ACADEMIA DA FORÇA AÉREA
INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Aero-Structural Optimization of Sailplane Wings

Bruno Jorge Pereira Cadete

133802-B

A thesis submitted in conformity with the requirements for the degree of Masters in

Aeronautical Engineering

Jury

President: Cor/EngEl/036605-G Joaquim Gonçalves Coelho Lopes
Supervisor: Doctor André Calado Marta
Examiner: Professor Paulo Rui Alves Fernandes

Sintra, December 2011

Acknowledgments

I would like to thank all those who directly and indirectly, gave me the opportunity and means to make possible the realization and completion of this work.

First, I would like to thank to my supervisor, Doctor André Marta, for its knowledge in the field of multi-disciplinary optimization (MDO), for all the suggestions and advices on the methodology developed, but most of all for the constant availability and interest in the discussions through the course of this work, which were essential for its development.

I wish also to thank Professor Joaquim Martins for its advices in the initial stages of the work and for putting at my disposal a set of multi-disciplinary tools; to Graeme Kennedy and Wenhu Wang for their support and availability to lean me information about the MDO toolset usage and for allowing me to help in development of the same.

On a personal level, I would like to thank to my family, specially to my parents, for all the support provided throughout my academic and military course. Also, to my friends, for the support and understanding of my absence in the last months. Lastly, a special thanks to Claudia for standing beside, giving the support and motivation that ultimately made possible the competition of my MSc thesis.

Abstract

This thesis presents a framework for the multi-disciplinary design analysis and optimization of sailplane wings. Its objective was to run an aero-structural optimization on sailplane wings. A literature review on the studies from various authors is presented and used as base for the establishment of the multi-disciplinary optimization (MDO) framework. The approach used employs a multi-disciplinary feasible architecture. The geometric parametrization method employed follows a free-form deformation method. To solve the aero-structural analysis problem, a panel method coupled with a finite-element solver is implemented in the framework, tested and used for the MDO of sailplane wings. The coupled non-linear system is solved using an approximate Newton-Krylov approach. The optimization algorithm uses sequential quadratic programming. Two study cases on sailplane wings are exploited within the MDO framework: a semi-tapered wing and a real sailplane wing, based on the L-23 Super Blanik from the Portuguese Air Force. Single disciplinary analysis assess the capabilities of the disciplinary modules of the framework. Results are presented for a drag minimization problem using aerodynamic and multi-disciplinary optimizations. They reveal important trade-offs between disciplinary optimum and multi-disciplinary optimum at the preliminary design stage.

Keywords: Aero-structural problem, Multi-disciplinary optimization, Free-form deformation method, Panel method, Finite-element method, Sailplane wings.

Resumo

A presente dissertação apresenta uma plataforma para o design multi-disciplinar de asas de planadores. O objectivo é a realização de uma optimização aero-estrutural em asas de planadores. Foi feita uma revisão bibliográfica sobre os estudos de vários autores da área, que serviram de base para o estabelecimento da plataforma de optimização multi-disciplinar. Foi utilizada uma arquitectura *Multi-Disciplinar Feasible*. O método de parametrização geométrica utilizado segue uma abordagem livre de software CAD, usando um método de deformação livre de forma. Para resolver o problema aero-estrutural, foram testados e implementados, um método de painéis juntamente com um método de elementos finitos. O sistema de equações acopladas não-lineares é resolvido utilizando um método aproximado de Newton-Krylov. O algoritmo de optimização faz uso de programação sequencial quadrática. Dois casos de estudo de asas de planadores são usados: uma asa com afilamento e uma asa real baseada no planador L-23 Super Blanik da Força Aérea Portuguesa. Foram realizadas análises disciplinares para avaliar as capacidades dos módulos disciplinares da plataforma. São apresentados resultados para um problema de minimização de arrasto, utilizando quer optimização aerodinâmica que optimização multi-disciplinar. Os resultados obtidos revelam importantes cedências, entre o óptimo disciplinar e multi-disciplinar, na fase de design preliminar de uma asa.

Palavras-chave: Problema aero-estrutural, Optimização multi-disciplinar, Método de deformação livre de forma, Método dos painéis, Método dos elementos finitos, Asas de planadores.

Contents

List of Figures

List of Tables

Nomenclature

Greek symbols

α	Angle of attack.
γ	Glide angle.
λ	Taper ratio.
ψ	Adjoint vector.
ρ	Density.
σ	Weighting parameter.
τ	Wall stress.
θ	Twist angle.

Roman symbols

C_D	Coefficient of drag.
C_L	Coefficient of lift.
C_p	Coefficient of pressure.
D	Drag.
L	Lift.
b	Span.
c	Chord.
p	Pressure.
q	Dynamic pressure.
V	Velocity magnitude.
$\mathbf{u}$	Velocity vector.
u	Structural state variables.

w Aerodynamic state variables.

Subscripts

0 Zero-lift angle.

∞ Free-stream condition.

a, A Axial component.

f Friction.

i, j, k Computational indexes.

KS Kreisselmeier-Steinhauser.

l Lower Surface.

LE Leading edge.

n, N Normal component.

r Root.

t Tip.

TE Trailing edge.

u Upper Surface.

x, y, z Cartesian components.

ref Reference condition.

Superscripts

' Per unit span.

T Transpose.

Glossary

Aero-Structural Problem	Engineering problem which accounts the fluid-structure interaction and involves the disciplines of aerodynamics and structures.
Angle of Attack	Angle between the chord and the vector representing the relative motion between the lifting body and the fluid through which it is moving.
Angle of Twist	Angle between the chord of the airfoil at the root of the wing (nearest to the fuselage) and the chord of the airfoil at the tip of the wing.
Architecture	Layout used to apply a certain technique in the solving of an engineering problem.
CFD	Computational Fluid Dynamics is a branch of fluid mechanics that uses numerical methods and algorithms to solve problems that involve fluid flows.
CSM	Computational Structural Mechanics is a branch of structure mechanics that uses numerical methods and algorithms to perform the analysis of structures and its components.
Chord	The straight line connecting the leading and trailing edges of a wing section.
Gliding/Soaring	Interchangeable terms to name the flight using a sailplane or glider.
L/D Ratio	Quotient between the lift and drag force used in aerodynamics to measure aircraft and components performance.
MDA	Tool used in the design of aircraft that accounts the interference of two or more disciplines to provide results for a coupled analysis.

MDF	Multi-Disciplinary Feasible is an MDO architecture in which the disciplines are directly coupled in a multi-disciplinar solver and the design variables are changed all at a system-level optimizer.
MDO	Multi-Disciplinar Optimization is an engineering technique that uses optimization methods to solve design problems incorporating two or more disciplines.
Optimization	Process that starting from a initial state, and, according to the specifications/constraints imposed, creates a final state, which optimizes a defined set of parameters that affect the current state.
Sailplane/Glider	Terms used to name a class of aircrafts which do not use an engine to propel.
Von Mises Stress	Named given to the equivalent stress in the Von Mises criterion for ductile failure, which, if superior to the yield stress of the material indicates a failure condition.

Chapter 1

Introduction

1.1 Motivation

The course of aircraft design, taught in the last year of the Master of Science (MSc) program in aerospace engineering at Instituto Superior Técnico (IST), addresses the main stages of an aircraft conceptual design.This course made the students realize that one of the most important stages, if not the most important one, is the aircraft preliminary design. It has to be made, considering not only the elegance and beauty of the aircraft but also the requirements in terms of aerodynamics, propulsion, structures, controls, among others. Typically, the integration of the disciplines is only handled in the latter stages of the design, when a scale model or prototype is tested in a wind tunnel or flown. This is the method that has been in use for the last 30 years and the reason for it is that the basic design of subsonic civilian aircrafts has not changed. With it, the basic wing shape has been thoroughly analyzed and well optimized by a generation of engineers that has more than a decade of experience designing these wing configurations. However, with the emergence of a new generation of aircrafts, with new design approaches like the blended-wing body, should a ”build and test” approach be used, it would be too time and resources consuming. This is where the power of multi-disciplinary optimization (MDO) techniques can make a difference.

As Abdo and Samareh (2005) of Bombardier Aerospace realized, ”The greatest potential benefits can be obtained by applying formal optimization at preliminary design stage”. However, the utilization of the MDO in aircraft design is relatively recent. In fact, it has only fully emerged as a technique viable for aircraft design in the last two decades. And within this time it has proven to provide good results either in terms of computational costs or in terms of accurate flow physics analysis. So the fact that a MSc thesis could be made on this relatively new area, that is now emerging to its full capability and that it is now beginning to be used in the major aircraft constructor companies, was an interesting opportunity. Other important aspect that led to the decision of making this thesis, was the fact that it focus on two of the most important disciplines of aeronautical engineering course: aerodynamics and aeronautical structures. So an opportunity arose to apply the lessons learned in a practical work, enriching theoretical knowledge with some computational experimentation.

Aero-structural optimization of wings is the subject of the thesis presented in this document. As a graduating aeronautical engineer of the Portuguese Air Force Academy, the interest in gaining knowledge that may enable the development of skills useful to the Air Force in its future projects is very motivating. At the present time, the Air Force Academy is involved in projects like PERSEUS - Protection of European borders and Seas through the intelligent Use of Surveillance (PERSEUS, 2010) or PITVANT - Project for Research and Technology in Unmanned Aerial Vehicles (Morgado and de Sousa, 2009), that may benefit from using MDO early in the design stages of aircraft project. With this approach, time can be gained and a better and unusual feasible result may be achieved, without the normal approach of ”do as others have done” or ”build and then test”. Recently the Air Force has created a new department, called Engineering and Programs Direction (DEP), whose objective is to provide technical skills required for the management of weapons systems at all stages of their life cycles, focusing on those that fall in the realization of modernization projects, as well as those related to quality management, environmental certification and airworthiness of military aircraft property of the Portuguese Air Force (PAF, 2009). This department may also benefit from the use of MDO techniques especially in modernization projects. So, the acquisition of knowledge in MDO can be a real asset to a graduating aeronautical engineer. The fact that the MDO in this thesis would be applied to sailplane wings also presents an extra appeal since, as an Air Force student, I was granted with the opportunity of taking a brief sailplane pilot course and perform some instruction flights. This gliding and soaring experience made possible the realization that sailplanes, although simple design aircrafts, present an interesting subject for the research in the multi-disciplinary field. This due to the fact that sailplane aircrafts are designed to have strong and flexible wings and a great flight performance. These aircrafts, especially their wings, make a simple but complete case study for the application of an MDO framework.

Therefore, this work may make a contribution not only to the development of a graduate aeronautical engineer, but also to the enrichment of the MDO community and, eventually, a contribution for its increasing use in the aircraft design process of future projects.

1.2 Project Objective and Synopsis

The thesis presented in this document under the subject of ”Aero-structural Optimization of Sailplane Wings” has the main objective of running an MDO on sailplane wings. To this end, the student was proposed to develop skills in the area of numerical modeling in both the aerodynamic and the structural aspects, which allow to simulate and evaluate the performance of various wings. With those skills, the student should establish a framework for an aero-structural MDO tool. Then this tool, should be applied in some exercises of multi-disciplinar analysis (MDA) and optimization. After these exercises with the MDO tool, the student should achieve the optimum geometric parameters for high performance wings at both the aerodynamic or structural levels. The methodology employed in the realization of the thesis was divided into seven phases, according to Fig. 1.1.

The first task was a literature review. Its main focus was on MDO history, its role in aircraft design and on works that have been and still being developed and published in this field. It provided the

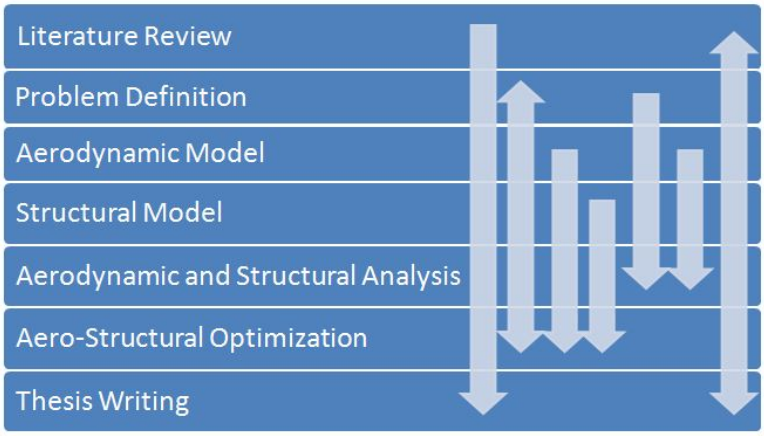


Figure 1.1: Work flow scheme for the thesis.

information needed to begin choosing the MDO architecture for the tool. Within this task, it has been made a research on gliding and soaring along with sailplane aircrafts and the case studies were chosen: one wing based on the standard class sailplane and one based on a real Portuguese Air Force aircraft. Then research was made on aerodynamic and structural models used to run the respective simulations. Software and code that have been and still being developed for this effect were researched, chosen and implemented within the framework of the MDO tool. Simultaneously, optimization methods were studied to find a suitable optimizer for the tool. Finally, exercises were made to test and assess the performance of the MDO tool. From the first task to the last, the record of all important information was made in an written document which was the base for the present document.

1.3 State-of-The-Art

The first surveys on MDO problems and approaches were published by the MDO Technical Commite (1991). Two years later, Sobieszczanski-Sobieski (1993) showed that MDO could be used as an efficient way to overcome the computational challenges on a new emerging method for aircraft conceptual design. During the following years, various papers were published that examined and tested one or various MDO architectures, as for their method of defining the problem formulation and/or the efficiency of their optimization algorithm. The most relevant architectures include Multi-Disciplinary Feasible (MDF) by Cramer (1994), Individual Discipline Feasible (IDF) by Dennis and Lewis (1994), Collaborative Optimization (CO) by Braun et al. (1996) and Bi-Level Integrated Synthesis System (BLISS) by Sobieszczanski-Sobieski et al. (1998).

The MDO research has matured in the last thirteen years, with the publication of many comparative studies. From Hulme and Bloebaum (1998) that compare several MDO methods, as for example the MDF and IDF, with five analytical examples of varying complexity or size, through Alexandrov and Kodiyalam (1998) who pointed out the importance of problem formulation evaluation detached from the traditional optimization metrics. Then Chen et al. (2002) who used two application examples with the same metrics to evaluate three different MDO methods (CO, CSSO, and BLISS) or Perez et al. (2004)

who through the evaluation of different MDO architectures, using an extended set of metrics, demonstrated the promising features of evaluation metrics based both on the formulation considerations and on the optimization performance criteria. More recently, Tedford (2006) or Martins and Tedford (2006) developed an MDO framework in Python to provide a platform for comparisons relating various MDO architectures and demonstrated its potential on identifying trends on the performance of that architectures. Yi et al. (2007) made a comparison study of six different methods using mathematical examples. Research of MDO architectures is still a very active field and many universities have specialized teams making and publishing studies. Today, specialized laboratories like the Multidisciplinary Design Optimization Laboratory in University of Michigan (MDO Lab of UoM, 2010-11) , Multidisciplinary Design Optimization Laboratory in University of Toronto's Institute for Aerospace Studies (MDO Lab of UoT, 2000-11) and Aerospace Design Laboratory in Stanford University's Department of Aeronautics and Astronautics (AD Lab of SU, 2010-11) are on the leading edge of MDO research

Aero-structural analysis techniques are a specialization of more general fluid-structure interaction (FSI) solution methods. Although these methods are a wider field of research, some studies are worth mentioning as their results have contributed with important ideas for the development of more aero-structural specialized techniques. Felippa et al. (2001) performed a review of solution techniques for coupled nonlinear problems using partitioned solvers. Later, Kim et al. (2003) developed a solution procedure for coupled multi-physics problems using a multi-level Newton's method. Applying their approach to a coupled FSI problem they have realized the importance of using accurate linearizations of the coupling terms. In 2005, there were two papers published (Biros and Ghattas, 2005a) and (Biros and Ghattas, 2005b), where a Lagrange–Newton–Krylov–Schur approach to the simultaneous solution of PDE-constrained optimization problems was presented. This approach was applied to a design problem using the incompressible Navier–Stokes equations. More recently, Heil et al. (2008) solved a time-dependent FSI problem by applying Newton's method to a second-order backward difference discretization of the coupled system and using a monolithic approach with both direct and iterative solvers to the resulting equations. In their study, they demonstrate that a monolithic approach is very reliable and competitive. Although there are several authors that have developed methods for MDA and MDO over the past decade, to limit the search field of the literature review, a focus was made on studies only in the aero-structural specialization.

In 1999, Reuther et al. published an article where an initial aero-structural analysis and optimization framework for MDO was presented. Later in 2002, Martins developed that aero-structural analysis and optimization framework with a method to calculate the sensitivities of aerodynamic and structural cost functions with respect to both aerodynamic shape and structural variables that was both accurate and efficient. That framework coupled a linear finite-element structural model to a finite-volume Euler CFD solver and achieved a coupled solution using a pseudo-time marching scheme with periodic updates of the displaced shape. A structural model composed of solid, three-dimensional elements was used to represent the stiffened aircraft wing. To transfer loads and displacements across the aircraft outer-mold line (OML), they used a systematic scheme based on the work developed by Brown (1997). Martins et al. (2005) developed a sensitivity analysis of the aero-structural equations for both the adjoint and

direct formulations, with a block Gauss–Seidel technique for solving the coupled adjoint system and applied it to the optimization of a supersonic business jet (Martins et al., 2002).

Around the same time, Maute et al. (2001) presented an aero-structural analysis that coupled the Euler equations to a linear finite-element model, where, following the previous work of Maman and Farhat (1995), a mesh movement strategy based on a spring analogy and a load and displacement transfer technique was employed. Formulations of both the adjoint and direct methods for computing the sensitivities of the coupled aero-structural system were presented and, to solve the coupled nonlinear equations, they used a nonlinear block Gauss–Seidel method with relaxation. In 2004, Maute and Allen developed an aero-structural optimization problem in which the internal structure of the wing box is parametrized using a single isotropic material with penalization approach to determine the topology of the optimal structure. It used a solution method similar to the previously described by Maute et al. (2001). The methods referred above were improved in terms of robustness and efficiency by Barcelos et al. (2006), who developed a class of Newton–Krylov–Schur methods for solving the coupled nonlinear fluid-structure-mesh movement problem. They realized that their technique was more robust and efficient than the original Gauss–Seidel method presented by Maute et al. (2001). It consists of using an approximate Newton's method for the solution of the nonlinear coupled equations and of using a Schur complement approach at each iteration to solve the coupled linear system that results from a linearization of the residual. The same authors, Barcelos and Maute , presented in 2008 an aero-structural solution technique coupling the Navier–Stokes equations with a turbulence model to a linear structure and mesh movement strategy.

