FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Master's Degree in Informatics Engineering
Dissertation

Aeroelastic Optimization of a High Aspect Ratio Wing

September 4, 2013

Ivo Daniel Venhuizen Correia
icorreia@student.dei.uc.pt

Advisor:
Professor Carlos M. Fonseca
cmfonsec@dei.uc.pt

Abstract

Two fundamental aspects of aircraft wings are considered in this work: their aerodynamics and their structural properties. Although originating in different disciplines, these two aspects should be studied together because structural behaviour influences aerodynamics and vice-versa, leading to what is known as the aeroelastic behaviour of the wing.

With respect to aerodynamics, lift and drag are the forces that allow the airplane to take off and sustain itself in the air. Although the absence of drag would make it impossible for the airfoil to fly, engineers generally seek to minimize it, as an increase in drag implies an overhead on aircraft maneuvers and greater fuel consumption. Drag is minimized by long elliptical wings, but such wings are difficult to manufacture in comparison to other shapes. Long wings are generally more aerodynamic, but a longer span naturally implies greater structural weight, thus reducing flight range. The range formulas of Breguet relate the lift and drag produced by the wing, the amount of fuel available and the weight of the aircraft to the maximum distance the aircraft can fly.

Aeroelasticity, on the other hand, is concerned with the fact that when in flight, the wing structure is under the influence of several forces that deform its original shape. Understanding these forces and how they change the aerodynamic behaviour of the airfoil is very important. In particular, such forces and the corresponding deformation may create a positive feedback loop, and the wing may bend so much that it breaks.

Accurately modelling the aeroelastic behaviour of a given wing may be computationally very demanding. Therefore, less accurate but simpler models are used for optimization purposes at the initial design stages. Such models must, nevertheless, remain valid to a certain extent, in order for optimized preliminary designs be useful at a later stage.

In this work, the integration between wing aeroelastic models and optimizers is considered, with a view to allowing more accurate and more computationally demanding wing models to be used for optimization. This was accomplished through two different approaches. In the first approach, the precision at certain intermediate steps was reduced without affecting the output. More specifically, Gauss-Seidel iterations were used to achieve faster but less precise solutions for systems of linear equations arising in given model. In the

second approach, a partial set of data was reused from one iteration to the next, reducing running time but still preserving the precision of the original method.

Although the models studied are not the most suited for incrementalization, it is shown that it is possible to reduce computation time without affecting model validity or the optimization results.

Keywords: Diederich's method, vortex lattice method, induced drag minimization, numerical optimization, wing modelling

Acknowledgements

I would like to thank Marc Mulkens for providing important bibliographic references about aeronautics, validation data and answering all my questions. Also, to my advisor Carlos M. Fonseca, for all the patience and support given during this whole year.

Finally, to all my companions from ECOS lab, family (especially parents and brother) and all other friends for helping me getting this far.

Coimbra, 4th September 2013

Ivo Correia

Contents

List of Tables

List of Figures

List of Acronyms

AoA Angle of attack

AD Automatic differentiation

DSM Direct search methods

GPGPU General-purpose graphics processing unit

GPU Graphics processing unit

GS Gauss-Seidel

LLT Lifting-line theory

VLM Vortex lattice method

1 Introduction

1.1 Motivation

Building a new aircraft wing requires much time and effort, and hence, it is crucial to optimize every single step of the process. As it is costly to perform experiments on real models, it is now common practice to do the design on computer simulators, which will approximate the considered wing shape close to its final form. Although computational power is expanding, it is also true that the complexity of the simulators is evolving at a fast pace and a single simulation run may take several days to finish.

Furthermore, simulation may be accompanied by an optimization process. There are then two distinct entities: the simulator, which holds a model and, given an input, will return useful values for the analysis (e.g., the drag and lift produced by the wing); and the optimizer, which will, given a set of parameters concerning the wing geometry, systematically perturb their values until the results are sufficiently close to optimal. An iterative process is then established, where the optimizer provides a set of parameters to the simulator, and the simulator returns the results back to the optimizer, which will decide whether and how to continue or to return a final answer.

This process is often seen as a communication between two completely separate objects, as the optimizer has no view of the simulator's internals and vice-versa. This means that different simulations are performed mostly independently and, therefore, little if anything is reused from previous compu-

tations.

The idea for this project was established after contacts with Embraer S.A., a Brazilian aerospace enterprise. Having Marc Mulkens as the main contact between the parties, the first models were implemented from the documentation and information provided by the company. Although code developed internally could not be made available, pseudo-code, documentation and answers to theoretical questions were all given. In a second phase, when looking for more computationally demanding models, guidance towards the best choices was offered as well.

1.2 Goals

The main goal of this work is to reduce the separation between the optimizer and the simulator in a simulation-optimization loop, so that the work performed by each component can be of use to the other. This is done by monitoring how data changes from one iteration to the next, so as to avoid performing the same calculations over and over again.

Even when the whole input data changes, different algorithms to address the same problem may be considered. This is the case of finding a solution for a system of linear equations, where full Gauss elimination may be replaced by decomposition techniques with lower complexity, when the solution to the original system is known. On the other, methods outputting less accurate solutions may, due to the less precision, run faster and still produce satisfactory results.

1.3 Contributions

Although starting from very simple models, this document presents a promising approach to speeding up simulation optimization, which may be useful for more advanced applications in order to reduce computing time.

The contributions include the adaptation of the original models to admit less accurate intermediate steps, namely with precision control in the Gauss-Seidel method. Precision needed to be sufficiently low to reduce computation time and yet, sufficiently high not to affect the output.

A way of incrementalizing the simulation process is described, although

its impact is not as high as initially expected. This was accomplished mainly through data reuse, including starting points of inner loops.

Finally, the document proposes a number of ideas that even though they were not applied to the considered models, may eventually be useful in connection with other approaches to wing modelling.

1.4 Document structure

This document is composed of three parts. In the first part, background on wing modelling, numerical optimization and systems of linear equations solving is provided. It encompasses Chapters 2, 3 and 4, respectively.

The main ideas concerning simulation and the implementation are discussed in Chapter 5. This Chapter defines the optimization design space, the specifications of the machines used for testing and provides a discussion of the ideas suggested to achieve greater speed-ups.

Finally, the analysis of the results, including model validation, and concluding remarks are presented in the last Chapters, 6 and 7. The document ends with the Nomenclature and the Bibliography.

2

Wing Modelling and Analysis

2.1 Aeronautical optimization

Optimizers and model simulators in aeronautics have naturally accompanied the evolution of airplane design. The methods of analysis evolved with the need for larger, faster and more aerodynamic aircraft, capable of carrying more weight and flying at higher speeds. Creating airfoils for new flight conditions was often driven by failures of the existing ones, with reports from the early days describing plane crashes due to flutter, as aircraft crossed their speed boundaries and their wings bent widely and broke [1].

To prevent such cases, aeronautical engineers started developing more and more detailed models as the underlying physics became better and better understood. This increase in model complexity was the entry point for computational models and optimizers, as they allowed preliminary wing designs to be produced that came closer to their final physical form, saving both time and resources in real experiments.

The very first models appeared with thin-airfoil theory, where a zero thickness airfoil with infinite wingspan was assumed. They relate the angle of attack to the lift produced by the wing, without taking into account the critical angle (see Section 2.2.1.1). Then, in the 1970s, linear models, such as panel methods, were developed. These methods divide the airfoil space into panels, which are usually flat and quadrilateral shaped. The lift distribution is calculated and, by combining the results of all the panels, a final value for the lift coefficient

is returned [2].

However, these methods only consider subsonic flights. Boundary-layer corrections were made during the 1980s, but still, the barrier of transonic flights was not crossed. Supersonic and hypersonic modelling could only be successfully achieved when fluid dynamics was taken into account. The Euler equations were firstly used, defining a set of equations that can model the behaviour of moving and inviscid flows, with conservation of mass, momentum and energy. Later, they were generalized by the Navier-Stokes equations, which no longer required that fluid viscosity be ignored.

In the 1990s, mesh-based models were introduced, which are used till the present day, and, most recently, time-dependent flow models [3]. In mesh-based model optimization, meshes can be structured (regular pattern) or unstructured (irregular patternas with Delaunay triangulations, for example). In general, these methods use a set of points, which, when properly linked, will form a surface. The mesh has to obery to a set of rules and restrictions, leading to mesh restructuring by changing the position of specific points during the optimization process.

2.2 Aerodynamic modelling

2.2.1 Concepts and notation

The fundamental concepts arising in aerodynamc modelling will be introduced next, in order to support a full understanding of every algorithm discussed here. Alongside the concepts, the notation used in this document is also defined. For a full listing of the variables here described, refer to the Nomenclature present at the end of this document.

2.2.1.1 Wing geometry

The **wingspan**, b, is the distance from one wingtip to the other. From the wingspan, y can be define, ranging from -$\frac{b}{2}$ to $\frac{b}{2}$, with the zero value at the fuselage line. The tips, denoted by the index t, are located at $y = \pm\frac{b}{2}$, while the root, denoted by the index r, is located at $y = 0$.

The **leading edge** is the set of points at the front of the airfoil that has maximum camber curvature. The **trailing edge** is defined similarly as the set

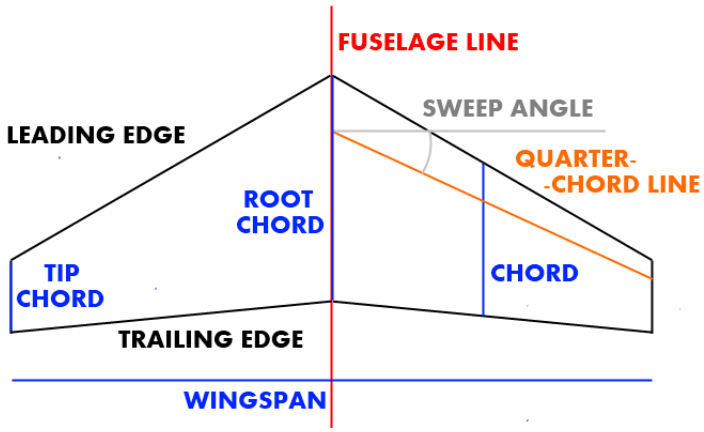

Figure 2.1: Wing top view.

of points of maximum camber curvature at the rear of the airfoil.

The **chord** c is a straight line joining the trailing and leading edge, parallel to the fuselage line. Taking chord length as a function of the position y along the wing leads to the chord-length distribution $c(y)$. As can be seen in Figure 2.3, the chord line does not necessarily need to be inside the wing profile. The **quarter-chord line** is the line which joins every point located at the quarter (from the leading edge) of every chord.

The **sweep angle** $\Lambda_{.25}$ (or simply Λ) is the angle defined by the quarter-chord and the vertical plane perpendicular to the fuselage. Although not usually considered, two more sweep angles may be defined. They are Λ_{TE} and Λ_{LE}, using respectively the trailing and the leading edge instead of the quarter-chord line.

The **dihedral angle** ϕ is the angle between the wing and the horizontal plane passing through the fuselage line. The **wing taper ratio** λ is the ratio between the lengths of the chord at the tip and the at the root:

(2.1)
$$\lambda = \frac{c_t}{c_r}$$

The **planform** (or wing area) S is the total area of the wing projected on the horizontal plane. The **aspect ratio** AR is then the square of the wingspan divided by the planform. They are defined as:

(2.2)
$$\begin{aligned} S &= 2\int_0^{\frac{b}{2}} c'(y)\,dy \\ AR &= \frac{b^2}{S} \end{aligned}$$

where $c'(y)$ is the distribution of chord-lengths projected in the horizontal plane.

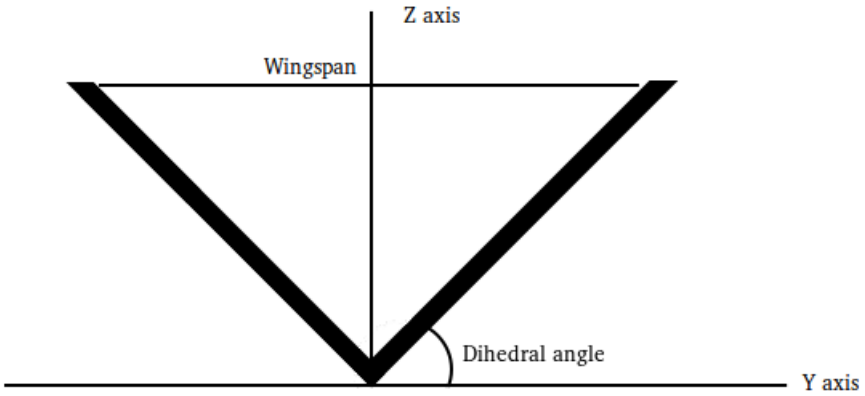

Figure 2.2: Wing side view.

The **angle of attack** α is the angle formed by the chord line of the wing and the direction of the air flow, while the **downwash angle** is the angle between the direction of air movement as it approaches the wing and as it leaves it.

The **camber** is an asymmetry between the top and the bottom of the wing. A cambered (or asymmetric) airfoil appears as opposed to a symmetric airfoil, where the two parts are shaped symmetrically. The **thickness** is the distance between the top and the bottom surface, defining a thickness distribution along the chord (see Figure 2.3).

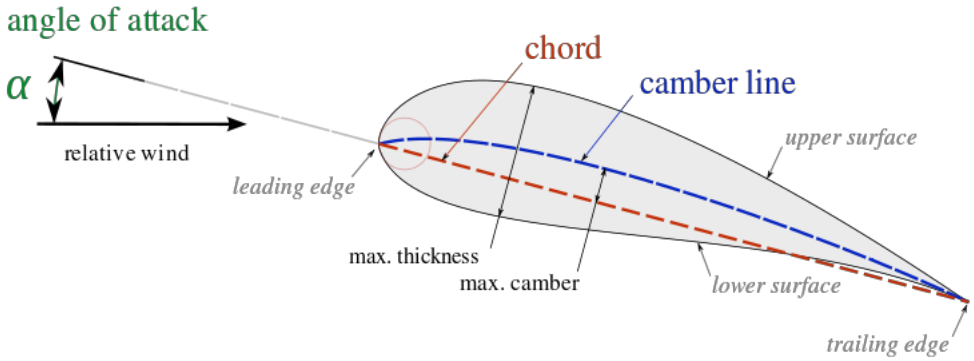

Figure 2.3: Wing profile.
(from http://en.wikipedia.org/wiki/Airfoil)

The **twist** ϵ is the difference between the angle of attack at given wing section and the root section. When the leading edge points upwards, we have a positive twist, named **washin**. When it points down, the most common situation, the twist is negative and is denominated **washout**.

Twist can be further split into geometric and aerodynamic twist. The **geometric twist** is the twist of the chord line with respect to the root's chord line. **Aerodynamic twist** is the angle formed by the corresponding zero-lift lines instead of the chord lines. The **zero-lift line** is influenced by the wing's geometry and thickness, and is the position of the given chord when the angle of attack is such that the wing section will produce no lift.

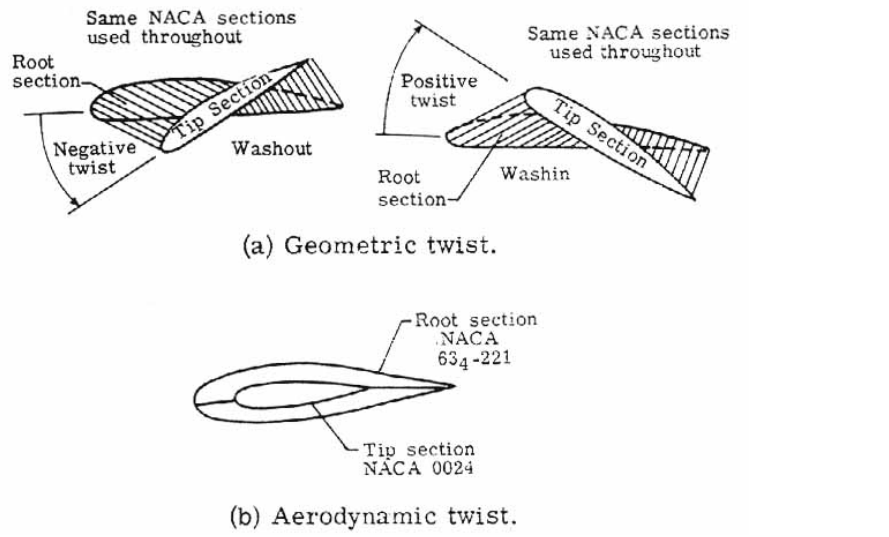

Figure 2.4: Geometric and aerodynamic twist.

2.2.1.2 Mach number

The Mach number is a dimensionless quantity that represents the speed of an object moving through a fluid, normalized by the local speed of sound:

$$M = \frac{V}{a} \tag{2.3}$$

where M is the Mach number, V is the speed of the source relatively to the medium and a is the speed of sound in the medium. Throughout this document, the Gallilean transformation from the ground-fixed-reference to the vehicle-fixed-reference will be applied more often. This means that the wing velocity $\vec{V}$ is null, while the fluid is considered to move with a velocity $\vec{U_\infty}$.

Regime	*Subsonic*	*Transonic*	*Sonic*	*Supersonic*	*Hypersonic*	*High-hypersonic*
Mach	< 1.0	0.8 - 1.2	1.0	1.0 - 5.0	5.0-10.0	> 10.0

Table 2.1: The six different flight categories.

Aerodynamically speaking, six different flight categories may be defined, depending on the Mach speed of the airplane. The categories do not overlap, with exception of the transonic regime. It defines a transition region between subsonic and supersonic speeds. In fact, during the transonic period, a wing will experience different speeds along its structure, some defined under (subsonic), at (sonic) and above (supersonic) the speed of sound.

2.2.1.3 Aerodynamic forces

Four different forces acting on the wing may be considered, namely thrust, weight, drag and lift. The last two are called aerodynamic forces. Generally speaking, we can say that the **lift**, L, is the component of the aerodynamic force perpendicular to the relative wind velocity; the **drag**, D, is the component parallel to the same relative velocity and the **weight** is the gravity force, pointing towards the center of the earth. Finally, the **thrust** is the force produced by the airplane engines.

2.2.1.4 Lift

Lift is the component of the total aerodynamic force that allows the airplane to elevate itself. In other words, it is the force exerted on the wing surface by the flowing air, perpendicular to this same flow. Lift can be increased either by increasing the aircraft's forward speed or by the angle of attack, though in the last case, when the critical angle is reached, a phenomenon known as stalling causes lift to decrease.

Lift can, in a simple way, be explained by the creation of different pressures above and under the wing. By Bernoulli's principle, an increase of speed is translated into a decrease of pressure, the opposite statement being correct too. This means that on the upper side of the wing, where the air flows more rapidly, the pressure is lower. The difference between particle speed is explained by the Kutta-Joukowsky theorem [2].

2.2.1.5 Kutta-Joukowski theorem and circulation

Circulation defines the line integral of the fluid velocity around any closed curve C [24]. The mathematical form is then:

$$\Gamma = \oint_C \vec{V}.\vec{dl} \tag{2.4}$$

where Γ is the circulation, $\vec{V}$ is the velocity and $\vec{dl}$ is the length of the element at which the velocity is considered.

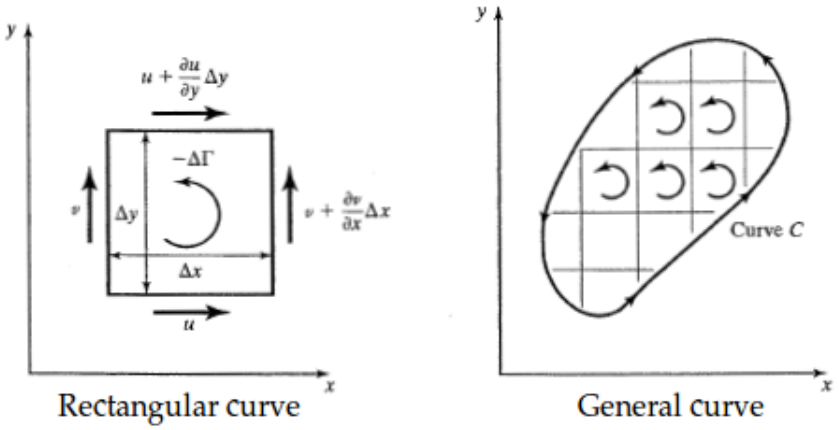

Figure 2.5: Circulation for elementary closed curves.

The Kutta-Joukowski theorem can be enumerated in three topics:

1. The force applied by a two-dimensional flow around a airfoil is perpendicular to the flow direction, the induced drag force being null.

2. The force depends directly on the circulation around the airfoil section.

3. The direction of the force is obtained rotating the fluid velocity vector, with a value of $\frac{\pi}{2}$ and in the direction opposed to the circulation.

The value of circulation is then used to calculate the lift generated by the airfoil. By the Kutta-Joukowski theorem, the lift is calculated by:

$$L = -\rho U_\infty \Gamma \tag{2.5}$$

where L is the lift, ρ is the fluid density and Γ is the circulation. The lift value is positive due to the fact that by convention, the value of constructive circulation is negative [2].

