotate towards the upper profile surface;

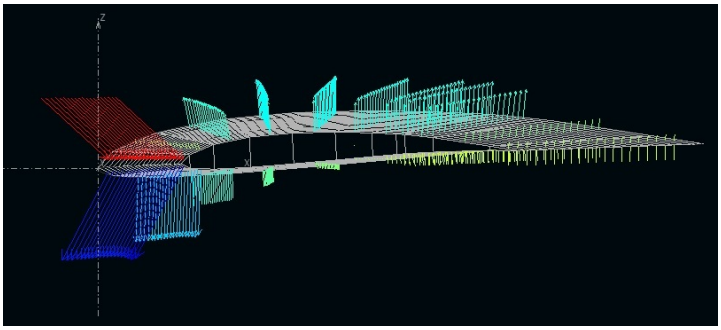


Figure 6.3: Pressure distribution plot

- Profile geometry is such that the flow:
  - Always expanses on top surface (after a first compression at the leading edge) → Pressure right before trailing edge has a minimum value;
  - Expanses during a first moment on bottom surface. Then it starts compressing from half chord till the trailing edge → Pressure right before trailing edge has not a minimum value.

  Aforementioned compression is due to the buckling geometry of the profile, a feature which becomes evident right after $x = c/2$.

Figure 6.3 shows a pressure distribution for:

- $\delta = 0$ [deg]
- $v = 12$ [m/s]
- $\alpha = -5$ [deg]

As it may be guessed, python is a useful tool for evaluating trends of functions and data. For instance, considering the dataframe df2 as a fixed δ bunch of configurations, it can be observed what happens to H by varying v and α:

[170]:
```
plt.figure(figsize=(12,8))
sns.scatterplot(x='alpha',y='H',
data=df2,hue='H',
palette='coolwarm',edgecolor=None,alpha=1)
```

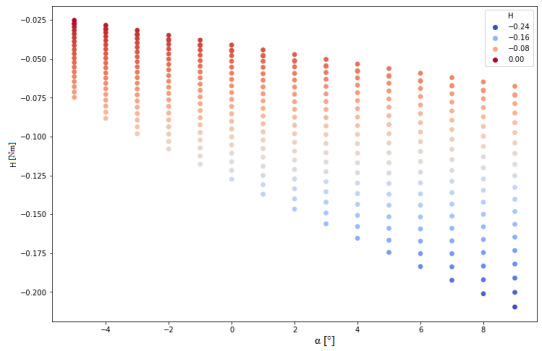


Figure 6.4: H vs α for "df2"

From fig 6.4 it can be observed that hinge moment has a sort of linear behaviour with respect to the incidence, in agreement with what has been said in section 2.1. For every α, more than one H value is plotted, because there are about 20 different speed values per incidence to consider. A better ex-

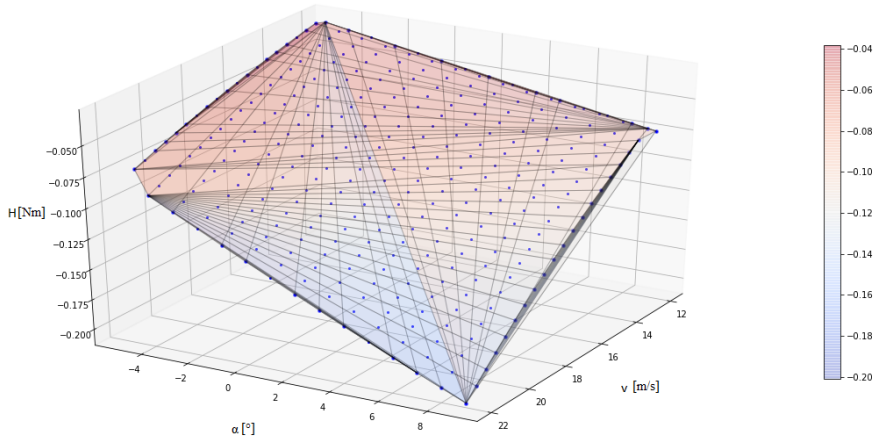


Figure 6.5: $\delta = 0°$ configuration

planation may be featured in figure 6.5, where a 3D view perfectly shows minimum and maximum hinge coordinates in terms of v and α.

However, all of this was just to set the field for the larger "df1" analysis. In this case there is one more dimension to consider and this can lead into mistakes. By the way, with respect to 6.4, it is useful to remember that δ values must result in similar behaviour of H, because of what has been said in section 2.1:

[14]:
```
plt.figure(figsize=(12,8))
sns.scatterplot(x='delta',y='H',
data=df1,hue='H',
palette='coolwarm',edgecolor=None,alpha=1)
```

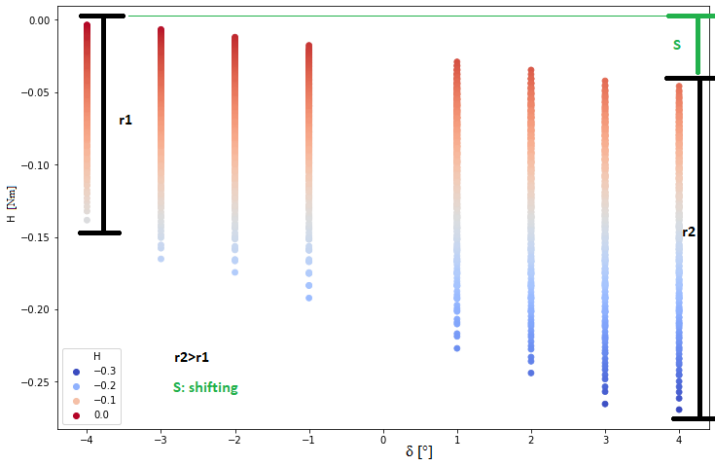


Figure 6.6: $\delta \neq 0°$ configurations (2D)

Figure 6.6 shows that by varying δ, hinge moment (considering fixed speed and incidence) has a linear trend. In other word, as δ steps forward, the range of resulting hinge moments shifts towards smaller values. Besides, in this graph it is more evident that by increasing aileron deflection, hinge moment range of values becomes larger. It implies low H results density in the R range for positive δ configurations. Figure 6.7 shows this feature and not only.

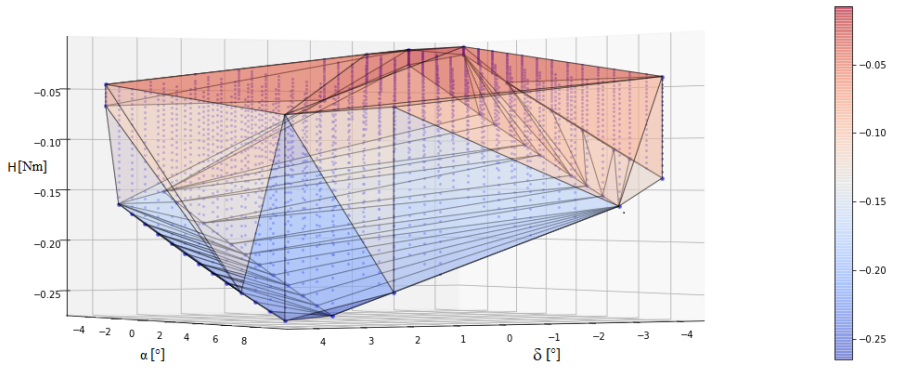


Figure 6.7: $\delta \neq 0°$ configurations (3D)

As regard dependency from speed, it can be observed (fig. 6.8) a slightly

quadratic trend, something which is in agreement with the classic expression of H (eq. 2.1.1). Using different views of previous graphs, minimum H

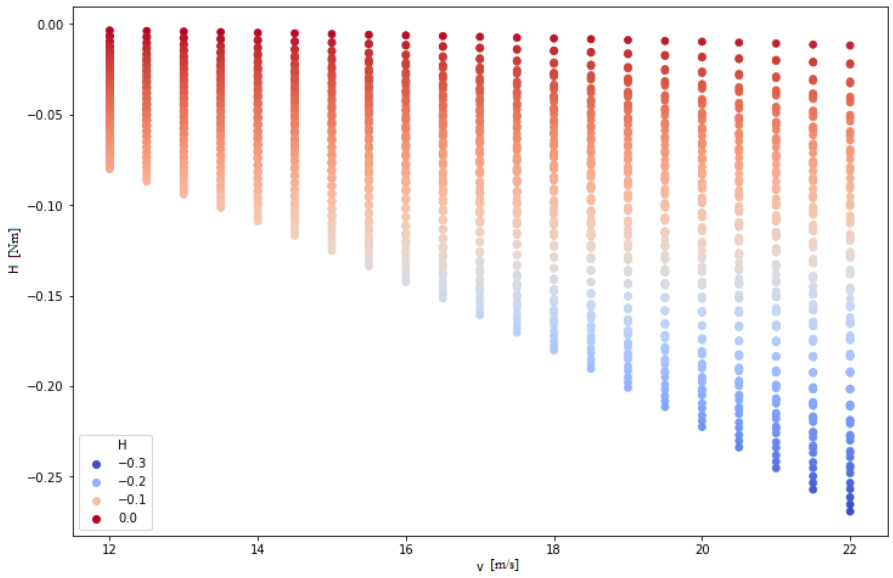


Figure 6.8: $v - H$ graph

value position may be estimated at $[\delta, v, \alpha] = [4, 8, 22]$. Actually, that is the exact set of parameters which yields to the minimum value $H_{MIN} = -0.2692$, because every representation is clear enough, as well as the trends. Nevertheless, different analyses strategies or a multitude of other configuration would have led to other not so easy conclusions.

6.3 Model building

6.3.1 Datasets split/scale

In order to realize the model, libraries set has to be extended with Tensorflow and Scikit Learn tools:

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from matplotlib import cm
from matplotlib.ticker import LinearLocator
from scipy.spatial import ConvexHull
from mpl_toolkits import mplot3d
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
```