Van Der Weide et al. (2006) showed results for the use of a CFD solver, *SUmb*, in unsteady turbomachinery computations. This flow solver was developed under the sponsorship of the Department of Energy Advanced Strategic Computing (ASC) Initiative. It can be used to solve the compressible Euler, laminar Navier-Stokes and RANS equations on multi-block structured meshes. Although the primary objective of this code within the Stanford ASC program is to compute the flows in the rotating components of jet engines, *SUmb* has been developed as a generic solver and it can be applicable to a variety of other types of problems, including external aerodynamic flows. It has also the advantage of having compatibility with a Python interface. This interface is available, for example, in a multi-disciplinary environment like the one presented by Alonso et al. (2004). Following these studies, an evaluation was made to use this finite-volume, cell-center multi-block solver. During this evaluation, it was also reviewed the possibility of using this code implementation with the adjoint technique for sensitivity analysis proper to the highly coupled nature of the aero-structural problem (Mader et al., 2008). Kenway et al. (2010) presented a method for the important and complex problem of geometric parametrization for high-fidelity MDO. This method follows a geometry parametrization approach with no resource to computer-aided design (CAD) software and it was shown that presents several advantages over other parametrization techniques, as for example the efficient computation of analytic derivatives for gradient-based optimization. Kennedy and Martins (2010) presented a comparison of methods for aero-structural analysis and optimization. In that study, they show that the approximate Newton–Krylov method is shown to be an efficient and robust solution technique to solve the coupled nonlinear aero-structural system. In the same

study, they develop an adjoint-based sensitivity method of high-level of accuracy and exploit three levels of parallelism: optimization-level, system-level and discipline-level. An aero-structural induced drag minimization problem is also solved using a panel method coupled to a finite-element solver for a typical subsonic turboprop aircraft wing. The optimizer used employed a state-of-the-art optimization algorithm, well-suited for large-scale optimization problems (linear and nonlinear).

These studies, with emphasis for the most recent ones, were the base for the proposed work. Therefore, to achieve its objective, i.e., run an MDO on a sailplane wings, this thesis aims to develop a similar work in the field of MDO.

Chapter 2

Gliding and Soaring

2.1 Introduction

This chapter describes the relevant information behind the activity of soaring and gliding. It starts by presenting a brief review of soaring history. Then it addresses to the principles of gliding and soaring flight, highlighting the differences in relation to the general power driven flight. The last section presents the glider aircraft, its uses and classes.

2.2 Brief History of Gliding and Soaring Flight

Since Man began observing the nature, free flying like an eagle was always a dream. Therefore, it is natural that through history, many used their imagination to build some practical and impractical ideas to try to lift themselves into the air. The first spoken tentatives are dated in 200 BC in China, with hot air balloons and kites [1]. There, war lords realized that kites could be used for scaring and at same time observing the enemy. Later, it was in Europe that some early attempts of glider flying were made, like the Benedictine Monk Eilmer of Malmesbury that flew 200 meters in a glider, before crashed and sustained injuries (White, 1961). In Renaissance, the famous Leonardo Da Vinci developed a sketch of a glider in which some control surfaces were placed towards the tips, trying to imitate bird wings. While his drawings exist and are deemed flight-worthy in principle, he never actually build it.

In the 18^{th} century, a first rigorous study of the physics of flight was made by an English engineer named Sir George Cayley. In 1799, he exhibited a plan for a glider which, except for platform, had a completely modern look. Over the next five decades, trying to improve its project, he invented most of basic aerodynamics principles, such as ”lift” and ”drag”. In 1856, Jean-Marie Le Bris made the first flight higher than his point of departure, in his glider *”L’Albatros artificiel”* pulled by a horse on a beach (Houard and Peslin, 1943). It was the first time that towing was used to make a take-off. The next two decades made gliders be of great relevance to the powered aviation history as they were the predecessors to the Wright Flyer I (Culick and Jex, 1985), the first aircraft to perform a sustained, controlled, powered

[1] Light frame covered with some thin material usually a form of cloth, to be flown in the wind at the end of a long string, they were the predecessors of gliders.

heavier-than-air flight. Before building the actual Flyer I, Orville and Wilbur built and tested a series of glider designs from 1900 to 1902. Although their first two gliders performed well below predictions from experiments and theory, their will prevailed and, through the development of their own wind tunnel and more sophisticated measuring devices (M.G.Dodson, 2005), they produced a third glider with far better performance (Fig. 2.1). This was successfully used as base for the design of the Flyer I (Sitek and Blunt, 1940). After 1903, gliders lost relevance to hot air balloons and powered aircrafts. Only after

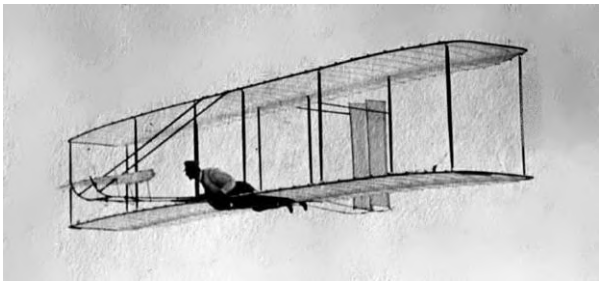


Figure 2.1: Wright brothers 1902 glider.

1919, gliders would emerge again. The Treaty of Versailles, consequence of the World War, imposed restrictions on powered aircraft usage in Germany. This enforced German engineers to develop ever more efficient gliders and to discover ways of using the natural forces in the atmosphere. The importance of developing training in gliders would grow. To promote the glider development, German held the first gliding competition at the Wasserkuppe [2], in 1920. In the following years, gliding sport spread to other countries like the U.S.A. or Russia. During this time, many records were set, as the Martens one hour marker in 1922 or the eight hour marker reached by Maneyrol (Sitek and Blunt, 1940). Gliding was even scheduled to be an Olympic sport in the 1940 Games (Welch, 1980), although that would never happen due to the World War II (WWII). During the war, gliding performed an important role, carrying troops and heavy equipment to combat zones. These gliders provided advantages, as carrying heavy equipment or ensuring quicker troop assemble on the ground. In the 1950's, after the war, many clubs and manufacturers emerged, many of which still exist today, led by the WWII trained pilots and engineers.

Since 1970's, the evolution of gliders has been following the exponential evolution of structural engineering, material science, computational fluid dynamics (CFD) and electronics. Many modern gliders are manufactured in new composite materials such as glass fiber and carbon fiber, which provide greater strength at lower weight. Also advances in CFD, allowed the development of new wing and airfoils shapes. Finally, the advances the Global Positioning System (GPS) and in weather forecasting, have allowed many pilots to make flights that were once unthinkable. To the present day, gliding sport has been actively growing (Roake, 2005), and its importance either as a training method for military and commercial pilots or as a recreational activity and competitive air sport is undeniable.

[2]The Wasserkuppe is a high plateau (elevation 950 m or 3,100 ft), the highest peak in the Rhon Mountains within the German state of Hessen.

2.3 Principles of Gliding and Soaring Flight

Although the terms gliding and soaring are used interchangeably, the air sport is called ”soaring”. In soaring, pilots fly unpowered aircrafts known as gliders or sailplanes using only the naturally currents of rising air in the atmosphere to remain airborne. This is the most relevant aspect of soaring and is what makes it unique.

Section 2.2 presented some early man's attempts to fly and the consequent failures. These were mostly due to the lack of knowledge of both the physical conditions needed to remain airborne and the structure and physical properties of the atmosphere. These early failures, however, provided the fuel for much thought. What started only as abstract observations became more and more of scientific nature. From the years preceding the World War One to the present day, sailplanes and soaring have been subject of many studies, so that the aircraft and its flying techniques are ever refined. The following paragraphs contain brief discussions on the basic principles of glider flight and how the atmosphere affect the different gliding phases.

Fundamental Principles of Flight The principles of flight for sailplanes are the same as for all the aircrafts: it is the action of forces on the entire vehicle that allows it to stay airborne. A sailplane is said to be at equilibrium when all the forces acting on the center of gravity cancel out. The forces that act on a sailplane are illustrated in Fig. 2.2. These forces act either on the center of pressure (C.P.) or on the

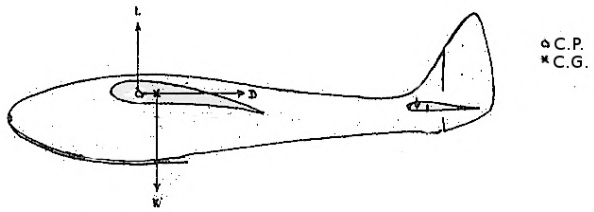
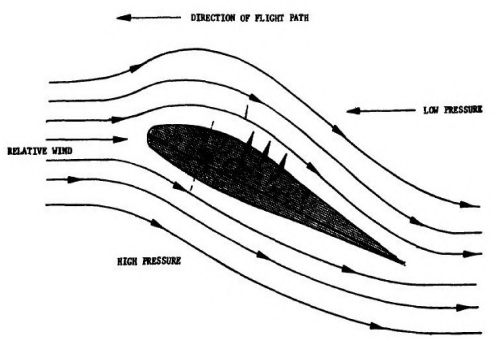
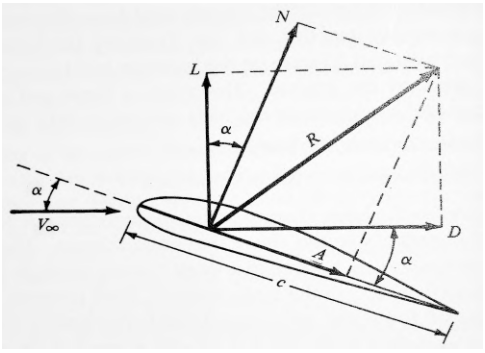


Figure 2.2: Forces acting upon a glider (Sitek and Blunt, 1940).

center of gravity (C.G.). The changes of these forces lead to changes in the relative positions of this two points. As for the aerodynamic forces, all parts of a sailplane either cause lift or drag or both. However, it is the wing the responsible for the major fraction of lift. To generate lift, the air flow has to be moving at a certain velocity and direction in relation to the wing as represented in Fig. 2.3, where α is the angle of attack, between the chord c and the free-stream velocity V_∞, R is the resultant aerodynamic force. The components of the resultant force, perpendicular and parallel to the free-stream, are lift L and drag D, respectively. In Fig. 2.3(a), the wing airfoil shape deflects the flow downwards as it passes the wing surface. The wing airfoil exerts a force on the air to change its direction, in turn the air must exert a force on the wing, equal in size but opposite in direction. This resultant force manifests itself as differing air pressures and flow velocities at the two sides of the wing surface. A region of lower air pressure and higher velocity is generated over the top surface of the wing and a region of higher pressure and lower velocity arises on the bottom. The resultant force can be divided into components normal and axial to the chord, N and A, respectively. The total normal and axial forces(*per unit span*) are obtained by



(a) Deflection of the air flow through the wing shape.



(b) Resultant Aerodynamic force and the components into which it splits.

Figure 2.3: Results of the interaction of the motion of air with a wing.

integrating the pressure p and the wall stress τ from the leading edge to the trailing edge, (Anderson, 2001):

$$N^{'} = -\int_{LE}^{TE} (p_u \cos(\theta) + \tau_u \sin(\theta)) ds_u + \int_{LE}^{TE} (p_l \cos(\theta) + \tau_l \sin(\theta)) ds_l, \tag{2.1}$$

$$A^{'} = \int_{LE}^{TE} (-p_u \sin(\theta) + \tau_u \cos(\theta)) ds_u + \int_{LE}^{TE} (p_l \sin(\theta) + \tau_l \cos(\theta)) ds_l, \tag{2.2}$$

where the subscripts u and l refer to the upper and lower surfaces, respectively. So, the total lift and drag can be obtained by relating L and D with N and A through the angle of attack α:

$$L = N \cos(\alpha) - A \sin(\alpha), \tag{2.3}$$

$$D = N \sin(\alpha) - A \cos(\alpha). \tag{2.4}$$

In aerodynamics, the dimensionless force coefficients, lift and drag coefficients, are defined as:

$$C_L \equiv \frac{L}{q_\infty S} \qquad \text{and} \qquad C_D \equiv \frac{D}{q_\infty S}, \tag{2.5}$$

where the free-stream dynamic pressure is defined as $q_\infty = (1/2) p_\infty V_\infty$, being p_∞ the free-stream pressure. There are also two additional dimensionless quantities of immediate use, which are the pressure coefficient and the skin friction coefficient,

$$C_p \equiv \frac{p - p_\infty}{q_\infty} \qquad \text{and} \qquad C_f \equiv \frac{\tau}{q_\infty}. \tag{2.6}$$

Sir George Cayley and the Wright Brothers were the firsts to study airfoil shapes, however, today aeronautical engineers have a large variety of databases with the qualities and characteristics of many airfoil shapes. Some of the key reference in airfoil selection is the ”Theory of Wing Sections” by Abbott and Doenhoff (1949), based on and the previous article from the same authors, ”Characteristics of Airfoil Sections” (Abbott and Doenhoff, 1945).

As important as lift, is the drag component of the resulting force. Drag results from three main sources:

- Form Drag, which results from the friction of the air around the airfoil and, in some cases, of the interference causes in transonic flow;

- Induced Drag, which is the result from the down-wash component on the net air flow over the wing, which is correlated with the trailing vortices that interact with the flow on the wing;

- Compressibility Drag, which is the result of the flow compressibility interaction with the other two drag components, when the Mach number is increased from the compressibility threshold.

The drag components are not of equal magnitude and their relation to airspeed is not the same. The compressibility drag effect, for instance, is negligible in soaring flight, as the speeds involved are too low to cause significant compressibility of the incident airflow. So, in the case studies of this thesis, the compressibility effect can be neglected. The remaining components also change with speed as shown in Fig. 2.4.

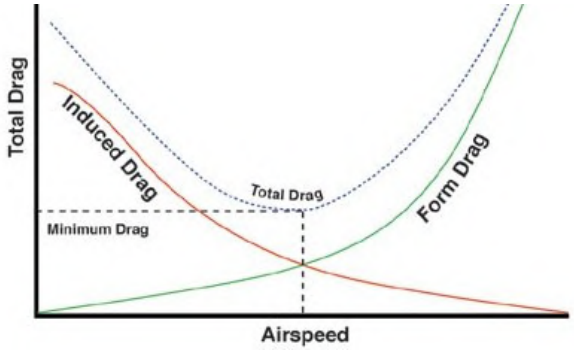


Figure 2.4: Relationship between airspeed and the different components of drag.

The total drag can be mathematically estimated from those two main components as presented in Eq. 2.7 (Corke, 2003),

$$C_D = C_{D_0} + kC_L^2, \tag{2.7}$$

where the first term in the form drag and the second in the induced drag. In the second term the k corresponds to the correlation factor, $k = 1/(\pi Ae)$ where the e, is the Oswald efficiency number, which accounts for the taper ratio and the fuselage effects on the wing. Another important quantity in aerodynamics, is the L/D ratio. This ratio depends on the wing area, the kind of airfoil, the relative wind and the *"aspect ratio of the wing"*. The last is the geometric relationship between the wing span and the wing mean chord, $A = (2b)/(C_r(1+\lambda)) = b^2/S$, where the taper ratio is defined as $\lambda = c_r/c_t$, being the c_r and c_t, the wing root and tip chords, respectively, and b the wing span. In general aircrafts the aspect ratio is relatively small, in the range of 6 to 9, however, in sailplanes it is often greater, from 16 to 23.

Sailplane Performance The main objective behind the study of aerodynamics is to make flight more efficient. The efficiency in gliding can be measured by the maximum range or maximum endurance. To understand the difference between these two concepts, it must be assumed that the sailplane is at

equilibrium, resulting the equations of motion

$$\begin{aligned} T - D - Wsin(\gamma) = m\dot{V} = 0, \\ L - Wcos(\gamma) = mV_S = 0, \end{aligned} \tag{2.8}$$

where γ is the flight path angle (Fig. 2.6) and $V_S = Vsin(\gamma)$ is the sinking speed. Dividing one equation by the other, the relation between the flight path angle and the L/D ratio arises, $tan(\gamma) = -D/L = -1/(L/D)$. This expression gives a negative flight path angle as would be expected in soaring. If the glide angle is defined as the negative of the flight path angle, the expression turns to

$$tan(\gamma_1) = \frac{1}{\frac{L}{D}}, \tag{2.9}$$

where the γ_1 is the glide angle. Therefore, in soaring, the glide angle is independent of the weight of the sailplane and the lowest glide angle corresponds to the maximum L/D ratio.

• **Glide Range** The gliding range, R, corresponds to the longest distance traveled along the ground during the glide descent. Assuming an initial altitude, h_1 and a ground altitude, h_2, the range can be calculated from

$$R = \frac{h_1 - h_2}{\tan \gamma_1} = \frac{L}{D}(h_1 - h_2). \tag{2.10}$$

Here the ratio L/D is also called *"gliding ratio"*. Other important term is the *"speed to fly"*, which corresponds to forward speed that provides the best gliding ratio. To find this speed, the *polar curve* is used. This graphic shows the relation between sink rate and forward speed. An example is illustrated in Fig. 2.5. The best gliding angle, forward and sinking speeds are obtained from the line, between the origin

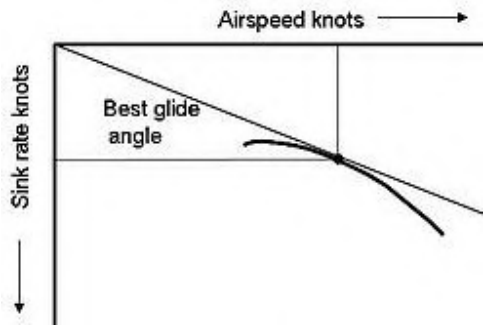


Figure 2.5: Example of a sailplane polar curve showing glide angle for best glide.

and the polar curve, with the least slope. Modern sailplanes have speeds to fly from around 20m/s to 30m/s.

• **Glide Endurance** To achieve the longest duration of gliding flight, the gliding angle has to be kept at a minimum, thus generating a minimum sinking speed. Mathematically, this speed is given as

$$V_S = Vsin(\gamma) = -V\frac{D}{W}. \tag{2.11}$$

As the gliding angle is usually small, a small angle assumption ($L = Wcos(\gamma) \approx W$) can be made, so

$$V_S = -V\frac{D}{W} \approx -V\frac{D}{L} = -\sqrt{\frac{W}{1/2\rho S}\frac{C_D}{C_L^{3/2}}}. \tag{2.12}$$

The sinking rate is directly related to the quantity $C_D/C_L^{3/2}$ and is also dependent of the weight, W. Therefore, to minimize the sink rate (Maximize endurance), these quantities must be minimized.

These two measurements of performance are important in the context of this thesis as they configure two possible optimization problems: one, where the objective is to maximize the range (by maximizing the L/D ratio), the other, where the endurance is maximized (by minimizing the drag and the weight).