A **potential vortex** (or simply vortex) is defined as a singularity about which the fluid flows with concentric circular streamlines, perpendicularly to the wing surface [24] (see Figure 2.6).

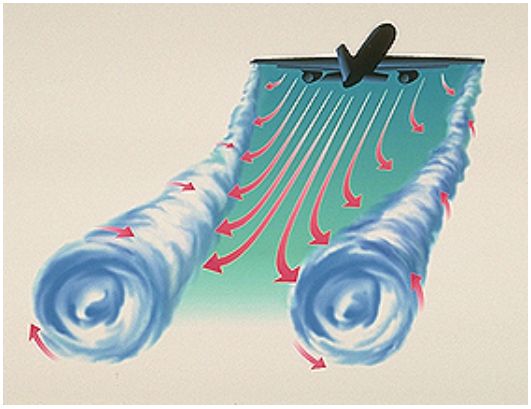

Figure 2.6: Horseshoe vortex representation.
(from http://www.flyingstart.ca)

Associated with the circulation are the **horseshoe vortices**, used for example in the vortex latticed method (see Section 2.2.4). An horseshoe vortex is composed by a wing bounded vortex, with constant circulation and denoted as the control point, and two trailing vortices, located on the wingtips or even outside the wing surface. The circulation will then be the value of the overall vortex strength.

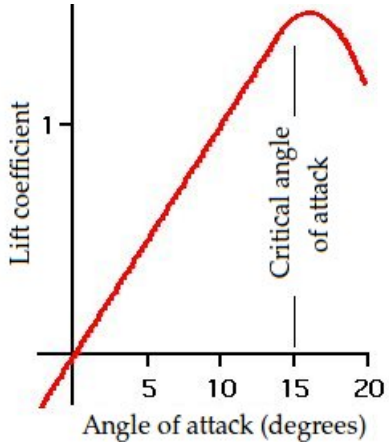

Figure 2.7: Relation between angle of attack and lift.

The angle of attack is of extreme importance, both to the airplane's speed

and to the lift produced by the airfoil. In fact, as shown in Figure 2.7, an increase on the angle of attack will be accompanied by an increase in lift, though it will reach a point where this starts producing the opposite effect. This point is the critical angle beyond which the aircraft will experience stalling and the consequent reduction of lift.

2.2.1.6 Drag

Drag is the set of forces that act on a solid object in the direction of the fluid flow and are dependent of its velocity, usually resulting in a loss of energy by the object. Hence, the airfoil should be built in order to minimize drag. The following equation translates the sum of the forces:

$$D = \frac{1}{2}\rho V^2 C_D S \tag{2.6}$$

where D is the drag, ρ is the density of the fluid, V the speed of the object relative to the fluid, C_D the drag coefficient and S is the wing area.

Drag can be divided into three main categories: parasitic drag (related to the size, shape and properties of the material), wave drag (resistance created by shock waves, which radiate energy out of the airplane) and lift-induced drag (or just induced drag). In this project, only lift-induced drag D_i will be considered, as the other forms of drag are used only in more advanced modelling stages.

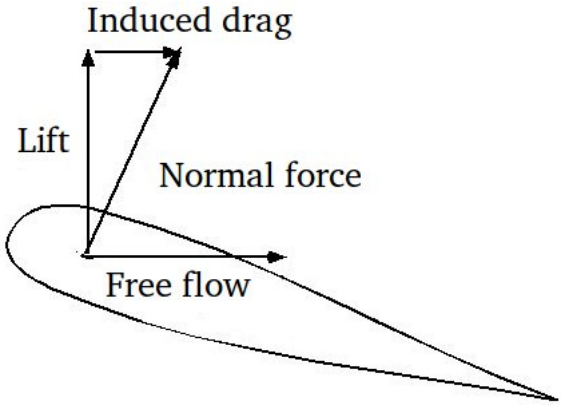

Figure 2.8: Induced drag on a non-zero angle of attack.

The induced drag occurs when the air flow must change its direction due to the pressure of the wing, usually in order to produce lift. As can be seen in

Figure 2.8, whenever the chord line is not aligned with the air flow, the normal force produced by the wing will not be perpendicular to the air flow. In those cases, the vertical component of the normal force will produce lift, while the horizontal component will contribute to the increase of the drag and hence the name, lift-induced drag. Note that when the downwash angle is zero, no lift is produced and consequently, induced drag is also zero.

2.2.2 Lifting-line theory

The lifting-line theory (LLT) was the first model studied and implemented in this work, a simple model with several restrictions. One of the first considerations to make is that, although the lift distribution is in fact 3-dimensional, it can be satisfactorily approximated by a 2-dimensional view. Because of this assumption, this model does not consider the thickness distribution, assuming instead a flat airfoil. It does not support flexible or swept wings (with sweep angle other than zero) either. Nevertheless, it can still be used to model tapered (trapezoid-shaped instead of rectangular) and twisted wings. All it needs as input are the following parameters:

1. Wing span;
2. Spanwise distribution of the following quantities:
   (a) Sectional profile or airfoil chord length;
   (b) Airfoil geometric angle of attack;
   (c) Airfoil zero-lift angle of attack;
   (d) Airfoil lift curve slope.

As previously referred, with increased angle of attack, lift increases in a roughly linear relation (see Figure 2.7), and the slope of the lift curve corresponds to the last item of the input. This parameter replaces some of the wing geometric characteristics not explicity defined in the model formulation, such as the wing profile.

In order to estimate the unknown spanwise lift distribution, sectionally divided by N points, LLT uses a Fourier series to approximate the vortex

strength distribution. As seen in Section 2.2.1, the Kutta-Joukowsky lift theorem states that the overall lift is directly proportional to the value of the vortex strength, also known as circulation.

Assuming that the wings are symmetric and so is the lift distribution, even components of the distribution will all be zero. The circulation distribution is then approximated as:

$$\Gamma(\theta) = 2bV \sum_{m=1}^{\frac{N}{2}} A_{2m-1} \sin((2m-1)\theta) \tag{2.7}$$

The lifting-line equation 2.8 that needs to be solved is:

$$\begin{aligned} D(k) &= \sum_{m=1}^{\frac{N}{2}} C(k, 2m-1) A_{2m-1}, \qquad k = 1, ..., M \\ C(y, n) &= \sum_{n=1}^{N} \left(\frac{4b}{a(y)c(y)} + \frac{n}{\sin(n\theta(y))} \right) \\ D(y) &= \alpha - \alpha_0(y) + \epsilon(y) \end{aligned} \tag{2.8}$$

where α is the angle of attack, $\alpha_0(y)$ is the zero-lift angle distribution (usually assumed to be linear, from the tip and root), $\epsilon(y)$ is the twist angle distribution (again, may be calculated linearly) and A_m represents the vector with the Fourier series coefficients.

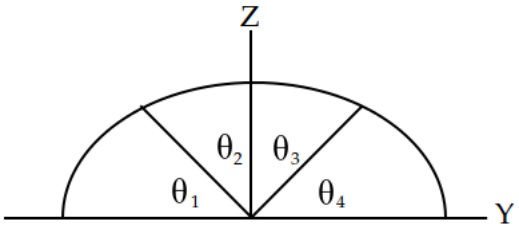

Figure 2.9: The lift distribution division over θ.

In these formulas, we also find θ, which is the angular component of the lift distribution, starting and ending at the wing tips (see Figure 2.9). Given that these angles must be evenly spaced over the wing, we have that $\theta(k)$ is obtained by the formula:

$$\theta(k) = \frac{(2k+1)\pi}{2N}, \qquad k = 1, ..., \frac{N}{2} \tag{2.9}$$

where N is the number of control points defined in the input. As the distribution is symmetric, the results can be mirrored for the other half wingspan. These angles are translated into the wing span coordinate using:

$$y(k) = \frac{b\cos(\theta(k))}{2} \tag{2.10}$$

where $y(k)$ is consequently a set of control points, located along half of the span. LLT will calculate lift coefficient values for those points, which will ultimately approximate the final lift distribution. These formulas insure that no control point is located at either the tip or the midpoint, as these positions provide no new information regarding the values of the Fourier coefficients.

Once the lift curve slope distribution is (linearly) computed from the input, as well as the chord distribution, from:

$$c(k) = c_r - \frac{2y(k)(c_t - c_r)}{b} \tag{2.11}$$

$C(k, m)$ can be calculated. Then, it is only a matter of solving the system of equations described by equation 2.8, which will return the Fourier series coefficients. The wing's lift coefficient is:

$$C_L = \pi AR.A_0 \tag{2.12}$$

The Oswald efficiency factor is defined as:

$$e = \frac{1}{1+\delta} \tag{2.13}$$

where:

$$\delta = \sum_{m=1}^{M}(2m-1)\left(\frac{A_m}{A_0}\right)^2 \tag{2.14}$$

And finally the induced drag coefficient:

$$C_{Di} = \frac{C_L^2}{\pi AR.e} \tag{2.15}$$

A more detailed explanation can be found in [4].

2.2.3 Diederich's method

The overall idea of Diederich's method is to divide the final lift distribution into two components, the basic and the additional lift distributions. According

to Diederich, the basic distribution for a certain twist angle is defined as the lift distribution for a given wing with the angle of attack reduced equally at every point until the total lift is zero. The additional lift distribution is defined as the distribution which the wing would carry if it were untwisted and the lift coefficient were equal to 1 [5].

The aeroelastic correction extends Diederich's method in order to consider the bending moment as the airplane shifts through the air. It basically uses the overall Diederich's method, but iteratively corrects the angle of attack distribution, repeating the basic lift calculation until it eventually (and hopefully) converges (see Section 2.4).

It also takes a larger number of input parameters when comparing to LLT. They are:

Aerodynamic inputs:

1. $\boldsymbol{M}$: Mach number;
2. $\boldsymbol{h}$: Cruising height;
3. $\boldsymbol{N_c}$: Load factor;
4. $\boldsymbol{C_L}$: Wing's lift coefficient.
5. $\boldsymbol{C_{D_0}}$: Markup value for the drag coefficient.
6. Wing geometry:
   (a) $\boldsymbol{AR}$: Aspect ratio;
   (b) $\boldsymbol{S}$: Wing area;
   (c) $\boldsymbol{\lambda}$: Taper ratio;
   (d) $\boldsymbol{\Lambda_{.25}}$: Sweep angle at the quarter-line.
7. $\boldsymbol{c_{l_\alpha}}$: 2D lift curve slope;
8. $\boldsymbol{c_{l_{max}}}$: Critical lift coefficient;
9. $\boldsymbol{\gamma}$: Air C_p/C_v ratio (specific heat capacity);
10. $\boldsymbol{\epsilon_t}$: Twist at the tip;

2.2.3.1 Wing properties

Given the input data, the algorithm starts by calculating the effective sweep angle:

$$\Lambda_\beta = \arctan\left(\frac{\tan(\Lambda_{.25})}{\beta}\right) \tag{2.16}$$

where β is the Prandtl-Glauert factor, which depends on the Mach number M and is given by $\sqrt{1-M^2}$; and the planform parameter, used in the additional distribution, which is given by:

$$F = \frac{2\pi AR}{c_{l_\alpha}\cos(\Lambda_{.25})} \tag{2.17}$$

An assumption made in the current work is that the initial twist distribution can be calculated as a (linear) interpolation between the value at the tip and at the root, which must be zero. This approach is not generic, as the shape of the wing does not necessarily need to follow this structure.

Other values needed are:

- **Mean geometric chord:** $c_g = \sqrt{\frac{S}{AR}}$
- **Wingspan:** $b = \sqrt{S.AR}$
- **Root chord:** $c_r = \frac{2.c_g}{1+\lambda}$
- **Tip chord:** $c_t = \lambda c_r$

For a straight-tapered (straight leading and trailing edges) wing, we also have that:

$$\left(\frac{c}{c_g}\right)(\eta) = \frac{c_r}{c_g} + \left(\frac{c_t - c_r}{c_g}\right)\eta \tag{2.18}$$

where η is the adimensional half-spanwise variable, with $\eta = \frac{2y}{b}, 0 \leq \eta \leq 1$.

The sweep angles of the trailing and leading edge (not necessarily the same as the sweep angle measured from the quarter-chord line) can be obtained by:

$$\begin{aligned} \Lambda_{TE} &= \arctan\left(\tan(\Lambda_{.25}) + \frac{1-\lambda}{AR(1+\lambda)}\right) \\ \Lambda_{LE} &= \arctan\left(\tan(\Lambda_{.25}) + \frac{3(1-\lambda)}{AR(1+\lambda)}\right) \end{aligned} \tag{2.19}$$

Finally, the Jone's edge velocity factor is given by:

$$E = \frac{1}{2\cos(\Lambda_{TE})} + \frac{1}{2\cos(\Lambda_{LE})} + \frac{c_t}{b} \tag{2.20}$$

At this point, it is possible to draw the undeflected wing, as shown in Figure 2.10.

2.2.3.2 Additional lift distribution

The additional lift distribution is calculated as:

$$L_a(\eta) = C_1\frac{c}{c_g} + C_2\frac{4}{\pi}\sqrt{1-\eta^2} + C_3 f(\eta, \Lambda_\beta) \tag{2.21}$$

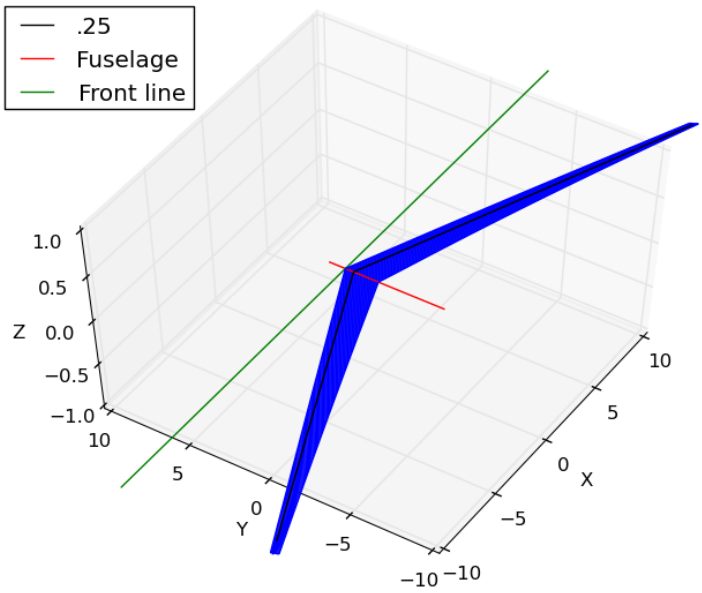

Figure 2.10: Plot of a wing.

where C_1, C_2 and C_3 can be obtained from Figure 2.11 (with the plan form parameter F as input, explained in Section 2.2.3.1) and $f()$ is the lift-distribution function from Figure 2.12 (with η and the effective sweep angle Λ_β as input). Note that Figure 2.12 only provides curves for a finite set of angles and hence, for all other values, some sort of interpolation must be performed.

Once L_a is calculated, we can obtain:

$$\left(\frac{c_{l_a} c}{c_g}\right)(\eta) = C_L L_a(\eta) \tag{2.22}$$

To assert the correctness of the implementation, the distribution may be checked through the value of the following integral:

$$\int_0^1 L_a(\eta)\, d\eta \approx 1 \tag{2.23}$$

The additional distribution, as it considers the untwisted version of the wing, does not require any type of correction after the structural analysis, as only the twist distribution is affected by the bending of the wing.

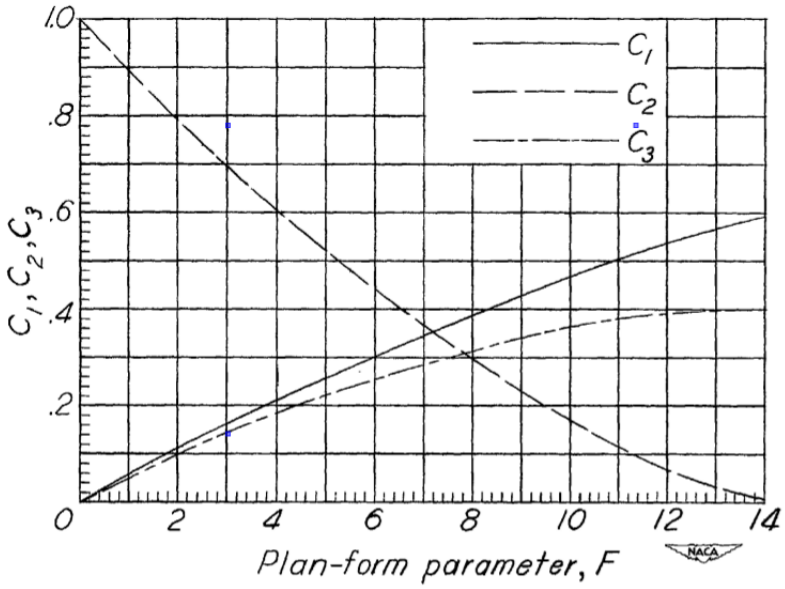

Figure 2.11: The lift-distribution constants C_1, C_2 and C_3, from [1].

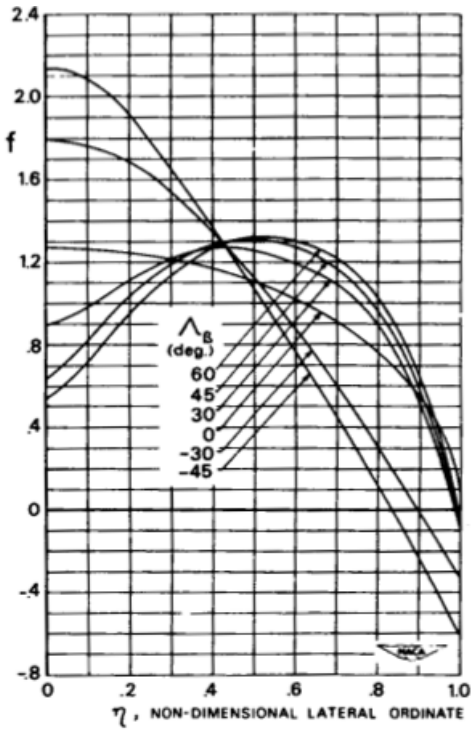

Figure 2.12: The lift-distribution function $f(\eta, \Lambda_\beta)$, from [1].

2.2.3.3 Basic lift distribution

In the original Diederich's implementation, the basic distribution was calculated only once. With the structural correction, it will be iteratively updated

till convergence. Though the form of calculation of the basic distribution will remain untouched, the effective twist distribution will be changing iteratively and so, it will influence the final result. The effective twist distribution can be calculated as:

$$\epsilon(\eta) = \epsilon_t \eta - \sin(\Lambda_{.25})\Theta(\eta) \tag{2.24}$$

where $\Theta(\eta)$ distribution is the correction factor, also known as "slope". In the first iteration, this distribution can be set to all zeros, so the effective twist distribution will be the same as that given as input. When considering only the aerodynamic side of the project, in fact $\Theta(\eta)$ is not necessary, as it will remain zero throughout the method. As it will be explained in Section 2.4.1, it only varies when considering the structural and aeroelastic components. The basic distribution can then be derived by the following formula:

$$L_b(\eta) = L_a(\eta) C_4 \cos(\Lambda_\beta) \left(\frac{\epsilon(\eta)}{\epsilon_t} + \alpha_0 \right) \beta \, E \tag{2.25}$$

Just like the additional distribution, we can also check the validity of the implementation with:

$$\int_0^1 L_b(\eta) \, d\eta \approx 0 \tag{2.26}$$

The two distributions may be summarized by the following statement: *The basic lift is the twist without the total lift, while the additional lift is the total lift without twist.*

2.2.3.4 Total distribution

The sectional lift coefficients can now be calculated using the following:

$$\left(\frac{c_l c}{c_g} \right)(\eta) = \left(\frac{c_{l_a} c}{c_g} \right)(\eta) + \left(\frac{c_{l_b} c}{c_g} \right)(\eta) \tag{2.27}$$

The sectional lift coefficient c_l can be isolated by multiplying the result by the mean geometric chord c_g and dividing it by the chord c.

2.2.4 Vortex lattice method

The vortex lattice method (VLM) defines a discrete sheet of horseshoe vortices to calculate the induced drag, typically using trapezoidal panels to build

this grid. The interactions between them define a system of linear algebraic equations for finding the vortex strengths that satisfy the boundary conditions of no flow through the wing [24]. The solution of the system will provide an approximation of the circulation distribution, used to calculate the wing's lift coefficient. The effects of wing thickness and fluid viscosity are not considered.

Before constructing the grid, a bound vortex line is initially placed along the quarter-chord line of the leading panels, making it in accordance with the sweep angle. The rest of the wing is then covered with panels, whose sides, parallel to the chord axis, are aligned with the free-stream follow.

Each panel has an associated control point, located on the chord axis at the three-quarter chord and located on wingspan axis at the mid-point between the lateral boundaries of the panel. An horseshoe vortex is influenced by the simple vortices located on both the wing and far from the aircraft, a position referred in this document as ∞. Figure 2.13 shows an example of a grid design.