```
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import␣
↪mean_squared_error,mean_absolute_error,explained_variance_score
```

First thing to do is to split dataset "df1" in training and testing data. Performed splitting turned out to be slightly in disagreement with what was assumed in chapter 2.4.4 (Table 2.3), for two main reasons:

1. Data loss, as discussed in previous section;
2. Before thinking about the model, it has been chosen to manually extrapolate a bunch of examples from the whole set ($\delta = 0$ configurations) to perform a clearer data analysis. Afterwards it has been decided to use this set as the "'Validation Set".

Considering this, actual established partition is reported in table 6.1. Par-

DataFrame	DF Percentage	Assigned to	Data Points
df1	70%	Train	1.680.7
	30%	Test	720.3
df2	100%	Validation	314

Table 6.1: Dataset partition

tition is performed directly by "*train_test_split*" function, passing the argument "*test_size*= 0.3". The problem of decimals approximation, such as in df1 splitting, is handled within the function itself.

Once all datasets are split, it is time to face a particular problem. Considering the range of possible values for H (fig. 6.2), it is evident that the model will deal with small values:

$$0 < |H| < 1$$

meaning $\frac{\partial C}{\partial w}$ (from eq. 5.4.8), may yield to the common issue of "**vanishing gradient**".

"*In machine learning, the vanishing gradient problem is encountered when training artificial neural networks with gradient-based learning methods and backpropagation. In such methods, each of the neural network's weights receives an update proportional to the partial derivative of the error function with respect to the current weight in each iteration of training. The problem is that in some cases, the gradient will be vanishingly small, effectively preventing the weight from changing its value. In the worst case, this may completely stop the neural network from further training.(...) Backpropagation computes gradients by the chain rule. This has the effect of multiplying n of these small numbers to compute gradients of the early layers in an n-layer network, meaning that the gradient (error signal) decreases exponentially with n while the early layers train very slowly.*" [Wik21e].

To avoid this problem, different scaling strategies may be adopted. For the present code, Scikit Learn provides an handy function which creates a scaler parameter based on the minimum and maximum values of a particular dataframe. This scaler has been generated on the training set and

applied to all dataframes. This is all about datasets preparation.

6.3.2 Model training

To be trained, the model needs its architecture, which has been created according to what has been said in chapter 5.5.1:

[9]:
```
model = Sequential()
model.add(Dense(3,activation='relu'))
model.add(Dense(3,activation='relu'))
model.add(Dense(3,activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam',loss='mse')
```

What is shown in the code is just the first experimented architecture, that is:

- 3-neurons input layer;
- 3-neurons hidden layers (x2);
- 1-neuron output layer.

Output layer will always have 1 neuron, reason for which all considered configurations will be referred to as:

$$[N, L]$$

meaning a structure made by N neurons for each of L levels (input+hidden). So the code above is about a $[N, L] = [3, 3]$ architecture.
"*Adam*" is the stochastic optimizer introduced at page 42, while "*mse*" refers to the metric used for loss evaluation during the training (Mean Squared Error).

The model can now be trained on the train set. It will try to predict values in the test set at the end of every epoch, to adjust hyperparameters[1]. An "*epoch*" can be considered as a complete pass on the train set.

[10]:
```
model.fit(x=X_train,y=y_train,
validation_data=(X_test,y_test),
batch_size=16,epochs=200)
```

As regard "batch_size", a longer discussion is needed.

THE BATCH SIZE PROBLEM *"The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters. Think of a batch as a for-loop iterating over one or more samples and making predictions. At the end of the batch, the predictions are compared to the expected output variables and an error is calculated. From this error, the update algorithm is used to improve the model, e.g. move down along the error gradient"* [Jas21].

In other words, training on data batches allows to perform a certain number

1 The model will not train on the test set, but it will use it just to verify its calculations.

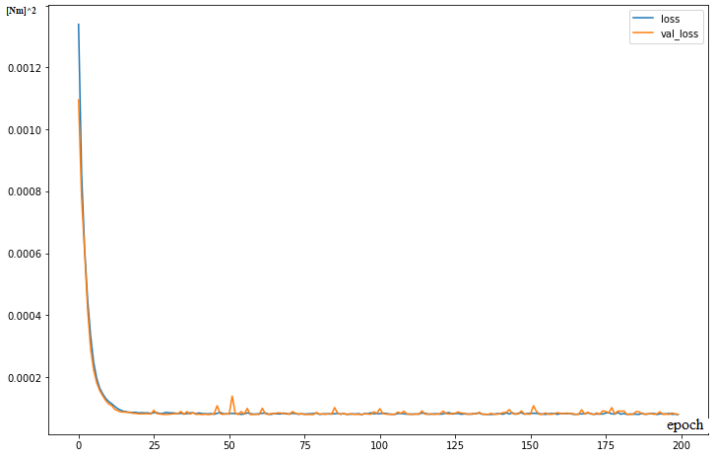
of output comparison (with predicted values) before evaluating the correction to be applied to the model. This affects network generalization[2] capabilities in a way which is still under study.
"*It is often reported that when increasing the batch size for a problem, there exists a threshold after which there is a deterioration in the quality of the model*" ([17]) which may lead to overfitting. Nevertheless, LBs allow to speed up the training and may provide as good generalization capabilities as SB (strictly context-related).
SBs, on the contrary, tend to reduce the risk of overfitting at the expense of the computational time. It appears that they can offer a regularizing effect [17], perhaps due to the noise they add to the learning process.
Something about this has been experimented for the present model, as it will be seen.

Back on topic, at the end of each epoch, two loss values are printed. For simple networks, losses gradually decrease with time.

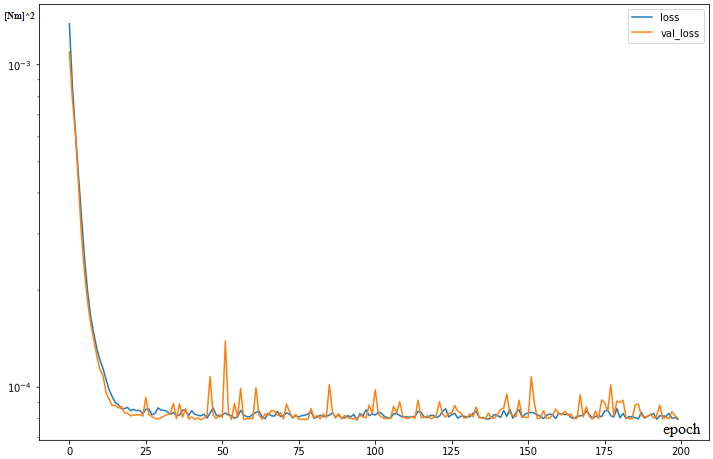
```
(...)
Epoch 6/200
105/105 [==============================] - 0s 3ms/
 ↪step - loss: 0.0015 - val_loss: 0.0014
Epoch 7/200
105/105 [==============================] - 0s 3ms/
 ↪step - loss: 0.0012 - val_loss: 0.0013
Epoch 8/200
105/105 [==============================] - 0s 2ms/
 ↪step - loss: 0.0012 - val_loss: 0.0012
Epoch 9/200
105/105 [==============================] - 0s 3ms/
 ↪step - loss: 0.0010 - val_loss: 0.0010
Epoch 10/200
105/105 [==============================] - 0s 2ms/
 ↪step - loss: 9.1057e-04 - val_loss: 7.5188e-04
Epoch 11/200
105/105 [==============================] - 0s 3ms/
 ↪step - loss: 6.4231e-04 - val_loss: 5.1092e-04