Gliding Flight Phases The profile for a soaring flight has four main phases: take-off and climb, cruise, descending and landing. Each of these phases has differences compared to the powered aircraft flight phases:

• **Take-off and Climb** Apart from a few exceptions, most of the sailplanes do not have engines. To take-off and climb, specific methods are used, each requiring specialized training either for the pilots or the technicians operating the launching devices. These methods can be bungee launch, auto-tow, winch launch and aerotowing. The first three methods rely on ground launching devices and are evolutions of each other. A hook in the sailplane is attached to a wire, which is pulled by either human power (bungee launch), a vehicle (auto-tow) or a winch (winch launch). The sailplane is then dragged until it becomes airborne and gains sufficient altitude to release the hook and begin the flight. The forth method, it is the most common, in which a powered airplane is used to tow the sailplane to the desired height and location.

• **Cruise Flight** In soaring, this flight phase can be also called cross-country. To overcome the need of thrust, the sailplane has to be off-balance, so that there is a force pulling it forward. Figure 2.6 shows an example of that off-balance. By changing the angle of attack of the sailplane, a pilot can change

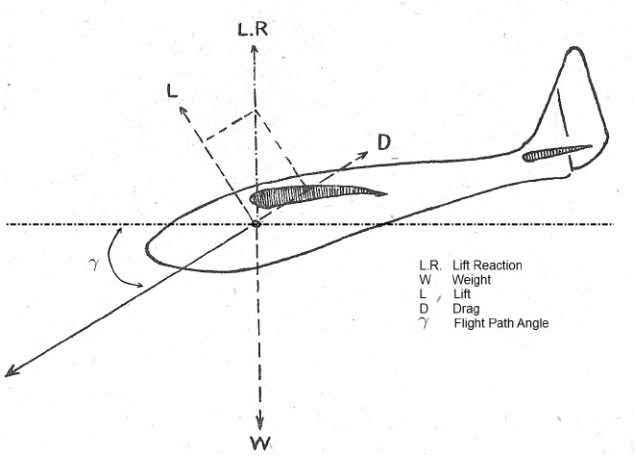


Figure 2.6: Imbalance of forces acting upon the sailplane to provide it with thrust (Sitek and Blunt, 1940).

the direction of the forces acting upon it, creating an horizontal resultant force. This force provides the acceleration or deceleration. However, this also affect the vertical resultant force and therefore the sinking speed of the sailplane. This reinforces importance of the L/D ratio as the pilot wants to gain thrust without losing much lift or increase the sinking speed. Another consequence is that to remain airborne or to ascend in air, the sailplane pilot needs to rely on outer sources of energy. These are associated with the atmospheric phenomena of air masses rising in the atmosphere. These phenomena can be classified as thermals, ridge lifts and wave lifts. The first is a stream of rising hot air are formed on the ground through the warming of the surface by sunlight. When encountered, these can be used to gain altitude before flying towards the destination or to the next thermal. This thermal to thermal flight is known as *"thermalling"* and can allow for great distance flights (Fig. 2.7(a)). Another type of soaring is *"ridge soaring"* (Fig. 2.7(c)). Ridge lift come from the the air flow deflected upward by the wind blows on the sides of hills. By flying in and out these ridge lifts, the sailplane can be kept airborne for long periods of time. The last type of cross-country soaring is the *"mountain wave soaring"* (Fig. 2.7(b)), which relies on wave lifts that occur when strong winds blow perpendicular to a mountain or ridge. The wind forces air flow to climb the mountain and over the top and then down the opposite side. There it bounces off in a layer of stable air near the ground and is deflected upward. By flying in and out these wave lifts, the sailplane can gain altitude faster.

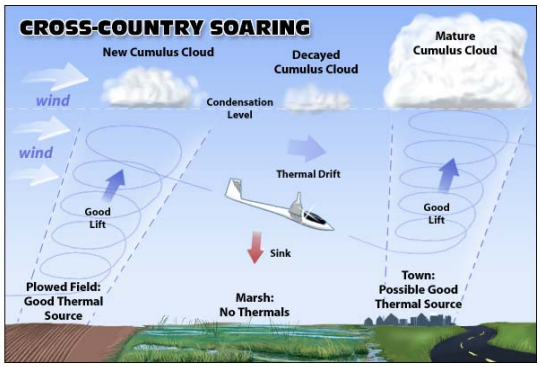
• **Landing** To land, the pilot lets the sailplane glide to the landing strip. As sailplanes have high L/D ratio, air-brakes are often needed. These devices disrupt the air flow, causing its separation over the wings, allowing the sailplane to sink faster due to reduced lift and increased drag.

2.4 Sailplane Aircraft

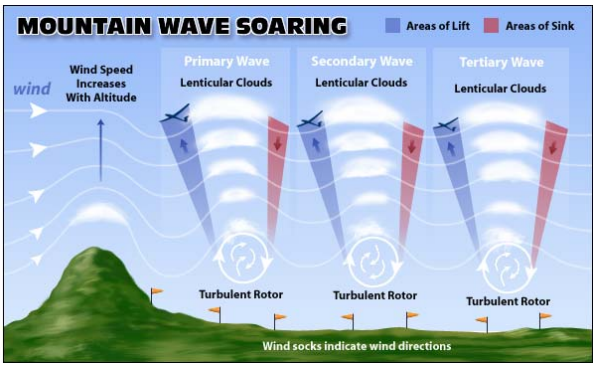
A sailplane is an heavier-than-air vehicle that flies through the dynamic reaction of the air against their lifting surfaces and which free flight does not depend on an engine (Federal Aviation Administration, 2003). Through history the activity of soaring, has been put to many practical purposes. In its early history, sailplanes were used mainly as research aircrafts. Today, though with less frequency, there are still some cases where sailplanes are used to test aircraft designs. Also, before the general use of meteorological balloons, instrumented sailplanes were the most accurate way to acquire meteorological data.

Since the goal of flight is to remain airborne, sailplanes were and are the best aircraft to learn how this is done. They are cheaper to produce and to fly, so anyone who wants to learn or perfect its flying skills, can do it easily with a sailplane. This fact was also acknowledge in the military field, where air forces recognize that pilots with sailplane training become more skilled than those who have not. In the Portuguese Air Force Academy, for instance, there are two types of sailplanes whose mission is to provide additional training to pilots and flight experience to non-pilot cadets, so all can experience the free flight.

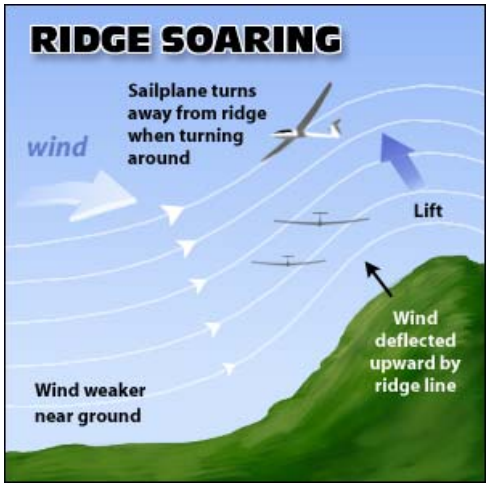
Today, however, most sailplanes are used in recreational and competition soaring. Initially the com-



(a) Cross-country soaring through thermal lifts.



(b) Cross-country soaring through wave lifts.



(c) Cross-country soaring through ridge lifts.

Figure 2.7: Cross-country soaring through atmospheric phenomena (Soaring Society of America, 2010).

petition objective was to increase the duration of the flight but, nowadays, there are competitions that require pilots to fly in races around pre-defined courses. These test the pilots abilities and the sailplanes performance to make best use of local weather conditions. These competitions also boost manufacturers and pilots to maximize sailplane performance and design. The Fédération Aéronautique Internationale (FAI)[3] divides competition sailplanes into seven main classes: Open Class,Standard Class,15 meter Class,18 meter Class,20 meter Two-Seater Class,Club Class and World Class. The first as the name states places no restrictions except a limit of 850 kg in the maximum take-off weight (MTOW). The standard class restricts the sailplane wings span and usage of lift-enhancing devices, as well as imposes a maximum MTOW of 525 kg. The 15 to 20 meters classes also restrict the wing spans and MTOWs, as class names state. The club class allows a wide range of older small gliders within a specified range of performances and the world class allows only the PW-5 glider.

[3]The Fédération Aéronautique Internationale is the world governing body for air sports and aeronautics and astronautics world records.

2.5 Summary

A brief description on the topic of gliding and soaring along it the sailplane aircraft was made in this chapter. First, approaching the history behind the use of sailplanes since the beginning of the aviation history as a research aircraft for flight requirements and conditions, to the present day where sailplanes are used mostly as a recreational activity and air sport. Also, Section 2.3 presented the fundamentals of aerodynamics in general sailplane flight. Finally, the most important types of sailplanes that existed in the past and those that exist actually according to the world governing body for air sports and aeronautics are presented.

Chapter 3

Multi-Disciplinary Analysis and Optimization

3.1 Introduction

This chapter presents the subject of MDA and MDO to the reader. It begins with a description of multi-disciplinary history in aeronautic industry, with a particular mention to its objective. Then, the theory behind each phase of the work developed is presented: from the theory of multi-disciplinary problem definition and approach strategies to the theory behind the MDO tool components.

3.2 Multi-Disciplinary Design History in Aeronautic Industry

An aircraft, from an engineering perspective, can be described as a complex multi-disciplinary system. As said by Ajmera et al. (2004), the design of an aircraft generally consists of a hierarchical and evolutionary sequence of steps starting from conceptual design phase through a preliminary design phase and a detailed design phase ending in a prototype building and testing.

Since the beginning of the aeronautical industry, the design was performed by various individual teams, each with expertise in a specific discipline, such as aerodynamics or propulsion. Although different teams were involved in the aircraft design, they did not work independently of each other. If that were to happen, we would end up with some strange aircrafts like those in the Fig. 3.1. Instead, every team would use its members experience and judgment to develop a workable design, usually sequentially starting from an outline of the shape of the body made by the aerodynamic team, which then, would be fitted with an inside structure by the structures team or modified and fitted with motors by the propulsion team. This methodology in aircraft design was widely used and led to good results in the early times of the aeronautical industry. However, in the 80's there were two major developments that led to the modification of the methodology used by aircraft design engineers. The first was the development of computer-aided design (CAD). The four decades of great computational improvements had

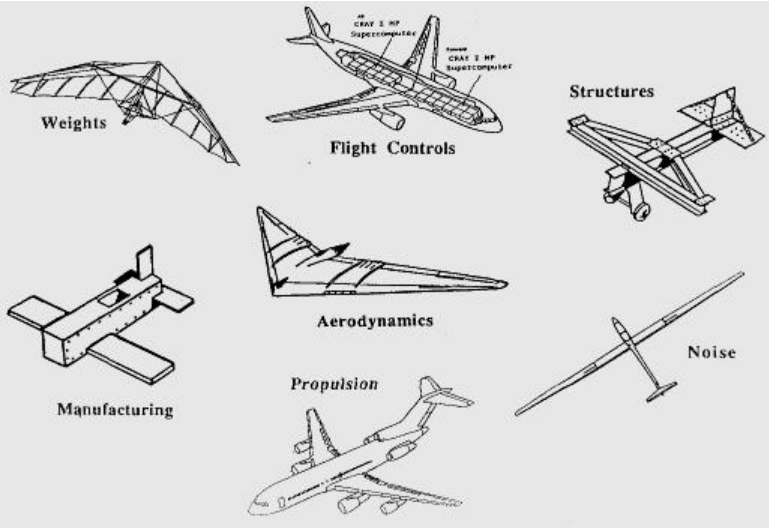


Figure 3.1: Aircraft design by specific design teams (Kroo, 2008).

a deeply effect in the aerospace industry, allowing designers to quickly generate, modify and analyze designs and ultimately validate them much earlier, in the design process, thus potentially saving costly redesigns later in the design stage (Kenway et al., 2010). The second development was the changes in the acquisition policy of most airlines and military organizations, from a performance-centered approach, to one that emphasis life-cycle cost issues. This made economic factors and attributes, known as the ”requirements” (including manufacture, reliability, maintainability), much more important than they were before. These major developments led to the emergence of multi-disciplinary design teams over the one discipline specialized teams. This way, the design process of an aircraft became mainly a compromise between different aspects and opposing constrains of the individual disciplines. Besides, it has been shown that repeated sequential optimization of individual disciplines does not necessarily result in an optimal multi-disciplinary system (Chittick and Martins, 2007).

An aircraft is a multi-disciplinary system as its analysis often require several fields of expertise. Performing computational analysis, together with numerical optimization, made MDO emerge as one of the fields of engineering that can provide optimal solutions to aircraft analysis problems.

The development of fluid flow and structural methods continue to be subject of active research to these days, but it can be considered a mature field. There are tools, such as CFD and computational structural mechanics (CSM), that can actually perform high-fidelity numerical analysis of disciplines like aeronautics and structures in an expedite way. In an MDO, these tools are coupled together to produce an optimal solution between two or more individual disciplines.

As shown in the history of aircraft design analysis, there is a tight coupling between several individual disciplines. Such interactions make aircraft design problems very suitable to the application of MDO methodologies but simultaneously very complex to perform. In theory, MDO strategies are expandable to an infinite number of disciplines, if the engineer or design team can pay the increased calculation cost. To surpass this difficulty, many studies addressed the best way to implement an MDO. These show that a modular scheme is the best way to accommodate the individual disciplines in an MDO framework (Isaacs et al., 2003), thereby making it possible to insert or remove individual disciplinary analysis mod-

ules, reducing the reprogramming costs associated (Alonso et al., 2004).

Today, the majority of the MDO studies focuses on two of the main disciplines of aircraft design, aerodynamics and structures. These, together, form the so called ”basic” aero-structural optimization problem. To solve this kind of problem, there mainly two levels of fidelity analysis. On one hand, low-fidelity analysis, used in the first stages of the design process, to determine the best general characteristics for the aircraft, regarding the requirements defined by the engineer. At this level, one can ignore certain design factors or reduce the level of detail of the analysis, to decrease the computational costs. Although not detailed, this level of analysis is generally used to obtain a comprehensive idea of the main aero-structural characteristics design space, such as lift-to-drag ratio and weight. On the other hand, high-fidelity analysis is used to get smaller improvements, that can refine the esoterical characteristics of the aircraft, such as refining the twist in the wing shape. This level requires detailed discretization of the computational domain and rigorous analysis methods, such as CFD or CSM. As a result, the computational resources required are much greater than that of the low-fidelity. However, it is the high-fidelity analysis that allow the achievement of an optimal detailed design.

Ultimately, there are two factors that have slowed industry's adoption of MDO (Kenway et al., 2010). The first is the increased computational cost and complexity of the optimization problems when running high-fidelity analysis. The second, and perhaps the main factor, is that the inter-disciplinary nature of MDO strategies does not integrate easily into well established aerodynamics and structural design groups. Yet, there are proven cases where MDO has performed successfully in the conceptual and preliminary design stages of aircraft design [(Liebeck, 2004),(Wakayama and Kroo, 1998) and (Kafyeke et al., 2002)].

3.2.1 Multi-Disciplinary Optimization Objective

The main objective of an MDO involving two or more individual disciplines is always to complement the engineer intuition about a design problem in a way that it can allow it to make better design decisions and trade-offs earlier and easier in the design process. For that, MDO will use an optimization process that will evolve the design upon the results of the individual analysis of each discipline according to a certain MDO approach strategy and come to a final result that may not be expected by the engineer. An important part of the objective of MDO is therefore to improve the efficiency of the optimization procedure or, in other words, the time needed to achieve a result. To save time, an engineer can adjust the level of coupling in an MDO problem. This adjustment can, however, lead the optimizer to unfeasible points in the design space where an individual discipline analysis provided a good result but the coupled analysis not. This is a drawback of granting the optimizer more freedom to explore the design space and the main reason why there are different approach strategies in MDO.

3.3 Multi-Disciplinary Optimization Problem Definition

The first step to understand an MDO problem is to know its common definition. In nature, disciplines are physically separated, so in MDO, a separate analyzer is also needed for each discipline. For each, an independent design optimization problem can be defined. These independent problems are often simpler than the overall MDO problem. So the coupling of the disciplines is only addressed in the overall system level. An MDO problem can be seen as a system containing multiple sub-systems, as illustrated in Fig. 3.2. Each of these sub-systems handles a discipline, having implicitly a set of discipline governing equations. These, solved with an appropriate set of inputs, will generate a disciplinary state. A generalized representation for these equations is

$$y_i = f(x_i, y_i, z), \qquad i, j = 1, ..., n \qquad , j \neq i, \tag{3.1}$$

where n is the number of coupled disciplines denoted by i, representing the i^{th} discipline, x_i is the local variable vector, the vector y_j corresponds to interdisciplinary couplings, and z denotes the global variable vector. When provided with a set of design variable inputs, the sub-systems will generate discipline feasible states and outputs. The set of inputs needed for each discipline, consists not only

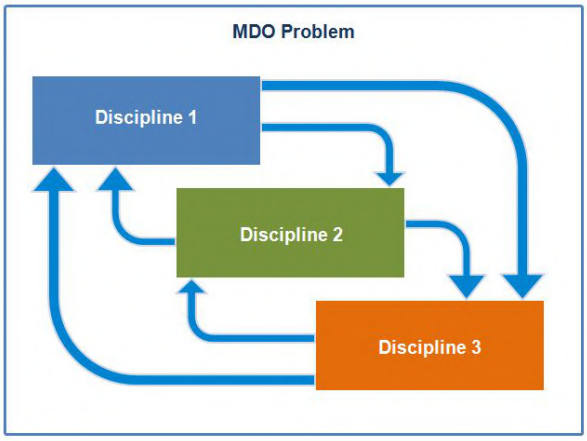


Figure 3.2: Overall system for an MDO problem.

of disciplinary variables but also of coupling variables. The last provide information regarding the state of the other disciplines, needed to solve the overall problem. As said by Martins and Tedford (2006), this relationship between the inputs of one discipline and the outputs of another is responsible for the coupled system of analysis. The common approach to MDO problems is using an iterative block method. When the difference in the coupling variable sets, between successive iterations, is so small that can be neglected, one can say that a convergence criterion is achieved and the MDO problem has reached a solution. In its formulation, the problem can be compared to an simple optimization problem as three entities need to be defined: the objective function, the design variable set and the constraint set.

The MDO problem has two main differences compared to a optimization problem with a single discipline (Yi et al., 2007). First, due to the coupling nature of most MDO problems, the analyzer of each discipline needs inputs that result from the analysis in other disciplines. Second, there are common objective

functions, design variables and constraints, shared by the disciplines. These differences make MDO problems larger and more complex than disciplinary optimization problems. In MDO problems, both the design variable and constraint sets can be grouped based on their effect, out of every discipline and in the global system. Therefore, two types of variables exist: local and global. The local variables are required to only one disciplinary system and are only used to solve that discipline governing equations. Likewise, there are local constraints. In contrast, those variables and constraints that are needed for multiple disciplines, are considered global. There are also two types of optimization constraints. They can set a range of values for the variables (inequality constraints) or they can be residual equations solved only at optima (equality constraints). Martins and Tedford (2006) stated that if instead of requiring the solution to the disciplinary governing equations at each design point, there are equality constraints in the MDO problem, then it can be stated as a standard optimization problem, without the presence of disciplinary sub-systems or a coupled analysis. Ultimately, the way how an MDO problem is converted into one or more standard optimization problems is what defines the MDO strategy or architecture.