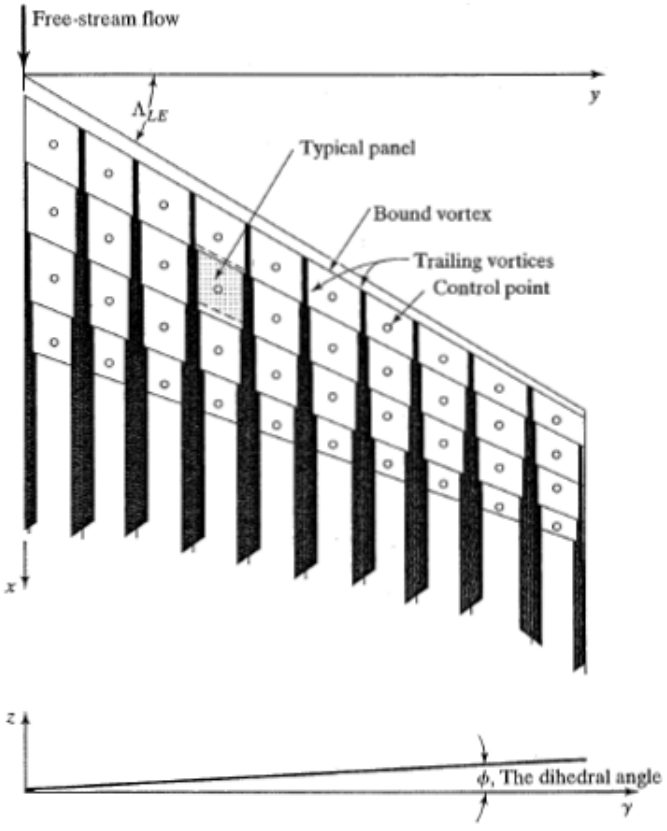

Figure 2.13: Example of a grid used in the vortex lattice method.

2.2.4.1 Velocity induced by a horseshoe vortex

As stated in 2.2.1.5, circulation and flow velocity are closely related. By the law of Bio-Savart [2], the velocity induced by a vortex filament of strength Γ_n is defined by:

$$d\vec{V} = \frac{\Gamma_n.(\vec{dl} \times \vec{r})}{4\pi r^3} \tag{2.28}$$

with magnitude (see Figure 2.15 for the nomenclature) given by:

$$dV = \frac{\Gamma_n \sin\theta dl}{4\pi r^2} \tag{2.29}$$

where Γ is the circulation, $\vec{V}$ is the velocity, $\vec{dl}$ is the length of the element at which the velocity is considered, $\vec{r}$ the vector pointing to the control point and θ its angle.

Let us define points A and B as the extremities of the leadinge edge, while C is the control point. A horseshoe vortex is then constituted by three segments: $\vec{AB}$, representing the leading edge, and the segments $\vec{AC}$ and $\vec{BC}$, joining the extremities to a control point.

Figure 2.14 illustrates two different types of horseshoes. The red one represents a horseshoe confined to a single panel, while the green vortex represents the case where the extremities and the control point are located on different panels. The three segments are mathematically defined as:

$$\begin{aligned} \vec{r_0} = \vec{AB} &= (x_{2n} - x_{1n})\hat{i} + (y_{2n} - y_{1n})\hat{j} + (z_{2n} - z_{1n})\hat{k} \\ \vec{r_1} = \vec{AC} &= (x - x_{1n})\hat{i} + (y - y_{1n})\hat{j} + (z - z_{1n})\hat{k} \\ \vec{r_2} = \vec{BC} &= (x - x_{2n})\hat{i} + (y - y_{2n})\hat{j} + (z - z_{2n})\hat{k} \end{aligned} \tag{2.30}$$

The basic expression for the calculation of the induced velocity by a horseshoe vortex n in the VLM is:

$$\vec{V_n} = \frac{\Gamma_n}{4\pi} \frac{\vec{r_1} \times \vec{r_2}}{|\vec{r_1} \times \vec{r_2}|^2} \left[\vec{r_0}.(\frac{\vec{r_1}}{\vec{r_1}} - \frac{\vec{r_2}}{\vec{r_2}})\right] \tag{2.31}$$

Using 2.31 to calculate the velocity induced at the control point by $\vec{AB}$, we can set:

$$\vec{V_{AB}} = \frac{\Gamma_n}{4\pi}.\{Fac1_{AB}\}.\{Fac2_{AB}\} \tag{2.32}$$

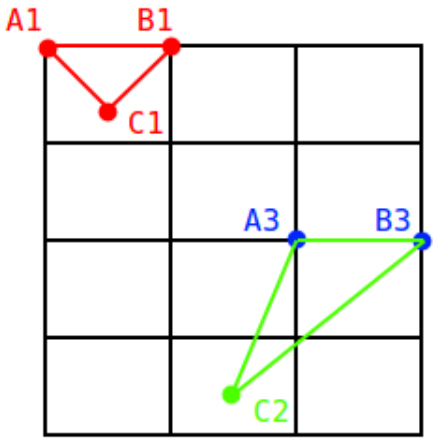

Figure 2.14: Representation of two horseshoe vortices.

where $Fac1_{AB}$ and $Fac2_{AB}$ are replacing:

(2.33)
$$\begin{aligned} Fac1_{AB} &= \frac{\vec{r_1} \times \vec{r_2}}{|\vec{r_1} \times \vec{r_2}|^2} \\ &= \{[(y-y_{1n})(z-z_{2n}) - (y-y_{2n})(z-z_{1n})]\hat{i} \\ &\quad -[(x-x_{1n})(z-z_{2n}) - (x-x_{2n})(z-z_{1n})]\hat{j} \\ &\quad -[(x-x_{1n})(y-y_{2n}) - (x-x_{2n})(y-y_{1n})]\hat{k}\}/ \\ &\quad \{[(y-y_{1n})(z-z_{2n}) - (y-y_{2n})(z-z_{1n})]^2 \\ &\quad -[(x-x_{1n})(z-z_{2n}) - (x-x_{2n})(z-z_{1n})]^2 \\ &\quad -[(x-x_{1n})(y-y_{2n}) - (x-x_{2n})(y-y_{1n})]^2\} \end{aligned}$$

(2.34)
$$\begin{aligned} Fac2_{AB} &= [\vec{r_0}.(\frac{\vec{r_1}}{\vec{r_1}} - \frac{\vec{r_2}}{\vec{r_2}})] \\ &= \frac{(x_{2n}-x_{1n})(x-x_{1n}) + (y_{2n}-y_{1n})(y-y_{1n}) + (z_{2n}-z_{1n})(z-z_{1n})}{\sqrt{(x-x_{1n})^2 + (y-y_{1n})^2 + (z-z_{1n})^2}} \\ &\quad -\frac{(x_{2n}-x_{1n})(x-x_{2n}) + (y_{2n}-y_{1n})(y-y_{2n}) + (z_{2n}-z_{1n})(z-z_{2n})}{\sqrt{(x-x_{2n})^2 + (y-y_{2n})^2 + (z-z_{2n})^2}} \end{aligned}$$

When extending the filaments from A or B to a control point located at

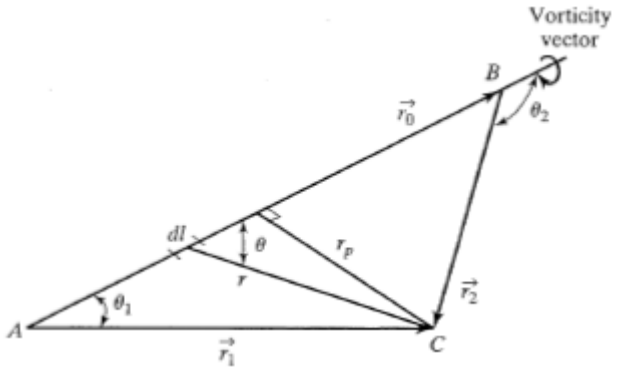

Figure 2.15: Velocity induced by a finite-length vortex segment.

∞ (see Figure 2.16), $\vec{V_{A\infty}}$ and $\vec{V_{B\infty}}$ are defined as:

$$\vec{V_{A\infty}} = \frac{\Gamma_n}{4\pi}\left\{\frac{(z-z_{1n})\hat{j}+(y_{1n}-y)\hat{k}}{[(z-z_{1n})^2+(y_{1n}-y)^2]}\right\}\times \left[1.0+\frac{x-x_{1n}}{\sqrt{(x-x_{1n})^2+(y-y_{1n})^2+(z-z_{1n})^2}}\right] \tag{2.35}$$

$$\vec{V_{B\infty}} = -\frac{\Gamma_n}{4\pi}\left\{\frac{(z-z_{2n})\hat{j}+(y_{2n}-y)\hat{k}}{[(z-z_{2n})^2+(y_{2n}-y)^2]}\right\}\times \left[1.0+\frac{x-x_{2n}}{\sqrt{(x-x_{2n})^2+(y-y_{2n})^2+(z-z_{2n})^2}}\right] \tag{2.36}$$

The vortex strength or downwash of the n horseshoe vortex will be the sum of the three induced velocity components:

$$\vec{\Gamma_n} = \vec{V_{AB}} + \vec{V_{A\infty}} + \vec{V_{B\infty}} \tag{2.37}$$

This downwash can be decomposed into its three axis components (X for chord axis, Y for the wingspan and Z for the chamber axis), respectively represented by the unit vectors $\vec{u}$, $\vec{v}$ and $\vec{w}$. The condition to be solved is:

$$-\vec{u_m}\sin\delta\cos\phi - \vec{v_m}\cos\delta\sin\phi + \vec{w_m}\cos\phi\cos\delta + \vec{U_\infty}\sin(\alpha-\delta)\cos\phi = 0 \tag{2.38}$$

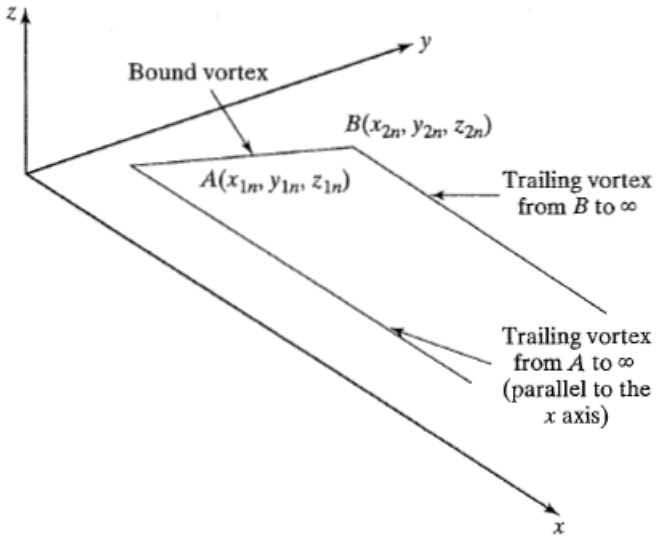

Figure 2.16: Horseshoe vortex with the control point in ∞.

where α is the angle of attack, δ is the mean camber line slope, ϕ is the dihedral angle, $\vec{U_\infty}$ is the velocity of the free stream and $\vec{u_m}, \vec{v_m}, \vec{w_m}$ are the vortex strength or downwash components.

Note that although for the shaping of the grid only the first halfspan vortices are considered, the influence of the ones placed on the second halfspan are also taken into account. The procedure is exactly the same, but the A and B points will be located in the other halfspan, keeping the positions of the vortex control points C untouched. In the end, the two results are added. The total vortex strength is:

(2.39)

$$\vec{w_m} - \vec{u_m} - \vec{v_m} = \sum_{n=1}^{N}(\vec{w_{m,ns}} - \vec{u_{m,ns}} - \vec{v_{m,ns}}) + \sum_{n=1}^{N}(\vec{w_{m,np}} - \vec{u_{m,np}} - \vec{v_{m,np}})$$

where m, ns and np represent the indexes of the considered panels. ns stands for starbord halfspan and np for port halfspan.

As a final remark, the downwash matrix will hold, for each position (m, n), the interaction between the vortex of panel m and the two extremities on panel n (as well as its corresponding panel on the other halfspan). Therefore, the position $(1, 1)$ will only consider the interaction within the first panel, while the position $(1, N)$ will consider the interaction between the control point of the first panel and the leading edge of the last panel.

2.3 Structural modelling

When considering only the above aerodynamic models, we assume that the wing does not suffer any deformations during the flight. However, this is not what happens in reality, as the forces that act on the airfoil will eventually produce a bending moment on the structure. Due to the wing's curvature, the considered angles will also be modified and consequently, there is the need to correct them. This is achieved in the wing's structural analysis.

Structural inputs:

1. Beam's material properties:
   (a) $\boldsymbol{E_s}$: Young's module;
   (b) $\boldsymbol{\sigma_s}$: Yield stress;
   (c) $\boldsymbol{\rho_s}$: Structural density.
2. Beam's geometric properties:
   (a) $\boldsymbol{r_r}$: Internal root radius;
   (b) $\boldsymbol{r_t}$: Internal tip radius;
   (c) $\boldsymbol{t_r}$: Root thickness;
   (d) $\boldsymbol{t_t}$: Tip thickness.
3. Other:
   (a) $\boldsymbol{m_m}$: Mass markup;

2.3.1 Beam properties

Given the internal radius and the thickness, the external radius is defined by:

- **External root radius:** $R_r = r_r + t_r$
- **External tip radius:** $R_t = r_t + t_t$

For the beam properties, we will need the external $R(\eta)$ and the internal $r(\eta)$ distributions. Just like the twist distribution, the two can be calculated using a (linear) interpolation from the values of $R_{r,t}$ and $r_{r,t}$. The following distributions can be calculated:

- **Transversal section surface:** $A(\eta) = \pi(R^2(\eta) - r^2(\eta))$
- **Inertial moment:** $I(\eta) = \dfrac{\pi}{4}(R^4(\eta) - r^4(\eta))$
- **Beam's mass:** $m_s = \dfrac{b\rho_s}{2\cos(\Lambda_{.25})} \int_{-1}^{1} A(\eta)\, d\eta$
- **Stiffness:** $st(\eta) = E_s I(\eta)$
- **Chord distribution:** $c(\eta) = \left(\dfrac{c}{c_g}\right)(\eta).c_g$

- **Wing's relative thickness:** $\left(\frac{t}{c}\right)(\eta) = \frac{2R(\eta)}{c(\eta)}$
- **Beam's linear density:** $\rho_l(\eta) = \rho_s A(\eta)$

2.3.2 The mass markup

The contents in this section provide the aircraft's mass.

1. **Dynamic pressure:** $q = \frac{1}{2}\gamma.p.M^2$
2. **Lift:** $L = C_L.q.S$
3. **Mean cruise flight mass:** $m = \frac{L}{N_c g}$

The mass markup m_m, defined in the input, encompasses both the payload and other additional masses not explicitly included on the model, such as the mass of the fuselage, crew or oil. The fuel mass compatible with the wing shape is given by $m_f = 2 \times (m - m_m - m_s)$. The factor of 2 is due to the fact that m is the mean flight mass instead of the initial cruise mass.

For the cruising flight, the beam's mass is limited by the total mass $m_s < m$. On the other hand, the fuel mass is limited by the condition $m_f > 0$. Violating these restrictions would mean that the airplane was not capable of transporting payload (e.g, luggage).

From those, we can calculate the initial and final cruising masses, for the Breguet range (a estimate of the distance a airplane can travel under certain conditions, see Section 2.5.4):

(2.40)
$$m_0 = m + \frac{m_f}{2}$$

$$m_1 = m - \frac{m_f}{2}$$

2.3.3 Atmospheric properties

Given the flight altitude h, the static pressure p, temperature T and gravity acceleration g must be obtained (likely given as input). For the Breguet range, the speed of sound at sea level will also be required.

2.3.4 Loads

For loads, we start by calculating the shear force, given by:

$$F_s(\eta) = qS \int \left(\frac{c_l c}{c_g}\right)(\eta)d\eta - \frac{bN_c g}{2\cos(\Lambda_{.25})} \int \rho_l(\eta)d\eta \tag{2.41}$$

where q is the dynamic pressure, S is the wing area, ρ_l the beam's linear density and N_c the load factor.

From the shear force, we obtain the bending moment:

$$M_b(\eta) = \frac{b}{2} \int F_s(\eta)d\eta \tag{2.42}$$

These three integrals, for the shear force and bending moment, must be calculated from the tip to the root [6]. The distributions F_s and M_b typically have the shape shown in figure 2.17.

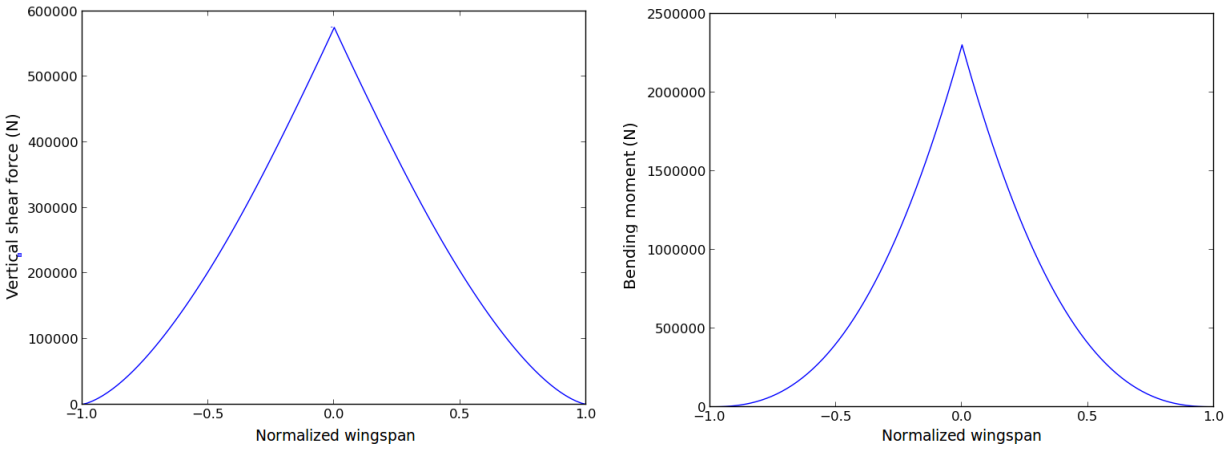

Figure 2.17: The shapes of the shear force and bending moment distributions.

2.4 Aeroelastic modelling

The aeroelastic component will be responsible for gathering the aerodynamic and structural parts into a single method. Previously, in Section 2.2.3.3, we have seen that only the basic distribution is dependent on the twist distribution, which is changed by the deformation of the wing. Hence, the aeroelastic analysis will consist of a loop, starting with the calculation of the basic distribution and updating it according to the values returned by the structural

component. A convergence criteria will be set to define when this loop should terminate.

Aeroelastic inputs:

1. $\boldsymbol{\delta_f}$: Maximum wing bending allowed;
2. $\boldsymbol{\eta_d}$: Typical section for divergence calculation.

Propulsion inputs:

1. $\boldsymbol{C_{T_0}}$: Specific fuel consumption parameter;
2. $\boldsymbol{C_{T_1}}$: Specific fuel consumption parameter.

C_{T_0} and C_{T_1} are used to calculate C_T, the kerosene mass consumption per second:

$$C_T = (C_{T_0} + C_{T_1} M)\sqrt{\theta_{ST}} \tag{2.43}$$

where θ_{ST} is the static temperature ratio for the flight altitude.

2.4.1 Displacements

In this section, the slope distribution is calculated, which allows the new effective twist and the deflection of the wing during flight to be determined. The slope $\Theta(\eta)$ and the deflection $\Delta Y(\eta)$ equations are given by:

$$\begin{aligned} \Theta(\eta) &= \frac{b}{2}\int \left(\frac{M_b}{E_s I}\right)(\eta)d\eta \\ \Delta Y(\eta) &= \left(\frac{b}{2}\right)^2 E \int \left(\frac{M_b}{E_s I}\right)(\eta)d\eta \end{aligned} \tag{2.44}$$

Note that in opposition to the integrals of 2.3.4, these are calculated from the root to the tip. As example of a $\Theta(\eta)$ distribution is displayed in Figure 2.18. The deflection $\Delta Y(\eta)$ can be used for plotting purposes, as it gives the elevation of each section according to the horizontal plane.

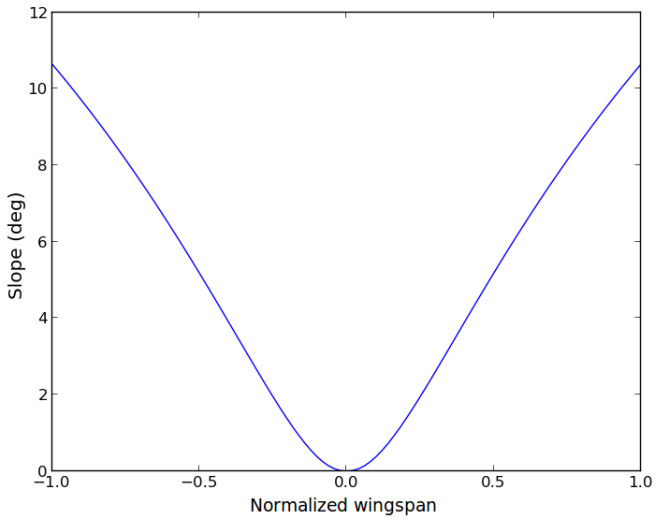

Figure 2.18: Shape of the slope distribution.