Epoch 12/200
105/105 [==============================] - 0s 2ms/
 ↪step - loss: 4.3388e-04 - val_loss: 1.8984e-04
Epoch 13/200
105/105 [==============================] - 0s 3ms/
 ↪step - loss: 1.7430e-04 - val_loss: 1.2261e-04
Epoch 14/200
105/105 [==============================] - 0s 3ms/
 ↪step - loss: 1.1582e-04 - val_loss: 9.4359e-05
Epoch 15/200
105/105 [==============================] - 0s 2ms/
 ↪step - loss: 1.0976e-04 - val_loss: 7.8603e-05
(...)
```

2 model ability to predict on brand new inputs

"loss" indication refers to training set loss, because the model learns by estimate train data, then it applies acquired knowledge to predict test labels. From this operation, "val_loss" indication is produced: fig. 6.9 shows the learning trend in terms of losses as a function of epochs.



(a) Normal plot



(b) Semi-log plot

Figure 6.9: Training

From the semi-logarithmic view, model seems to well behave although there is some low order fluctuation in the convergence zone. Noise itself does not necessarily mean that model will be bad, since it may be caused by the low batch size adopted in the training. Indeed, this is a problem just when fluctuations rise with time in terms of amplitude and frequency.
By the way, all produced model have been tested on both "test set" from df1:

[13]:
```
predictions = model.predict(X_test)
```

and "validation test" from df2 (brand new data):

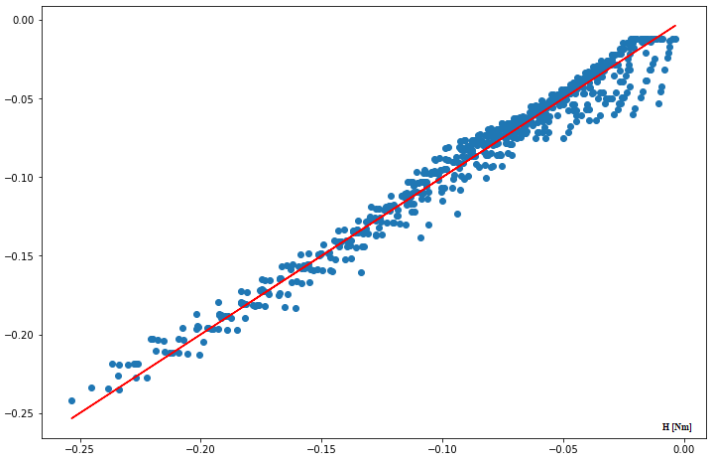
[20]:
```
newpredictions = model.predict(X2_scaled)
```

6.4 Model evaluation

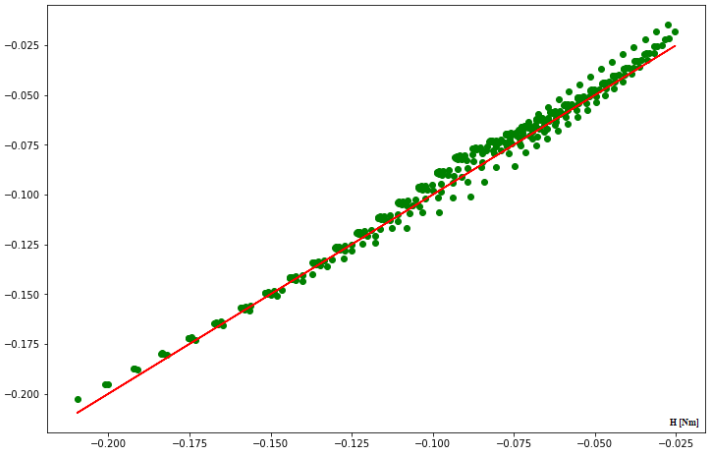
As regards previous model testing, two different scatter plot (fig. 6.21) may be generated. These images are built as the superposition of two graphs:

- First graph has x-axis= H_{true} and y-axis= H_{true}, so that the red line represents the bisector of the quadrant, as well as the points locus of exact H values (true labels);
- Second graph has x-axis = H_{true} and y-axis = $H_{predicted}$.

This kind of superimposition allows to show a comparison between true and predicted values as a vertical distance between the points (prediction) and the red line (true labels).



(a) Test set prediction



(b) Validation set prediction

Figure 6.10: Mk1 True labels line (red) VS Predictions

From 6.10a it can be observed that something went wrong during training, because of the weird behaviour near zero value (flat prediction placing), perhaps due to overfitting or to the lack of training data in that range, most likely the first. By the way predictions on validation set (fig. 6.10b) seem to be nice and very close to true values.
Conclusions must be provided in terms of numbers and for this reason, some evaluation metrics have been adopted:

1. Mean Absolute Error: from section 5.3;
2. Root Mean Squared Error: from section 5.3;
3. Explained Variance Score;
4. Relative Error.

Explained Variance Score is basically an index from 0 to 1 of model goodness. $EVS = 1$ means a perfect prediction. Considering:

- P: Predictions;
- Y: True values;
- $\hat{Y}$: True values mean;

for evaluating explained variance of N datapoints, user needs to:

1. Evaluate all Absolute Deviations $AD = |P_i - \hat{Y}|$;
2. Evaluate explained deviance or Summary Squared Regression $SSR = \frac{1}{N}\sum_i AD_i^2$;
3. Final calculation:
$$EV = \frac{SSR}{N} \tag{6.4.1}$$

The dual index is Residual Variance:

1. Evaluate all Absolute Errors $AE_i = (P_i - Y_i)$;
2. Evaluate residual deviance or Summary Squared Error $SSE = \sum_i AE_i^2$;
3. Final calculation:
$$RV = \frac{SSE}{N} \tag{6.4.2}$$

The less is RV, the best is the model.
According to this, tab 6.2 shows results for the two present situations.

	Test set	**Validation set**
MAE	0.006618908	0.004382582
RMSE	0.008927034	0.005339941
EVS	0.971682142	0.988075351
H **Mean Value**	$\hat{H}_{df1} = -0.089814244$	$\hat{H}_{df2} = -0.090131528$

Table 6.2: Mk1 Overall Prediction Performances

As it can be seen, the model works well enough on forecasting brand new data, but it has to be considered that validation set barely meet that range of values in which the model seemed to train worse: in other words validation set prediction are performed with the "well trained side of the model". This may be clearer looking x-axes of figure 6.9 and will surely be, during the next step of the discussion.

It was previously said that EVS is a goodness index, and in this case, model

performances seems to be very good, neglecting the issue for $H \to -0.01, -0.02$. Anyway, there is an other parameter which allowed to make more practical considerations about the error entity:

$$RE = \frac{|AE|}{\hat{Y}} \tag{6.4.3}$$

Relative Error allows to quickly estimate if a generic error is small or big with respect to a specific measured quantity. In other words, it let understand if a measured error is tolerable in the present system. It is used as a percentage of the correct value.

It makes sense to evaluate the relative error on the validation set, which summarizes model generalization capabilities.

[23]:
```
Rel_Err=[]
Mean_Prep=0
Range=range(y2_resh.size)
for k in Range:
    Err=(y2_resh[k][0]-newpredictions_resh[k][0])/y2_resh[k][0]
    Rel_Err.append(Err)
type(Err)
```

In order not to write down all numeric values, a scatterplot has been realized (fig. 6.11).

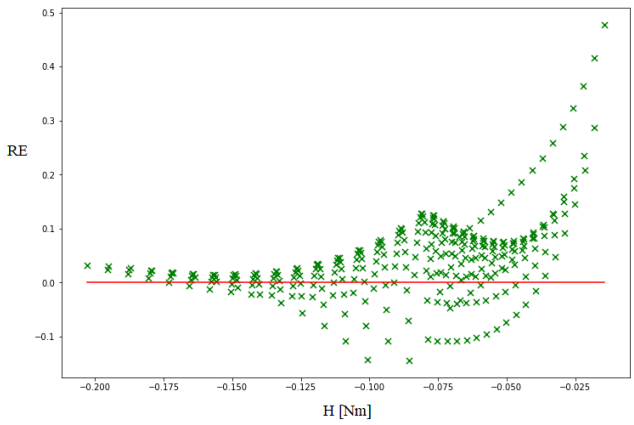
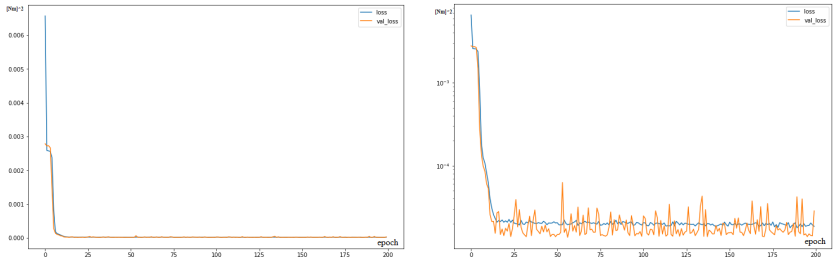


Figure 6.11: Mk1 Relative Errors

- RE max range: 0.6221 → It is a quite large range of variation, but it has to be considered that most of values are placed around the zero line, so it is a "no-weighted" estimation of error entity;
- RE arithmetic mean: 0.0456 → A simple arithmetic mean offers a more statistic description of what will most likely happen using this model. 4.56% of mean tolerance is a good value.

By the way, picture shows a particular growing behaviour which warns about what has been said at the very beginning of the present section. Model works bad around $H = -0.025$ [Nm] and beyond (till H_{max}).

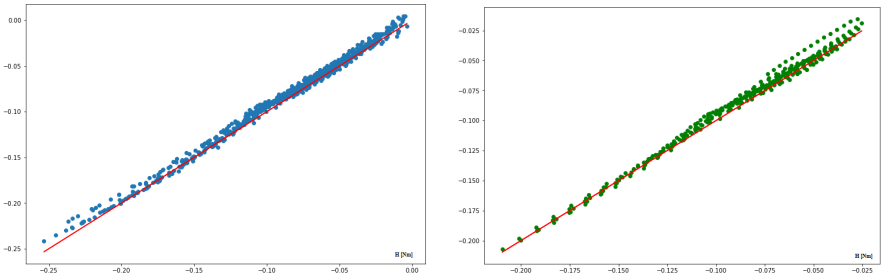
IMPROVING PERFORMANCES To improve training performances, it was firstly chosen, maintaining the same [N, L] configuration, to lower the batch size, resulting in a slightly noisier training (6.12).