3.3.1 Multi-Disciplinary Optimization Architectures

With the definition of the MDO problem comes the need for an MDO architecture, so that the problem turns into one or more standard optimization problems. Then classical optimization algorithms may be employed.

A wide variety of MDO architectures have been proposed and evaluated either by defining a different problem formulation or by finding the most efficient optimization algorithms [MDO Technical Commite (1991) and Sobieszczanski-Sobieski (1993)]. Also, research in the advantages and disadvantages of each MDO architecture has been made by many authors, such as Sobieszczanski-Sobieski and Haftka (1997) ,Tedford and Martins (2009) or Perez et al. (2004). An important aspect of approaching an MDO problem is the fact that its formulation can vary according to the architecture used. Before selecting an architecture, there are some aspects of the MDO problem that have to be examined, like for instance, the number of the disciplines involved, the number of design variables or if they are local, global or coupling variables. Other aspect is the method used to solve the optimization problem created. Currently must studies use gradient-based methods, although some recent research has been made to use non-gradient-based methods such as genetic algorithms, neural networks or simulated annealing.

The MDO architectures can be classified in: single-level methods and multi-level methods. Single-level methods, like Individual Discipline Feasible (IDF) (Dennis and Lewis, 1994), or Multi-Disciplinary Feasible design (MDF) (Cramer, 1994), include only one optimizer at a system-level, which runs a system analysis in each step. This has the authority over the global system and dictates its current state. In each system iteration, the coupled relationship of the disciplines is solved. These methods are simple to implement and are more appropriate for simple problems with two or three disciplines. Yet, as problem complexity grows, a single-level method may not be the best approach, as combining multiple disciplines in a single-level structure can become too difficult. Multi-level methods which include Concurrent Subspace Optimization (CSSO) (Sobieszczanski-Sobieski, 1988) and Bi-level Integrated Systems Synthesis

(BLISS) (Sobieszczanski-Sobieski et al., 1998), create for each discipline a subspace, in which optimizations are made. Each individual discipline has a separate local optimizer, that modifies the design. Also, there is a global optimizer at the system-level, that manages the relationship between disciplines. These methods create a hierarchical structure in the global system. These approaches mimic an industrial setting where each disciplinary sub-group was some degree of freedom to work independently, based on design objectives determined by the system-level optimizer (Martins and Tedford, 2006).

Comparison of Architectures Extensive studies on architecture comparison are referenced [(Balling and Wilkinson, 1996), (Chen et al., 2002) and (Park, 2007)]. However, this topic makes a brief description of some of advantages or disadvantages of the most common architectures.

Yi et al. (2007) made a comparison between MDO methods using mathematical examples. In their study, seven methods were qualitatively and quantitatively compared based on their performance through several mathematical examples. In the end, they have presented their conclusion summarized in the Table 3.1, which presents the need for additional information in the formulation process. Figure 3.3 shows the relationship between the number of function calls and the additional required information, where AAO stands for All-At-Once method (Cramer et al., 1993), CO for Collaborative Optimization method (Braun, 1996), MDOIS for MDO based on Independent Subspaces (Shin and G.J.Park, 2005) and the O/x, signal where additional information is required or not, respectively. The additional required

Table 3.1: Required information for each MDO method (Yi et al., 2007).

Methodology	Design variables	Equality constraints	System analysis	Optimizer	OSA	GSE
MDF	x	x	O	x	x	x
IDF	O	O	x	x	x	x
AAO	O	O	x	x	x	x
CSSO	O	O	O	O	O	O
BLISS	x	x	O	O	O	O
CO	O	O	x	O	O,x	x
MDOIS	x	x	O	O	x	O,x

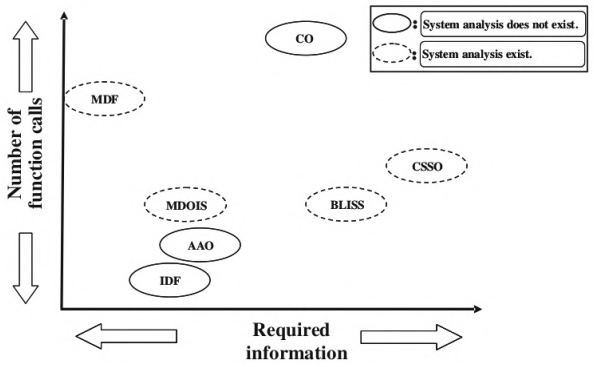


Figure 3.3: Number of function calls versus required information (Yi et al., 2007).

information consists of design variables, equality constraints, system analysis, additional required op-

timizer, Global Sensitivity Equation (GSE) and Optimum Sensitivity Analysis (OSA). The addition of information in design variables or equality constraints makes it difficult to find an optimum solution. Moreover, the number of function calls can increase if additional GSE or OSA information is needed. In sum, MDF is recommended if a system analysis is simple and MDOIS is recommended if each discipline is able to modify the design. Another comparison study made by Perez et al. (2004), evaluated MDO architectures using an extended set of proposed metrics which took into consideration optimization and formulation characteristics. The metrics proposed in this study were simplicity, transparency, portability, efficiency and accuracy. The summary of the results is reproduced in the Table 3.2. The results show

Table 3.2: MDO comparison summary Perez et al. (2004).

	Accuracy	Efficiency	Transparency	Simplicity	Portability
Best	MFD	IDF	MFD	MFD	CO
.	IDF	BLISS	IDF	IDF	CSSO
.	BLISS	CSSO	CO	CO	BLISS
.	CO	CO	CSSO	CSSO	IDF
Worst	CSSO	MFD	BLISS	BLISS	MFD

MDF as the most accurate method since it performs full disciplinary system analysis. Unfortunately, its efficiency suffers with the increase in complexity, so its better used with simple system analysis (Yi et al., 2007). The IDF method is shown to be a feasible alternative to MDF when portability analysis is not an issue. Bi-level optimization schemes proved to be computationally expensive but their accuracy is similar to centralized schemes. Its main advantage lies in the portability for distributed analysis which is in accordance with the study made by Martins and Tedford (2006). This study also presented an MDO framework in Python[1] for comparisons in MDO architecture performance. The conclusion of this study is consistent with the other two mentioned above, as in terms of robustness, MDF and IDF proved to be able to consistently return optimal solutions, with the least number of failures.

As for this thesis, the objective is to perform a simple aero-structural optimization on sailplane wings. Therefore, the methodology in which the MDO problem is to be approached should be the simpler, the most transparent and at same time have high accuracy and robustness. Looking at the results and conclusions of the studies mentioned above, the architectures that seem best adapted to this thesis objective, are the single-level methods, IDF and MDF.

Multi-Disciplinary Feasible The MDF architecture is often viewed as the most traditional approach. In it, an optimizer is placed over an MDA module. This takes in the optimizers set of design variables, optimal global z and local variables x and iterates over the discipline analyses until a consistent set of coupling variables has been generated. Then, the complete variable set is used to compute the values of the objective and constraint functions. The MDA is typically solved by a block-iterative procedure like the Gauss-Seidel iteration[2] and is considered to be converged once the coupling variables generated by each discipline analysis have remained constant within a specified tolerance over successive iterations.

[1] Python is a general-purpose, high-level programming language whose design philosophy emphasizes code readability.
[2] The Gauss–Seidel method is an iterative method used to solve a linear systems of equations.

This architecture requires a solution of the MDA at each design point. This aspect ensures that a multi-disciplinary feasible solution is present throughout the optimization process, so if the optimization is terminated prematurely, a physically realizable design point will still be achieved. The computation of the MDA at each design point also negates the need to include the discipline coupling variables as optimization variables. A schematic representation of the flow of information using MDF architecture is presented in Fig. 3.4. Mathematically, this architecture can be described as

$$\begin{aligned} \underset{z,x}{\text{Minimize}}: \quad & f(z, y_i(x, y_j, z), x), \quad i, j = 1, ..., n \quad j \neq i, \\ \text{s.t.:} \quad & g(z, y_i(x, y_j, z)) \leq 0, \end{aligned} \tag{3.2}$$

where f is the objective function and g represent all the global and local system constraints.
The MDF main advantage is that it ensures that a global feasible solution is present throughout the optimization process. The amount of effort required to implement MDF for a given problem, is directly related to the effort required to build an appropriate MDA module. General iterative solvers, such as block Gauss-Seidel, are commonly used, but may suffer from convergence difficulties. Also an important consideration is that general iterative schemes are much less efficient than fully integrated MDA solvers. For example, if all the discipline analyses are linear, Newton solvers are computationally much more efficient than generalized iterative solvers. However, additional time is need to develop and implement them, if they are not already developed. The sensitivity analysis method employed is also important as the use of gradient-based optimizers with MDF method can result in low performance. If either a finite difference or complex-step method is used to compute the the sensitivities of the objective and constraints with respect to the design variables, an MDA must be solved for each sensitivity which can have a prohibitive cost (Martins et al., 2003). To solve this, Martins and Tedford (2006) proposes the use of semi-analytic sensitivity analysis methods. A disadvantage of MDF is that the space for parallel processing, outside the MDA module, is very limited because the multi-disciplinary system is addressed as a whole. So, if it is to be used within a parallel framework, the computational costs may be higher than if a decoupled approach is used.

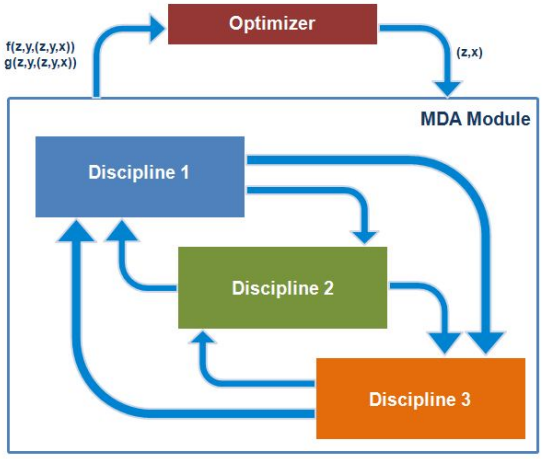


Figure 3.4: Multi-disciplinary design feasible architecture.

Individual Discipline Feasible (IDF) The IDF approach can be seen as a decoupled version of the MDF approach. Instead of enforcing multi-disciplinary feasibility, IDF only requires a discipline feasibility at each design point. At each iteration, the discipline residuals are satisfied but a multi-disciplinary feasible solution may or may not be present. Therefore, if for some reason the optimization process fails at some point, the design solution achieved will be fully realizable in each discipline spectrum but, at an overall system level, it may not be feasible as no coherent multi-disciplinary state is present and the optimization process did not fully converged. As the disciplinary analysis are decoupled, they no longer rely on one another for their coupling variable inputs. Instead, the extra coupling variables are introduced in the formulation, adding coupling variables to the optimization variable set. Then, the optimizer provides each discipline with both design variables and an estimate of the other disciplines coupling variables. To ensure that a multi-disciplinary feasible solution is achieved, one additional feasibility constraint is added to the problem formulation for each coupling variable. These constraints ensure that at the optimum, the estimate of the coupling variables provided by the optimizer and the actual coupling variables are equivalent. A schematic representation of the flow of information using a IDF architecture is in Fig. 3.5. Mathematically, it can be described as

$$\begin{aligned} \underset{z,y^{'}x}{\text{Minimize :}} \quad & f(z, y_i(x, y_j^{'}, z), x) \qquad i, j = 1, ..., n \qquad j \neq i, \\ \text{s.t.:} \quad & g(z, y_i x, y_j^{'}, z), x) \leq 0; \quad y_i^{'} - y_i(x, y_j^{'}, z) = 0 \end{aligned} \tag{3.3}$$

where $y^{'}$ is the extra coupling variable vector created to decouple the disciplinary analysis.
The IDF main advantage is that need for an MDA is eliminated. The disciplinary analyses are uncoupled and can be evaluated in parallel, which can increase its overall performance. This uncoupled nature allows that each disciplinary analysis can be solved only once per design point, making IDF objective evaluation typically less costly than MDF evaluation. As IDF increases both in dimensionality of the optimization problem and in the number of constraints, the number of sensitivity analyses required can become too costly. If the problem has a large number of coupling variables, calculating the sensitivities of the optimization variables with respect to the constraints may be prohibitive since the matrix of sensitivities nearly squares with the number of optimization variables.

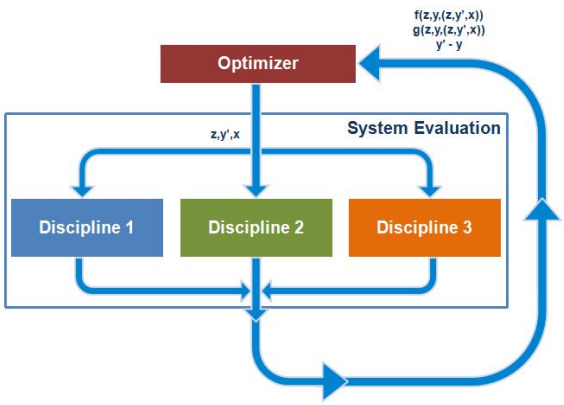


Figure 3.5: Individual discipline feasible architecture.

3.4 MDO Framework

Ultimately, the architecture chosen from the two presented was the MDF. The fact that it ensured a global feasible solution throughout the optimization process made it preferable over the IDF architecture. This section presents the MDA framework established for the MDO architecture chosen to be employed in this thesis. The disciplines for the proposed MDA are aerodynamics and structures, so a solver for each discipline analysis had to be chosen. But the first step required for the disciplinary analysis to be computed was a proper geometric parametrization method. Kenway et al. (2010) presented a method that follows a CAD-free geometry parametrization approach and showed that it presents several advantages over other parametrization techniques. As the study presented good results and the fact that having no need to use a CAD software for the geometric parametrization is an advantage itself, the CAD-free method presented was chosen to be used within the MDO framework. This decision has come with other implications. The study presented was made using tools developed at the University of Toronto (UoT) MDO Lab. Therefore, to reproduce the stated method, access to the same tools was requested. With the access granted, a repository of MDO tools became available. Several tools were tested to find a viable combination for an MDO framework to be established to achieve the objective of this thesis, i.e., running an MDO on sailplane wings. The two disciplines involved in the presented objective are coupled by nature, since aerodynamic loads cause changes in the wings structural deflection, which in turn, changes the aerodynamic characteristics of the wing. So, the MDO framework would need to use two discipline analyzing tools that could be easily coupled together. Using the available repository, two aerodynamic disciplinary solvers and one structural solver were tested. Based on the fidelity of the results, the ease of implementation and the possibility of coupling, two were chosen: a panel code named *Tripan* for the aerodynamic analysis, and a parallel finite-element analysis package named *TACS* for structural analysis. As result of the choice of an MDF architecture only one global optimizer was needed to handle the non-linear, constrained, aero-structural optimization problem. After testing some optimizers with mathematical optimization problems, the optimizer chosen was a sparse sequential quadratic programming (SQP) algorithm for non-linear problems, called SNOPT (Gill, 2008). This algorithm was one of the fully integrated algorithms present in the *pyOpt* MDO Lab module (Perez et al., 2011). This object-oriented framework for formulating and solving non-linear constrained optimization problems was then chosen to handle the optimization process. So the overall scheme of the MDF architecture established for the MDO tool was the on represented in the Fig. 3.6.

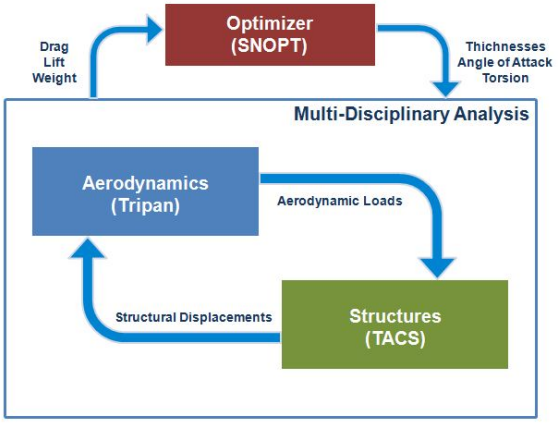


Figure 3.6: MDO framework established for the aero-structural optimization of sailplane wings.

3.5 MDO Tool Structure

This section presents the structure of the MDO tool established in this thesis. As stated in the previous section, the MDO architecture chosen was MDF. However, the MDF architecture only covers the MDA and optimization. So a more wider structure had to be established to the MDO tool. For that structure a modular scheme was chosen. Also an interface had to be elected for the tool. Following the work of the authors mentioned in the previous sections, the interface chosen for the tool was through script files[3]. Although the core components were mostly written in Fortran and C languages, all could be wrapped or were already wrapped in Python language. This fact made clear the choice to go with Python as the scripting language. With the interface, disciplinary solvers and optimizer components chosen, the structure to the final MDO tool was created. Figure 3.7 shows the scheme of the overall MDO tool structure established and used to run the aero-structural optimization of sailplane wings. The next sections will address to each module in the established MDO tool.

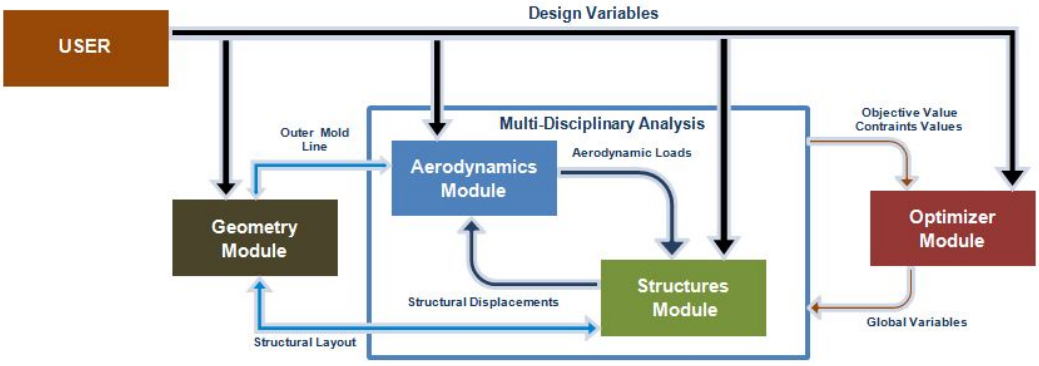


Figure 3.7: MDO tool structure established for the aero-structural optimization of sailplane wings.

[3]A script is a file written in a scripting language or extension language that allows control of one or more applications. Usually the scripting language is different from the core programming language of the application it controls to provide the user a more comprehensive interface through a higher level programming language.

3.6 Geometry Module

The geometry generation and parametrization is of fundamental importance to an MDO problem. This section presents the geometry module of the MDO tool. For that, a first sub-section about aircraft wings will be presented, followed by a sub-section with information about the employed methods. A last sub-section will address to the script created to use this module.