2.4.2 Convergence evaluation

The convergence evaluation will determine whether the loop should continue or not. The relevant values are:

$$\delta_{b_i} = L_{b_i} - L_{b_{i-1}} \tag{2.45}$$

$$L_n(\vec{\delta_{b_i}}) = \left(\sum_{k=1}^{N} |\delta_k|^{n'} \right)^{\frac{1}{n'}} \tag{2.46}$$

where N is the number of control points and n' is the type of norm. The usual setting is $n' = 2$ for the Euclidean norm. When $L_n(\vec{\delta_{b_i}}) < \xi$, where ξ is the convergence bound provided as input, the solution is said to be convergent.

If $|\Theta|_{max} \geq \frac{\pi}{2}$ or the number of iterations exceeds the maximum allowed, the solution is considered divergent. Divergence may indicate an inappropriate beam design when considering its flexibility against the structural load.

2.5 Post-processing

2.5.1 Diederich's method induced drag value

When the process converges, the induced drag value can be calculated with the Garner Method for untwisted wings, as long as, in the end, the corresponding

correction for twisted wings is applied [1]:

$$\delta = 46.264. \left(n_{cp} - \frac{4}{3\pi} \right)^2 \tag{2.47}$$

where the spanwise centre of pressure n_{cp} is given by:

$$n_{cp} = \int_0^1 \left(\frac{c_l c}{C_L c_g} \right) (\eta)\, d\eta \tag{2.48}$$

with a correction:

$$\Delta C_{D_i} = 3.7e^{-5} {\epsilon_t}^2 \tag{2.49}$$

and having the twist angle at the tip ϵ_t in degrees, the final value of the induced drag can be approximated as:

$$C_{D_i} = (1 + \delta) \frac{{C_L}^2}{\pi AR} + \Delta C_{D_i} \tag{2.50}$$

The total drag is given by:

$$C_D = C_{D_i} + C_{D_0} \tag{2.51}$$

where C_{D_0} is the markup value for the drag coefficient, given in the input.

2.5.2 Vortex lattice method lift coefficient and induced drag

Once the lift distribution is returned by the VLM, the lift coefficient and the induced drag are calculated. The lift coefficient can be defined as:

$$C_L = 16\pi \sin\alpha \cos\phi\, U_\infty \rho_\infty \sum_{n=1}^{N} (\Gamma_n \Delta y_n) \times \frac{1}{S} \tag{2.52}$$

where α is the angle of attack, ϕ the dihedral angle, U_∞ is the free-follow speed and ρ_∞ its density, Γ_n the circulation value and Δy_n the wingspan step.

The value of the induced drag is obtained through the following expression:

$$C_{Di} = \frac{{C_L}^2}{\pi . AR . e} \tag{2.53}$$

where C_L is the lift coefficient, AR the aspect ratio and e the Oswald efficient factor, which, for our particular case, was set to 0.95. The value for the total drag is calculated exactly as in Equation 2.51.

2.5.3 Security Margins

Another important aspect to consider are the security margins. As they define maximum values that the wing can support, they will serve as constraints during the optimizing process. The flutter security margin can be obtained by:

$$SM_{flutter} = \frac{\delta_f - \delta_{max}}{\delta_f} \tag{2.54}$$

with δ_f (the maximum wing deformation allowed) given at the input and δ_{max} calculated in section 2.4.1.

The divergence security margin is given by:

$$SM_{divergence} = \frac{q_d - q}{q_d} \tag{2.55}$$

where q is the dynamic pressure at flight condition (calculated in 2.3.2) and q_d is the divergence dynamic pressure, calculated from [7].

Finally, there is the stress security margin, given by:

$$SM_{tension} = \frac{\sigma_s - \sigma_{max}}{\sigma_s} \tag{2.56}$$

where the maximum allowed tension, σ_s, is given as input, and σ_{max} is calculated from:

$$\sigma(\eta) = \frac{|M_b(\eta)|R(\eta)}{I(\eta)} \tag{2.57}$$

where $M_b(\eta)$ is the bending moment, $R(\eta)$ the external beam diameter distribution and $I(\eta)$ the inertial moment, all previously calculated.

2.5.4 Breguet Range

Finally, the Breguet range provides an estimation of the distance an airplane can fly, given the atmospheric conditions, the drag coefficient and the structural mass.

$$R_b = \frac{Ma_0\sqrt{\theta_{ST}}C_L}{C_T C_D} \ln\left(\frac{m_0}{m_1}\right) \tag{2.58}$$

where M is the Mach number, a_0 the speed of sound at sea level, θ_{ST} the static temperature ratio for the flight altitude, C_T the specific fuel consumption, C_L

the lift coefficient, C_D the drag coefficient, m_0 the mass of the airplane at the beginning of the flight and m_1 the mass at the end of the flight.

The C_T parameter is calculated as:

$$C_T = (C_{T_0} + C_{T_1} M)\sqrt{\theta_{ST}} \tag{2.59}$$

Since the Breguet range represents an efficiency measure of the propulsion, aerodynamics and structural components, it can be used as a multi-disciplinary objective function for the optimizer.

A limitation of the estimate is that M, C_L and C_D vary during the flight, whereas the mass is measured at beginning and the end. To solve this problem, the inputs of the formula are mean values, with m_0 and m_1 obtained as explained in Section 2.3.2.

This Section concerning Diederich's method was written based on a document provided by Marc Mulkens, which in turn, was based on [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21] and [22].

2.6 Concluding remarks

Diederich's method takes quite a different approach when compared to the lifting-line theory (LLT) and vortex lattice method (VLM). It drops the vortex distributions and introduces a strong empirical component. In order to perform the calculations, there are some steps that use values returned by functions constructed from real data.

Another, and likely more important, difference is the starting point that each method takes to accomplish induced drag calculation. VLM requires as input the angle of attack and the wing geometry. In addition, LLT also needs the slope of the lift function in order to calculate the lift coefficient C_L. On the contrary, Diederich's method receives the value of C_L as input, assuming that the wing will be built in a way that the defined angles of attack of each section will lead to the desired C_L.

In general, we can say that both LLT and the original Diederich's method are fairly straightforward from a computational point of view. The differences are more notorious if we consider Diederich's method with the aeroelastic correction, as the convergence of the algorithm may take longer than the other approaches. The relative computational simplicity of Diederich's is due to the

fact that it approximates the final lift distributions through the use of a chord-length sectioned wing, and that most calculations, if not all, can be performed in constant time per section.

On the other hand, VLM (and LLT too, though this method is lighter generally than VLM) carries the additional weight of having to solve a potentially huge system of linear equations and having to maintain a grid of panels as well. Although the method is easy to understand and implement, solving the system of equations can make this method computationally heavy and hence, the opportunity for looking for incremental approaches.

3

Optimization

Optimization is the minimization or maximization of a function subject to constraints on its variables [25]. We can define:

- x as the vector of variables;
- $f()$ as the objective function, a function of x to be minimzed or maximized;
- $c_i()$ as the constraint functions, scalar functions of x which define the set of constraints the vector x must respect during the optimization.

Then, an optimization problem can be defined as:

$$\begin{aligned} \text{minimize } & f(x), x \in \mathbb{R}^n \\ \text{s.t. } & c_i(x) = 0, i \in E, \\ & c_j(x) \geq 0, j \in I \end{aligned} \tag{3.1}$$

where E is the set of equality constraint indices and I the set of inequality constraints indices. A maximization problem can be transformed into a minimzation problem by using the symmetric of the inverse of the objective function $f()$.

3.1 Direct search, linear-search and trust-region interpolation methods

In[26], three different families of optimization approaches are presented, namely direct search methods (DSM), line-search algorithms and trust-region interpolation based methods. All of them share the fact that they avoid using function derivatives, as they may be hard or even impossible to compute, for example, because the objective function is not continuous.

Briefly, DSM use an iterative process over a set of finite points to assess which of them present a better value for the objective function. By comparing pairs of points, it does not need to consider representations of the objective function or its derivatives.

The method can also define two different steps, namely a poll step and a search step. The first will ensure that the algorithm converges (delimiting the searching area to a finite set of points), while the second defines the range of a local search around the current iterate. At the end of each iteration, if a better point is found, the iteration will be marked as successful. The step size will be consequently increased or reduced, depending on whether the iteration was successful or not.

As stated before, the alternatives are line-search algorithms and trust-region interpolation based methods, both inspired on derivative approaches. The first class of algorithms defines a search direction which it follows while looking for a better solution, while the second uses an interpolated model from a set of points to approximate the objective function.

3.2 Gradient-based optimization

Gradient-based optimization is an iterative process which, by analyzing the derivative at certain points, decides which direction the algorithm will take next, by moving to the next evaluation point in search for the minimum of a scalar function of N real variables.

Each iteration is given by:

$$x_{k+1} = x_k + \alpha_k p_k \tag{3.2}$$

where α_k is the step length, which must be carefully selected [25]. Vector p_k is called the steepest descent direction, the direction for which the function varies the most. It can be represented as:

$$p_k = -B_k^{-1} \bigtriangledown f_k \tag{3.3}$$

where B_k is a symmetric and nonsingular matrix. This matrix can vary from the identity matrix in the simple steepest descent method to the Hessian $\bigtriangledown^2 f(x_k)$ in Newton's method. If B_k is positive definite, then we have that:

$${p_k}^T \bigtriangledown f_k = - \bigtriangledown f_k^T B_k^{-1} \bigtriangledown f_k < 0 \tag{3.4}$$

The following loop will be executed till convergence:

1. Check the convergence criterion of $f(x_k)$; terminate with $x = x_k$ if the solution if fulfilled.

2. Compute a direction p_k and a step length α_k.

3. Evaluate the new point $x_{k+1} = x_k + \alpha_k.p_k$; compute the target $f(x_{k+1})$ and go back to step 1.

The step α_k should be large enough to promote fast convergence, but not too large, or the minimum will not be found. For further discussion, please refer to [25].

Surrogate models can also be considered. They are built from real training data to later produce an output when in the presence of unseen values. Some of the best examples are least-squares polynomials, multi-layer perceptrons, radial basis functions (RBF) and Kriging (techniques to interpolate unknown values in geostatistics, though successfully applied to other areas of engineering, including aeronautics [27]).

3.2.1 Automatic Differentiation

Associated with the gradient methods, we can find the automatic differentiation (AD) techniques. The AD can be seen as a way of efficiently computing all the necessary derivatives of a given function [28]. Basically, all the functions present in a computer algorithm can be decomposed into primitive functions and constants. By applying the chain rule to the decomposition of the overall

function, we get the resulting tree-graph. Numerical evaluations of the successive operations applied over the nodes are also stored in this graph. The derivation of the nodes is applied to the original graph, resulting in the derivative tree.

Table [29], illustrates the process, considering that we are evaluating the derivative at the point $x = 3$:

Function	**Derivative**
$x = 3$	$x' = 1$
$y_1 = x^2 = 9$	$y_1' = 2xx' = 6$
$y_2 = sin(y_1) = 0.4212$	$y_2' = cos(y_1)y_1' = -5.46668$
$y = xy_2 = 1.2363$	$y' = x'y_2 + xy_2' = -15.9883$

3.3 Optimization algorithms

3.3.1 DIRECT

The implementation of DIRECT, a direct-search algorithm, was based in [37]. The algorithm starts by transforming the optimization space into an unit hyper-cube by normalizing all the variables. Working from the centre of this hyper-cube, the space is successively divided into sections, generating new centres for new subspaces. Each centre is verified to check if it is potentially optimal. If it is not, it is simply discarded. The process continues until no more points are considered potentially optimal, outputting the best evaluation found.

3.3.1.1 Initialization

The first step is to normalize the problem domain into an hyper-cube:

$$\bar{\Omega} = x \in R^N : 0 \leq x \leq 1 \tag{3.5}$$

After the normalization, the value of the objective function $f(c_1)$ is calculated for the hyper-cube's centre. The subspace of each variable is divided into thirds, creating the new centres at $c_1 \pm \delta e_i$, for $i = 1, ..., n$, as shown in Figure

3.1. δ represents the third of the side-length of the current hyper-cube, while e_i is the unit vector corresponding to the ith design variable.

Each dimension is assigned the minimum value of the objective function of its two centres, designated as v:

$$v_i = \text{minimize } (f(c_1 + \delta e_i), f(c_1 - \delta e_i)),\ 1 \leq i \leq N \tag{3.6}$$

After ordering all the dimensions by the corresponding values of v, new hyper-cubes (that may turn into hyper-rectangles) are successively divided. The centres of the current dimension previously defined by $c_1 \pm \delta e_i$, get a third of the current dimension length. Then, for all the other centres, the length of the current dimension is shrunk also by a factor of three.

This means that the dimension with the highest v will be the one whose hyper-cubes are smallest, as all its dimensions' lengths have been shrunk. Figure 3.1 depicts the process for the first iteration, where the horizontal dimension has a smaller value of v and hence, gets a greater subspace for its hyper-rectangles. Pseudo-code description of DIRECT is given in Algorithm 1, and a set of images representing several iterations of DIRECT is provided in Figure 3.2.

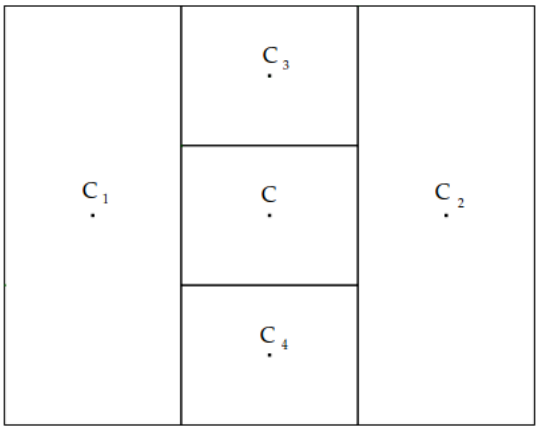

Figure 3.1: Initial 2-dimensional space division in the DIRECT algorithm.

3.3.1.2 Potentially optimal hyper-cubes

Given all the new defined subspaces, the potentially optimal hyper-cubes must be defined. Once selected, the new hyper-space is divided in the same way as

Algorithm 1 DIRECT algorithm

Input: Optimization bounds

Output: Maximum solution

```
1: P ← {} // The potential optimal list
2: evaluate(centre) // Evaluate the normalized hyper-cube centre
3: insert(P, centre)
4: while P > 0 & diff < TOLERANCE do
5:    A ← {} // The analysis points list
6:    for all c in P do
7:       for i ← 1, NUMBER OF DESIGN VARIABLES do
8:          sc.position[i] ← c.position[i] - δe_i
9:          pc.position[i] ← c.position[i] + δe_i
10:         evaluate(sc, pc)
11:         insert(sc, pc, A)
12:         update(current best value)
13:      end for
14:      sort(A)
15:      for m ← 1, A.size() do
16:         dim ← A[m].dimension;
17:         for j ← m + 1, A.size() do
18:            if A[j].dimension ≠ dim then
19:               A[j].dimension_lengths[dim] *= 1/3;
20:            end if
21:         end for
22:         if is_potentially_optimal(A[m]) then
23:            insert(A[m], P)
24:         end if
25:      end for
26:      diff = previous best value - current best value
27:   end for
28: end while
```

previously described, although now the length of each dimension has been reduced and may not be the same for all the dimensions.

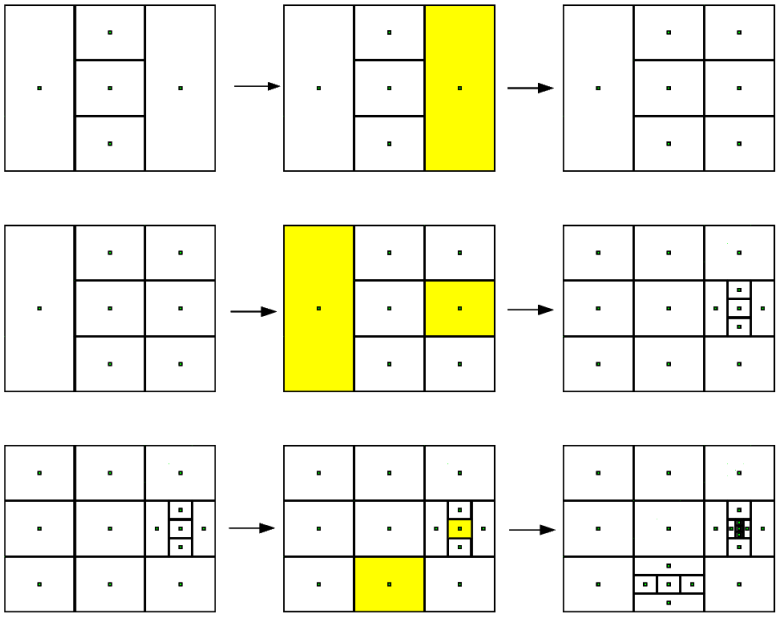

Figure 3.2: Several iterations of DIRECT.

3.3.2 Hill-climber

An alternative to DIRECT is the hill-climber. Given a set of neighbours, it selects the one that has the best value to continue exploring the optimization space. From the starting point, which can be chosen randomly, the value of the objective function $f(x)$ is calculated. Given a finite set S of the neighbours of x, the one presenting the best objective function value, $\min f(s)$, $s \subset S$, is selected. It then enters in a loop, restarting the process from the selected neighbour s. The algorithm stops when none of the neighbours present a greater value of the objective function.

In the implementation used in this work, to define the neighbourhood set S, a vector of step sizes for each dimension is maintained. Then, for each dimension, the value of the step is subtracted/added to the current solution, meaning the number of neighbours will euqal to the double of the dimensions.

The implementation is also an adaptation of the original hill-climber to support a variable step size, to promote faster convergence. When a solu-

tion is chosen, the size of the step for the selected dimension is increased. If not selected in the current iteration, the step is decreased, till a lower bound previously defined.

3.4 Concluding remarks

In this Chapter, various optimization approaches were briefly reviewed. Gradient-based optimizers are suited to problems where the derivatives are available or can be easily approximated, whereas alternatives such as direct search method can be used when the functions are discontinuous. Concerning this last topic, two optimizers were presented, namely DIRECT and a hill-climber.

For this project, direct search methods are the more suitable, as the shape of the objective function is relatively unknown. Although the model functions seem to be smooth, there are no guarantees about discontinuty points and therefore, gradient-based methods cannot be confidently used.

4

Systems of Linear Equations

Many engineering problems make use of systems of linear equations. There are different ways of finding a solution and so, the best approach must be carefully selected. In this Chapter, some of those options will be presented and their advantages concerning our specific problem will be discussed.

4.1 Solving and decomposition

The basic way to solve a system $Ax = b$ is to successively transform the matrix of left side coefficients A, in a process known as Gaussian elimination, in order to find the solution x. In each step, all the coefficients beneath the current diagonal element (named the pivot) are eliminated, by subtracting a scalar multiple of the current row from each of the remaining rows.

Two types of pivoting are defined. In partial pivoting, the largest absolute value in the current column is chosen as the next pivot, only considering row permutations. In complete pivoting, all the remaining matrix positions can be considered for pivoting, selecting again the largest absolute value and considering both row and column permutations. Complete pivoting offers the highest accuracy at the expense of additional computations. Usually, partial pivoting is suitable for most applications.

Provided that the system has a single solution, the original matrix is transformed into an upper triangular matrix, with all the subdiagonal elements equal to zero.

4.1.1 LU decomposition

An alternative to Gaussian elimination is to perform the LU decomposition. Given the matrix A, the method decomposes it into three matrices (or four, if column permutations are applied for complete pivoting), namely P, L and U. P is the row permutation matrix, while L and U are lower and upper triangular matrices, respectively. If complete pivoting is used, some columns may be permuted as well, resulting in a fourth matrix Q.

The idea behind the LU decomposition is to perform Gaussian elimination, saving the elimination coefficients in L, while U will hold the result of the elimination. When these matrices are obtained, the original system of equations can be solved in two steps:

$$\begin{aligned} Ly &= b \\ Ux &= y \end{aligned} \tag{4.1}$$

As matrices L and U are triangular matrices, the system can be easily solved by forward and back substitution. The speed of solving these linear systems is greater when comparing to the application of Gaussian elimination to the original matrix. However, it is also true that to perform the LU decomposition, Gaussian elimination will have to be performed in first place, making the whole process just as slow (or even slower) as the first approach.

In the special case where the same linear system must be solved for different values of b, however, once the LU factorization is available, the system can be solved quickly as many times needed.