(a) Normal plot **(b)** Semi-log plot

Figure 6.12: Mk2 Training

By the way, according to what has been said before about batch size, training resulted in a better model, well trained also around $H = -0.025$ [Nm] (fig. 6.13). By the way Comparing 6.13b and 6.10b, it can be observed that batch



(a) Test set prediction **(b)** Validation set prediction

Figure 6.13: Mk2 True labels line (red) VS Predictions

size changing affected the whole training, since last predictions are more precise than the first ones. Improved model performances are reported in table 6.3. However, the best way comprehend the meaning of these values

	Test set	**Validation set**
MAE	0.0043127210	0.003862890
RMSE	0.0053392147	0.004976332
EVS	0.9950987408	0.991969700
H Mean Value	$\hat{H}_{df1} = -0.089814244$	$\hat{H}_{df2} = -0.090131528$

Table 6.3: Mk2 Overall Prediction Performances

is to consider the RE trend (fig. 6.14).

- RE max range: 0.5255 → Again, this is a quite large range of variation, but same considerations as before may be done.
- RE arithmetic mean: 0.0500 → A bit greater then before, 5% of mean tolerance.

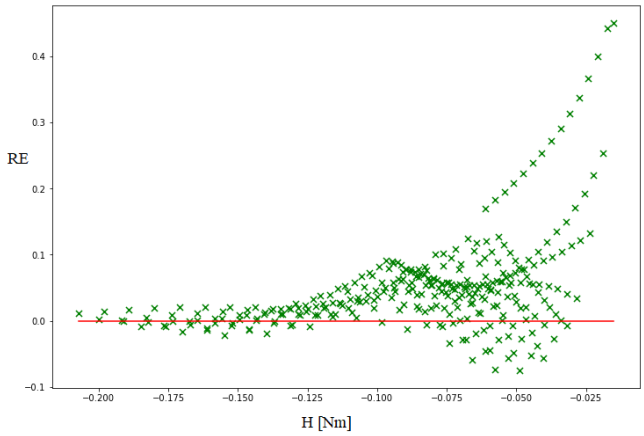


Figure 6.14: Mk2 Relative Errors

Comparing these two indications with the Mk1 ones, it may be said that error maximum range has been lowered by 0.1 at the expanse of a little shifting for the mean value. This is a good compromise: by shifting by 0.5% the mean tolerance, it has been reduced by 10% the worst case error.
In conclusion, Mk2 is a good model, able to perform quite nice predictions from H_{min} to $H = -0.05\,[Nm]$. As well as Mk1, after this threshold, RE stretches in both negative and positive directions. This is something normal and it depends on dataset composition: this is the "R" range of dataset (page 51) , the most susceptible to overfitting and related prediction error growing.

6.5 Model optimization

Mk2 model is not "ready to deploy" yet. 40% error, even if on 5 out of 314 predictions on validation set, is something that cannot happen on a true application. A better model has to be created.

Generally speaking, the great is the number of neurons and levels, the better the model will perform in terms of training goodness and computational time. However, there might be situations in which a "less" powerful configuration performs better than a slightly "higher" one. In other words, sometimes a right combination of neurons and levels may be better than an "all maximum" in terms of computational and time cost. For this reason it has been chosen to realize an optimization algorithm which tries to evaluate the right combination of neurons and levels within a prearranged range, for reaching the best result in terms of relative error and computational time.
Combined set of levels and neurons were:

$$N = [1, 2, 4, 6, , 8, 10]$$
$$L = [1, 2, 3, 4, 5, 6]$$

The algorithm basically trains 36 different models and evaluates relative errors and related features of each one.
Figure 6.15 shows relative errors (absolute values) on "test set" with a 3D view. It appears that there is a form of threshold at $[N, L] = [4, L]$ and $[N, L] = [N, 3]$. For less neurons/levels, model seems to generate huge errors (blue points), at least for used batch size and epochs number.
Other configurations work well enough. The same visualization is provided for "validation set" prediction REs (fig. 6.16).

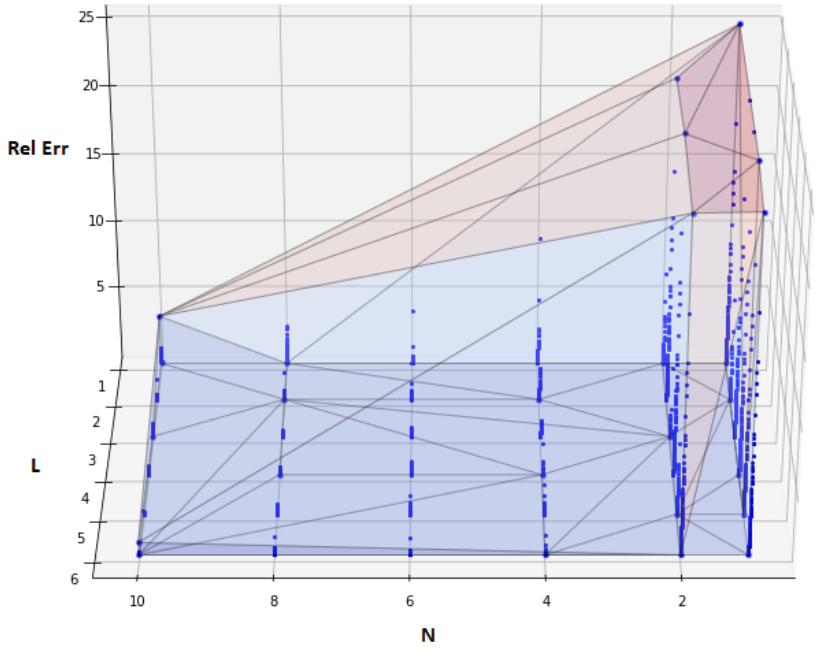


Figure 6.15: Optimization algorithm results - "Test Set"

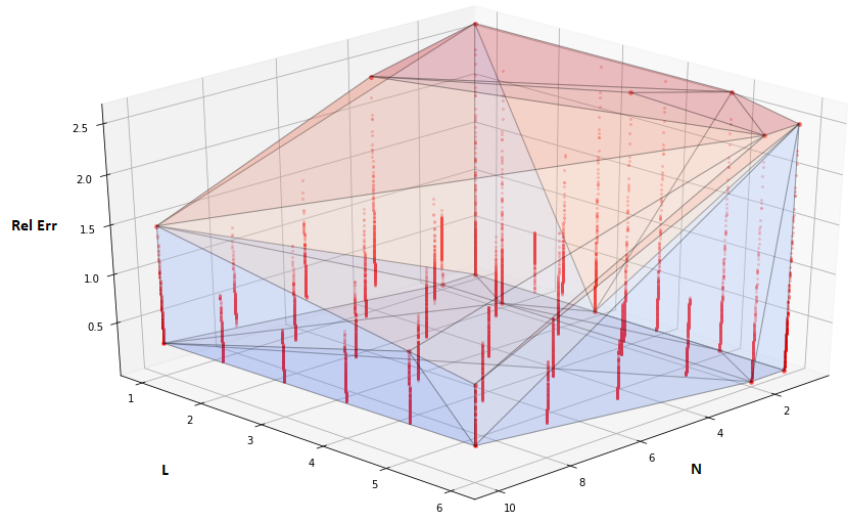


Figure 6.16: Optimization algorithm results - "Validation Set"

In order to select the best model, it has been chosen to evaluate the configuration with smaller value of:

$$\Delta RE = RE_{max} - RE_{min}$$

This means that highest RE density configurations have been selected, both from test and validation set predictions. Results are described in table 6.4. Results must be explored under different prospectives:

Set	Configuration		RE_{min}	Configuration		ΔRE_{min}
	N	L		N	L	
Test	8	4	0.000007	10	5	0.255799
Validation	2	2	0.000396	6	4	0.511165

Table 6.4: Selected configurations

- Configuration $[8, 4]$ leads to the minimum RE value on test set, while the minimum on validation set is reached for configuration $[2, 2]$. However, none of this configurations is representative for the present study. This because the minimum RE may be reached even for bad models, exactly what happened with validation set predictions;
- Configuration $[10, 5]$ leads to the minimum ΔRE value on test set, while the minimum on validation set is reached for configuration $[6, 4]$. Between these two choices it has been decided to adopt the configuration which performed better on the test set, because it has been taken into account the problem of "randomization".

THE RANDOMIZATION FEATURE Some algorithms are stochastic. *"This means that their behaviour incorporates elements of randomness". Nevertheless, "stochastic does not mean random. Stochastic machine learning algorithms are not learning a random model. They are learning a model conditional on the historical*

data user has provided. Instead, the specific small decisions made by the algorithm during the learning process can vary randomly. The impact is that each time the stochastic machine learning algorithm is run on the same data, it learns a slightly different model. In turn, the model may make slightly different predictions, and when evaluated using error or accuracy, may have a slightly different performance" [Jas20].

By adding randomization, algorithm may improve its performances. It's not a bug, it's a feature. By the way it also means that every building of the model may be slightly different from the previous one, but considering that the main trend must be the same each time.

Back on topic, it has been judged unwise to select the configuration with the ΔRE_{min} evaluated on validation set because of the randomization feature of the algorithm. Starting the code many times indeed, has shown that ΔRE_{min} for validation set switches among different configurations, while the one evaluated on the test set switches from two configurations, $[10, 5]$ and $[8, 3]$, mainly the first. Figure 6.17 actually shows that ΔRE is quite close between these two cases, something that explains this behaviour.