3.6.1 Case Study

The subject of study in this thesis are sailplane wings. Two case studies were chosen: an academic semi-tapered sailplane wing and a real sailplane wing. The general aerodynamic characteristics of sailplane wings have been already been mentioned in Chapter 2. Therefore, this sub-section will focus on the presentation of the L-23 Super Blanik sailplane and the main structural components of sailplane wings. The LET L-23 Super Blanik is one of the most popular sailplanes in the world of flight instruction. Due mainly to the excellent cost-to-performance ratio, it was a success throughout the world. In its composition, it is an ”all-metal” sailplane. It exists in two versions, single and two-seater sailplane. It has a simple and robust cantilever, mono-spar (supported by an auxiliary spar), tapered, high-wing and a T-tail. The wing also has some twist, sweep and dihedral. Last but not least is the fact that, nowadays, it is one of the two sailplanes operated by the Department of the Air Corps Student Activities, in the Portuguese Air Force Academy. Other countries like Brazil or U.S.A., have also adopted this glider as their instruction sailplane. Figure 3.8 shows a cutaway of the L-23 sailplane.

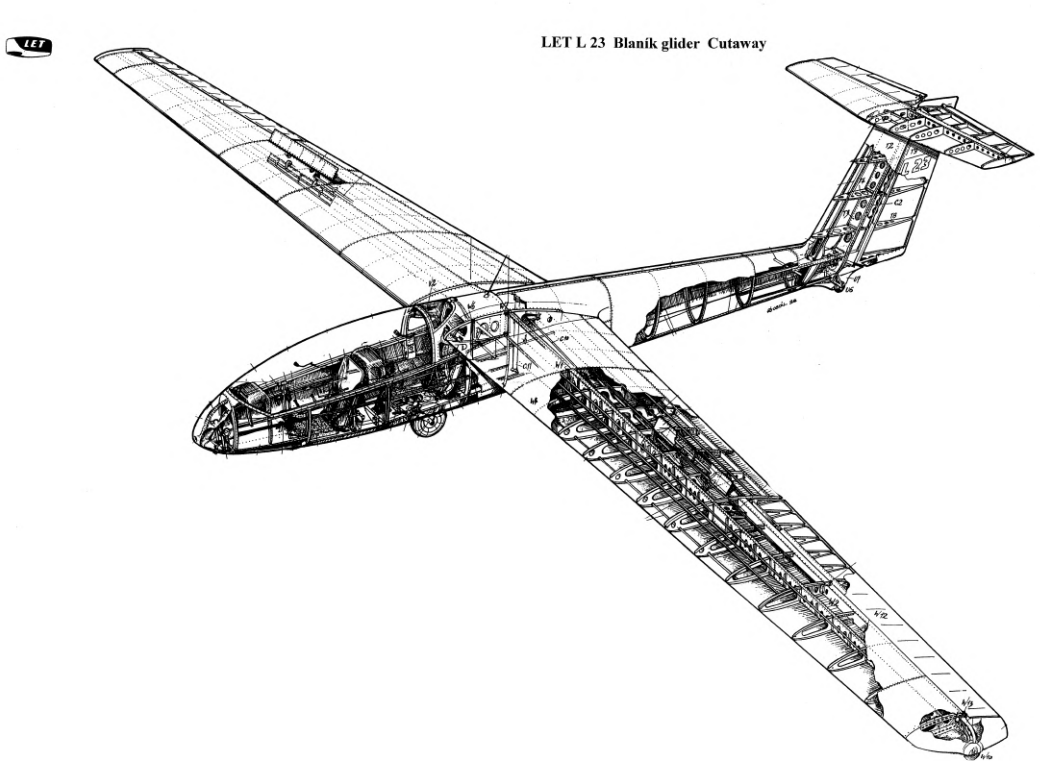


Figure 3.8: Cutaway of the L-23 Super Blanik sailplane (L-23 sailplane maintenance manual, 2011).

Wing Components The main structural components that form a sailplane wing internal structure are (Megson, 2007):

Spars The spar is often considered the main structural member of the wing. It is the component that supports the main aerodynamic loads and serves as structural interface between the wing and the fuselage. Generally, this component is configured approximately normal to the flow direction, extending outward from the fuselage to the wing tip, although it can be placed with a certain angle related to the sweep angle of the wing. The most common current wing structural layout for personal aircrafts is the cantilever, where a single large beam, designed as the main spar, is placed nearer the leading edge at about 25% of the total chord to carry the lift load through the fuselage to the other wing. To resist forward and aft movement, the wing has usually a second smaller drag-spar near the trailing edge. This auxiliary spar is tied to the main spar with ribs or a stressed skin (forming a wing-box structure) providing the wing rigidity needed either in flight or on the ground.

Ribs The ribs are structural members of the wing and often considered its skeleton. They serve as connectors between spars and are responsible along with stringers for the support and shaping of the skin, also they serve as attachment points for control surfaces, flaps, undercarriage or engines. In the traditional layout, ribs are placed so they have the orientation of the flow direction.

Stringers In conjunction with the ribs the stringers are structural members that are responsible for giving an airfoil shape to the wing skin as well as to help support wing bending and act as interrupter to the spread of cracks. They have the function of supporting the skin panels and prevent thin-wall buckling under compression or shear loads. In the traditional layout, stringers are placed between ribs, assembled to the skin and oriented parallel to the spar direction.

Skin The last structural member described is the skin on the wing. The skin in conjunction with ribs and spars compose a wing-box structure. Generally, the wing skin consists of panels comprising the top and bottom of the wing-box attached to each other and attached to ribs giving the wing its aerodynamic shape.

Figure 3.9 shows the generic components of a traditional wing-box structure.

3.6.2 CAD-Free Method for Geometry Generation and Parametrization

To use high fidelity tools like CFD and CSM, detailed discretization of the computational domain are required (the accuracy of the solutions may depend on it). In CFD analysis, a model of the *"wetted surface"* or *"outer mold line"* (OML) of the wing is required. Also the computational domain for a CFD analysis is three dimensional and may include a large number of nodes. On the other hand, CSM analysis of that same wing, require not only the OML but also a description of the internal aircraft structure components like ribs, skins, spars and stiffeners. So to chose the geometric parametrization method some research was made. Samareh (2001) presented a survey of shape parameterization techniques

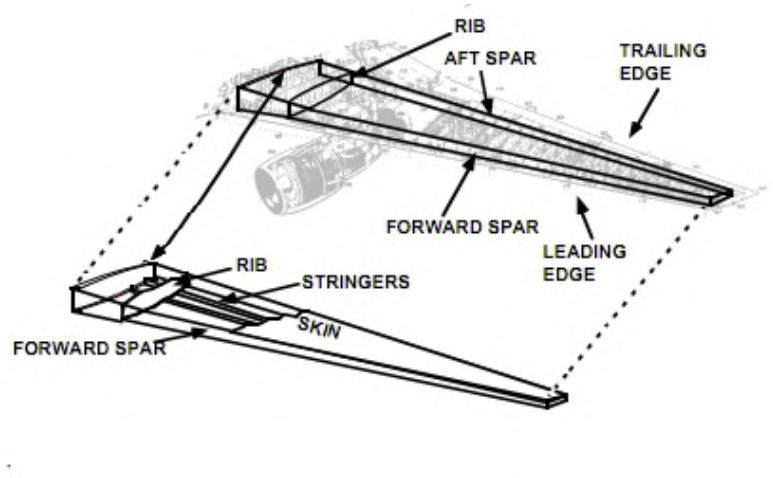


Figure 3.9: Representation of a wing-box configuration.

for high-fidelity MDO, where he identifies eight types of geometric parameterizations: basis vector, domain element, partial differential equation, discrete, polynomial and spline, CAD-based analytical and free-form deformation (FFD). Kenway et al. (2010) presented a CAD-free approach method that uses both spline and FFD volume based approaches. This method presented various advantages like the efficient computation of analytic derivatives for gradient-based optimization. Thus, it was the method followed and employed in the geometry module for the geometric parametrization of the proposed MDO. In order to implement the method in the geometry module, a set of geometry tools from the MDO Lab was used. These tools include functionalities with both B-spline curves and surfaces. The tools are called *pySpline* and *pyGeo* and their usefulness will be detailed in the next paragraphs.

pySpline is a underlying B-spline library for curves and surfaces developed by Gaetan Kenway for the MDO Lab of the UoT. Its evaluation routines are written in Fortran90 and wrapped using f2py [4] (Martins et al., 2001) to provide a high-level, object-oriented application programming interface in Python. The recurrence relations for the B-spline basis functions are

$$N_{i,0}(u) = \begin{cases} 1 & \text{if} t_i \leq u \leq t_{i+1} \\ 0 & \text{otherwise} \end{cases}, \tag{3.4}$$

$$N_{i,p}(u) = \frac{u - t_i}{t_{i+p} - t_i} N_{i,p-1} + \frac{t_{i+p+1} - u}{t_{i+p+1} - t_{i+1}} N_{i+1,p-1}, \tag{3.5}$$

where u is the parametric location with respect to knots ti. The parametric equations for B-spline curves and surfaces can then be written as

$$C(u) = \sum_{i=0}^{N_u - 1} N_{i,p_u}(u) P_i, \tag{3.6}$$

$$S(u,v) = \sum_{i=0}^{N_u - 1} \sum_{j=0}^{N_v - 1} N_{i,p_u}(u) N_{j,p_v}(v) P_{i,j}, \tag{3.7}$$

[4] F2PY is a tool that provides an easy interface between Python and Fortran languages.

$$V(u,v,w) = \sum_{i=0}^{N_u-1} \sum_{j=0}^{N_v-1} \sum_{k=0}^{N_w-1} N_{i,p_u}(u) N_{j,p_v}(v) N_{k,p_w}(w) P_{i,j,k}, \tag{3.8}$$

where the control points $p_{i,j}, i = 0, ..., N_u - 1; j = 0, ..., N_v - 1; k = 0, ..., N_w - 1$ exist is three spatial dimensions. The $N_{i,p_u}, N_{j,p_v}, N_{k,p_w}$ functions are the polynomial B-spline basis functions of degree p_u, p_v and p_w, respectively. One of the biggest advantages of B-splines is the analytic formulation of their derivatives. So, differentiating the basis functions l times, the l^{th} order derivative with respect to the parameter value can be evaluated. Hence the derivatives for curves, surfaces and volumes are, respectively,

$$\frac{\partial^l}{\partial^l u} C(u) = \sum_{i=0}^{N_u-1} N_{i,p_u}^{(l)}(u) P_i, \tag{3.9}$$

$$\frac{\partial^{l+m}}{\partial^l u \partial^m v} S(u,v) = \sum_{i=0}^{N_u-1} \sum_{j=0}^{N_v-1} N_{i,p_u}^{(l)}(u) N_{j,p_v}^{(m)}(v) P_{i,j}, \tag{3.10}$$

$$\frac{\partial^{l+m+n}}{\partial^l u \partial^m v \partial^n w} V(u,v,w) = \sum_{i=0}^{N_u-1} \sum_{j=0}^{N_v-1} \sum_{k=0}^{N_w-1} N_{i,p_u}^{(l)}(u) N_{j,p_v}^{(m)}(v) N_{k,p_w}^{(n)}(w) P_{i,j,k}. \tag{3.11}$$

The first derivative of the B-spline basis function is expressed as

$$N_{i,p}^{(l)} = \frac{p}{t_{i+p} - t_i} N_{i,p-1}(u) - \frac{p}{t_{i+p+1} - t_{i+1}} N_{i+1,p-1}(u). \tag{3.12}$$

The compact nature of the B-spline basis functions results that most $p + 1$control points in a given direction will affect a fixed parametric location. Moreover, the linear nature of the B-spline shape functions results that the derivative of a point in a volume at parametric location u, v, w, with respect to a control point $P_{i,j,k}$ is the product of shape functions expressed as

$$\frac{\partial V(u,v,w)}{\partial P_{i,j,k}} = N_{i,p_u}(u) N_{j,p_v}(v) N_{k,p_w}(w). \tag{3.13}$$

For configurations of some complex geometry, as an airplane wing, using only isolated curves, surfaces or volumes are not enough. It is necessary to combine the entities together in some topological manner. *pyGeo* is the tool that handles this function, working as a geometry surfacing engine. It performs multiple functions including producing surfaces from cross sections, fitting surfaces and has built-in design variable handling. The actual B-spline surfaces are of the *pySpline* surface type and come from the previously seen tool, *pySpline*. *pyGeo* is able to generate, from argument inputs, lifting surface objects (from splined surfaces), load in a plot3D surface patches and load the surface patches from an IGES format file[5]. *pyGeo* has also the capability to generate outputs with the geometry object (splined surfaces) in Tecplot Data file format [6] or IGES file format. Figure 3.10 shows examples of B-spline surface geometries generated with *pyGeo*.

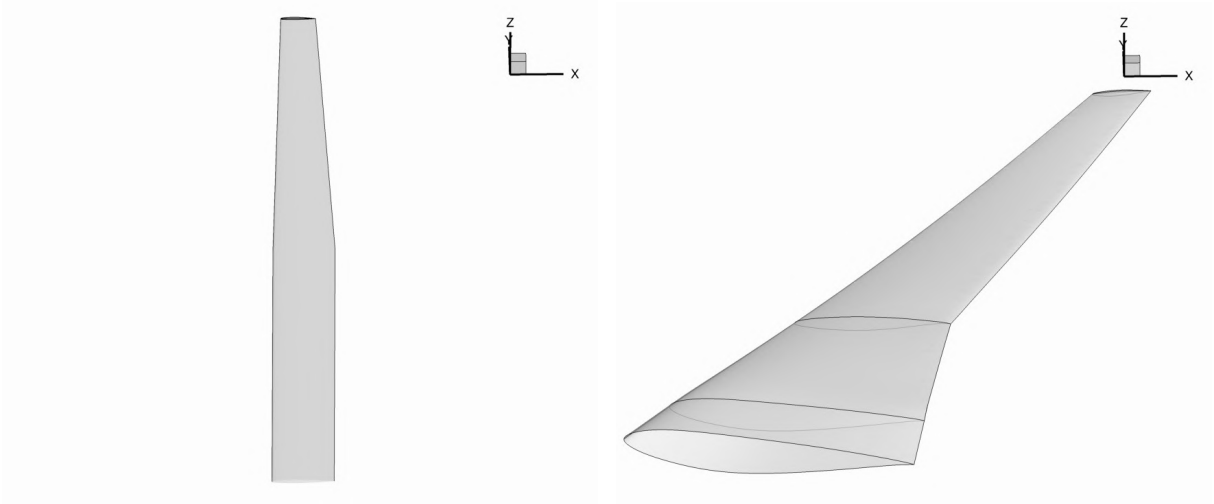
Finally, the tool used to generate the finite-element analogues to the structural members within the

[5]The Initial Graphics Exchange Specification (IGES) is a file format which defines a vendor neutral data format that allows the digital exchange of information among Computer-Aided Design (CAD) systems.

[6]Tecplot is the name of a family of visualization software tools developed by Tecplot, Inc.. Tecplot 360 for example is a CFD and numerical simulation software package that is generally used for post-processing simulation results.

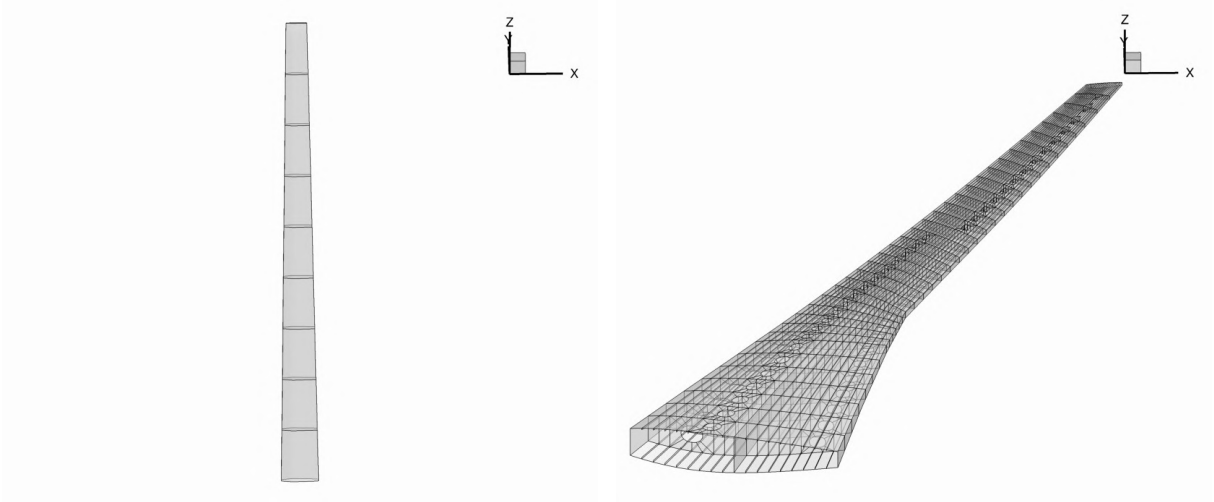
wing is called *pyLayout*. This Python module is used for automatic parametric structure generation of wings. Given a description of the structural layout within the OML of the wing, *pyLayout* automatically generates a wing-box finite-element model that mimics the structural characteristics of the real wing. The description required can be simple, with only the required information of position and number of ribs and spars, or complex, with various optional informations like the element order for the finite-elements, the number of elements between each rib (span-wise), the number of elements between each spar (chord-wise), the number of elements in the thickness or the number and positioning of holes in ribs, spars or skins. This module possesses features that allow it to generate a wide range of structural layouts, from simple wing-box finite-element models like the one in Fig. 3.11(a), to more complex wing-box finite-element models like the one in Fig. 3.11(b).

After generating the structure, *pyLayout* module creates an output file that can be loaded by the structures module.



(a) B-spline surface representation generated from input file. (b) B-spline surface representation generated from IGES file.

Figure 3.10: B-spline surface representations generated with *pySpline* and *pyGeo*.



(a) Simple wing-box finite-element model. (b) Complex wing-box finite-element model.

Figure 3.11: Wing-box finite-element models generated by *pyLayout* module.

Free-form Deformation The CAD-free geometry parametrization method uses a technique known as FFD. This technique is an approach to the free-form solid modeling used in soft object animation in the computer graphics field. It was first presented by Sederberg and Parry (1986). A good physical analogy that is often used to explain the FFD approach, is the one where an object (or objects) that one wants to deform, is embedded in a clear, flexible, plastic material. The object is assumed to be flexible, so that it deforms along (in a consistent motion) with the material surrounding it. This material containing the object usually takes form of a simple geometric shape and usually is mapped using a $R^3 \rightarrow R^3$ map, like tri-variate Bézier or B-spline volume. With this technique, any object can be embedded inside the volume. To perform that, a Newton search can be used to determine the u, v, w values, mapping parameter space to physical space. After the embedding, the objects can be indirectly deformed by making high level modifications to the FFD volume geometry. The use of this technique allows easier parameterizations of solid object models since it is not the object geometry itself that is parametrized but the volume where it is embedded. As a consequence, this technique only uses a set of design variables which will produce the desired modifications to the object, rather than the objects geometry itself. Although it may have the disadvantage of making the design variables not have physical meaning for design engineers, this technique can be adapted to be used in MDO applications (Samareh, 1999).