4.1.2 Decomposition update

For some models, it is reasonable to apply optimization steps which only affect few rows and/or columns of matrix A. Therefore, it is wise to consider updates of lower complexity for low rank changes.

Being aware that, due to pivoting, some of the rows in the LU decomposition may have changed their places comparatively to the original positions, some conversions must be made in the ongoing calculations to take this in account.

4.1.2.1 Column exchange

If one column is exchanged in the original matrix, the resulting matrix is defined as:

$$S' = PL\tilde{U}Q \tag{4.2}$$

where $\tilde{U}$ is the altered U matrix. Let c be the column altered in the original matrix. Then, c' is defined as the correspondent column in $\tilde{U}$, given by:

$$Lc' = P^{-1}c \tag{4.3}$$

If q is the length of c' (row number of the last nonzero element), c' can be shifted to the $q-1$ position in the altered matrix $\tilde{U}$, meaning the columns $c+1, c+2, ..., q-1$ are shifted to the left one position, as shown in Figure 4.1. This results in an upper Hessenberg matrix H, with subdiagonal nonzeros at the shifted positions. If more than one column is exchanged simultaneously, there may be more than a single nonzero subdiagonal element. This column permutation can be defined in the matrix $\bar{Q}$.

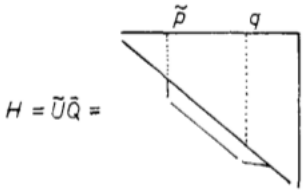

Figure 4.1: Hessenberg matrix resulting from $\tilde{U}$.

The matrix H must be turned back into an upper triangular matrix, resulting in the following equation:

$$S' = (P\tilde{P}^{-1})(\tilde{P}LM^{-1})(M\tilde{U}\bar{Q}\tilde{Q})(\tilde{Q}^{-1}\bar{Q}^{-1}Q) = P'L'U'Q' \tag{4.4}$$

To find the missing matrices $\tilde{P}$, $\tilde{Q}$ and M, one can follow this procedure. For getting the product of $M\tilde{U}\bar{Q}$, the normal Gaussian elimination is applied to $\tilde{U}\bar{Q}$, getting an upper triangular matrix. During the elimination, the elimination coefficients are stored in the subdiagonal positions of the matrix M^{-1}, whose diagonal is equal to the identity matrix. If during the Gauss elimination no pivoting is made, $\tilde{P}$ and $\tilde{Q}$ will be equal to the identity matrix, different otherwise to take in account the permutations applied.

4.1.2.2 Row exchange

Row exchange is quite similar to column exchange, taking the symmetric operations to accomplish the final result. If one row is exchanged in the original matrix, the resulting matrix is defined as:

$$S' = P\tilde{L}UQ \tag{4.5}$$

where $\tilde{L}$ is the altered L matrix. Let r be the row altered in the original matrix. Then, r' is defined as the correspondent row in $\tilde{L}$, given by:

$$U^T r'^T = r^T Q^{-1} \tag{4.6}$$

If q is the length of r' (column number of the last nonzero element), r' can be shifted to the $q-1$ position in the altered matrix $\tilde{L}$, meaning the rows $r+1, r+2, ..., q-1$ are shifted up one position, as shown in Figure 4.2. This results in an upper Hessenberg matrix H, with superdiagonal nonzeros at the shifted positions. If more than one row is exchanged simultaneously, there may be more than a single nonzero superdiagonal element. This row permutation can be defined in the matrix $\bar{P}$.

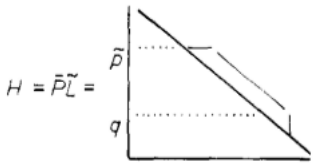

Figure 4.2: Hessenberg matrix resulting from $\tilde{L}$.

The matrix H must be turned back into an lower triangular matrix, resulting in the following equation:

$$S' = (P\bar{P}^{-1}\tilde{P}^{-1})(\tilde{P}\bar{P}\tilde{L}M^{-1})(MU\tilde{Q})(\tilde{Q}^{-1}Q) = P'L'U'Q' \tag{4.7}$$

To find the missing matrices $\tilde{P}$, $\tilde{Q}$ and M, one can follow this procedure. For getting the product of $\bar{P}\tilde{L}M^{-1}$, the transposed Gaussian elimination can be applied to $\bar{P}\tilde{L}$, getting a lower triangular matrix. The transposed Gaussian elimination consists in starting at the first column and successively eliminating the nonzero elements in the row, counting from the position $i+1$ and on. During the process, the elimination coefficients are stored in the superdiagonal

positions of the matrix M, whose diagonal is equal to the identity matrix. If during the Gauss elimination no pivoting is made, $\tilde{P}$ and $\tilde{Q}$ will be equal to the identity matrix, different otherwise to take in account the permutations applied.

4.2 Iterative methods

Iterative methods appear as an alternative to the traditional approaches for solving systems of linear equations. The problem with, for example, the LU factorization or Gaussian elimination, is that in cases where the size of the matrix is very large, it may be impossible to save all the fill-in resulting from these methods. In contrast, iterative methods work in-place and they do not require more than a few extra vectors [31].

However, Gaussian elimination always produces exact results. Iterative methods, if desired, may produce not so accurate results, but still applicable for many situations.

Given the left side coefficient matrix A and the solution vector b, the linear system to be solved is:

$$Ax = b \tag{4.8}$$

However, instead of looking for an exact solution x, an approximation $\tilde{x}$ can be used. Define the residual as:

$$r = b - A\tilde{x} \tag{4.9}$$

and the error $e = x - \tilde{x}$ that satisfies the equation:

$$Ae = r \tag{4.10}$$

In iterative methods, the approach is to solve:

$$Se = r \tag{4.11}$$

where S is an approximation to A. Corrections will be added to $\vec{x}$ till it is close enough to the exact solution x, executing the following loop:

1. Define x^{old}, the current approximation to x.
2. Compute the residual $r = b - Ax^{old}$.

3. Solve $Se = r$.
4. Set $x^{new} = x^{old} + e$ and go back to step 1.

When multiplying $x^{new} = x^{old} + e$ by S, the following equation is obtained:

(4.12)
$$\begin{aligned} Sx^{new} &= Sx^{old} + Se \\ &= Sx^{old} + b - Ax^{old} \\ &= (S - A)x^{old} + b \\ &= Tx^{old} + b \end{aligned}$$

(4.13)

where $T = (S - A)$ is called the splitting of matrix A. All the three methods here presented (Jacobian, Gauss-Seidel and Successive over-relaxation (SOR) methods) have a similar terminology.

The choice of S gives rise to different iterative schemes [31]. For instance, matrix A can be split as:

(4.14)
$$A = D - L - U$$

where D is its diagonal, $-L$ the strictly lower triangular and $-U$ the strictly upper triangular matrix. On SOR, there are two additional constants. They are σ and ω, related by $\omega = \frac{1}{\sigma}$. A value of $\omega > 1$ will speed up convergence of the process; $\omega < 1$ slows it down or establishes convergence in a divergent process; $\omega = 1$ will turn SOR into the Gauss-Seidel method. The mathematical description of the methods is presented in the following sections.

4.2.1 Jacobian method

(4.15)
$$S = D;\ T = L + U$$

(4.16)
$$Dx^{new} = (L + U)x^{old} + b$$

(4.17)
$$x_i{}^{new} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j{}^{old} - \sum_{j=i+1}^{n} a_{ij} x_j{}^{old} \right)$$

4.2.2 Gauss-Seidel method

$$S = D - L;\ T = U \tag{4.18}$$

$$(D - L)x^{new} = Ux^{old} + b \tag{4.19}$$

$$x_i{}^{new} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j{}^{new} - \sum_{j=i+1}^{n} a_{ij} x_j{}^{old} \right) \tag{4.20}$$

4.2.3 Successive over-relaxation (SOR) method

$$S = \sigma D - L;\ T = (\sigma - 1)D + U \tag{4.21}$$

$$(D - \omega L)x^{new} = (1 - \omega)Dx^{old} + \omega Ux^{old} + \omega b; \omega = \frac{1}{\sigma} \tag{4.22}$$

$$\hat{x}_i^{new} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j{}^{new} - \sum_{j=i+1}^{n} a_{ij} x_j{}^{old} \right) \tag{4.23}$$

$$x_i{}^{new} = (1 - \omega)x_i{}^{old} + \omega \hat{x}_i^{new} \tag{4.24}$$

4.3 Concluding remarks

The LU decomposition update is a method that clearly promotes the use of an incremental approach, with greater impact when using large matrices subject to small changes.

From another point of view, the iterative solving methods can be quite useful when applied together with the structural model. As the structural model uses an iterative approach as well, there is no need to solve the linear system with great accuracy in the first steps, as it will not immediately output the final solution.

Therefore, the precision required may be increased as the solution approaches the final state, allowing saving iterations during the solving in earlier stages of the structural loop.

5

The Proposed Computational Approach

In this Chapter, the fundamental ideas of the project will be presented. It is an exploratory view about the potential speed-up the implemented work can achieve. Simple cases will be studied and their speed-ups will give an approximation of what can be achieved in the optimization process, or in other cases, in future work.

5.1 Hardware specifications

The programming languages used for this project were Python (including the packages Numpy and Scipy) for the early development, and C++ (including the BLAS and OpenMP libraries) for the final implementation. Matlab was also considered as an alternative to Python, but preliminary tests showed that it was slower (around 5% to 10%), besides not offering any special advantage in terms of code development. Finally, OpenCL was used to interact with the GPU.

For running most of the tests, a laptop LG R510 with the following specifications was used:

- **CPU:** Intel(R) Core 2 Duo 2.4GHz
- **Cache:** 3MB L2 Cache
- **RAM:** 4GB
- **Swap:** 3GB

- **Graphics:** NVIDIA GeForce 9600M GS
  - **Cores:** 32
  - **Dedicated memory:** 512MB
  - **Maximum PCIe Link Speed:** 2.5 GT/s
- **Operating system 1:** Linux, Mint 13, 64bits
- **Operating system 2:** Windows, 7, 32bits

While the Linux operating system was the main choice, Windows was reserved for running the GPU tests. This was due to the problems encountered in setting up OpenCL for Linux Mint.

Since the hill-climber registered considerably longer executions when compared to DIRECT, the tests involving this algorithm where ran in the cluster kindly provided by the ECOS group. The specification of a cluster machine is the following:

- **CPU:** AMD Athlon(tm) 64 X2 Dual Core Processor 3800+ 2.0GHz
- **Cache:** 512 KB L2 Cache
- **RAM:** 3.8GB
- **Swap:** 3.9GB
- **Operating system:** Linux, Ubuntu 12.04, 32bits

5.2 Aeroelastic optimization

Wing optimization based on the proposed aeroelastic model involved the design variables summarized in Table 5.1, the values of which are set by the optimizer in the search for the best solution. They are grouped into two sets, namely wing geometry and beam properties related variables. Values range between the given lower and upper bounds and they serve as constraints for the optimizer.

Design variables			
Variable	L-bound	U-bound	Units
Wing geometry			
Wing span	15.00	32.00	**m**
Root chord	2.00	5.00	**m**
Tip chord	1.00	4.00	**m**
Sweep angle	-45.00	45.00	**deg**
Twist angle	-12.00	12.00	**deg**
Dihedral angle	0.00	15.00	**deg**
Beam properties			
Root inner radius	0.000	1.000	**m**
Root thickness	0.002	1.002	**m**
Tip inner radius	0.000	0.500	**m**
Tip thickness	0.002	0.502	**m**

Table 5.1: Design variables.

5.3 Problem formulation

The problem formulation can be defined as:

$$
\begin{aligned}
\text{5.1} \qquad \text{maximize} \quad & breguet \quad && (b, r_c, t_c, \Lambda, \epsilon, \phi, r_{inner}, r_{thick}, t_{inner}, t_{thick}) \\
\text{s.t.} \quad & b && \in [15, 32], \\
& r_c && \in [2, 5], \\
& t_c && \in [1, 4], \\
& \Lambda && \in [-45, 45], \\
& \epsilon && \in [-12, 12], \\
& \phi && \in [0, 15], \\
& r_{inner} && \in [0, 1], \\
& r_{thick} && \in [0.002, 1.002], \\
& t_{inner} && \in [0, 0.5], \\
& t_{thick} && \in [0.002, 0.502]
\end{aligned}
$$

where b is the wingspan, r_c is the root chord length, t_c is the tip chord length, Λ is the sweep angle, ϵ is the twist angle, ϕ is the dihedral angle, r_{inner} is the

root internal radius, r_{thick} is the root thickness, t_{inner} is the tip internal radius and t_{thick} is the tip thickness. This design space is summarized in Section 5.2 and the meaning of each variable is explained in Chapter 2.

Note that the *breguet*() function presented in this formulation may use any of the three models presented in Section 2.2, namely lifting-line theory, Diederich's method or the vortex lattice method.

5.4 Vortex lattice method grid

One of the explored techniques was the incrementalization of the downwash calculation, applied to the vortex lattice method (VLM). For this purpose, an alternative grid format was considered, built from linearized theory concepts.

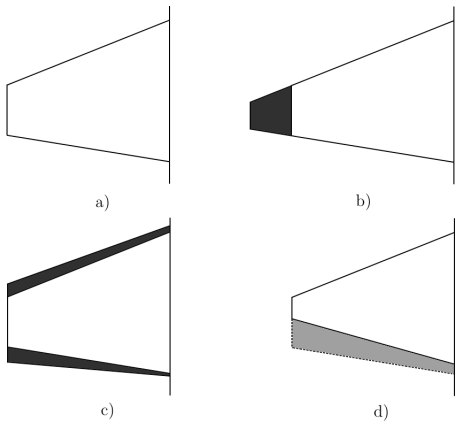

Figure 5.1: a) Original; b,c) augmented; d) decreased wings.

Figure 5.1 gives a general idea of the process. Imagine that induced drag has been calculated for the original wing *a)*. Then, in a next iteration, the optimizer would decide to increase the wingspan, without changing the direction of both trailing and leading edge. In a perfect situation, this would simply require calculate the drag contribution on the shadowy area. We can also consider the cases shown by *c)* and *d)*, where the wing is modified in different ways.

The problem of implementing this idea is that the original VLM adapts the panels' size to fit in the considered wing (top of Figure 5.2). This approach has a clear inconvenience. If, for example, the root chord is modified, even with a small step, it will affect all the panels and the downwash matrix must

be fully recalculated.

Another approach, more in accordance with the incremental view, is representing the grid has depicted at the bottom of Figure 5.2. By building a fixed grid, with the panels' borders aligned with the chords, the panels are simply marked as being in or out of the wing coverage. This will lead to rectangular panels on the wing's centre and triangular or trapezoidal shapes in the borders.

The advantage of this second grid is that, for the same example of changing the root chord's length, only the panels located at the borders are affected. Hence, it is only needed to update part of the downwash matrix, which can be accomplished by the factorization update techniques previously described in Chapter 4.

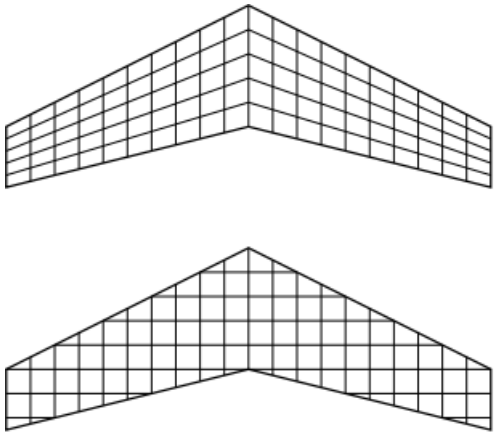

Figure 5.2: The original grid on the top, the linearized at the bottom.

5.4.1 Potential speed-up analysis

The LU update analysis concerning the optimization process was performed through the replacement of columns and rows in a matrix of size 400x400, with the results shown in Figure 5.3 and Table 5.2 (at the end of this Chapter). The changes are made in pairs, i.e. one column update is always accompanied by a row update. The justification is that in the VLM, due to the interactions between all the panels, changing a column necessarily means changing a row too. Therefore, Figure 5.3 shows a total of 2 to 800 changes.

A Python version of a Gaussian elimination solver was implemented, instead of using the optimized BLAS [35] libraries via the Numpy and Scipy

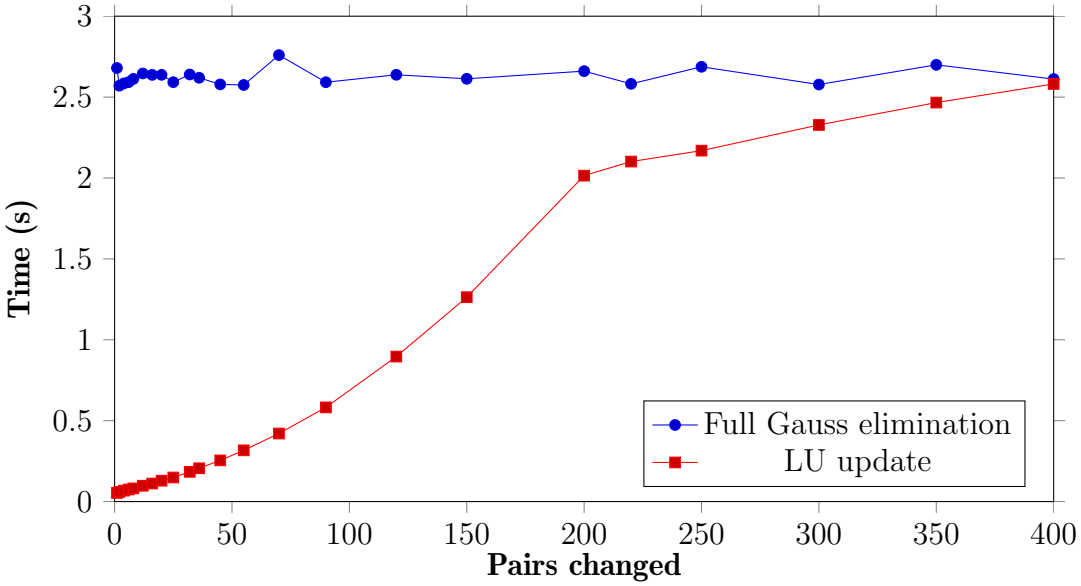

Figure 5.3: Time comparison for the LU factorization update.

packages. The reason for this is that the LU update code was all developed in Python, while BLAS is compiled in C and Fortran, making the time comparison unfair.

Finally, the decomposition time is not taken into account. This is justified by the fact that the decomposition, in the LU update, is performed only once. This time will then be diluted in the total execution of the algorithm and consequently, can be considered negligible for the current analysis.

A maximum speed-up of 50 is reached, though this value corresponds to a single pair change, an unlikely event in a VLM run. To get a better idea of the average speed-up, let us take the example of a wing covered by a grid of 14x28 panels (392 panels in total). If only the panels in the edges are altered, the group will be sized around 28 to 46 panels, depending on whether both the leading and trailing edges are affected or not. For the chord axis, 14 panels would be changed at most, as the root panels are fixed and only the tip position is changed. This gives a lower and an upper bound of 14 and 46 panels. Looking at Table 5.2, these values match speed-ups between 10 and 27. Even in the worst case scenario, a speed-up of 10 is quite remarkable.

5.4.2 Implementation of the linearized grid

The idea of incrementalization through a linearized grid was not further explored due to unstable preliminary results. It required a much large number of panels (up to 100 times more) to achieve the same lift coefficient as the rigid grid. Divergence was also registered for configurations where it did not occur with the original vortex lattice method.

Reference [39] states that the panels should have an aspect ratio between one and three. While a value of three is suited for a transonic flight, supersonic flights require an aspect ratio close to one. Besides, for better results, the shape of the panels should follow the wing geometry, having a greater concentration on discontinuity points, i.e. at the tip borders.

Both conditions cannot be guaranteed by the linearized grid, although the first, keeping a correct aspect ratio, is easier to achieve than the panel density. This might be the reason why the linearized grid did not produce correct and stable results during the experiments conducted.

Nevertheless, the idea may still be held for future work because this grid format has actually been considered before [24]. Following the linearized theory of airfoils, Carlson and Miller proposed an approach valid for calculating the pressure distributions in supersonic flows.

5.5 Optimizer parallelization

The advantage of using direct search algorithms, which samples new points around the current iterate, is that it makes the optimization process embarrassingly parallel. As the design variables are tested independently from one another, the evaluations can potentially be split by the same number of cores.

The construction of the downwash matrix can be easily parallelized as well. Since the matrix elements are independent, in the limit, they could be assigned to a different thread each. Besides, with the working space so well defined, there will not be any conflicts in memory access. Locks are then unnecessary, avoiding in this way one of the pitfalls of parallelization effectiveness.

Therefore, tests were made concerning the use of general-purpose graphics processing unit (GPGPU) in the downwash calculation.

5.5.1 Potential speed-up analysis

In order to assess the potential of the GPGPU in downwash calculation, a simple case test was conducted, using the wing design that will be presented for the Diederich's method validation (see Section 6.1). All the code was developed using the standard OpenCL [36], a language for GPGPU. Code was developed in C/C++.