In conclusion, to be sure, both $[6, 4]$ and $[10, 5]$ have been separately realized and tested ($[8, 3]$ as been considered unimportant for present treatment).

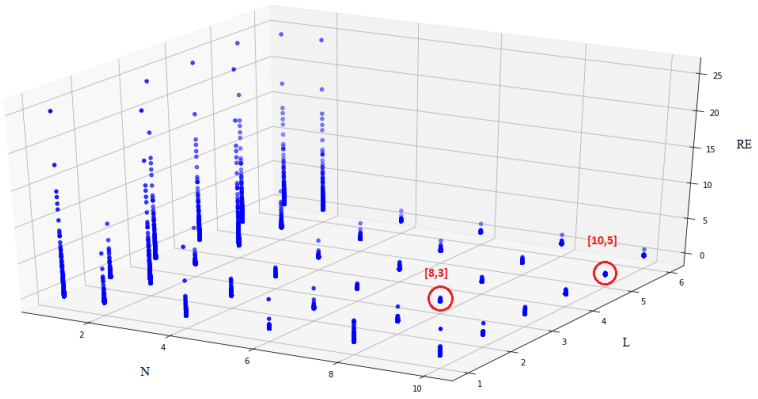


Figure 6.17: Scatter plot for test set configurations

6.6 Final Model

As regards configuration [6, 4], training phase resulted in an overall better behaviour with respect to previous model. The main difference surely is the reduction on RE indices. However, even if it has been obtained a mean

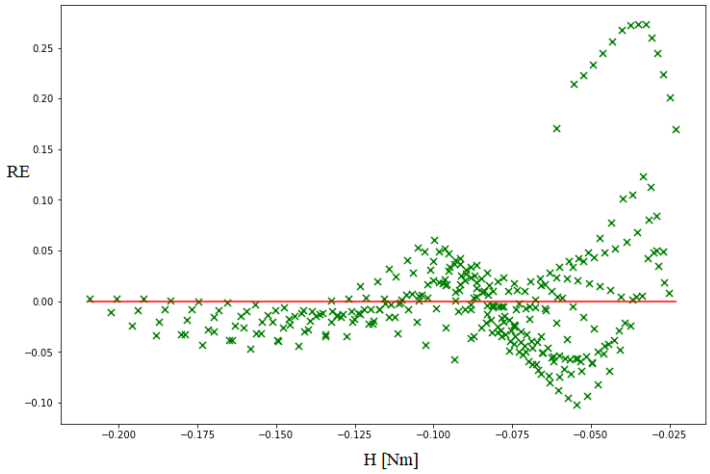


Figure 6.18: Mk3 Relative Errors

tolerance of 0.4% is great, ΔRE_{max} could be better. Turned out that the optimized [10, 5] configuration was able to improve this model feature (fig. 6.19). Here, REs appear to be more scattered, but just because y-axis has a

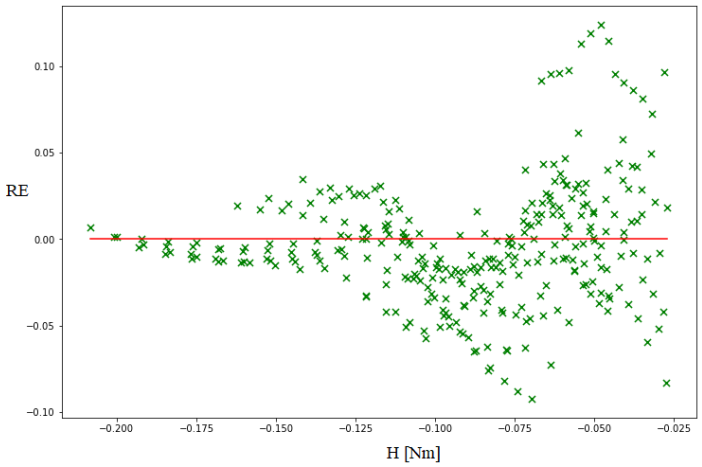
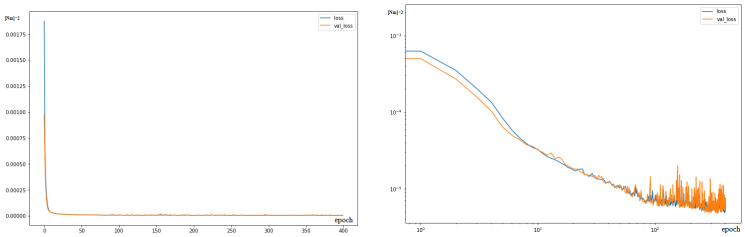


Figure 6.19: Mk4 Relative Errors

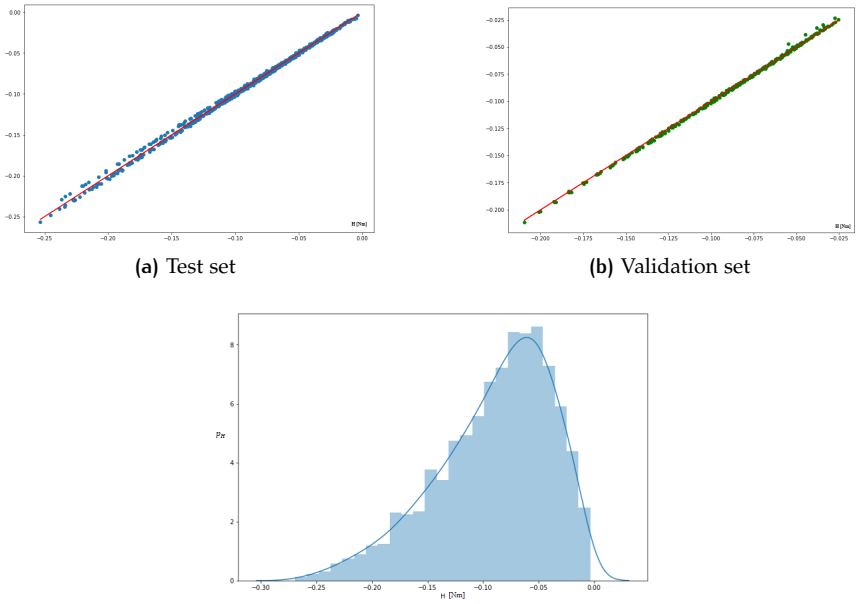
reduced range to show, with respect to the previous case: ΔRE_{max} has been reduced by about 0.1.
Nevertheless, this wasn't the ultimate model, because something more suitable has been reached working with a low batch size. Figure 6.20 shows a quite noisy training during final epochs, something normal for what has been previously said (page 57).



(a) Normal plot **(b)** Logarithmic plot

Figure 6.20: Mk5 Training

Training has resulted in a very coherent trend, as regards predictions. Recalling the H distribution plot (fig. 6.21c), figure shows that the more data density is high, the more model local performances are good, at least until a threshold inside the "R" range after which it starts to make some noticeable errors (on "validation set" predictions).



(a) Test set **(b)** Validation set

(c) Distribution plot

Figure 6.21: Mk5 True labels line (red) VS Predictions

As regard the relative error (fig. 6.22), it can be observed a drastic improvement, comparing to Mk1 model (fig. 6.11). This time:

- RE max range: 0.192 → Still quite large, but only 5 REs stretched around 0.16;
- RE arithmetic mean: 0.0053 → A bit greater then before, but 0.53% of mean tolerance allowed to reduce RE maximum range in value and primarily in terms of points which reach it.

Table 6.5 summarizes final model features. All graphics and details about every model is reported in appendix A.1.

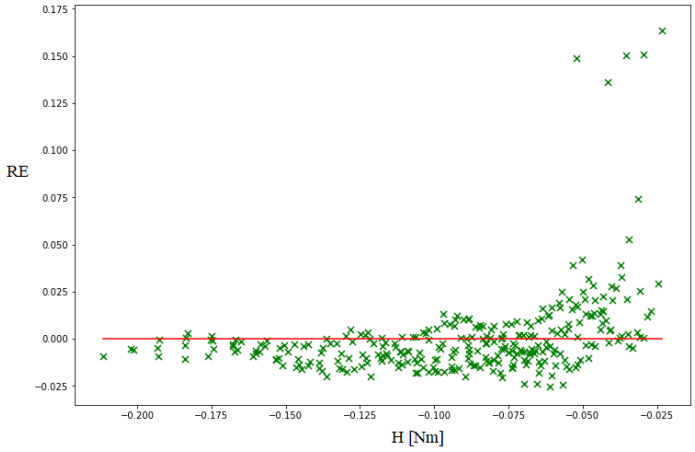


Figure 6.22: Mk5 Relative Errors

	Test set	**Validation set**
MAE	0.001597852	0.001153364
RMSE	0.002226205	0.002050664
EVS	0.998240684	0.998442156
$\mathbf{RE_{mean}}$		0.005320076
$\mathbf{\Delta RE_{max}}$		0.192899931
H **Mean Value**	$\hat{\mathrm{H}}_{df1} = -0.089814244$	$\hat{\mathrm{H}}_{df2} = -0.090131528$

Table 6.5: Mk5 Overall Prediction Performances

7 RESULTS AND FUTURE DEVELOPMENTS

7.1 Results

Present study has shown feasibility and limits of some hinge moment forecasting models. Last one is able to predict with accuracy H values in most of configurations.
Anyway, the entire study has shown the hinge moment is able to vary very quickly for little modifications in input parameters. Indeed, this is the reason why all of the models fail in some points inside the R range. Obviously, for the last model the entity of this failure is small enough: about 1.6% of the predictions may be affected by an error of about 15% of the correct value.
Moreover, it has to be considered that hinge moment high variation ratio (with δ,ν and α) means that, as regard real applications, the model will quickly switch from one result to an other, meaning in turn that wrong values may be easily recognized among all correct ones. Nevertheless, something might be done to reduce the error on validation set, and future developments will take this into account, not strictly improving the code itself, but even adding more analysis, in order to build a more complex database. Some solutions are described in further sections.

7.2 Adding a structural input

For the present model, it has been used a particular set of inputs, which were also aerodynamic analysis parameters. This does not mean δ, ν, α are the only possible inputs to estimate hinge moment through a neural network. Moreover, it is not so easy to calibrate flow incidence sensors, because most of them need to acquire pressure distribution data. It might be easier to have displacement/strain sensors available. Assuming this, for every considered input configuration it would be possible to extrapolate a particular strain input to feed the DLN model with. It could be a feasible improvement for the model to provide a structural analysis of the control surface for evaluating the strain on the hinge axis, where a strain gauge is assumed to be placed.
For this purpose, adopting a Data Handling strategy which is slightly different from the one described in chapter 4.2 it is possible to map a mesh for a pre-modelled surface to use Xflr5 pressure distribution as an input for a FEM analysis. Actually, this code, along with the model has been realized during the first phases of the project and it works like this:

1. From a triple loop cycle, as previously done, each Xflr5 .tex output file may be explored;

2. Inside the .tex file it can be found the coordinate position of every force application point. These forces are pressure vectors and points the centroid of each mesh element.

Main Wing Cp Coefficients

Panel	CtrlPt.x	CtrlPt.y	CtrlPt.z	Nx	Ny	Nz	Area	Cp
Main WingCp Coefficients								
Strip 1								
11	1.807e-01	-1.850e+00	1.683e-04	-0.020	0.001	-1.000	1.688e-03	0.1663
12	1.637e-01	-1.850e+00	1.892e-04	0.017	-0.001	-1.000	1.688e-03	0.1208
13	1.467e-01	-1.850e+00	-2.333e-04	0.032	-0.001	-0.999	1.688e-03	0.0792
14	1.318e-01	-1.850e+00	-7.517e-04	0.038	-0.001	-0.999	1.266e-03	0.0505
15	1.190e-01	-1.850e+00	-1.247e-03	0.039	-0.001	-0.999	1.266e-03	0.0266
16	1.063e-01	-1.850e+00	-1.748e-03	0.039	-0.001	-0.999	1.266e-03	-0.0009
17	9.353e-02	-1.850e+00	-2.231e-03	0.037	-0.001	-0.999	1.266e-03	-0.0314
18	8.077e-02	-1.850e+00	-2.664e-03	0.031	-0.001	-1.000	1.266e-03	-0.0712
19	6.802e-02	-1.850e+00	-2.980e-03	0.018	-0.001	-1.000	1.265e-03	-0.1313