3.6.3 Code for the Geometry Module

This sub-section presents a generic script for the parametrization of a wing geometry to be used in an MDO problem. To lighten this document, it is only presented a brief description of the functions of each section of the script. The full transcription of the script, along with the programming comments, can be found in Appendix A. The script is structured in five parts and its structure is represented in Fig. 3.12.

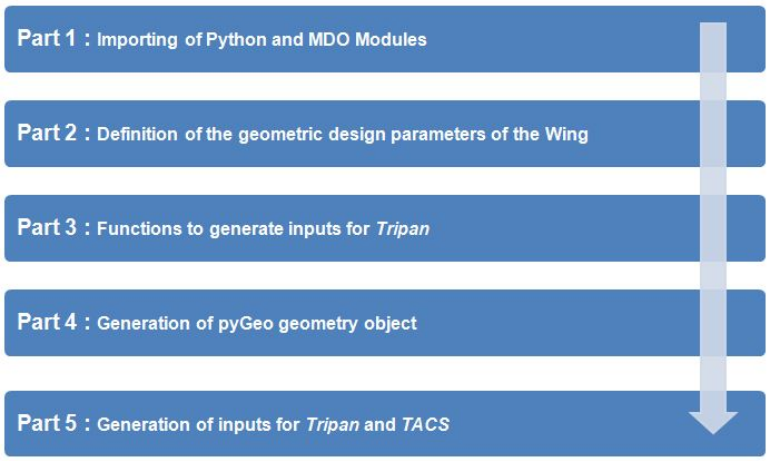


Figure 3.12: Script structure to generate a wing geometry and structural layout.

The first part has the lines needed to import standard and external Python modules as well as the specific MDO extension modules. In part 1.1, some specific functions from the scientific computing package for Python (NumPy)[7] are imported. Also an the MPI extension of package mpi4py [8] is imported,

[7]For detailed information on NumPy package for Python see http://numpy.scipy.org/.
[8]MPI for Python is a Message Passing Interface (MPI) standard package for the Python programming language. It

to allow this script to be extended to run in multiple processors. In part 1.2., the MDO Lab tools are imported, namely, the *pyGeo*, *pySpline* and *pyLayout*, reviewed in Sub-section 3.6.2.

The second part handles the geometry variables. It is in this part that the user sets the variables to obtain the desired wing geometry, either in terms of OML and in terms of structural layout. These variables are span, wing chord at tip and root (it can be also defined with the tapper ratio of the wing) and airfoil shapes at specified sections in the wing. Each of these sections has a set of variables associated: span location, chord, position of the leading edge and rotation. By manipulating these values, the user can add sweep, dihedral and twist angles to the wing geometry. Part 2.2 handles the information about the inner structural layout. First the number of spars, ribs and stringers are defined. Then, the position of the leading and trailing edge spars are specified. Finally, there are some options on the order and number of the finite-elements to be used for the structural mesh. Also, the number of control points used by *pyGeo* to generate the wing geometry object can be defined.

Part 3 should not be modified by the user as it contains the code that defines the functions to generate .wake, .edge and .tripan files with the panel mesh for *Tripan*.

The forth part of the script calls *pyGeo* to generate the geometry object.

The last part defines the code for generating flow and structural solver input files.

Once the script is run, the output files for the aerodynamic and structural analysis are generated and can be used as presented in Sections 3.7 and 3.8.

3.7 Aerodynamics Module

This section presents information about the module that handles the aerodynamic analysis. A first sub-section introduces computational methods for aerodynamic analysis. Then, a sub-section addresses the methods employed. Finally, a sub-section documents an example script used to implement and test the aerodynamics module.

3.7.1 Computational Methods and Computational Aerodynamics

Computational Methods In Sub-section 2.2, it was highlighted that in the early years of the aviation history, the developments in aircraft design resulted mostly from experimentation. Today, computational methods have become standard practice, resulting from the fast and exponential evolution of electronics and digital computers over the last three decades. They allowed the emergence of new complex calculation methods, only possible through numerical analysis, that revolutionized a wide range of disciplines. Computational methods have become an indispensable tool to engineers in aircraft design. Alonso (2011) gives a general overview of the capabilities of computational methods and potential problems that they present. From that overview, it is worth to highlight the fact that computational methods allow the analysis of the behavior of complex systems that otherwise would not be possible using just analytic theories, they also allow the reduction of the costs of design and production by giving detailed

allows any Python program to exploit multiple processors. For detailed information on Mpi4py package for Python see http://http://mpi4py.scipy.org/.

information without the need of experimentation. However, it is important to know that computational methods are just tools and, therefore, have to be used with caution. There are four main aspects to avoid problems from the misguided use of these methods (Alonso, 2011): the model used must be in accordance with the phenomena that is being analyzed, otherwise the solution will not be valid; the accuracy of a numerical solution is heavily dependent on the domain discretization used; the solution of a computational method is only valid in the range of the model used; the solutions obtained should be carefully observed to assess if they follow the expected trends, based on the theoretical principles.

Computational Methods applied to Aerodynamics As with all the major aerospace disciplines, aerodynamics was also changed by the use of computational methods. It was in 1967 that the first paper on a practical three-dimensional method to solve the linearized potential equations was published at the Douglas company (Hess and Smith, 1967). The presented method discretized the surface of the geometry with panels, giving birth to a new class of programs that would use the surface panel methodology, the *"Panel Methods"*. Although the first method was simple and did not include lifting flows, many researchers and aircraft companies realized its potential. During the following years, additional capabilities were added, as in 1968, when Paul Rubbert and Gary Saaris of Boeing Aircraft introduced the fist lifting Panel Method. Many more companies like, Lockheed, McDonnel Aircraft or NASA followed the trend and began to create and use the panel methodology, to run aerodynamic analysis. Today, panel methods have many more capabilities like the use of higher order codes, the solution of unsteady flows or the coupling with boundary layer formulations. This versatility made them gain wide-spread acceptance, throughout the aerospace industry. The big advantages that this computational method presented, and still presents, are simplicity and speed. The fact is that, any modern personal computer can easily setup and perform the analysis of potential flows around bodies, given an appropriate panel code. Nevertheless, the *" solution is only as good as the model that is being solved"*, so it is important to acknowledge that panel methods have limited applicability when high-speed non-linear flow is present. Even though panel methods represented a revolution in the field of aerodynamics, they were just the beginning. Figure 3.13 gives a good illustration on the evolution of computational methods in aerodynamics. As the complexity of the modeled flow is increased, more complex computational methods are required. Though panel methods with boundary layer corrections can provide quite accurate predictions of lift and drag when the flow remains attached, when the inviscid outer solution interacts with the inner boundary layer, the solution becomes increasingly difficult to obtain with the onset of separation. Also when the modeled flows have high Reynolds numbers, the turbulent effects have to be evaluated and taken into account, usually by Reynolds averaging of the fluctuating components. This requires a turbulence model like Reynolds-averaged Navier-Stokes (RANS) equations, which is the most common approach to turbulence modeling. As the computing power is growing rapidly, new and more accurate methods are rising, as Large Eddy Simulation (LES), in which the smallest scales of the turbulent flow are removed through a filtering operation and their effect modeled using sub-grid scale models. This allows the largest and most important scales of the turbulence to be resolved. In time, even the extremely computational costing Direct Numerical Simulation (DNS) will be generally used, allowing the resolution

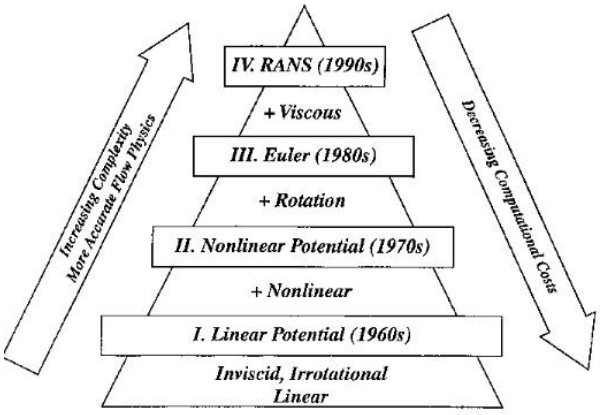


Figure 3.13: Hierarchy of aerodynamic models with corresponding complexity and computational cost (Alonso, 2011).

of the entire range of turbulent length scales through marginalizing the effect of models.

3.7.2 Aerodynamic Analysis

Modern gliders have average speeds to fly is in the range of 20m/s to 30m/s, which for an average flight altitude of 1000 meters, give low Mach numbers. This is an important fact, as it allows the airflow to be considered incompressible. Also, modern sailplanes are designed to be smooth and have wing geometries that avoid flow separation and minimize viscous effects. Therefore, is a valid assumption to consider an inviscid, incompressible and irotational model to accurately simulate flow in which a sailplane flies. Reminding Sub-section 3.4, two flow solver were tested: the *SUmb* and the *Tripan*. *SUmb* is a multi-block structured flow solver developed in the Stanford University Center for Integrated Turbulence Simulations (CITS). It is a code that solves the compressible Euler, laminar Navier-Stokes and Reynolds-Averaged Navier-Stokes equations (Weide et al., 2005). On the other hand, *Tripan* is an unstructured, three-dimensional panel code. The two modules could be used for the aerodynamic analysis, however, as the course of the work revealed, the *Tripan* flow solver was better suited to be coupled with the structural solver. As the other modules presented for the geometry parametrization, the *Tripan* module is wrapped in Python, allowing the easy integration in the established MDO tool. These reasons led to the choice of this panel code as the aerodynamic solver for the MDO tool. Though not implemented, *SUmb*, as an high-fidelity model, was validated and used to verify the accuracy of *Tripan*, as Sub-section 4.3.1 will present.

Tripan uses a first-order panel method with constant source and doublet singularity elements, distributed over the surface of a body, discretized with quadrilateral and triangular panels. Further information on panel methods and its implementation can be found in references [(Anderson, 2001) and (Hess and Smith, 1967)]. This method allows the calculation of aerodynamic forces, moments and pressures for inviscid, incompressible, external lifting flows. Yet, it has well known limitations, especially of accuracy when computing drag (Smith, 1996).

To perform the aerodynamic analysis on the proposed MDO, the *Tripan* panel method determines the source strengths based on the onset flow conditions while the boundary conditions for the doublet

strengths constitute a dense linear system, represented by

$$\mathbf{A}(u, w) = 0, \tag{3.14}$$

where u and w are the vectors of the structural and aerodynamic state variables. The linear system represented in Eq. 3.14 is solved using the parallel, linear algebra routines in PETSc [9] (Balay et al., 2004) and using the Krylov subspace method GMRES [10] (Saad and H.Schultz, 1986), with a block Jacobi Incomplete LU(ILU) preconditioner formed using a sparse approximate-Jacobian. The study presented by Kennedy and Martins (2010) shows that this is an effective method, requiring the least overall computational time, amongst the range of preconditioning options considered. The preconditioner is assembled by considering only those panels that are within a predetermined radius from the current panel centroid (closest panels have the strongest effect on a given panel). The effect of a higher ILU preconditioner fill-level can be achieved by selecting a larger radius. A final important feature in the *Tripan* is that, it is implemented with an adjoint sensitivity method which will be discussed in Sub-section 3.11.2.

3.7.3 Code for the Aerodynamic Analysis

In the scheme presented in Fig. 3.7, the aerodynamics module receives input files from the geometry module and then it communicates with the structural module to perform the coupled aero-structural analysis. Though this is the final purpose of this module, to better understand how it works and how the user can define the aerodynamic design variables, this sub-section presents a description of a generic script for a simple aerodynamic analysis. The a full script with programming comments is listed in Appendix B. Figure 3.14 shows a representation of the script structure used.

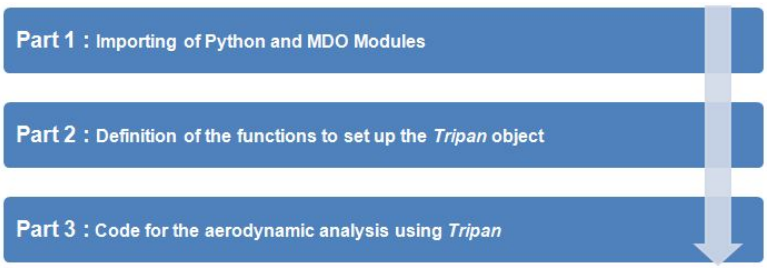


Figure 3.14: Script structure to perform an aerodynamic analysis.

The script is structured in three parts. The first one is used to import the Python and MDO extension modules, namely *Tripan*, and to define the output directory for the results.

The second part is used to set up *Tripan* panels. It should be used as presented in the Appendix B.

Part 3 is the core of the script, as it is here that the input files are defined and the aerodynamic analysis is performed. It is divided in four sub-parts. The first one is here the user can define the input files generated with the geometry module. Then the script calls the setting up of the *Tripan* object and

[9] PETSc is an open-source suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations.

[10] The generalized minimal residual method (GMRES) is an iterative method for the numerical solution of a system of linear equations. The method approximates the solution by the vector in a Krylov subspace with minimal residual.

solver. The second sub-part is here the user can define the design parameters for the airflow model. These properties are the Mach number, the air density, speed of sound and angle of attack. Air speed and dynamic pressure are calculated from the defined properties. The third sub-part sets the *Tripan* load case and handles the solution of the aerodynamic analysis using the implicit GMRES method. The last sub-part handles the outputs. The possible outputs are the surface solution, the wake, the aerodynamic loads, the lift distribution (semi-span) and the sectional pressure coefficient.

3.8 Structures Module

This section presents the second disciplinary module of the MDO tool. A first sub-section is dedicated to computational methods. Then, a sub-section presents the the methods and the tools employed in the structures module. Lastly, a sub-section documents a generic script for the structural analysis.

3.8.1 Computational Methods for Structural Mechanics

As the computational power rapidly evolved, analytical methods have given their space to more accurate and flexible numeric methods. The discipline of structures is the perfect proof. This discipline addresses to structural analysis, whose focus is the determination of the effects of loads on physical structures and components. There are many branches in this discipline, often related to the type of structures that is being studied, as for example aircraft structures. Structures can also be viewed as a wider study field of other disciplines like applied mechanics or materials science. This makes it one of the main disciplines in aircraft design. Throughout the history of structural analysis, various methods were developed to predict the behavior of structures under the effect of loads. From the Euler–Bernoulli beam equation, in the 18th century, to the introduction of the name ”finite-element method” (FEM) in 1956 by Turner et al. (1956), many methods were presented and used to perform structural analysis and determine information, such as structural loads, geometry, support conditions, and materials properties. Today, however, this analysis is performed using mainly two approaches, the analytic methods, such as the mechanics of materials or the elasticity theory and computational methods like the finite-element approach. The first approach uses analytical formulations whose applicability is limited to simple linear elastic models, leading to closed-form solutions. The second approach uses actually numerical methods for solving differential equations generated by structure mechanics theories, such as elasticity theory and mechanics of materials. Similarly to computational aerodynamics, structural analysis software are composed of numerical methods used to solve the discretized structural equations of motion, on a suitable mesh, created from the geometry of the structure. These programs are also being more and more used in aircraft structural design, to optimize the shape and properties of structures and materials. Given the proper time and study, their integration in MDO frameworks will be even more used.

3.8.2 Structural Analysis

Structures represent the second discipline in the proposed MDO. The methods and tools chosen to perform the structural analysis followed the most recent studies published by Kennedy and Martins (2010), Kennedy (2011) and Kenway et al. (2010). In these, structural analysis was performed by a finite-element code developed by Graeme J. Kennedy of UoT. This code, called Toolkit for the Analysis of Composite Structures (*TACS*), has been tested and it has been developed to have an easy coupling with aerodynamic codes for MDAs. This reason made it the elected tool for the structures module.

TACS was designed for the analysis of stiffened, thin-walled, composite structures using either linear or geometrically non-linear strain relationships. It can use higher-order finite-elements to enhance the stress prediction capability. The residuals of the structural governing equations are expressed as

$$\mathbf{S}(u, w) = \mathbf{S}_c(u) - \mathbf{F}(u, w), \tag{3.15}$$

where where u is a vector of displacements and rotations (structural state variables), w is a vector of aerodynamic state variables, S_c are the residuals due to conservative forces and internal strain energy and F are the follower forces due to aerodynamic loads. The Jacobian of the structural residuals involves two terms. The first is the tangent stiffness matrix $\mathbf{K} = \partial \mathbf{S}_c / \partial u$. The second is the derivative of the consistent force vector with respect to the structural displacements. These terms are computed using a matrix-free approach. Mathematically the Jacobian of the structural residuals is represented by the expression in Equation 3.16.

$$\frac{\partial \mathbf{S}}{\partial u} = \mathbf{K} - \frac{\partial \mathbf{F}}{\partial u}. \tag{3.16}$$

TACS uses the Krylov subspace method GMRES and the the Krylov method GCROT [11](Hicken and Zingg, 2010), to solve the non-symmetric, linear systems of Eq. 3.16. It handles stress constraints by applying a local failure constraint at each Gauss point in the finite-element model. These local failure constraints compute a load factor, λ_k, required for that point to fail. The load factor implies that the current point will fail at λ_k times the current stress level. For a safe-life design, the criterion $min\{t_k\} > Fs$ is applied, where Fs is the safety factor. This method applied to an optimization has some specificities. Instead of using the minimum value directly, a Kreisselmeier-Steinhauser (KS) constraint aggregation technique is applied to groups of these local constraints (Wrenn, 1989). Normally these groups are aggregated amongst similar structural components. In *TACS* code, the KS function is computed as

$$\lambda_{KS} = min\{\lambda_k\} - \frac{1}{\sigma} ln \left[\sum_{i=1}^{N} exp\{-\sigma(\lambda_i - min\{\lambda_k\})\} \right], \tag{3.17}$$

where σ is a weighting parameter that controls the degree of approximation and λ_{KS} is the aggregated KS value. This approach has the advantage that it reduces the number of constraints required in the optimization, while keeping a conservative approximation, in that λ_{KS} is a lower bound. Values of σ between 30 and 50 are recommended.

[11] GCROT is a Krylov method that uses Generalized Conjugate Residual with inner orthogonalization and Outer Truncation.

A final mention to the fact that *TACS*, although written in C++ language, contains a Python interface. This allows the easy modification and flexibility in building the desired finite-element models.