As can be seen in Figure 5.4, the CPU has a clear advantage over GPU, presenting values around 4 to 6 times faster. However, it is also clear from Figure 5.5 that the CPU time grows quicker than the GPU. This graph was built with the ratios between the time of each evaluation and the time of the base test with 150 panels.

This leads to the conclusion that eventually, for a large number of panels, the GPU may actually outperform the CPU. However, due to the difference in execution time registered for this wing, it is also true that this breakpoint may not stand for a reasonable number of panels.

One of the problems encountered was that the GPU was limited to 4100 working units, meaning not all the positions could be calculated at the same time for the larger test cases. The solution was to implement a loop, only filling a single row on each turn. Although the time complexity decreases from $O(N^2)$ to $O(N)$, it is still slower than constant time, achieved if all the positions were calculated in parallel.

5.5.2 Implementation of the optimizer parallelization

While the parallelization of the downwash calculation was made using GPGPU, the optimizer parallelization was accomplished using the OpenMP API [38], which provides an easy and simple solution for multithread management.

Concerning the parallelization of the algorithms, refer that the analysis of the design variables was split into two by its two categories (wing geometry and beam variables, see Table 5.1). The reason behind this is that when was tried an uniform approach, treating all the variables together, the solutions tended to diverge or stop too early.

Therefore, a first iteration is conducted, looking for the best Breguet range increase in the wing geometry design space. For the potentially optimal points of this space, a second phase is conducted, now using the beam variables.

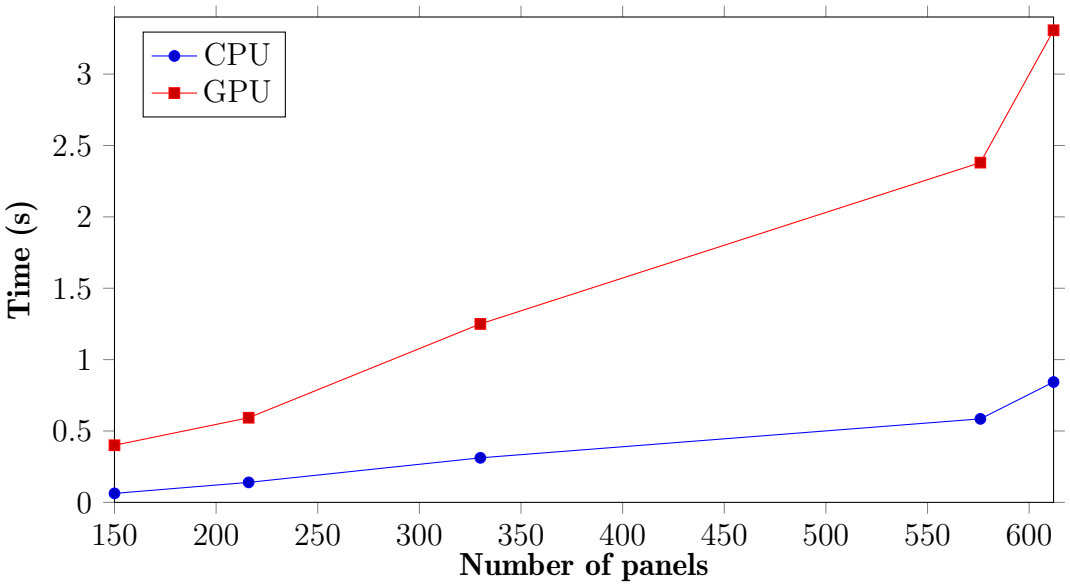

Figure 5.4: Downwash calculation over CPU and GPU.

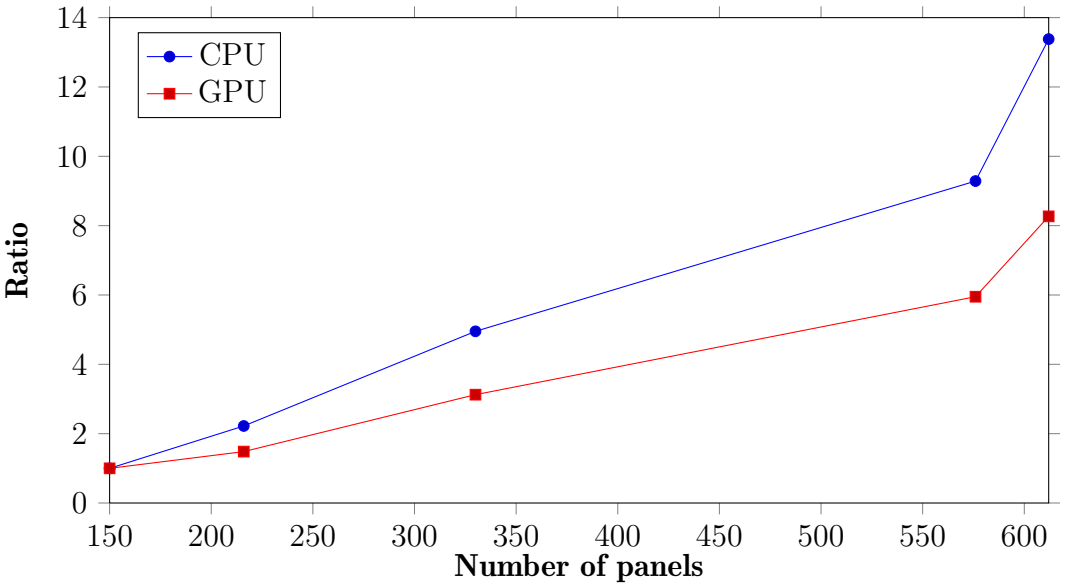

Figure 5.5: Time growth proportion for the downwash calculation.

Again, for reaching stable results, the optimization in this phase is cumulative, meaning the internal diameters are tested and, if modified, the new values are used in the thickness.

Because of this, it was hard to create a parallel solution for the structural analysis and still promote a good optimization performance in terms of objective function evaluation. Consequently, the structural optimization is done sequentially.

5.6 Structural and Gauss-Seidel solutions reuse

Taking advantage of the fact that optimization evaluations are not independent from each other but consist in a progressive refinement of the current iterate, there is no need to start every analysis from the scratch. Instead, the structural distributions may be initialized with the values found by the parent node, only using zero values in the first iteration.

The same technique may be applied to the Gauss-Seidel method. Instead of initializing the approximation $\tilde{x}$ with zero or random values, the method can start from the vector reached by the node that originated the current iterate, hopefully bringing the initial guess closer to the true solution.

5.6.1 Potential speed-up analysis

To assess the speed-up offered by the reuse of solutions, the following experiment was made. A normalized configuration was used (for the optimization boundaries, see Section 5.2) with a wingspan of 0.8, root chord of 0.1, tip chord of 0.07, sweep angle of 0.5, twist of 0.95, dihedral angle of 0.3, root and tip internal radius of 0.01, thickness in the root of 0.14 and thickness in the tip of 0.01. The angle of attack was set to 7.5º and the precision to $1e^{-5}$.

Starting from this point, each variable's value is modified by a given percentage, from 0.5% to 5% and with a 0.5% step. In order to promote feasible solutions, the percentage of change reflects increments in the wingspan, sweep angle, twist and dihedral angle, decreasing the values for all the other design variables.

The results obtained are illustrated in Figure 5.6, with a decreasing speed-up of around 1.3 for 0.5% of change and 1.15 for 5% of change. The "Standard"

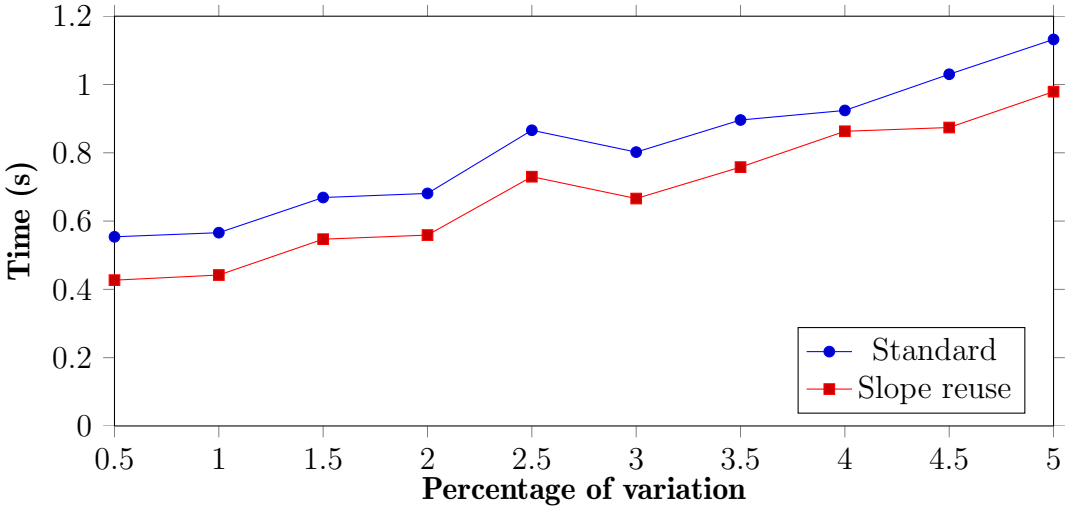

Figure 5.6: Influence of the incremental approach on successsive iterations.

label corresponds to the execution of two consecutive VLM evaluations without any solutions reuse. Note that the percentage of change is always applied to the original wing and so, the modifications are not cumulative.

The results indicate that, as expected, the smaller the variation, the greater is the speed-up. However, this improvement is not as significant as initially expected, speeding up only to around 30% for the best case.

Besides, it was also concluded that the incrementalization of the structural loop has a greater importance than in the Gauss-Seidel method. In fact, the speed-up for the Gauss-Seidel seems negligible, although when combining both solutions, it tends to be greater than the sum of the individual contributions.

5.7 Gauss-Seidel precision

Due to nature of the structural loop, it does not require full precision in all iterations, as the first iteration will likely be relatively distant from the final solution. The trick here is to start with a smaller precision for the Gauss-Seidel method (either by reducing the maximum number of iterations allowed or increasing the convergence tolerance) for the initial iterations of the loop, and by tightening it as the algorithm progresses, provided both loops converge successfully.

This technique has some problems that should be carefully considered. First, reducing excessively the number of Gauss-Seidel iterations may have undesired effects, as it will require more iterations from the structural loop. Since this loop is computationally heavier than Gauss-Seidel's, it will have the opposite effect, increasing the solving time.

Second, a lower precision may make the evaluation results invalid. Imagine that two wings, A and B, are evaluated, A having a greater Breguet range when using high precision. However, B has an associated Breguet range close to the one of A, so close that when evaluating it with lower precision, the value output by the model may actually be greater than the one of A.

If this happens too many times, the optimizer may be misguided and take directions that should otherwise avoided. Once in the wrong path, it may be hard for the optimizer to recover, even when precision is later increased.

5.8 Concluding remarks

Several algorithmic improvements were discussed. They encompass changes to the original method, introducing a linearized grid instead of the standard format. Also, alternative views to the problem of solving systems of linear equations were considered, both in terms of the LU factorization update and iterative approaches, which can directly use incrementalization by starting from previously calculated distributions.

The speed-ups for the simpler cases show that it is valid to consider each one of them for the optimization process, though the real results of their implementations are going to be viewed on the next Chapters. Some of the ideas, such as the use of the GPGPU, will not be developed any further, as the results presented are not sufficiently satisfactory.

Pairs changed	*Full solving time (s)*	*LU update time (s)*	*Speed-up*
1	2.6800	0.0535	50.09
2	2.5705	0.0598	42.98
4	2.5849	0.0678	38.13
6	2.5923	0.0747	34.70
8	2.6126	0.0816	32.02
12	2.6460	0.0979	27.03
16	2.6373	0.1115	23.65
20	2.6377	0.1291	20.43
25	2.5923	0.1489	17.41
32	2.6398	0.1840	14.35
36	2.6195	0.2058	12.73
45	2.5784	0.2546	10.13
55	2.5749	0.3168	8.13
70	2.7597	0.4208	6.56
90	2.5920	0.5821	4.45
120	2.6380	0.8963	2.94
150	2.6131	1.2632	2.07
200	2.6605	2.0145	1.32
250	2.6873	2.1691	1.24
300	2.5777	2.3279	1.11
350	2.6995	2.4662	1.09
400	2.6114	2.5812	1.01

Table 5.2: Speed-ups achieved with the LU update.

6

Results

The results can be divided into three main categories. The first is related to the validation of the models, an aspect highly valued, in order to guarantee that all the improvements implemented may have a real application. For the validation of Diederich's method, the base of comparison was provided by Marc Mulkens, while for the vortex-lattice method, it was taken from the bibliography.

The second category encompasses a comparison using the three the methods analyzed, namely the lifting-line theory, Diederich's method and the vortex lattice method. The lift coefficient and the shape of the lift distribution are compared.

Finally, the third category is related to the optimization results, by running DIRECT and the hill-climber over grids with different panel densities. Each improvement discussed in Chapter 5 was firstly activated alone, to assess the

Diameter / Twist	**-10**	**-8**	**-6**	**-4**	**-2**	**0**	**2**	**4**	**6**	**8**	**10**
dr*1.5	122	109	98	90	85	83	84	87	94	103	115
dr*1.4	123	110	99	90	85	83	83	87	92	102	114
dr*1.3	126	111	100	92	86	83	83	86	91	100	111
dr*1.2	130	115	103	94	88	84	83	84	89	95	106
dr*1.1	146	129	115	104	95	88	84	83	84	87	92

Table 6.1: **Induced drag coefficient** results (normalized by $1e^{-5}$).

Diameter / Twist	**-10**	**-8**	**-6**	**-4**	**-2**
dr*1.5	-0.82%	0.0%	0.0%	0.0%	0.0%
dr*1.4	-1.63%	-0.91%	0.0%	-1.11%	-1.18%
dr*1.3	-0.79%	-1.8%	-1.0%	0.0%	0.0%
dr*1.2	-1.54%	-1.74%	-1.94%	-1.06%	0.0%
dr*1.1	-0.68%	-0.78%	-0.87%	-0.96%	-1.05%

Table 6.2: Deviations for **induced drag coefficient** I.

Diameter / Twist	**0**	**2**	**4**	**6**	**8**	**10**
dr*1.5	0.0%	1.19%	0.0%	1.06%	1.94%	0.87%
dr*1.4	0.0%	0.0%	1.15%	0.0%	0.98%	1.75%
dr*1.3	0.0%	0.0%	1.16%	0.0%	1.0%	1.8%
dr*1.2	0.0%	0.0%	0.0%	1.12%	0.0%	1.89%
dr*1.1	-1.14%	-1.19%	0.0%	1.19%	1.15%	0.0%

Table 6.3: Deviations for **induced drag coefficient** II.

speed-up it could provide. In the end, all the improvements were used simultaenously to return the speed-ups obtained with the overall solution. The wing configurations achieved by each optimizer are also presented.

6.1 Diederich's method validation

The first validation set concerns Diederich's method. Tables 6.1 and 6.4 describe the output of the author's implementation, while Tables 6.2, 6.3, 6.6 and 6.7 refer to deviations registered relatively to Marc Mulkens's output.

Here, a single wing was used. It has an aspect ratio of 7.8, area of 51.18m^2, taper ratio of 0.25, sweep angle of 22.73º, an internal root radius of 0.205m, an internal tip radius of 0.06m and a thickness at the tip of 0.001m. Only the thickness at the root suffered variations, from 0.1% to 0.5% relatively to the internal root radius, with steps of 0.1% (as described in the first column of the Tables here presented).

Diameter / Twist	**-10**	**-8**	**-6**	**-4**	**-2**
dr*1.5	2334	2446	2542	2614	2660
dr*1.4	3447	3613	3754	3862	3929
dr*1.3	4502	4719	4903	5043	5131
dr*1.2	5498	5763	5987	6159	6266
dr*1.1	6437	6747	7010	7211	7337

Table 6.4: **Breguet range** results I, in kilometers.

Diameter / Twist	**0**	**2**	**4**	**6**	**8**	**10**
dr*1.5	2676	2660	2614	2542	2446	2334
dr*1.4	3952	3929	3862	3754	3613	3447
dr*1.3	5161	5131	5043	4903	4719	4502
dr*1.2	6303	6266	6159	5987	5763	5498
dr*1.1	7379	7337	7211	7010	6747	6437

Table 6.5: **Breguet range** results II, in kilometers.

Relatively to the results, there is little to be said, as the deviations registered are quite low and are likely due to different approaches when approximating the empiric functions defined in [5]. Therefore, this work's implementation of the Diederich's method can be considered valid.

6.2 Vortex lattice method validation

Unlike the validation accomplished for the Diederich's method, more than a single wing was used to validate the vortex lattice method (VLM). Several properties of the wing are separately tested throughout five different cases, considering aspects such as the flexibility of the wing (in the aeroelastic analysis) or the use of twist and dihedral angle.

The test cases are arranged by their features' coverage. The results are compared with the ones provided in the bibliography. It encompasses results from other VLM software, namely Tornado [32] and SURFACES [33].

Diameter / Twist	**-10**	**-8**	**-6**	**-4**	**-2**
dr*1.5	0.22%	0.21%	0.16%	0.12%	0.04%
dr*1.4	0.35%	0.31%	0.24%	0.18%	0.1%
dr*1.3	0.43%	0.39%	0.33%	0.26%	0.18%
dr*1.2	0.49%	0.47%	0.4%	0.33%	0.24%
dr*1.1	0.41%	0.39%	0.37%	0.32%	0.28%

Table 6.6: Deviations for **Breguet range** I.

Diameter / Twist	**0**	**2**	**4**	**6**	**8**	**10**
dr*1.5	-0.07%	-0.19%	-0.27%	-0.39%	-0.49%	-0.59%
dr*1.4	0.0%	-0.1%	-0.23%	-0.34%	-0.49%	-0.6%
dr*1.3	0.08%	-0.08%	-0.2%	-0.34%	-0.48%	-0.63%
dr*1.2	1.72%	0.0%	-0.16%	-0.32%	-0.48%	-0.64%
dr*1.1	0.21%	0.12%	0.03%	-0.08%	-0.21%	-0.32%

Table 6.7: Deviations for **Breguet range** II.

6.2.1 Case 1 - Bertin & Smith wing

The Bertin & Smith wing is represented in Figure 6.1, with the image scale variable b (not to be confused with the b for the wingspan) set to 1. This means the tested wing has a span of 1m, a constant chord along the chord of 0.2m, a sweep angle of 45°. No dihedral angle or twist are applied and the number of panels used was 1 along the chord axis and 4 for the wingspan.

This example is taken from [24], where it is accompanied by a detailed exaplanation of the implementation, including intermediate results. The deviation of 0.32% for this first case is explained by the round-off applied in the original example, a situation which does not occur in SURFACES, which outputs the same lift coefficient value as the author's implementation.

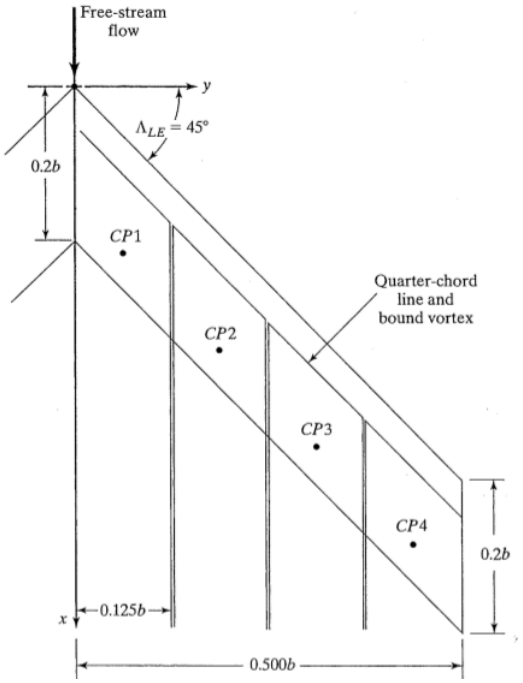

Figure 6.1: The Bertin & Smith wing configuration.

6.2.2 Case 2 - Rigid and unswept wing

The tested wing is the same as the one used in the previous case. The only difference is that instead of setting a fixed angle of attack, Figure 6.2

Method	C_L *coefficient*	*Deviation*
Bertin & Smith	0.05992%	0.32%
SURFACES	0.06011	0.00%
Tornado	0.06080	1.13%
VLM	0.06011	- - -

Table 6.8: Bertin & Smith wing results.