20	5.526e-02	-1.850e+00	-3.007e-03	-0.014	-0.001	-1.000	1.265e-03	-0.2331
21	4.251e-02	-1.850e+00	-1.318e-03	-0.243	-0.004	-0.970	1.304e-03	-0.2687
22	4.251e-02	-1.850e+00	3.952e-03	-0.499	-0.009	0.867	1.460e-03	0.1997
23	5.526e-02	-1.850e+00	8.833e-03	-0.187	-0.005	0.982	1.288e-03	-0.4555
24	6.802e-02	-1.850e+00	1.060e-02	-0.087	-0.005	0.996	1.270e-03	-0.4596
25	8.077e-02	-1.850e+00	1.132e-02	-0.025	-0.005	1.000	1.266e-03	-0.4248
26	9.353e-02	-1.850e+00	1.136e-02	0.019	-0.005	1.000	1.265e-03	-0.3809
27	1.063e-01	-1.850e+00	1.089e-02	0.053	-0.005	0.999	1.267e-03	-0.3375
28	1.190e-01	-1.850e+00	1.003e-02	0.082	-0.005	0.997	1.270e-03	-0.2930
29	1.318e-01	-1.850e+00	8.823e-03	0.106	-0.006	0.994	1.272e-03	-0.2451
30	1.467e-01	-1.850e+00	7.019e-03	0.131	-0.006	0.991	1.702e-03	-0.1856
31	1.637e-01	-1.850e+00	4.449e-03	0.167	-0.007	0.986	1.712e-03	-0.0845
32	1.807e-01	-1.850e+00	1.503e-03	0.174	-0.008	0.985	1.714e-03	0.0308

Figure 7.1: .tex file layout

3. By extraction of this coordinates, a pressure distribution can be realized on a FEM software. For the present study, it has been chosen Altair Hyperworks.

The model has been realized using the same geometry discussed in chapter 2.4.4. From CAD model, it has been isolated the central control surface (red

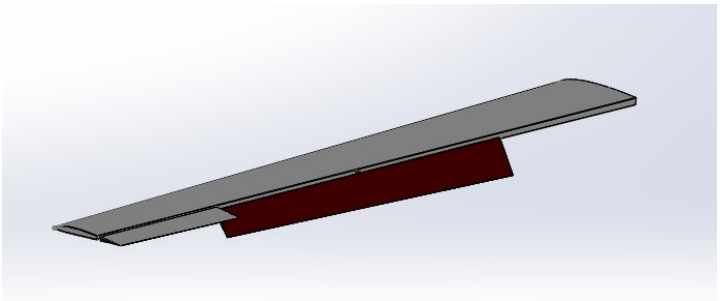


Figure 7.2: Ailerons CAD view

aileron), in order to import it on HW.
After having applied the same mesh used on Xflr5, it has been possible to realize the pressure distribution through a field generator function. This powerful tool recognized the coordinates of the centroids (mesh control points) and adapted related pressure forces on the present FEM mesh (fig. 7.3). Built model can be associated to a fixed surface FEM model to evaluate hinge axis strain as a function of the aerodynamic features. This strain could be used to replace one of the inputs used in the present job, or additionally. By the way, as previously said, this operation requires more studies and all the process is demanded to future developments.

7.3 Hinge approximation

Calling back equation 2.1.3, it can be observed that hinge moment coefficient can be evaluated knowing the variation of C_H as a function of other angular values:

$$C_H = b_0 + \frac{\partial C_H}{\partial \alpha}\alpha + \frac{\partial C_H}{\partial \delta}\delta + \frac{\partial C_H}{\partial \delta_{tab}}\delta_{tab}$$

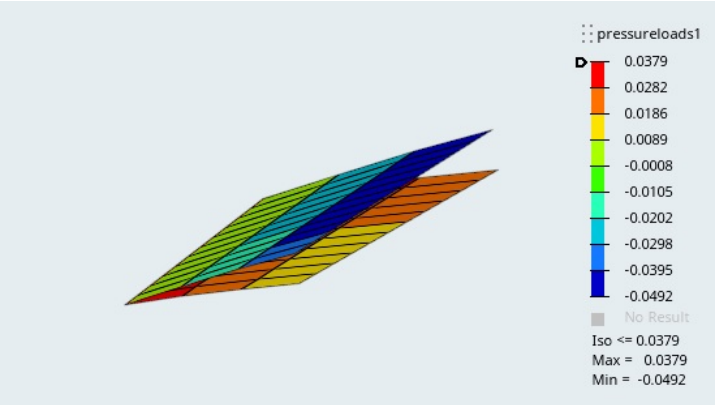


Figure 7.3: Pressure contour on HW ($v = 12\,[m/s], \alpha = 0[deg]$)

For a symmetric profile with a single tab, it can be said that:

$$C_H = \frac{\partial C_H}{\partial \alpha}\alpha + \frac{\partial C_H}{\partial \delta}\delta$$

Using software simulation, some sort of $\frac{\partial C_H}{\partial \alpha}$ and $\frac{\partial C_H}{\partial \delta}\delta$ trend may be estimated, or at least approximated using some interpolation feature. After all, it has be observed that hinge moment has a sort of linear behaviour with respect to α and δ (see sec. 6.2).
All of this means the possibility to estimate hinge moment with a different analysis tool, probably faster than a whole wing aerodynamic analysis: given two points and a function which connects them, it is possible to evaluate a third point between the first two. In other words, with a bunch of aerodynamic analysis, it is possible to estimate inputs and outputs of other configurations.
However, also this strategy is demanded to future studies. By now, all that can be said is:

- Process accuracy depends on the user and on the case study (it should be performed as a semi-empiric process);
- Ideally, this process should be able to double the size of a DLN dataset with small costs.

7.4 Experimental studies

The best way to produce a good forecasting model is, whenever possible, to acquire data using real tests. For example, a wind tunnel, combined with a proper angular position sensor on the control surface would be able to provide all data to build all DLN example in the way Xflr5 did for the present work. Obviously, there would be high costs behind this choice and one might prefer to adopt this strategy only at the end of the whole project, just to deploy a perfectly customized forecasting model.

7.5 GUI improvement

A graphical interface is a powerful tool to speed up the data handling development process. Different users may find the code tough to be red, but using the GUI, every analysis variation may be implemented to the DLN network without any problem. However, it has to be improved. For example:

- The code depends on the specific folders where the Xflr5 output files are placed in. This means that every reached string match will point to a series of separated sub-folders. This is necessary using the present aerodynamic tool because it allows to generate different file with a particular auto-updating string name which does not take into account flap configuration (δ) but just speed and incidence (v and α). This means, for instance, that the file corresponding to the configuration $[\delta, v, \alpha] = [-2, 12, 5]$ will have the same string name as the one corresponding to the configuration $[\delta, v, \alpha] = [4, 12, 5]$. This issue has been solved using three iteration loops which are used to point the specific δ folder and the specific file inside this folder. Considering this, some sort of folder selection tool inside the GUI would be useful for lots reasons, first of all to allow the user to select at least the main work path without entering the code;

- An even more useful tool would be the possibility to set a sample of the string to find directly inside the GUI. For instance, present output files have the following string name:

  *"MainWing_a=**(value)**_v=**(value)**ms"*

  The only varying parameters inside the string are α (incidence) and v (speed). A nice improvement would be the possibility to set the fixed part of the string inside the GUI itself. This operation, by now, is still performed inside the code and it is not user friendly;

- A less important improvement, at the end, would be to allow the user to see more information or previews like some sort of loading bar with an estimated remaining time information, or a preview of the final result.