3.8.3 Code for the Structural Analysis

This sub-section documents the script created to test the structures module of the established MDO tool. To achieve this objective, a simple structural analysis of a generic wing was run, in which a single point load to the tip was applied. The input structural mesh of the wing was created with the script presented in Sub-section 3.6.3. Figure 3.15 shows a representation of the script structure used to perform a structural analysis. The full transcript of the script can be found in Appendix C.

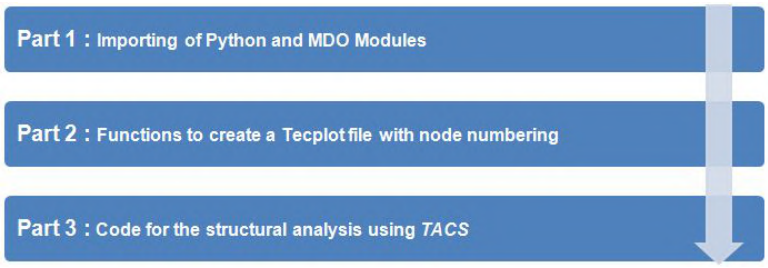


Figure 3.15: Script structure to perform a structural analysis.

The script is structured in three parts. The first part handles the importation of Python and MDO extension modules and the definition of the output directory for the results. The MDO extension modules imported are, 'TACS', 'elements', 'contitutive' and 'functions'. The first one is the *TACS* core code, the second and the third are the extensions with the functions needed to define the finite-element class and the constitutive class. The last one handles the functions needed for auxiliary calculations.

The part 2 has an auxiliary function code to create Tecplot visualization file with the numeration of the nodes in the structural mesh. This is important as it is only when the *TACS* object is created that the numeration of the nodes is done. This is required if a nodal point load is to be applied.

Part 3 is the core of the script. It starts with a sub-part used to load the structural mesh file created with the geometry module. There, the variables 'nribs','nspars' and 'ncomponents' store the number of ribs, spars and components. The second sub-part handles the set-up of the KS function domains, the definition of the design material properties and the finite-element type used in the structural mesh. Part 3.3 has the code for the creation of the *TACS* object and KS functions. The first is created assigning six variables to each node, for the six degrees of freedom. Also the number of load cases that with be computed is defined. Then, the KS functions are created for each domain defined above. Part 3.4 is where the user defines the design parameters for each load case. In the example listed in Appendix C, a point load of 500 N is applied to a node at the tip of the wing-box structure. The fifth sub-part operates the setting of the parameters for the solver. Sub-part 3.6 processes the solving of the structural system. The last sub-part is used to set up the solution files, consisting of the displacements, stresses and strains in the structure.

3.9 Aero-Structural Coupling

The present section explains how the two disciplinary modules couple together. This is an important aspect of the MDO tool as it is important to maintain the level of accuracy when the iteration goes from one solver to another. When using a coupling method in an MDO framework, it is important that the level of fidelity in the coupling guarantees that the accuracy of the individual disciplines is not affected (Martins, 2002). Also, the discretization in each discipline (the aerodynamic mesh and the finite-element mesh) must preserve the geometric consistency during the analysis process.

3.9.1 Displacement Transfer Between Modules

The objective of the load-displacement transfer process is to accurately translate the nodal displacements of the structural model to aerodynamic mesh point displacements. The flow solution is affected by the position and shape of the solid boundary, which is dictated by the structural displacements. In turn, these displacements are affected by the forces applied to the structure due the flow pressures at that boundary. In the tool established for this thesis, the load and displacements transfer scheme follows the method described by Brown (1997). This method rely on extrapolation functions for the displacements of the internal structure to obtain the aerodynamic mesh displacements. These extrapolation functions must satisfy two conditions. The first one is that these functions must accurately reproduce a rigid body motion. In other words, for a given set of nodal displacements corresponding to a rigid body mode, the extrapolation must yield a rigid body displacement of the aerodynamic mesh. The second condition is that the resulting aerodynamic mesh displacement field must be continuous over the whole surface. To extrapolate the structural displacement field, each point of the aerodynamic mesh, $\mathbf{x}_A$, must be associated to a point on the structural model, $\mathbf{x}_S$. The association is made so that the distance between the two points is minimized. When the association is made, it remains the same either in the initial and perturbed geometries. Figure 3.16 shows a representation of the displacement procedure.

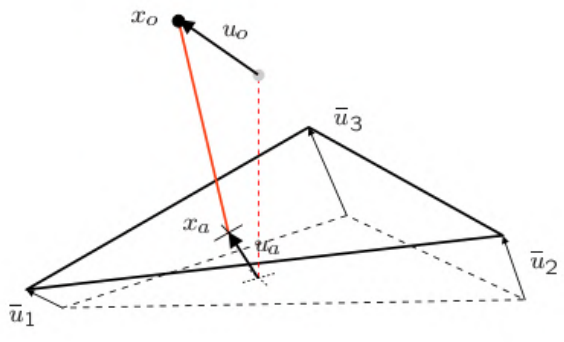


Figure 3.16: Illustration of the displacement extrapolation procedure (Martins, 2002).

The link between the point in the aerodynamic mesh and the point on the structural model is made through the vector, $\mathbf{r} = \mathbf{x}_A - \mathbf{x}_S$, which maintains its position and orientation relative to the associated

finite-element point. The displacement of the aerodynamic mesh point, $\mathbf{u}_A$, can then written as

$$\mathbf{u}_A = \mathbf{u}_S - \mathbf{r} \times \theta_\mathbf{S}, \tag{3.18}$$

where u_S is the displacement of the structural model point, and θ_A and θ_S are equal rotations ($\theta_A = \theta_S$). Note that the equality uses small angle approximation. Once the displacements for each aerodynamic mesh point have been found, the displacement field can be obtained by interpolating between the points using the aerodynamic mesh parametric description stored in the geometry database. In other words, the mapping from the aerodynamic mesh to the finite-element model is performed on explicit point by point basis, for a finite number of points. Displacement field continuity in the aerodynamic mesh is then enforced directly, without requiring continuity from the underlying structural model.

3.9.2 Load Transfer Between Modules

The load transfer procedure is similar to the displacement transfer. The pressures calculated by the aerodynamic flow solver are transferred to the structural nodes through aerodynamic mesh points. To perform the transfer, an appropriate cell and the parametric location of each mesh point within this cell, is identified. The aerodynamic pressures are then calculated by using bilinear interpolation on the surface of the aerodynamic mesh. The accuracy of this transfer is assumed ensured by the fact that the OML surface is of comparable or better fidelity than that of the aerodynamic mesh, and that the two surface representations are consistent. The distributed pressure load, applied to a structural finite-element model, must first be transformed into an equivalent set of nodal forces. This transformation has two requirements. The first is that the resultant nodal forces and moments are the same as those that result from the pressure field for each element. The second is that the load transfer must be conservative. To ensure the former, the virtual work performed by the load vector, $\mathbf{f}$, undergoing a virtual displacement of the structural model, δu, must be equal to the work performed by the distributed pressure field, p, undergoing the equivalent displacement of the aerodynamic mesh, u_A. So the virtual work is the finite-element model is given by

$$\delta W_S = \mathbf{f}\delta u, \tag{3.19}$$

and the virtual work of the aerodynamic pressure forces is given by

$$\delta W_A = \int p\mathbf{n}\mathbf{u}_\mathbf{A} dS, \tag{3.20}$$

where the integral is taken over the entire aerodynamic surface and n represents the unit vector normal to the surface. So conservation requires that $\delta W_S = \delta W_A$ and that the load vector is consistent and conservative. That load vector may be given by

$$f = \int p\mathbf{n}\mathbf{N} dS, \tag{3.21}$$

where N is a transfer matrix between the aerodynamic and structural meshes, calculated in a pre-processing step. Figure 3.17 shows a representation on how a distributed pressure field is integrated to produce a force vector that is translated into the nodal forces of a finite-element using Eq. 3.21.

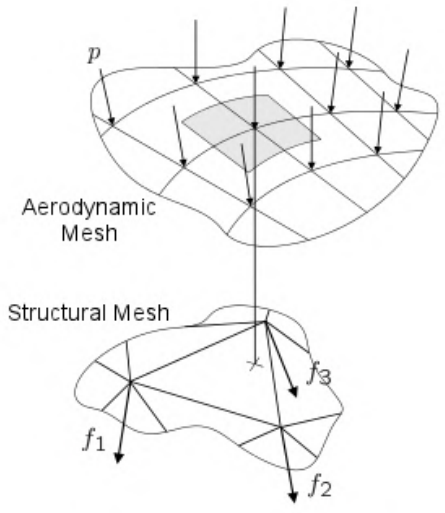


Figure 3.17: Load transfer scheme (Martins, 2002).

3.10 Aero-Structural Solution

In Sections 3.7 and 3.8, the modules for the aerodynamics and structures were presented. In each, the focus was the disciplinary sub-systems. This section focuses on method for solving the coupled aero-structural system. This coupled non-linear system of equations is a combination of the aerodynamic and structural residuals, Eqs. 3.14 and 3.15, respectively, represented by

$$\mathbf{R}(q,x) = \begin{vmatrix} \mathbf{A}(w,u,x) \\ \mathbf{S}(w,u,x) \end{vmatrix} = 0, \tag{3.22}$$

where, x are the design variables and q is the combination of aerodynamic and structural states, $q^T = \begin{bmatrix} w^T u^T \end{bmatrix}$. During the solution procedure, a point is considered converged when the relative tolerance of both residuals is reduced below a specified tolerance, such that

$$\begin{matrix} \left\| \mathbf{A}(w^{(n)},u^{(n)}) \right\|_2 < \epsilon_r \left\| \mathbf{A}(w^{(0)},u^{(0)}) \right\|_2 \\ \left\| \mathbf{S}(w^{(n)},u^{(n)}) \right\|_2 < \epsilon_r \left\| \mathbf{S}(w^{(0)},u^{(0)}) \right\|_2 \end{matrix}. \tag{3.23}$$

This stop criterion is applied to each discipline separately, rather than to the aero-structural system, to avoid situations where the initial residual of one discipline is significantly greater than the initial residual of the other.

3.10.1 Approximate Newton-Krylov Method

To solve the aero-structural system, an approximate Newton-Krylov Method is used. When applied to Eq. 3.22, this method results in the linear system of equations for the update, $\Delta q^{(n)}$, expressed as

$$\frac{\partial \mathbf{R}}{\partial q}\Delta q^{(n)} = -\mathbf{R}(q^{(n)}). \tag{3.24}$$

This method can converge quadratically if the starting point is sufficiently close to the solution and the Jacobian remains non-singular. However, to achieve convergence when the starting points are far from the solution, the Newton method may have to be globalized with some strategy, to ensure progress is made towards the solution until a suitable starting point is found. So, solving Eq. 3.24 inexactly for each update is typically more efficient than finding an accurate solution. This is the methodology used, so a tolerance of $\epsilon_{nk} = 10^{-3}$ was set to the Newton update (Kennedy and Martins, 2010),

$$\left\| \mathbf{R}(q^{(n)}) + \frac{\partial \mathbf{R}}{\partial q}\Delta q^{(n)} \right\|_2 < \epsilon_{nk} \left\| \mathbf{R}](q^{(n)}) \right\|_2, \tag{3.25}$$

with the update, $q^{(n+1)} = q^{(n)} + \Delta q^{(n)}$. To guarantee that Eq. 3.23 is satisfied, the stop criterion used for the Newton–Krylov approach is

$$\left\| \mathbf{R}(q^{(n)}) \right\|_2 < \epsilon_{nk} min \left\| \mathbf{A}(q^{(o)}) \right\|_2, \left\| \mathbf{S}(q^{(o)}) \right\|_2. \tag{3.26}$$

At each iteration, the Newton update is determined using a preconditioned Krylov method. The preconditioner is based upon a single application of block Jacobi. This is applied to each discipline set of equations. So, a preconditioner can be seen as a sub-Krylov method for each discipline.

3.11 Optimizer

This section addresses the optimizer module of the MDO tool. First, documents some background information on optimization methods. A second sub-section presents the sensitivity analysis method used. Finally, a sub-section addresses the optimization algorithm used in the MDO tool.

3.11.1 Optimization Methods

MDO frameworks are based on numerical analysis methods that evaluate the relative merit of a set of feasible designs. The merit of a design is based on the value of an objective function that is computed using numerical simulations, such as *Tripan* and *TACS*. The choice of that function is very important and requires a knowledge of the multi-disciplinary design problem. To efficiently achieve a feasible design point, numerical simulations must be combined with automatic optimization procedures. These are optimization algorithms created to find the design variables that yield the optimum point for a design problem. At the moment, there are two main categories of algorithms. The first includes the *'zeroth order methods'*, such as grid searching, random searches and evolutionary algorithms. These only need

information of the value of the objective function. Grid searching (Nocedal and Wright, 2006), performs systematical surveys to the design space by evaluating each point in a multi-dimensional grid. The number of function evaluations associated with this method increases exponentially with the number of design variables, making it prohibitive in problems that yield more than a few design variables. Random searches (Alexandrov et al., 1997) are a non-systematic way of restrict the design space. These methods do not require as many function evaluations. However, they cannot guarantee that the system optimum will be found. Also, its cost tend to increase if large numbers of design variables are used. Evolutionary algorithms (Floreano and Mattiussi, 2008) are another type of methods that offer simplicity and robustness. These use computational models of evolutionary processes to choose the design parameters. They often achieve multiple optimal solutions but, again, their cost grows with the increase of design variables. So this first category of *'zeroth order methods'* has a general disadvantage, because, as the number of design parameters increases, the number of function evaluations needed to reach the optimum solution rapidly increases to a point where the computational cost is unbearable.

The second category of algorithms for numerical optimization includes the *'gradient-based methods'*. These methods use the value of the objective function and the value of its gradient with respect to the design variables. They use the interpretation of first and sometimes second order sensitivity information to make the steps in the design space toward the optimum point. These methods have the advantage that they will converge to the optimum with a smaller number of function evaluations. Yet, these methods are not guaranteed to succeed as they rely on the fact that the objective function must vary smoothly within the design space. Also, they often converge to a local optimum and not the global optimum. Within the gradient methods, the steepest descent (Snyman, 2005) is the simplest. In it, each optimization step is taken in the direction of the gradient vector. Other methods are Newton and Quasi-Newton (Gill et al., 1982). The first requires second order derivatives (the Hessian matrix) in addition to the first derivatives. This method show a higher rate of convergence. Quasi-Newton, Conjugate Gradient, and variable metric strategies approximate the Hessian during the search. The general characteristic of these methods is that they use the sensitivity information to identify a search direction in the design space. Then, a one-dimensional search in that direction is performed, until a new search direction is found.

Both zeroth and gradient categories of optimization algorithms have a role in solving optimization problems. In a problem with a low number of design variables and multiple local minima or discontinuities, the zeroth order methods are more suitable. On the other hand, in optimizations like those in aircraft MDO, that feature a large number of design variables and a smooth design space, the benefit goes for the gradient-based methods. In particular, gradient methods are regular practice for aerodynamic shape optimization problems. This because, they are often parameterized with hundreds of design variables and require computationally expensive high-fidelity analyses. Therefore, a gradient-based strategy is also employed in the established MDO tool, virtually enabling the use of hundreds of design parameters and providing a level of detail that cannot be treated by non-gradient methods.

3.11.2 Aero-Structural Sensitivity Analysis

Efficient gradient-based optimization requires the accurate and efficient computation of the objective and constraint gradients. In a typical aero-structural optimization problem, there are fewer objective and constraint functions than there are design variables. For that reason, an adjoint implementation of the sensitivity equations is appropriate. Following Martins (2002), an aero-structural adjoint method that is based entirely on analytical derivatives was employed. This method was proven to be highly accurate at low computational cost when compared with finite-difference or complex-step calculations. The implicit aero-structural adjoint equations are

$$\frac{\partial \mathbf{R}^T}{\partial q}\psi = \frac{\partial \mathbf{f}}{\partial q}, \tag{3.27}$$

where ψ refers to the adjoint vector and f is either an aerodynamic or structural function of interest. The total derivative is then determined using

$$\frac{d\mathbf{f}}{dx} = \frac{\partial \mathbf{f}}{\partial x} - \psi^T \frac{\partial \mathbf{R}}{\partial x}. \tag{3.28}$$

This method is implemented for all the objective functions and constraints considered in the MDO problem, namely,the aerodynamic lift, drag and mass as well as the KS functions. To compute the adjoint, it is necessary to solve the adjoint Eq. 3.27 using a Krylov method. This method solves the adjoint equation using a fully-coupled approach. Matrix-vector products use the exact Jacobian-transpose of the coupled aero-structural system. One iteration of a transpose block Jacobi is used as the preconditioner, with similar settings to those used in the Newton–Krylov solution method. Once the adjoint vector ψ has been determined, the total sensitivities must be computed using Eq. 3.28. This requires the partial derivative of the residuals with respect to the design variables.

3.11.3 Optimization Algorithm

The optimization algorithm used to implement the gradient-based optimization in the established MDO tool is called *SNOPT* (Gill, 2008). This module was created in Fortran but has been compiled with a Python interface, named *pySNOPT*, for an easy integration in MDO frameworks. *pySNOPT* implements a SQP algorithm used with user provided gradients, from the already stated adjoint method. This optimizer has also the capability of automatically computing sensitivities, using the finite-differences method or the complex-step method. Ultimately, this module is able to solve non-linear problems with a high number of variables and can even deal with discontinuities in the design space unless they are near the optimum.

3.12 Summary

This chapter presented the subject of MDA and MDO. Starting from the multi-disciplinary problem definition and its approach strategies and finishing with the methods and modules implemented, in the MDO tool established in this thesis.

Chapter 4

Results

4.1 Introduction

The aim of this thesis is to run an MDO on sailplane wings, considering both the aerodynamics and structures disciplines. To do that, an MDO tool was established. Then, disciplinary analysis were run to evaluate the disciplinary modules. Finally, an MDA and MDO were performed. Thus, the exercises run were: geometry parametrization; aerodynamic and structural analysis; aerodynamic and aero-structural optimizations. The following sections document and discuss, the results obtained for each exercise.

4.2 Case Studies

4.2.1 Geometry of Sailplane Wings

This sub-section is intended to present the geometric parameters of the two case studies chosen: a standard class sailplane wing and the L-23 Super Blanik sailplane wing.

Standard Class Sailplane Wing Design requirements are needed to create an initial geometry for the optimization process. Considering a standard class sailplane, the main design requirements are:

- Maximum wing-span of the sailplanes of 15 m;
- Fixed wing sections;
- Maximum take-off gross weight of 525 kg.

To complete the design parameters, a semi-tapered wing geometry was chosen. The geometric parameters used are listed in Table 4.1. The semi-tapered wing geometry was chosen because the use of tapper ratio in sailplane wing design has become standard practice. One major reason for that is because taper ratio is a proven way to reduce the induced drag, which is very important in sailplane performance. It was decided not to include sweep, dihedral or twist angles. Sweep angle is not relevant since as its objective is to reduce the incident Mach number, but, in this case, it is already in the

incompressible range. Dihedral was also not chosen as its objective is to ensure roll stability and not to directly affect the cruise performance. Twist angle was not used because it is a design parameter that it will be studied in the optimization. For the chosen wing geometry, a well studied academic airfoil, the NACA 2412, was used for all wing sections. At the structural level, the internal layout chosen was a configuration with two spars and twelve ribs. The thickness values were set to 5 mm in the skin, 10 mm in the spars and 8 mm in the ribs and the maximum take-off weight to 430 kg. This initial settings were merely academic and do not mimic any specific sailplane wing.