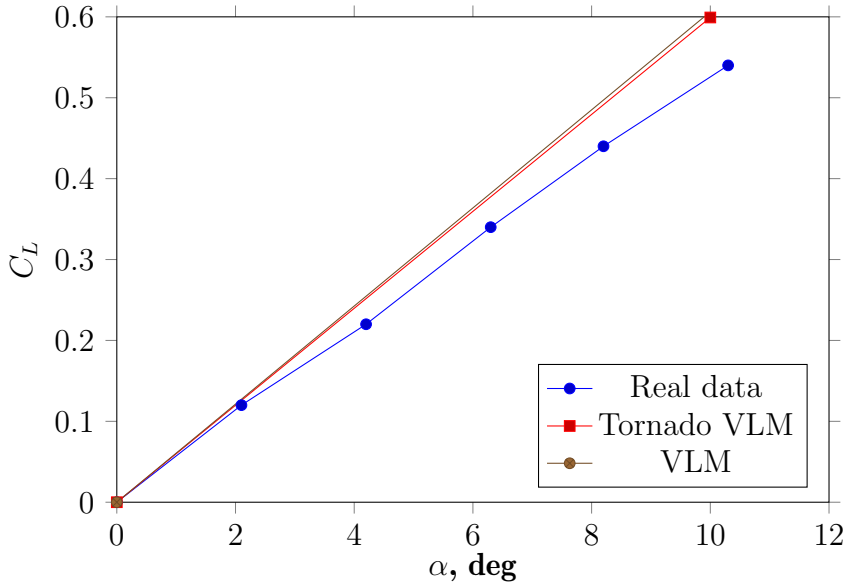

Figure 6.2: Comparison between real data and two VLM implementations.

shows the variation of the lift coefficient C_L with the angle of attack α. The graph is plotted from the results available in [24], [32] and [33]. The line labeled as "Tornado VLM" represents the results for the Tornado VLM software [32], while the line "Real data" is, as the name indicates, built from real data collected and described in [24]. The graph shows that the author's implementation has the same deviation relatively to the real data as the Tornado line, a deviation resulting from approximations of the theoretical model.

6.2.3 Case 3 - Flat plate airfoil (0º and 35º sweep angle)

The classic method referred in Tables 6.9 and 6.10 is the one described in [34], which gives the expected results based on theoretical results. The number of panels used was 8 along the chord axis and 32 for the wingspan. The wing has

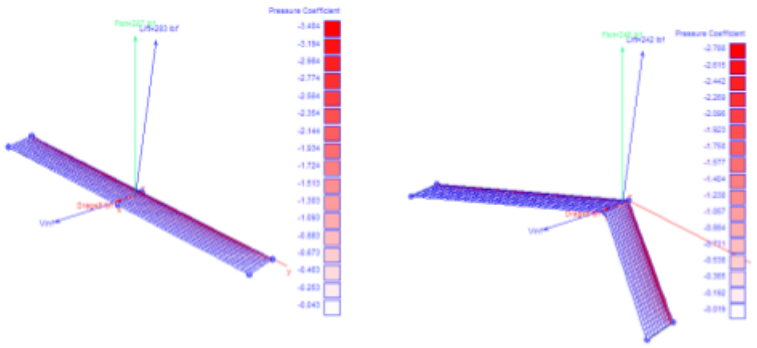

Figure 6.3: The flat plate wing configuration, with 0º and 35º sweep angle.

an aspect ratio of 20, unity root and tip chords and with the sweep angle 0º for the first example and 35º for the second.

Though the implementation used deviates 4.29% from the classic method, it returns a result close to SURFACES, with variations only for the third decimal place of significant digits.

Method	C_L *coefficient*	*Deviation*
Classic method	0.885	4.29%
SURFACES	0.845	0.29%
VLM	0.847	- - -

Table 6.9: Flat plate airfoil with 0º of sweep angle results.

Method	C_L *coefficient*	*Deviation*
Classic method	0.748	3.07%
SURFACES	0.723	0.24%
VLM	0.725	- - -

Table 6.10: Flat plate airfoil with 35º of sweep angle results.

6.2.4 Case 4 - Warren 12 wing

In reference [33] this wing is represented as the standard wing for VLM validation. Three different grids were considered, with panels along the chord axis

and wingspan being respectively 6x16, 8x24 and 16x36. The wing configuration can be found in Figure 6.4.

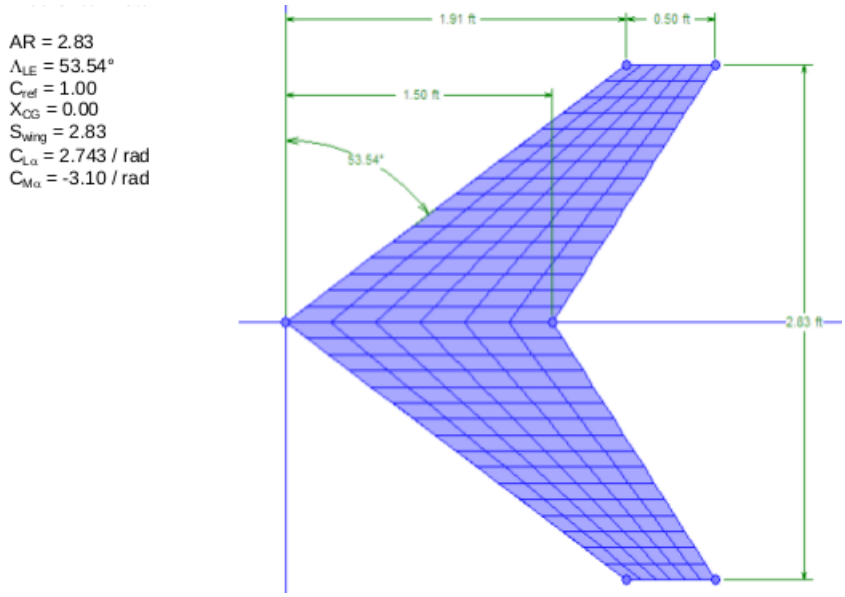

Figure 6.4: The Warren 12 wing configuration.

For the three cases, the author's implementation gives the same results as SURFACES. This is the last validation case where no twist or dihedral angle is applied. All the wings analyzed so far do not consider the Z camber axis.

Method	C_L *coefficient*	*Deviation*
SURFACES (6x16)	0.487	0.0%
VLM (6x16)	0.487	- - -
SURFACES (8x24)	0.485	0.0%
VLM (8x24)	0.485	- - -
SURFACES (16x36)	0.483	0.0%
VLM (16x36)	0.483	- - -

Table 6.11: Warren 12 wing results.

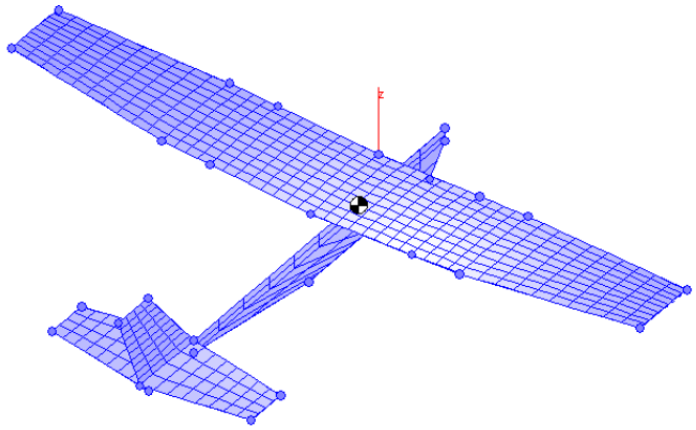

Figure 6.5: The Cessna 172 wing.

6.2.5 Case 5 - Cessna 172 wing

The Cessna 172 wing has a wingspan of 11m, 1.62m for root chord, 1.13m for tip chord, an angle of attack of 1.3º, an approximated sweep of 2.6º, -3º of twist in the tip and finally, 1.44º of dihedral angle. The lift coefficient C_L was calculated using a grid of panels with size 16x32. As can be seen in Figure 6.5, the Cessna 172 wing is not completely uniform, having two sections with different sweep angles. As this cannot be achieved by this implementation and the variation in the original wing is not significant, the global sweep angle was approximated.

Despite these shortcomes, this example was used due to the difficulty in finding another with full wing description and the corresponding lift coefficient. Besides, the variation of the sweep angle is not very large and hence, an approximation at this level is compatible with the validation of the overall model. This example encompasses all the wing properties, requiring the use the three analysis axis, including the Z camber axis.

The author's implementation does not exactly match any of results produced by the other software, though the lift coefficient returned is neither too distant from any of them (except maybe for "TEST Data"). The deviations can be explained, for the cases of SURFACES and Tornado, by the greater detail implemented by their software, as for example, they consider the effect of the chamber thickness. This difference was not visible when validating the untwisted and dihedral-less wings. As for "TEST Data" and AVL, the documentaion was vague and therefore, little explanations can be drawn for

Method	C_L *coefficient*	*Deviation*
TEST Data	0.080285	14.22%
AVL	0.086917	5.5%
Tornado	0.092089	0.42%
SURFACES	0.090413	1.42%
VLM	0.091699	- - -

Table 6.12: Cessna 172 wing results.

them.

6.3 Model comparison

In this Section, a comparison between the three models will be presented, in order to understand what is at stake when choosing between one of them. The wing configurations were naturally chosen for scenarios where the compared models are valid, also to guarantee that, when used correctly, they are in accordance with each other.

The comparison starts with LLT and VLM, considering the lift curve slope, followed by Diederich's method against VLM, comparing the shapes of the lift and slope distributions.

6.3.1 Lift-line theory and vortex lattice method

The first case of study concerns lifting-line theory (LLT) and the vortex lattice method (VLM), where the coefficients produced by the wing are compared. The example comes from [24], where the untwisted wing used is shaped by a wingspan of 4.572m, root chord of 0.726m, tip chord of 0.290m and a -1.2º zero-lift angle of attack.

Figure 6.6 shows that the LLT and VLM are in accordance with lift coefficient produced by the different angles of attack. This was expected, as [1] states that the LLT can produce reliable results for unswept and thin wings, for low values of the angle of attack.

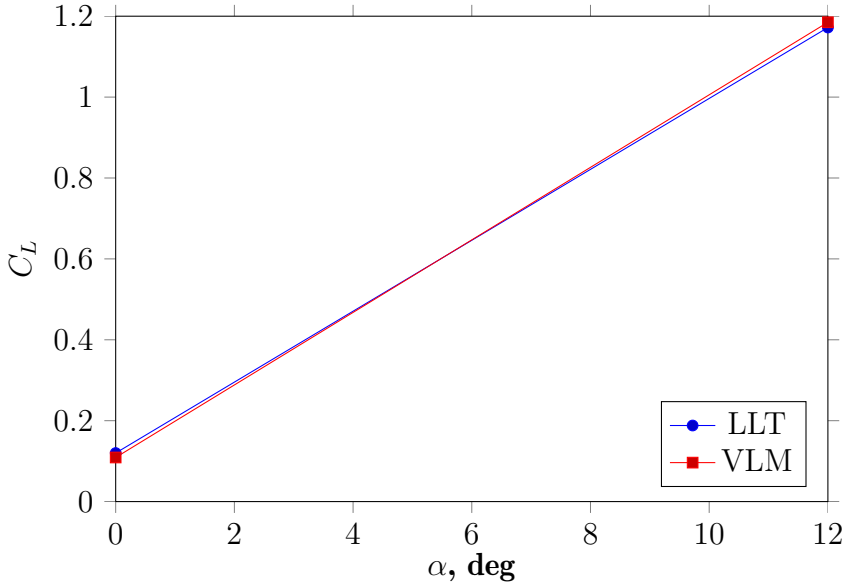

Figure 6.6: Lift coefficients produced by LLT and VLM.

6.3.2 Diederich method and vortex-lattice method

Next, Diederich's method and the VLM lift and slope distributions are compared. For collecting data, the Diederich's validation wing was used (see Section 6.1). As referred in Chapter 2, the two methods model the lift coefficient differently. While Diederich's method received this coefficient as input, VLM uses the angle of attack to adjust it. Therefore, to match the 0.45 lift coefficient used in Diederich's, the angle of attack in VLM was set to 5.497°.

Figure 6.7 shows that the shapes for the lift distributions produced by the Diederich's method and the VLM are distinct for the same wing. They have a mean difference of 16.98%, with a maximum difference of 21.66% and minimum of 12.60%. The VLM distribution ranges from a minimum of 0.205 and a maximum of 0.755, while the Diderich's distributions ranges from 0.063 to 0.598.

This naturally leads to different slopes reached by each method, with their differences depicted in Figure 6.8. The deviations registered are according to what is stated in [1], which refers that Diederich's method may differ from real values by a margin of 10% to 15%. Taking in account that the VLM is neither a highly precise method, the mean difference of around 17% may be considered acceptable.

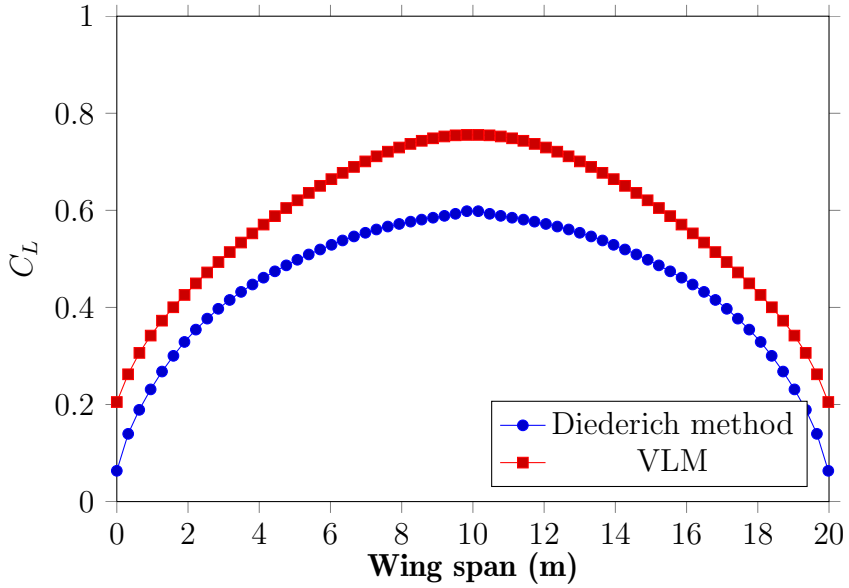

Figure 6.7: Lift distributions produced by Diederich and VLM.

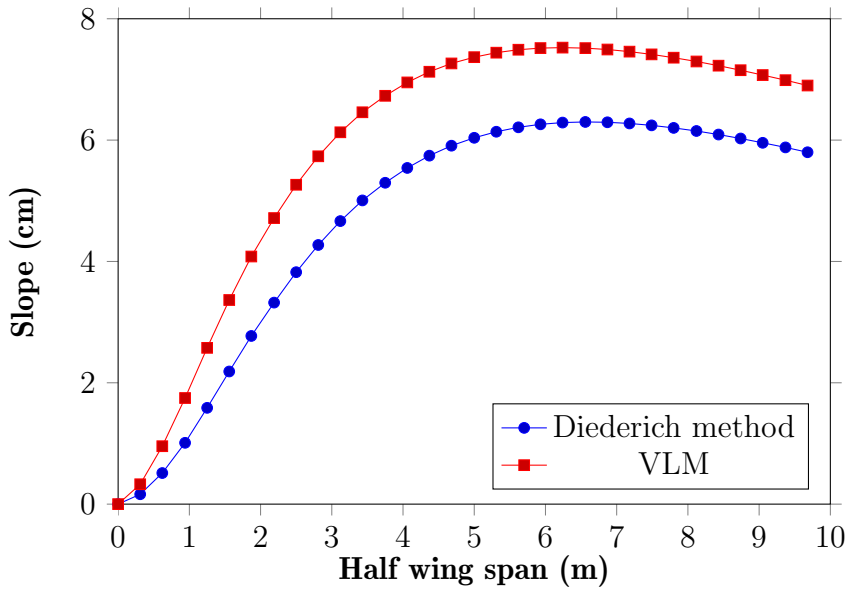

Figure 6.8: Slope distributions produced by Diederich and VLM.

6.4 Optimization results

All the tests were made using a precision value of $1e^{-5}$ and with grids of 10x28, 14x32 and 16x38 panels. The blue lines correspond to the use of the Gaussian elimination (GE) to solve the system of equations, while orange stands for the Gauss-Seidel (GS) method (see Section 4.2). If this text was printed in

greyscale, lines 1, 3, 5 and 9 are related to the GE, while the others correspond to the GS.

The first line of the Tables ("Standard") corresponds to the version of the code without any improvements and using Gauss-elimination. The speed-ups are relative to the standard version of the optimizer, while the last column corresponds simply to the mean time taken by each iteration, in order to calculate a relative speed-up.

All the tests, for both DIRECT and hill-climber, were run 30 times. As stated in Section 5.1, they were executed on different machines, as the hill-climber, took 2 to 3 times longer to conclude in the LG R510. The Tables show the mean execution time, the standard deviation, the number of iterations, the gross speed-up and the speed-up proportional to the number of iterations, for each set of runs.

6.4.1 DIRECT Algorithm

The first set of Tables shows the results for DIRECT (see Section 3.3.1). The maximum-speed up reached by the different grids varies from 1.92 to 4.09, depending on the number of panels. Note that these speed-ups concern the ration between the absolute times, eventually differing if the time taken by a single iteration is considered.

The incremental approaches used for speeding up the execution were not as successful as initially hoped. This was expected after analysing the preliminary results of Section 5.6. Nevertheless, they still provide a relatively significant contribution to the final speed-up. Also, when activated simultaneously, the data reuse in both the structural loop and the Gauss-Seidel solution proved to be faster than being separately activated and having the individual contributions added.

The reason for a low performance of the solutions reuse, both in the Gauss-Seidel method and the structural loop, was naturally investigated. Both loops share a common behaviour, which can be seen in Figure 6.9. It represents the sum of the differences between the current slope distribution and the previous one (see Section 2.4.2). At the beginning, the loop has a fast convergence speed, significantly modifying the distributions, but it quickly drops as the number of iterations increase.

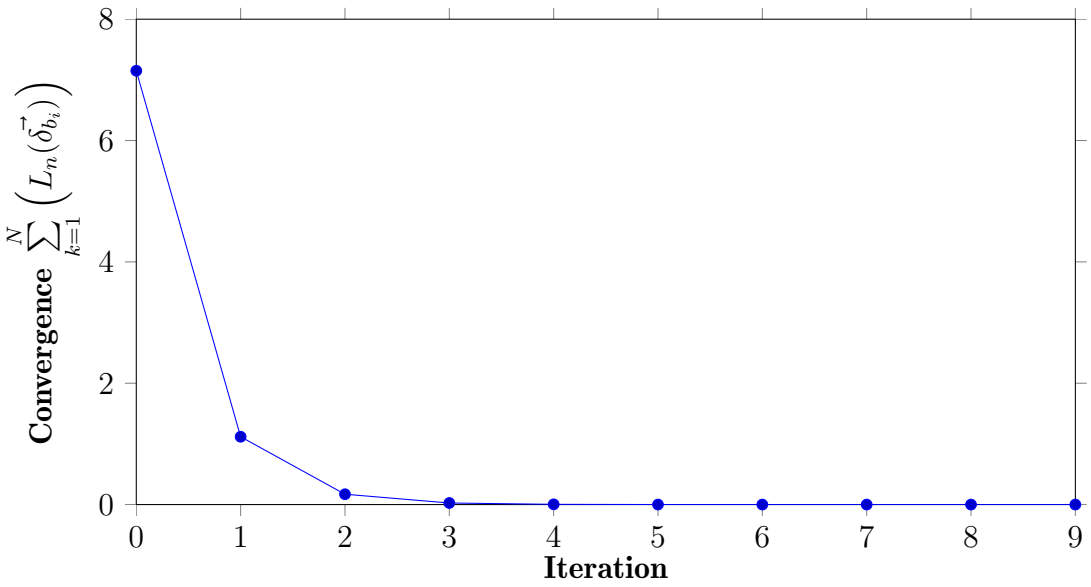

Figure 6.9: The convergence rate behaviour of the structural loop.

Regardless of the precision used, activating this option invariably reduced by a factor of two the number of iterations. Looking at the graph, these first two iterations present the greater variation. By reusing the solution, this initial phase is cut off, but unfortunately, it is quite small when compared to the time taken to conclude the slow convergence phase.

This means that the reuse has a greater impact in terms of speed-up for lower precision. If with lower precision the structural loop used four iterations, cutting two of them means a speed-up of 2. If the precision is increased and the loop now takes eight iterations, cutting the same two iterations result in a speed up of only one third. This explains why the solutions reuse tend to provide poor performance in the optimization execution.

Finally, the speed-up gained by using parallelization increases with the number of panels. As was stated before in Section 5.5.2, only the wing geometry analysis was parallelized and therefore, the speed-up may be lower than the number of cores, ideally achieved. However, the speed-up achieved solely by this component grows from the first case to the last, meaning that as the number of panels is increased, the weight of the wing geometry analysis over optimization grows comparatively to the structural analysis.