Obviously, these improvements are not as important as the others, and easy to be implemented, so they may be neglected to focus on what has been said in previous sections of the present chapter.

8 CONCLUSION

The analysis performed so far has shown that machine learning can be applied to the present avionic problem with good results. Summarizing:

- Good precision may be reached in forecasting hinge moments using a quite small quantity of data (less than 3000 examples);
- IT computational costs are low, and the greatest part of time costs is about producing the samples (aerodynamic and, eventually, structural analysis or others);

As a procedure, the whole job has been made to easily adapt to changes, so each phase may be refined and improved by just paying attention to required outputs for the next step: indeed, present work could be considered the beginning of a bigger project, which has to be improved with other studies and experiments, in order to be able to represent, one day, a suitable solution for different scopes, such as:

- Automatic flight stabilization assistance for RC aircraft models;
- Design sizing of servo motors used to move control surfaces on a specific aircraft;
- A tool for fast preliminary analyses and checks;

However, a cutting edge technology hides by definition some disadvantages:

- A powerful network needs a great quantity of data, which means a great number of real experiments or accurate analyses;
- Considering a real situation, various nature effects take part to the generic phenomenon. This means that related DLN model should be fed with lots of input parameters to well describe the system;
- DLN based algorithms can handle just one problem at the time. It is not possible (yet) to realize models able to perform different tasks (multitasking) at the same time;
- Some effects like overfitting and batch size consequences are still under study.

By the way, "*neural networks are certainly not new, they have been taught at university for some time. What happened is that algorithms were written in order to make machine training operations more efficient. Simultaneously, GPUs and language libraries came out*" [Tre].
This means that DLN future is strictly in dependence with IT progress itself and may reach great performance in the future.

By now, for the present avionic application, it can be said that the state of the art of DLN technology is more than enough and that further developments will primarily focus on data acquisition criteria and on alternative input choices and experimentations.

A APPENDIX

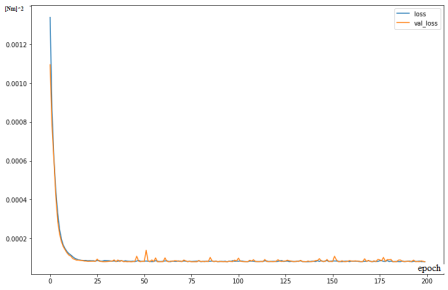
A.1 Configurations Summary

Model performance details and graphics are reported here. Type "$\mathcal{M}$" near configuration indication, means "batch size modification" for improvement.

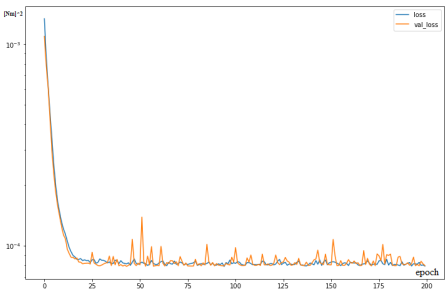
A.1.1 Configuration $[3, 3]$ - Mk1

	Test set	Validation set
MAE	0.006618908	0.004382582
RMSE	0.008927034	0.005339941
EVS	0.971682142	0.988075351
H **Mean Value**	$\hat{\mathrm{H}}_{df1} = -0.089814244$	$\hat{\mathrm{H}}_{df2} = -0.090131528$
$\mathbf{RE_{mean}}$		0.045615970
$\mathbf{\Delta RE_{max}}$		0.622115186

Table A.1: Mk1 Overall Prediction Performances

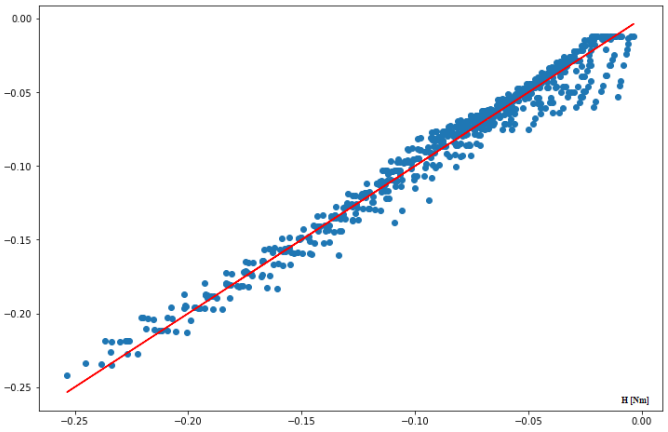


(a) Normal plot

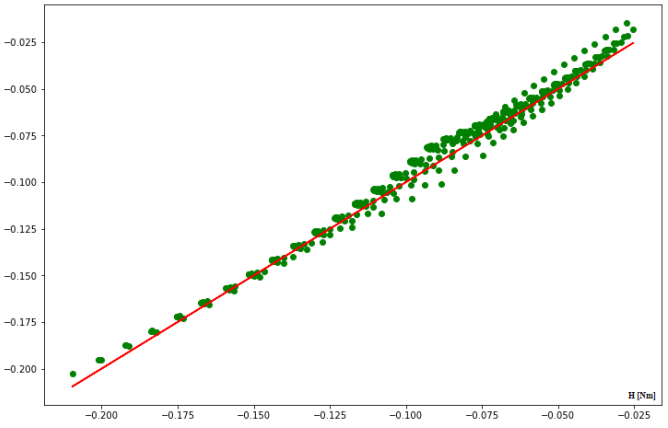


(b) Semilogarithmic plot

Figure A.1: Mk1 Training



(a) Test set prediction



(b) Validation set prediction

Figure A.2: Mk1 True values (red) VS Predictions

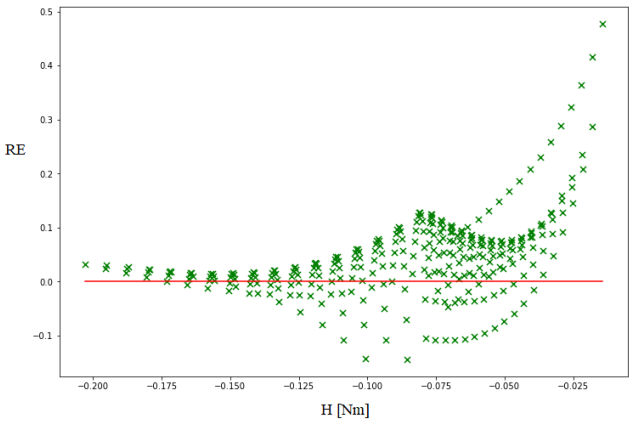
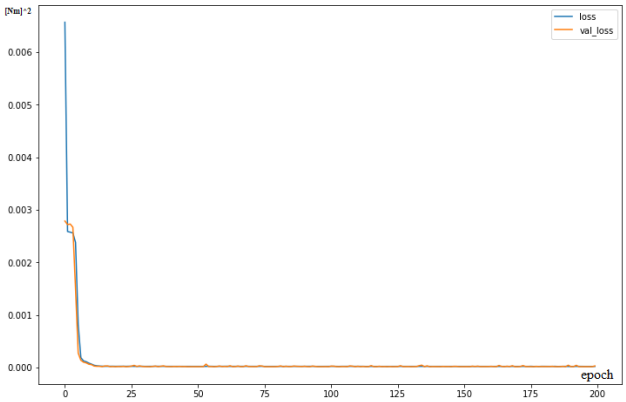


Figure A.3: Mk1 Relative Errors

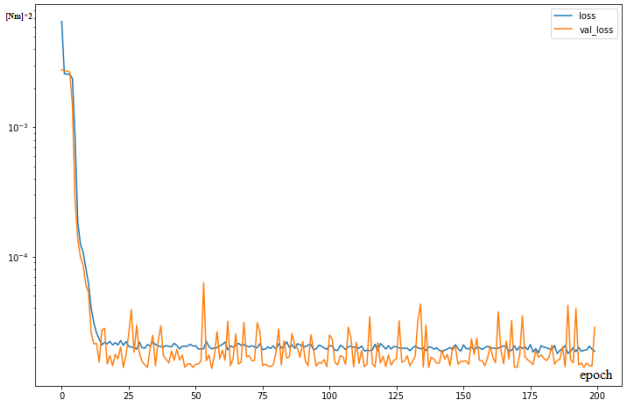
A.1.2 Configuration $\mathcal{M}[3,3]$ - Mk2

	Test set	Validation set
MAE	0.0043127210	0.003862890
RMSE	0.0053392147	0.004976332
EVS	0.9950987408	0.991969700
H Mean Value	$\hat{H}_{df1} = -0.089814244$	$\hat{H}_{df2} = -0.090131528$
$\mathbf{RE_{mean}}$		0.049982570
$\mathbf{\Delta RE_{max}}$		0.525476595

Table A.2: Mk2 Overall Prediction Performances

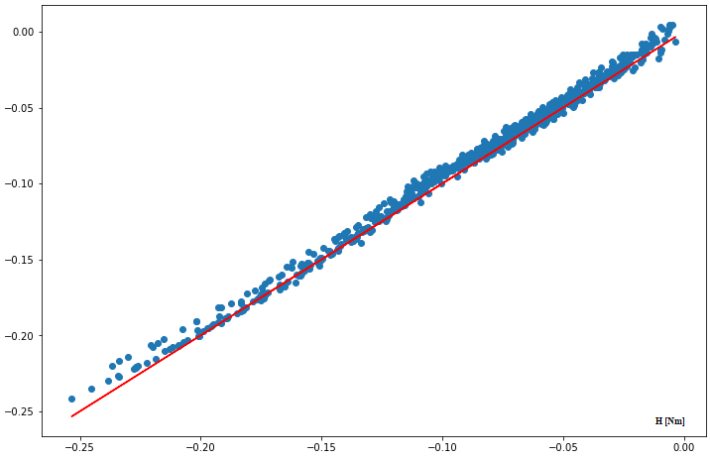


(a) Normal plot

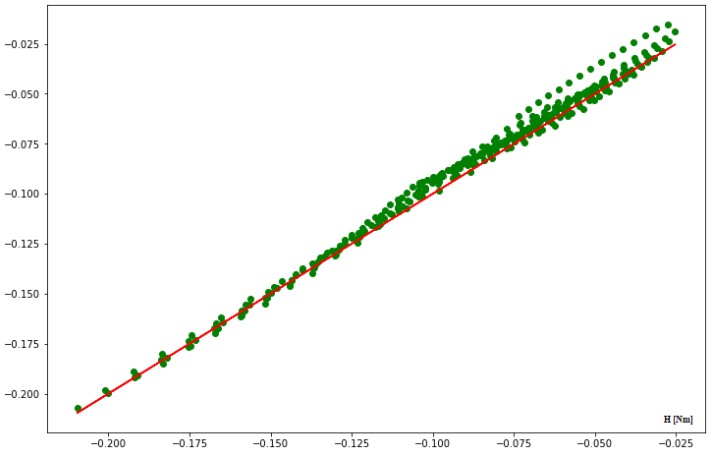


(b) Semilogarithmic plot

Figure A.4: Mk2 Training



(a) Test set prediction



(b) Validation set prediction

Figure A.5: Mk2 True values (red) VS Predictions

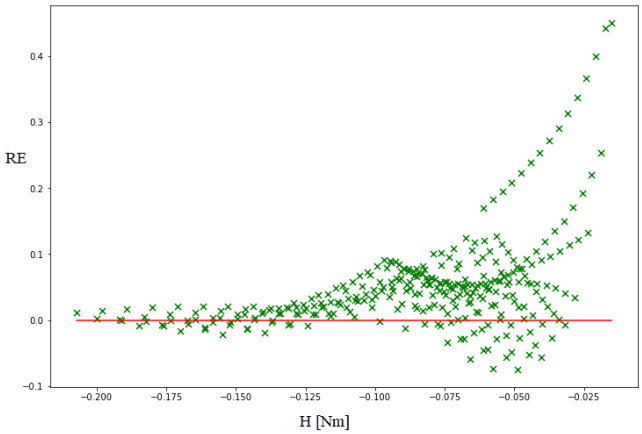
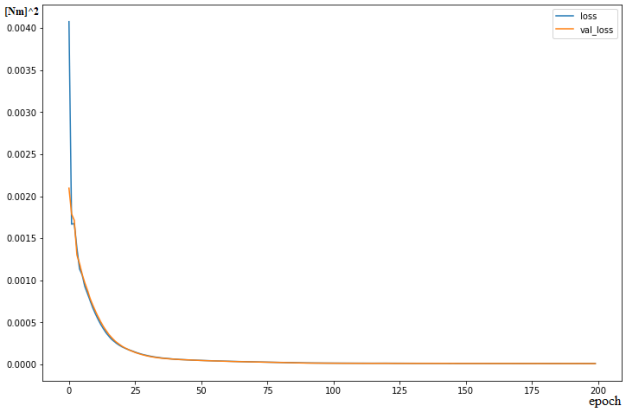


Figure A.6: Mk2 Relative Errors

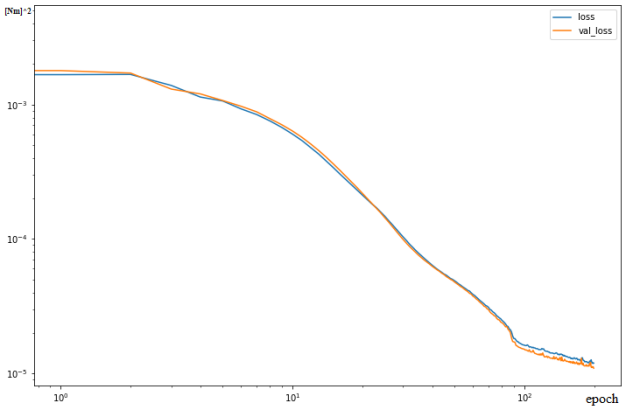
A.1.3 Configuration $[6, 4]$ - Mk3

	Test set	Validation set
MAE	0.0026338365	0.002714916
RMSE	0.0033046255	0.003814072
EVS	0.9961214854	0.991167269
H Mean Value	$\hat{H}_{df1} = -0.089814244$	$\hat{H}_{df2} = -0.090131528$
$\mathbf{RE_{mean}}$		0.004153418
$\mathbf{\Delta RE_{max}}$		0.375617169

Table A.3: Mk3 Overall Prediction Performances

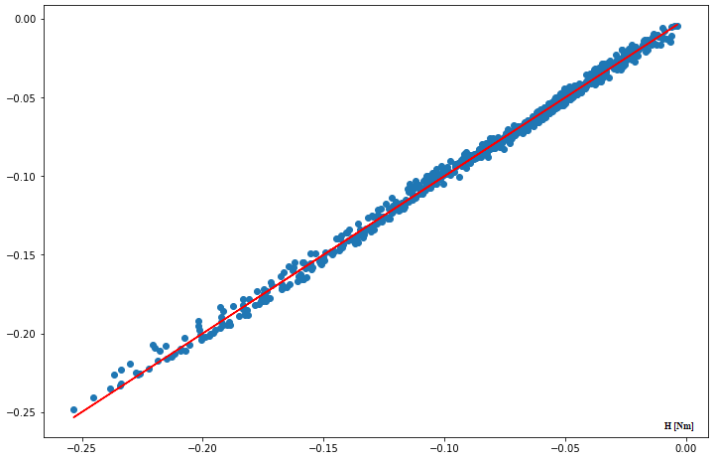


(a) Normal plot

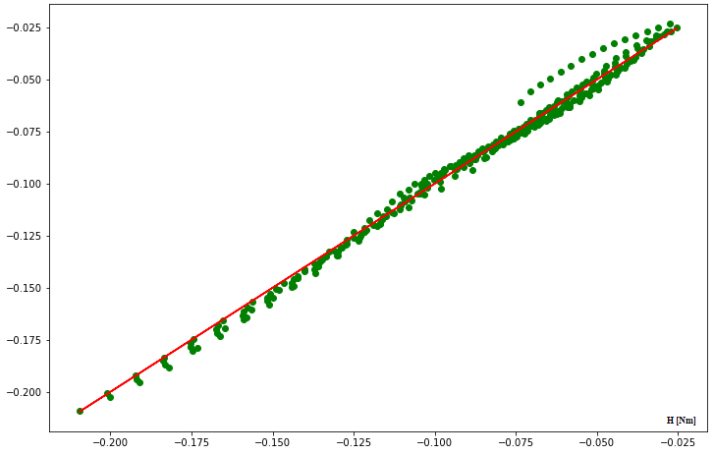


(b) Logarithmic plot

Figure A.7: Mk3 Training



(a) Test set prediction



(b) Validation set prediction

Figure A.8: Mk3 True values (red) VS Predictions

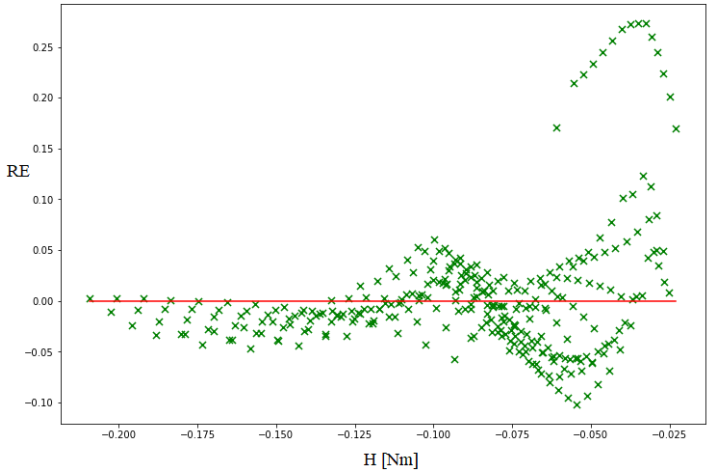
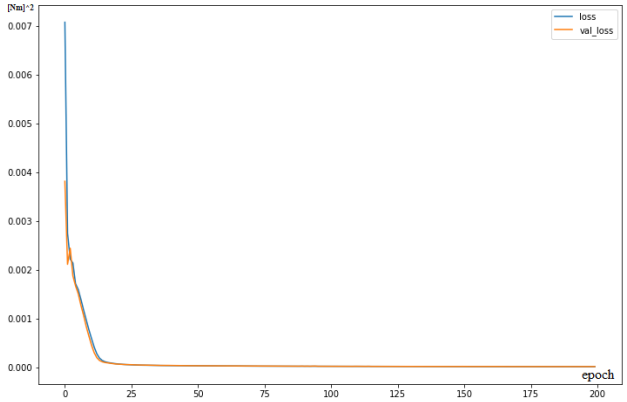


Figure A.9: Mk3 Relative Errors

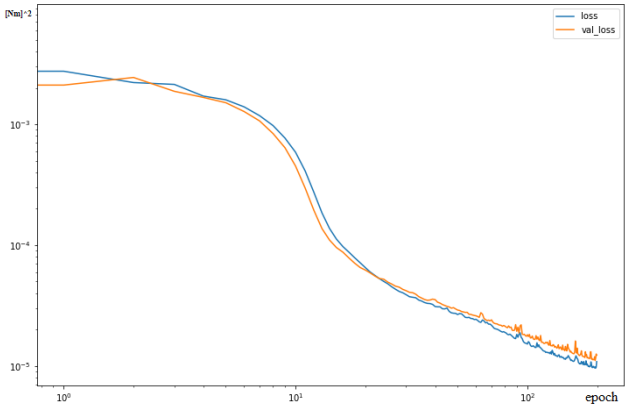
A.1.4 Configuration $[5, 10]$ - Mk4

	Test set	Validation set
MAE	0.0028237234	0.001966437
RMSE	0.0035050066	0.002522917
EVS	0.9960847812	0.996256053
H Mean Value	$\hat{H}_{df1} = -0.089814244$	$\hat{H}_{df2} = -0.090131528$
$\mathbf{RE_{mean}}$		-0.00391814
$\mathbf{\Delta RE_{max}}$		0.216450786

Table A.4: Mk4 Overall Prediction Performances

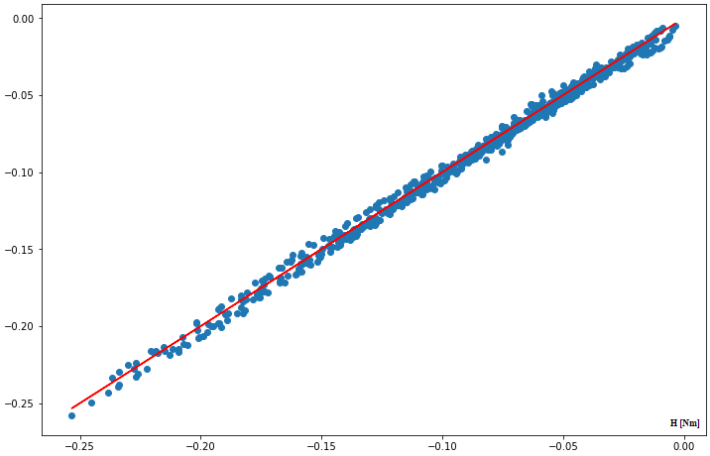


(a) Normal plot

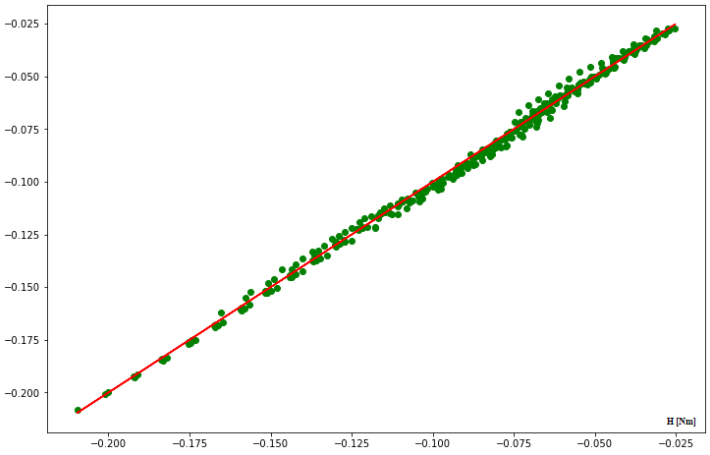


(b) Logarithmic plot

Figure A.10: Mk4 Training



(a) Test set prediction



(b) Validation set prediction

Figure A.11: Mk4 True values (red) VS Predictions

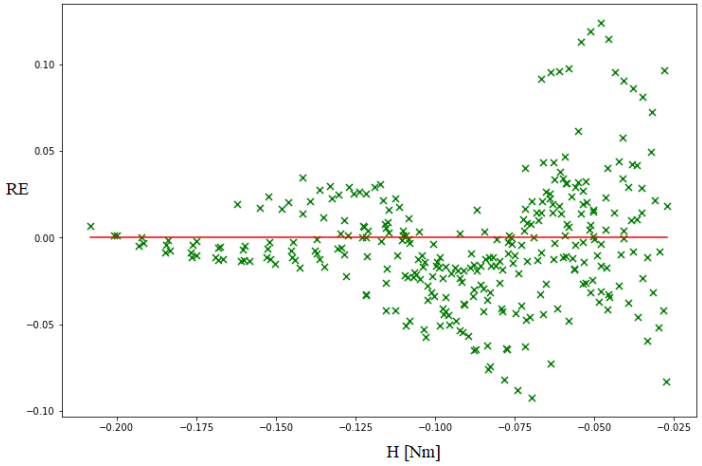
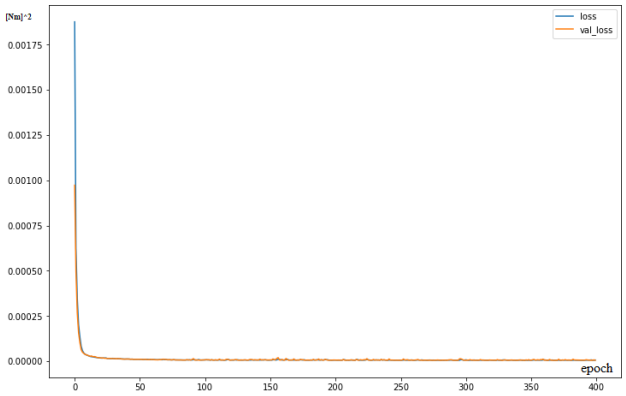


Figure A.12: Mk4 Relative Errors

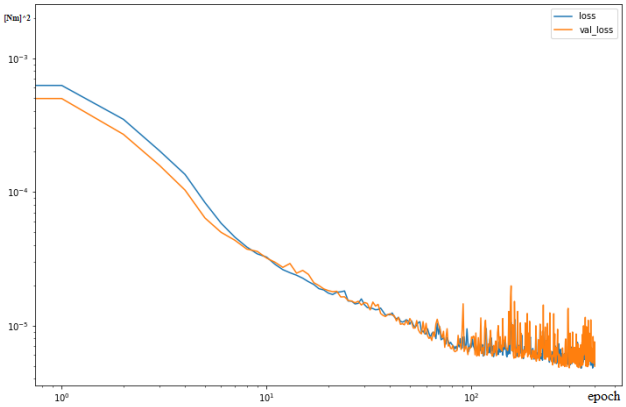
A.1.5 Configuration $\mathcal{M}[5, 10]$ - Mk5

	Test set	Validation set
MAE	0.001597852	0.001153364
RMSE	0.002226205	0.002050664
EVS	0.998240684	0.998442156
H Mean Value	$\hat{H}_{df1} = -0.089814244$	$\hat{H}_{df2} = -0.090131528$
$\mathbf{RE_{mean}}$		0.005320076
$\mathbf{\Delta RE_{max}}$		0.192899931

Table A.5: Mk5 Overall Prediction Performances

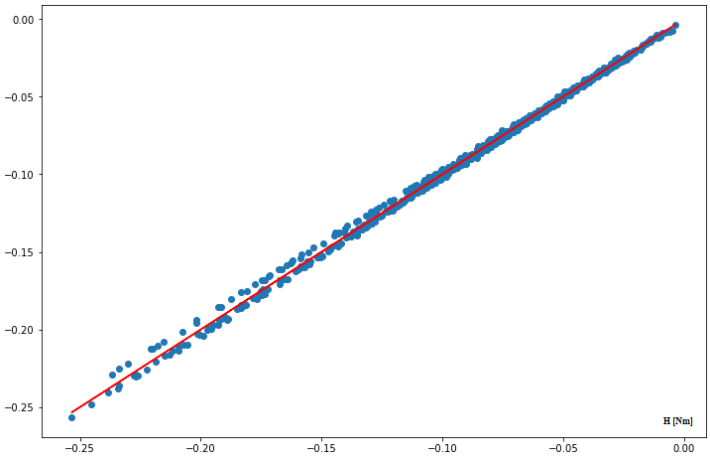


(a) Normal plot

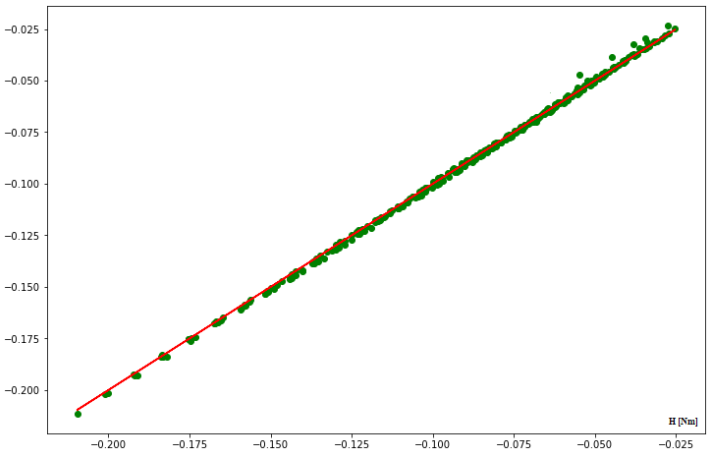


(b) Logarithmic plot

Figure A.13: Mk5 Training



(a) Test set prediction



(b) Validation set prediction

Figure A.14: Mk5 True values (red) VS Predictions

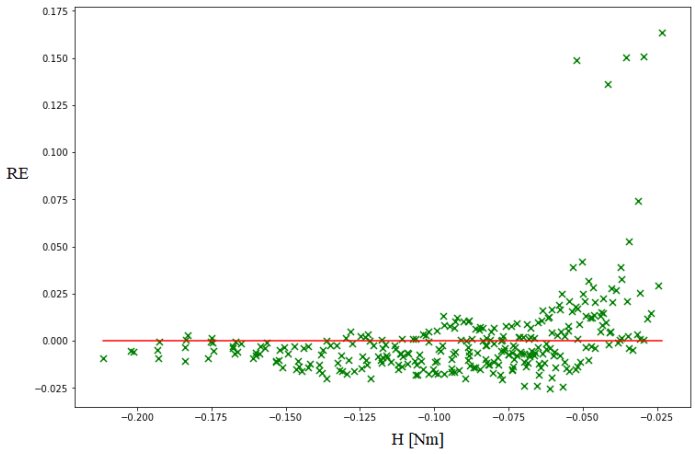


Figure A.15: Mk5 Relative Errors

B REFERENCES

Bibliography

[17] *"ON LARGE-BATCH TRAINING FOR DEEP LEARNING: GENERALIZATION GAP AND SHARP MINIMA"*. ICLR, 2017.

[Amio8] Donato Amitrano. *"Ricostruzione ed analisi di incidenti aerei. Il caso studio dell'aliante acrobatico Grob G103C Twin Astir Acro."* Napoli, Italy, 2007/2008.

[Caco6] Paolo Caccavale. *"Un moderno metodo a potenziale per analisi fluidodinamiche"*. Napoli, 2005/2006. URL: %7Bhttp://www.din.unina.it/tesi%5C%20dottorato/Un_Moderno_Metodo_a_Potenziale_per_Analisi_Fluidodinamiche.pdf%7D.

[Cas19] Pietro Casalone. *"Interazione fluido-struttura di hydrofoil in materiale composito"*. Torino, Italy, 2018/2019.

[Coro9] Cesare Corrado. *"Moti di fluido ideale irrotazionale: metodologie di soluzione"*. Padova, 2009. URL: %7Bhttp://www.image.unipd.it/s.lanzoni/teaching/PDF/Irrotational_Inviscid_Flows.pdf%7D.

[Joh84] Jr.. John D. Anderson. *"Fundamentals Of Aerodynamics"*. United States of America: McGraw-Hill, inc, 1984.

[Tea17] Icarus Team. *"Report Anubi"*. Torino, Italy, 2017.

Consulted web sites

[Bri20] Alessandro Bria. *Perché la funzione di attivazione ReLU è la più utilizzata nelle reti neurali?* [Online; in data 13-dicembre-2020]. 2020. URL: %7Bhttps://it.quora.com/Perch%5C%C3%5C%A9-la-funzione-di-attivazione-ReLU-%5C%C3%5C%A8-la-pi%5C%C3%5C%B9-utilizzata-nelle-reti-neurali%7D.

[Bru] traduzione di Samuele Bolotta Bruno Campello de Souza. *Qual è il QI medio dei premi Nobel?* [Online; in data 18-Marzo-2021]. URL: %7Bhttps://it.quora.com/Qual-%5C%C3%5C%A8-il-QI-medio-dei-premi-Nobel%7D.

[Gov] Lorenzo Govoni. *L'Overfitting e l'Underfitting nel machine learning*. [Online; in data 18-Marzo-2021]. URL: %7Bhttps://lorenzogovoni.com/overfitting-e-underfitting-machine-learning/%7D.

[Jas20] Jason Brownlee. *Why Do I Get Different Results Each Time in Machine Learning?* [Online; accessed 21-February-2021]. 2020. URL: %7Bhttps://machinelearningmastery.com/different-results-each-time-in-machine-learning/%7D.

[Jas21] Jason Brownlee. *Difference Between a Batch and an Epoch in a Neural Network*. [Online; accessed 19-February-2021]. 2021. URL: %7Bhttps://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/%7D.

[Jos] Jose Marcial Portilla. *Python for Data Science and Machine Learning Bootcamp*. [Online; accessed 07-March-2021]. URL: %7Bhttps://www.pieriandata.com/p/python-for-data-science-and-machine-learning-bootcamp%7D.

[Jup] Jupyter.org. *Jupyter home page*. [Online; in data 17-Febbraio-2021]. URL: %7Bhttps://jupyter.org/%7D.

[Met] Meteorologia.it. *Tabelle meteo ICAO*. [Online; in data 13-Febbraio-2021]. URL: %7Bhttps://www.meteorologia.it/Tabelle/index.htm%7D.

[Num] Numpy.org. *Numpy home page*. [Online; in data 18-Febbraio-2021]. URL: %7Bhttps://numpy.org/about/%7D.

[Som19] Alice Sommacal. *Lo stato dell'arte dell'IA. Cosa fa, ad oggi, l'intelligenza artificiale? Le principali applicazioni*. 2019. URL: %7Bhttps://www.unilab.eu/it/articoli/coffee-break-it/ia/%7D.

[Tre] Luca Tremolada. *Quello che il Deep learning non sa (ancora). Limiti e potenziale dell'Ia*. [Online; in data 18-Marzo-2021]. URL: %7Bhttps://st.ilsole24ore.com/art/tecnologie/2016-10-17/quello-che-deep-learning-non-sa-ancora-limiti-e-potenziale-dell-ia--130342.shtml?uuid=ADsLoEYB&refresh_ce=1%7D.

[Unso9] "Unspecified". *"Xflr5 analysis of foils and wings operating at low reynolds numbers"*. [Online; in data 13-Febbraio-2021]. 2009. URL: %7Bhttps://engineering.purdue.edu/~aerodyn/AAE333/FALL10/HOMEWORKS/HW13/XFLR5_v6.01_Beta_Win32%5C%282%5C%29/Release/Guidelines.pdf%7D.

[Web] XFLR5 Website. *About XFLR5 calculations and experimental measurements*. [Online; in data 11-Marzo-2021]. URL: %7Bhttp://www.xflr5.tech/docs/Results_vs_Prediction.pdf%7D.

[Web21] XFLR5 Website. *Theoretical Background (of Xflr5)*. [Online; in data 13-Febbraio-2021]. 2021. URL: %7Bhttp://www.xflr5.tech/xflr5.htm%7D.

[Wik19] Wikipedia. *Palomar Sky Survey — Wikipedia, L'enciclopedia libera*. [Online; in data 13-dicembre-2020]. 2019. URL: %7Bhttp://it.wikipedia.org/w/index.php?title=Palomar_Sky_Survey&oldid=104599434%7D.

[Wik20a] Wikipedia. *Apprendimento automatico — Wikipedia, L'enciclopedia libera*. [Online; in data 13-dicembre-2020]. 2020. URL: %7Bhttp://it.wikipedia.org/w/index.php?title=Apprendimento_automatico&oldid=116605164%7D.

[Wik20b] Wikipedia. *Intelligenza artificiale — Wikipedia, L'enciclopedia libera*. [Online; in data 13-dicembre-2020]. 2020. URL: %7Bhttp://it.wikipedia.org/w/index.php?title=Intelligenza_artificiale&oldid=117092823%7D.

[Wik20c] Wikipedia. *Rivoluzione digitale — Wikipedia, L'enciclopedia libera*. [Online; in data 13-dicembre-2020]. 2020. URL: %7Bhttp://it.wikipedia.org/w/index.php?title=Rivoluzione_digitale&oldid=116452888%7D.

[Wik21a] Wikipedia contributors. *Categorical variable — Wikipedia, The Free Encyclopedia*. [Online; accessed 16-February-2021]. 2021. URL: %7Bhttps://en.wikipedia.org/w/index.php?title=Categorical_variable&oldid=1001628444%7D.

[Wik21b] Wikipedia contributors. *Machine learning — Wikipedia, The Free Encyclopedia*. [Online; accessed 18-March-2021]. 2021. URL: %7Bhttps://en.wikipedia.org/w/index.php?title=Machine_learning&oldid=1012767353%7D.

[Wik21c] Wikipedia contributors. *Rectifier (neural networks) — Wikipedia, The Free Encyclopedia*. [Online; accessed 17-February-2021]. 2021. URL: %7Bhttps://en.wikipedia.org/w/index.php?title=Rectifier_(neural_networks)&oldid=1006601306%7D.

[Wik21d] Wikipedia contributors. *Step function — Wikipedia, The Free Encyclopedia*. [Online; accessed 23-March-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Step_function&oldid=1008876359.

[Wik21e] Wikipedia contributors. *Vanishing gradient problem — Wikipedia, The Free Encyclopedia*. [Online; accessed 19-February-2021]. 2021. URL: %7Bhttps://en.wikipedia.org/w/index.php?title=Vanishing_gradient_problem&oldid=1006243644%7D.