In terms of effectiveness, the improvements tend to produce the same ef-

Improvement	Mean time (std)	No. iterations	Speed-up	Mean it. speed-up
Standard	0min 58s (72ms)	26	-	-
Standard	0min 47s (69ms)	26	1.23	1.23
Parallelization	0min 43s (156ms)	26	1.34	1.34
Parallelization	0min 34s (90ms)	26	1.71	1.71
Structural reuse	0min 40s (30ms)	26	1.45	1.45
Structural reuse	0min 32s (41ms)	23	1.81	1.60
GS solution reuse	0min 46s (61ms)	26	1.26	1.26
GS Precision	0min 43s (54ms)	28	1.34	1.45
All	0min 27s (160ms)	28	2.14	2.31
All (Sequencial)	0min 21s (23ms)	32	2.76	3.40
All	0min 30s (270ms)	32	1.92	2.38

Table 6.13: Results for DIRECT algorithm with a 10x28 grid.

fect when using Gauss-elimination or the Gauss-Seidel method, with a small advantage to the second. Although the results show a greater difference, the time is always compared to the "Standard" Gauss-elimination. If compared with the "Standard" Gauss-Seidel, the differences tend to be less significant.

6.4.2 Hill-climber

The results for the hill-climber show a different perspective than the one for DIRECT. Although the improvements applied to both of them deliver greater speed-ups for a larger number of panels, in the first grid configuration, Gaussian elimination actually runs faster than the Gauss-Seidel method, even if the difference is less notorious when all the improvements are used.

However, as the number of panels grows, this situation is inverted. This can be explained by the fact that loops tend to have a fast convergence at the beginning, slowing down as the final solution is approached, as mentioned earlier. Also, DIRECT and the hill-climber have a different approach. While the first defines larger steps at the beginning, reducing the jump progressively, the hill-climber has a more local approach, defining smaller steps along the whole optimization process.

Improvement	Mean time (std)	No. iterations	Speed-up	Mean it. speed-up
Standard	3min 30s (139ms)	31	-	-
Standard	2min 41s (150ms)	31	1.30	1.30
Parallelization	2min 26s (258ms)	31	1.44	1.44
Parallelization	1min 54s (182ms)	31	1.84	1.84
Structural reuse	2min 31s (202ms)	31	1.39	1.39
Structural reuse	2min 1s (141ms)	31	1.74	1.74
GS solution reuse	2min 41s (194ms)	31	1.30	1.30
GS Precision	2min 25s (144ms)	31	1.45	1.45
All	1min 29s (171ms)	33	2.36	2.51
All (Sequencial)	1min 31s (188ms)	34	2.31	2.53
All	1min 4s (117ms)	34	3.28	3.60

Table 6.14: Results for DIRECT algorithm with a 14x32 grid.

Improvement	Mean time (std)	No. iterations	Speed-up	Mean it. speed-up
Standard	6min 45s (166ms)	28	-	-
Standard	4min 47s (164ms)	28	1.41	1.41
Parallelization	4min 41s (459ms)	28	1.44	1.44
Parallelization	3min 20s (279ms)	28	2.02	2.02
Structural reuse	4min 38s (224ms)	28	1.45	1.45
Structural reuse	3min 41s (178ms)	28	1.83	1.83
GS solution reuse	4min 46s (453ms)	28	1.42	1.42
GS Precision	3min 42s (521ms)	28	1.82	1.82
All	3min 8s (300ms)	28	2.15	2.15
All (Sequencial)	2min 21s (355ms)	28	2.87	2.87
All	1min 39s (628ms)	28	4.09	4.09

Table 6.15: Results for DIRECT algorithm with a 16x38 grid.

This means that in the hill-climber, the loops' faster convergence period is shorter than in DIRECT. Therefore, it is natural that the several improvements implemented cannot achieve the same speed-ups. Nonetheless, a maximum speed-up of 2.61 shows that the cases studied here can be successfully deployed in different contexts, although with less effect.

Also, note that in the transition from a grid of 14x32 to 16x38 panels, the overall speed-up is actually reduced for both telimination and Gauss-Seidel method. Nevertheless, the increasing gap between the two methods, in favour of Gauss-Seidel, is maintained.

The same tests were executed too on the same machine as DIRECT, although for only 5 runs. This was made in order to find the influence of the architecture and the compiler version (G++ 4.6 on the ECOS cluster against the G++ 4.7 on the LG R510) over the results. Table 6.19 shows that for the same environment, the results between the two algorithms are much more similar than when executed in different machines.

Although the maximum speed-up is only of 3.00 when DIRECT reached a maximum of 4.09, the tendency is that for a larger number of panels, the speed-up is increased, too. Also, the Gauss-Seidel always produces better results than the elimination version, something that did not necessarily happen when executing the code in the cluster machines.

6.4.3 Final configurations

Table 6.20 summarizes the configurations reached by the two optimizers. The column "Value" defines the optimization output, while the column "Per." translates it into the normalized design space (see Table 5.1 for the optimization bounds).

Both DIRECT and the hill-climber reached similar configurations, arriving at long and thin wings. In fact, the upper and lower bounds are reached by several variables, with special incidence on the wingspan and the chords.

Concerning the sweep, twist and dihedral angles, those are kept in values around the 0^{o}. As for the structural model, clearly the wings with greater weight at the root are favoured. Looking at Table 6.20, although presenting low values as well, the diameter and the thickness at the root are still relatively far from the lower bound, a situation that does not occur for the tip.

Improvement	Mean time (std)	No. iterations	Speed-up	Mean it. speed-up
Standard	1min 7s (11ms)	46	-	-
Standard	1min 10s (70ms)	46	0.96	0.96
Parallelization	0min 38s (41ms)	46	1.74	1.74
Parallelization	0min 40s (7ms)	46	1.67	1.67
Structural reuse	0min 47s (11ms)	46	1.42	1.42
Structural reuse	0min 49s (3ms)	46	1.38	1.38
GS solution reuse	1min 10s (2ms)	47	0.97	0.99
GS Precision	1min 10s (7ms)	46	0.96	0.96
All	0min 28s (24ms)	46	2.34	2.34
All (Sequencial)	0min 47s (1ms)	46	1.43	1.43
All	0min 29s (1ms)	46	2.33	2.33

Table 6.16: Results for hill-climber with a 10x28 grid.

Improvement	Mean time (std)	No. iterations	Speed-up	Mean it. speed-up
Standard	3min 5s (8ms)	44	-	-
Standard	3min 5s (16ms)	44	1.0	1.0
Parallelization	1min 51s (109ms)	44	1.67	1.67
Parallelization	1min 51s (28ms)	44	1.66	1.66
Structural reuse	2min 3s (10ms)	44	1.51	1.51
Structural reuse	2min 2s (11ms)	44	1.51	1.51
GS solution reuse	3min 4s (15ms)	44	1.01	1.01
GS Precision	3min 3s (17ms)	44	1.01	1.01
All	1min 13s (9ms)	44	2.51	2.51
All (Sequencial)	1min 57s (11ms)	44	1.57	1.57
All	1min 10s (29ms)	44	2.61	2.61

Table 6.17: Results for hill-climber with a 14x32 grid.

Improvement	Mean time (std)	No. iterations	Speed-up	Mean it. speed-up
Standard	5min 24s (176ms)	42	-	-
Standard	5min 28s (14ms)	42	0.99	0.99
Parallelization	3min 12s (132ms)	42	1.68	1.68
Parallelization	3min 16s (49ms)	42	1.65	1.65
Structural reuse	3min 44s (25ms)	42	1.44	1.44
Structural reuse	3min 44s (19ms)	42	1.44	1.44
GS solution reuse	5min 42s (94ms)	42	0.95	0.95
GS Precision	5min 44s (7532ms)	42	0.94	0.94
All	2min 52s (17290ms)	42	1.88	1.88
All (Sequencial)	3min 49s (182ms)	42	1.41	1.41
All	2min 18s (42ms)	42	2.34	2.34

Table 6.18: Results for hill-climber with a 16x38 grid.

Improvement	Grid	Mean time	Speed-up
Standard	10x28	2min 04.43s	-
Standard		1min 45.65s	1.17
All		1min 11.26s	1.75
All		0min 51.56s	2.41
Standard	14x32	4min 46.43s	-
Standard		3min 30.32s	1.36
All		2min 38.16s	1.81
All		1min 53.81s	2.52
Standard	16x38	10min 33.73s	-
Standard		6min 08.92s	1.71
All		5min 19.48s	1.98
All		3min 31.22s	3.00

Table 6.19: Speed-ups for hill-climber, executed in the laptop LG R510.

Finally, although the two optimizers reached similar configurations, the hill-climber returns a greater Breguet range. Due to a more steady pace, with shorter optimization steps, the hill-climber is likely to better explore the whole optimization space than DIRECT. However, this greater accuracy comes at the expenses of running time, which is considerably longer for the hill-climber.

6.5 Concluding remarks

Concerning model validation, the small deviations registered for both Diederich's method and the vortex lattice method show that the author's implementation used for this work can be trusted, under the considered conditions.

Design variables	**DIRECT**		**Hill-climber**	
	Value	*Per.*	*Value*	*Per.*
Wingspan	31.97m	99.9%	32.0	100.0%
Root chord	2.0m	0.0%	2.0m	0.01%
Tip chord	1.0m	0.0%	1.0m	0.01%
Sweep angle	0º	50.0%	4.5º	55.0%
Twist	0º	50.0%	-1.2º	45.0%
Dihedral angle	0º	0.0%	0º	0.01%
Root diameter	0.037m	3.7%	0.04m	4.0%
Tip diameter	0m	0.0%	0.001m	0.01%
Root thickness	0.14m	14.0%	0.1m	10.0%
Tip thickness	0.007m	0.7%	0.02m	2.0%
Breguet range	10173.5m	-	11279.0m	-

Table 6.20: Optimized configurations.

The software used to compare the vortex lattice method results is more complete, as it encompasses variations of the camber along the wing. This is even more noted when the wing is under the influence of twist and dihedral angle, as the camber thickness plays an more important role in this cases. Aside that, the implementation used in the current work can produce satisfying results.

As for the optimization results, it was concluded that the parallelization

of the algorithms and the introduction of the Gauss-Seidel method for solving the linear system had the most impact in the speed-ups achieved, with the incremental use of the distributions not contributing as much as initially expected.

Finally, the improvements produced different effects on the optimizers. DIRECT, which uses a less local search, benefited more than the hill-climber, achieving greater speed-ups. For both the optimizers, the speed-ups grow with the number of panels, although this condition may be affected depending on the machine where the code is executed.

7

Concluding Remarks

Aeroelastic modelling is of great importance in aeronautics. Ideally, it should be considered right at the preliminary stage, but aeroelastic optimization depends on the availability of simple enough, yet sufficiently accurate models. For that reason, any effort that allows more faithful models to be used at this stage is likely to be of interest to the aircraft industry.

The goal of this dissertation was to define a new optimization approach over the conceptual design of a wing, eventually extending it to further development stages. To fulfill such an objective, several techniques were proposed. They encompasse iterative methods for solving linear algebraic systems of equations (more specifically, the Gauss-Seidel method), the control of the solving precision, parallelization of the optimization and finally, the use of incrementalization in order to avoid repeated computation.

Being an exploratory work, the potential of the proposal could not be fully predicted at the beginning and so, the speed-ups achieved were not as good initially hoped. Nevertheless, they certainly cannot be considered irrelevant. Looking through a more industrial perspective rather than academic, these are actually promising results, especially if the same speed-up can be successfully obtained in later stages of development, where the execution time is far greater.

The results have also shown that the speed-up grows with the number of panels. This is an interesting starting point for future work, if extending optimization further than solely wing analysis.

Although more than 600 panels can be considered excessive (at least for

the basic vortex lattice method, more advanced methods may require denser surface coverage), in the limit, the whole aircraft can be shaped by panels, as every section contributes to both lift and drag generation. If, besides the wing, the fuselage, breaks, fin, rudder, tails and engines are taken into account too, a larger number of panels must be used.

However, it was also registered that the performance of the Gauss-Seidel method deteriorates with the increase of the precision required by the optimizer. This can be a severe problem for the more advanced stages, as at that point, it is no longer feasible to simply have good approximations of the final wing.

Nevertheless, if maintaining a speed-up of at least 4 for those stages, it would be an achievement. The whole development process in Embraer can reach up to between 12 and 17 hours of computation. Reducing this time to simply 3 or 4 hours is certainly an option to analyze. Of course, the development process does not only include optimization, so this reduction would be smaller, unless the other parts of the cycle could be speeded up as well.

More than only setting a new approach for the models studied, this document also opens another door for future work, by proposing some incrementalization ideas to be implemented. The LU factorization update may be suitable for more complex models, that eventually use meshes for modelling and due to the greater volume of data, may benefit more from incrementalization.

All this considered, this project has served its initial purpose and the results obtained show that it is possible to reduce computation time without jeopardizing the accuracy of the models. As long as optimization is well structured and organized, recalculations and other unnecessary and heavy steps may be avoided or simplified.

Nomenclature

Variables

$A(\eta)$	Transversal section surface
AR	Aspect ratio
C_T	Specific fuel consumption parameter
C_{D_0}	Drag coefficient markup value
C_{Di}	Induced drag coefficient
C_D	Drag coefficient
C_L	Lift coefficient
D	Drag
D_i	Induced drag
E	Jone's edge velocity factor
E_s	Young's module
$F_s(\eta)$	Shear force
$I(\eta)$	Inertial moment
L	Lift
$L_a(\eta)$	Additional lift distribution
$L_b(\eta)$	Basic lift distribution
M	Mach number
$M_b(\eta)$	Bending moment
N_c	Load factor
$R(\eta)$	External radius
S	Planform or wing area
T	Temperature
U_∞	Fluid speed
V	Object speed
X	Chord axis
Y	Wingspan axis
Z	Camber axis
ΔY	Wing deflection
Γ	Circulation
Λ	Sweep angle
Λ_β	Effective sweep angle
$\Theta(\eta)$	Slope
α	Angle of attack
α_0	Zero-lift angle
β	Prandtl-Glauert factor
δ	Mean camber line slope
δ_f	Maximum wing bending allowed
ϵ	Twist angle
η	Adimensional half-spanwise
η_d	Typical section

γ	Specific heat capacity
λ	Wing taper ratio
ϕ	Dihedral angle
ρ	Fluid density
$\rho_l(\eta)$	Beam's linear density
ρ_s	Structural density
σ_s	Yield stress
θ	LLT angular component
θ_{ST}	Static temperature ratio
$\vec{u}$	Chord axis unit vector
$\vec{v}$	Wingspan axis unit vector
$\vec{w}$	Camber axis unit vector
a	Speed of sound
b	Wingspan
$c(y)$	Chord
c_g	Mean geometric chord
$c_{l_a}(\eta)$	Sectional additional lift
$c_{l_b}(\eta)$	Sectional basic lift
c_{l_α}	2D lift curve slope
$c_l(\eta)$	Sectional lift
e	Oswald efficiency factor
g	Gravity acceleration
h	Cruising height
m	Mean cruise flight mass
m_f	Fuel mass
m_m	Mass markup
m_s	Beam's mass
n_{cp}	Spanwise centre of pressure
p	Static pressure
q	Dynamic pressure
$r(\eta)$	Internal radius
$st(\eta)$	Stiffness
$t(\eta)$	Thickness
y	Ordinate of wingspan

Subscribers

.25	Quarter-chord line
LE	Trailing edge
TE	Leading edge
r	Root
t	Tip

Bibliography

[1] Torenbeek, Egbert: "Synthesis of Subsonic Airplane Design"; Delft University Press, 1981.

[2] Oliveira, L.; Lopes, A.: "Mecânica dos Fluidos"; Lidel, 2006.

[3] Zang, Thomas A.: "Aifoil/Wing Optimization"; NASA Langley Research Center, Hampton, USA, 2010.

[4] Hadi Winarto: "Lifting Line Theory - Tutorial Example (AERO 2258A)"; RMIT University, May 2004.

[5] Diederich, F.W.: "A simple approximate method for calculating spanwise lift distributions and aerodynamic influence coefficients at subsonic speeds"; NACA TN 2751, 1952.

[6] Young, Warren C.; Budynas, Richard G.: "Roark's Formulas for Stress and Strain"; 7th Edition, McGraw- Hill, 2002.

[7] Hodges, Dewey H.; Alvin Pierce, G.: "Introduction to Structural Dynamics and Aeroelasticity"; Cambridge University Press, 2002.

[8] Abbot, I.H.; von Doenhoff, A.E.: "Theory of wing sections"; Mc.Graw-Hill, New York, 1949.

[9] Bisplinghoff, R.L.; Ashley, H.; Halfman, R.L.: "Aeroelasticity (Corrected Version)"; Dover Publications, 1996.

[10] Anderson, John D.: "Fundamentals of Aerodynamics. Second edition"; McGraw-Hill International Editions, Aerospace Science Series, 1991.

[11] Raymer, Daniel P.: "Aircraft Design: A Conceptual Approach"; AIAA Education Series, American Institute of Aeronautics and Astronautics, 1999.

[12] Wright, Jan R.; Cooper, Jonathan E.: "Introduction to Aircraft Aeroelasticity and Loads"; AIAA Education Series, John Wiley and Sons, 2007.

[13] Bruhn, E.F.: "Analysis and Design of Flight Vehicle Structures'; Jacobs Publishers, 1975.

[14] Martins, Joaquim R.R.A.: "A Coupled-Adjoint Method for High-Fidelity Aero-Structural Optimization"; Doctoral Thesis, Stanford University, 2002.

[15] Mattingly, Jack D.; Heiser, William H.; Pratt, David T.: "Aircraft Engine Design"; 2nd edition, American Institute of Aeronautics and Astronautics, 2002.

[16] Diederich, Franklin W.; "Calculation of the aerodynamic loading of swept and unswept flexible wings of arbitrary stiffness"; NACA Report 1000, 1948.

[17] Philips, W.F.: "Lifting-Line Analysis for Twisted Wings and Washout-Optimized Wings"; AIAA Journal of Aircraft, January-February 2004, Vol.41, no.1.

[18] DeYoung, John; Harper, Charles W.: "Theoretical Symmetric Span Loading at Subsonic Speeds for Wings Having Arbitrary Plan Form"; NACA Report 921, 1948.

[19] Giunta, Anthony A.: "Sensitivity analysis method for aeroelastic aircraft models"; Elsevier Science, 1999.

[20] Pant, Rajkumar; Fielding, J.P.: "Aircraft configuration and flight profile optimization using simulated annealing"; Elsevier Science, 1999.

[21] Mason, W.H.: "Analytic Models for Technology Integration in Aircraft Design". AIAA Paper 90-3262, September 1990.

[22] Malone, Brett; Mason, W,H.: "Multidisciplinary Optimization in Aircraft Design Using Analytic Technology Models"; Journal of Aircraft, Vol. 32, No. 2, March-April, 1995.

[23] Moran, Jack: "An Introduction to Theoretical and Computational Aerodynamics"; John Wiley & Sons, 1984.

[24] Bertin, John; Smith, Michael: "Aerodynamics for Engineers", Prentice Hall, Third Edition, 1998.

[25] Nocedal, J. and Wright, S. J., "Numerical Optimization", second edition, Springer, 2006.

[26] A.L Custódio, M. Emmerich and J.F.A. Madeira, "Recent Developments in Derivative-free Multiobjective Optimization", April 5, 2012.

[27] W. Yamazaki, M. Rumpfkeil and D. Mavriplis, "Design Optimization Utilizing Gradient/Hessian Enhanced Surrogate Model", University of Wyoming, American Institute of Aeronautics and Astronautics, USA, 2010.

[28] Reed, James A.; Utke, Jean; Abdel-Khalik, Hany S.: "Combining Automatic Differentiation Methods for High Dimensional Nonlinear Models"; North Carolina State University, 2010.

[29] Neidinger, Richard D.: "Introduction to Automatic Differentiation and MATLAB Object-Oriented Programming"; Siam Review, Vol. 52, No. 3, 2010.

[30] Gondzio, J.: "Stable algorithm for updating dense LU factorization after row or column exchange and row and column addition or deletion", Optimization: A Journal of Mathematical Programming and Operations Research, 1992.

[31] Chu, M. T.: "System of Linear Equations - Iterative Approach", Lecture notes on Numerical Analysis I, Chapter 3, 2007.

[32] Melin, Tomas: "A Vortex Lattice MATLAB Implementation for Linear Aerodynamic Wing Applications", Master Thesis, Royal Institute of Technology (KTH), 2000.

[33] User manual: "Surfaces - Vortex Lattice Module", Great OWL Publishing - Engineering Software, August 2009.

[34] Abbott, I. H.; von Doenhoff, E.: "Theory of Wing Sections", Dover Books, June 1959.

[35] "Basic Linear Algebra Subprograms" (BLAS) libraries: http://www.netlib.org/blas/.

[36] "Open Computing Language" (OpenCL): http://www.khronos.org/opencl/

[37] Finkel, Daniel: "DIRECT Optimization Algorithm User Guide", North Carolina State University, March 2003.

[38] OpenMP: http://www.openmp.org/

[39] User's guide: "MSC.FlightLoads and Dynamics User's Guide", MSC Software, 2006.

[40] Shewchuk, Jonathan Richard: "Lecture Notes on Delaunay Mesh Generation"; University of California at Berkeley, 1999.