L-23 Super Blanik Wing The LET L-23 Super Blanik has an all-metal, cantilever, mono-spar, tapered wing that consists of two assemblies. The basic dimensions of the entire sailplane are presented in Fig. 4.1. The main geometry parameters of the wing are summarized in the Table 4.1.

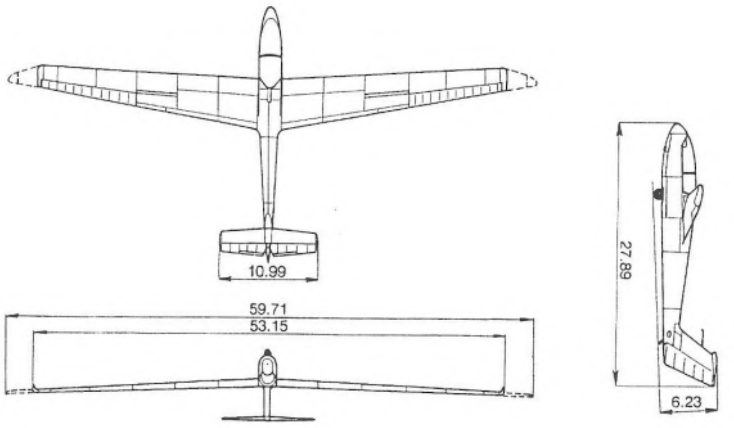


Figure 4.1: Basic dimensions for the L-23 sailplane (L-23 sailplane flight manual, 2011).

Table 4.1: Initial geometry parameters for the case studies.

Design Parameter	L-23 Wing	Semi-tapered Wing	
Span	16.2	15	m
Aspect Ratio	13.7	22	
Reference Area	19.15	10.22	m^2
Taper Ratio	0.429	0.4	
Dihedral Angle	3	0	$^\circ$
Sweep Angle	-5	0	$^\circ$
Twist Angle	-3	0	$^\circ$

The real structural layout of the L-23 wing, presented in the technical drawing of Fig. 3.8, was used as reference for its structural modeling. Therefore, the wing internal layout is modeled with seventeen ribs, one main spar and an auxiliary spar. Although it is not the exact modeling of the real layout, it was the best approximation that was possible to recreate using the geometry module. Also a maximum take-off weight of 530kg was used (L-23 sailplane maintenance manual, 2011).

4.2.2 Modeled Geometries

This sub-section presents the geometry objects created using the geometry module of the MDO tool. The script used to created the OML, aerodynamic and structural meshes is referred in Sub-section3.6.3. Figure 4.3 shows the geometry objects created. For each case study, an OML and an aerodynamic/structural mesh are presented. As observable, the aerodynamic meshes are almost perfectly coincident with the OML of the desired wing geometries. Also, the structural models of the wing boxes fit perfectly in the OMLs. So, the use of *pySpline* in combination with *pyGeo* has proven to provide quality aerodynamic geometry objects and the use of *pyLayout* also assured well suited internal structural layouts. These results show that the CAD-free approach is a valid method for the generation of geometry objects for high-fidelity MDO.

4.3 Aerodynamics

The results obtained with the aerodynamics module of the MDO tool are presented in this section. First, to attest these results, a verification and validation study of the *Tripan* code is made. Then, the aerodynamic analysis and optimization results for the case studies are presented.

4.3.1 Verification and Validation of the Aerodynamic Analysis

Validation of *SUmb* Flow Solver To verify the fidelity of the aerodynamics simulation code, it is important to do a validation. As it was presented in Chapter 3, the code used for the aerodynamics analysis is *Tripan*. Doing an experimentation on a real or model wing for the validation of this code was a nearly impossible task due to the time and resources required for the construction, instrumentation and run of tests. Thus, the methodology chosen was to perform a verification of *Tripan* with another code that could be validated using experimental data available. The work-flow for this task is presented in Fig. 4.2.

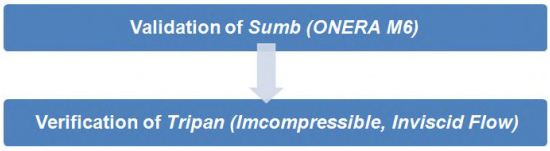
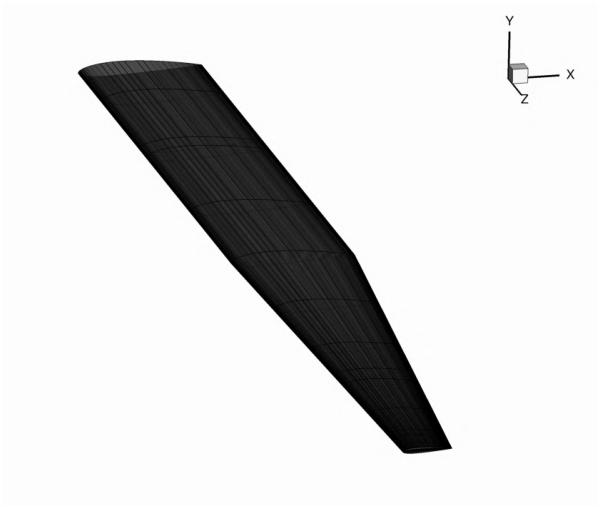


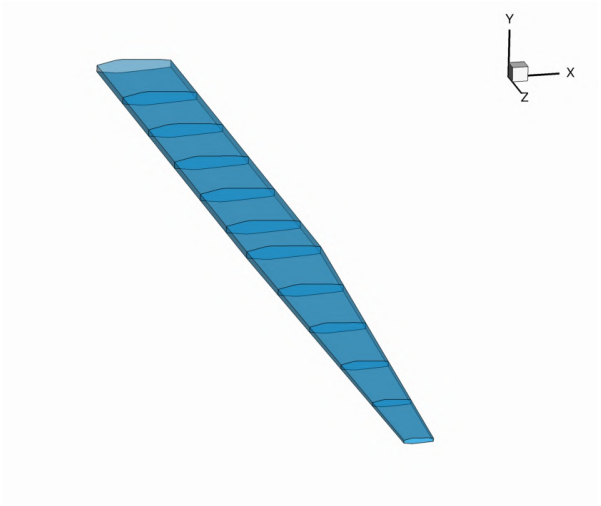
Figure 4.2: Task scheme for the verification of *Tripan* code.

The first task was to validate *SUmb* so that it may be used for the *Tripan* verification. To fulfill this, a classical study of three dimensional turbulent transonic flows was chosen, the rebuilding of the ONERA M6 wing wind tunnel experiments (Schmitt and Charpin, 1979). In this, the flow over the ONERA M6 wing is studied by testing it in a wind tunnel at transonic Mach numbers (0.7, 0.84, 0.88, 0.92) and various angles-of-attack, up to 6 degrees. The Reynolds numbers were about 12 million based on the mean aerodynamic chord. Records were taken of the upper and lower pressures for seven wing sections along the span, as shown in Fig. 4.4. To rebuild the wind tunnel tests, the flow was numerically build using *SUmb*. The flow conditions for the simulation were set to match the experiment values of Mach

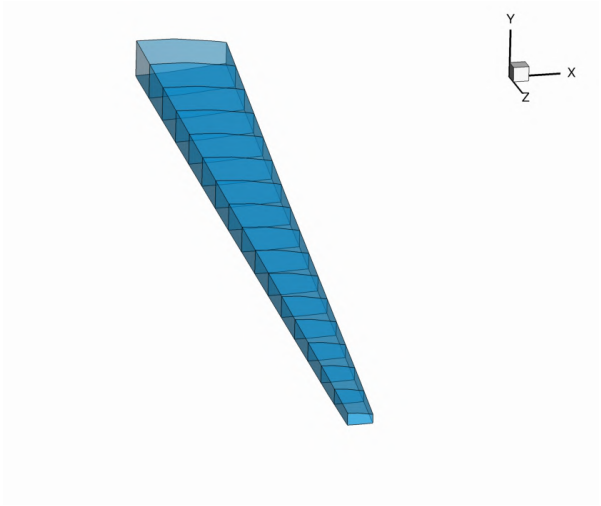


(a) Aerodynamic mesh for the semi-tapered wing geometry.

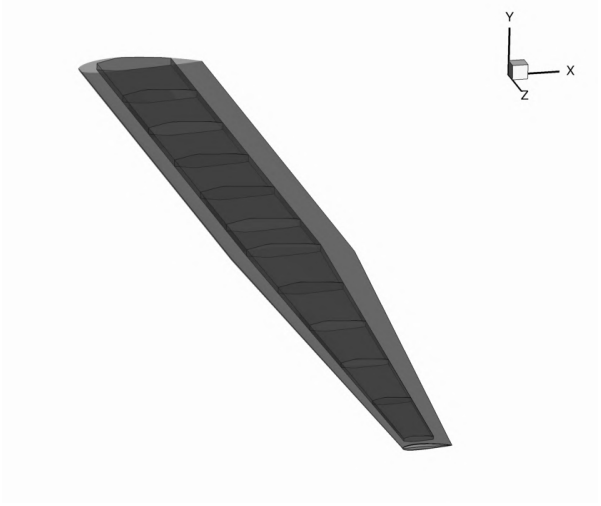
(b) Aerodynamic mesh for the L-23 wing geometry.



(c) Structural mesh of the semi-tapered wing geometry.



(d) Structural mesh of the L-23 wing geometry.



(e) Aero-structural combination for the semi-tapered wing geometry.

(f) Aero-structural combination for the L-23 wing geometry.

Figure 4.3: Objects generated with the geometry module.

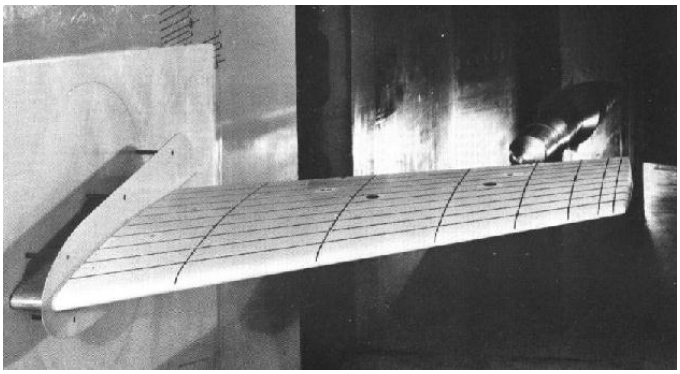


Figure 4.4: ONERA M6 wing in the wind tunnel (Schmitt and Charpin, 1979).

number, Reynolds number and angle-of-attack. Table 4.2 summarizes the experiment conditions and the corresponding simulation conditions.

Table 4.2: Free-stream conditions.

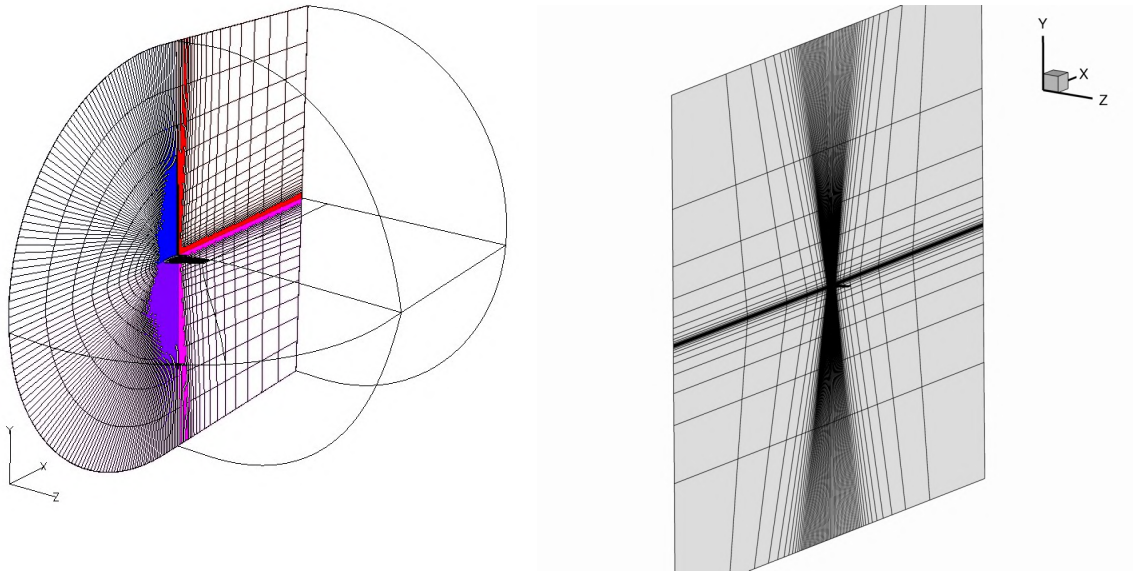
Design Parameter	Experimental	Simulation	
Mach	0.8395	0.8395	
Reynolds	11.72E+06	-	
Angle of Attack	3.06	3.06	$^{\circ}$
Pressure	-	315980	Pa
Temperature	-	255.6	K
Density	-	1.367	Kg/m^3

The ONERA M6 wing is a swept, semi-span wing with no twist. It uses a symmetric airfoil. The semi-section of the airfoil is the ONERA D section (Schmitt and Charpin, 1979). The properties of the geometric layout of the wing are summarized in Table 4.3.

Table 4.3: ONERA M6 wing layout data (Schmitt and Charpin, 1979).

Design Parameter		
Span	1.1963	m
Mean Aerodynamic Chord	0.64607	m
Aspect ratio	3.8	
Taper Ratio	0.562	
Leading-edge Sweep	30	$^{\circ}$
Trailing-edge Sweep	15.8	$^{\circ}$

To improve this study, a cross comparison of the obtained results was done against the *WIND* code results from NASA (Slater, 2008). Figure 4.5 presents the views of the computational domains used in each code. Also a view of the *SUmb* mesh is presented in the Fig. 4.6. Through the view of wall pressures in Fig. 4.7 or the Mach number contour in Fig. 4.7, it is possible to identify where is the localization of the forming transonic shock. A comparison of the *SUmb* results with the experimental data and with the numerical *WIND* data is presented in the six sectional Cp graphics of Fig. 4.8. On note for the fact that the Cp is negative for pressures less than free-stream, which occurs on the top of the wing. Thus, Cp values are plotted to indicate that the lower pressure region on the top of the wing (top



(a) Surfaces, zones and grids for *WIND* (Slater, 2008).

(b) Surfaces, zones and grids for *SUmb*.

Figure 4.5: Comparison between surfaces, zones and grids for *SUmb* validation.

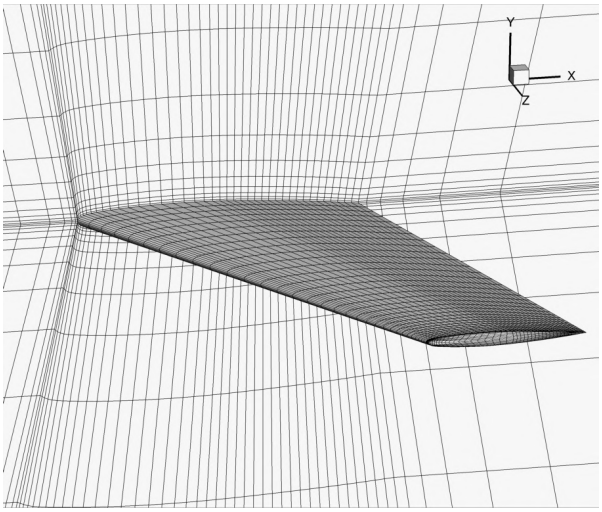
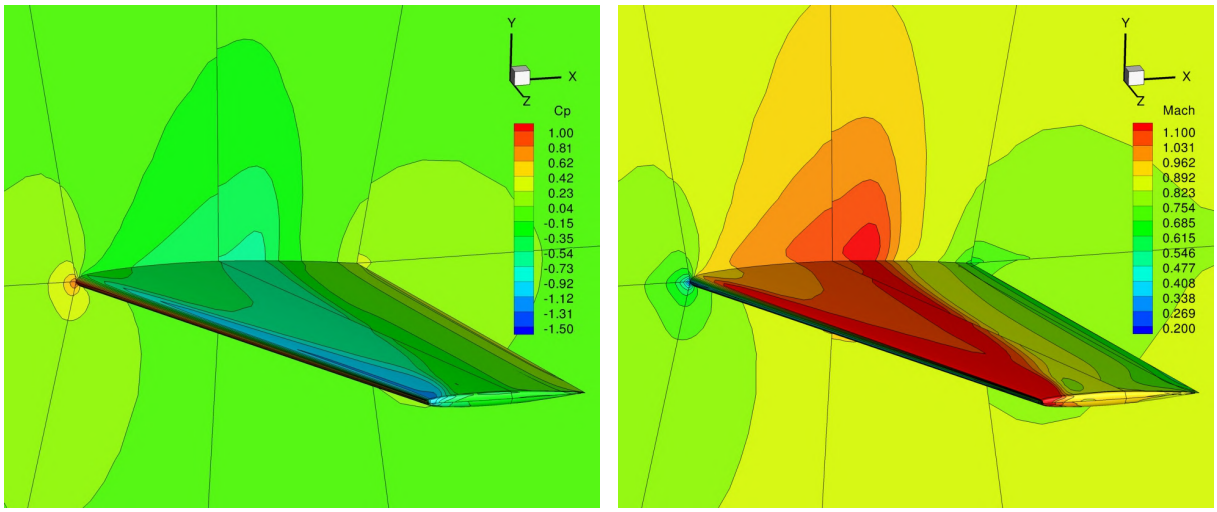


Figure 4.6: ONERA M6 mesh for *SUmb* validation.

line of graphics) and the high pressure region is on the bottom of the wing (bottom line of graphics).

From the results, it is clear that the two numerical solvers are very close to one another. In some cases, like sections (d) and (e), it is even possible to see localizations where *SUmb* tends to be slightly closer to the Cp values observed in the experimental data. Overall, its clear that the agreement between numerical solvers is good. Compared to the experimental data, the flow structure computed by the two numerical solvers seems to be not sharp enough in the shock resolution. One reason for that may be that the aerodynamic meshes bring some level of dissipation. Also a note for the wing tip section where at the trailing edge, the results were less accurate, probably due to the increased effect of vorticity. However, for the validation purpose it is meant for, i.e., to validate an incompressible, inviscid flow solver (*Tripan*), the good agreement between *SUmb*, *WIND* and experimental data was considered enough.



(a) Flow pressure coefficient zones on the symmetry plane and wing surfaces.

(b) Flow Mach zones on the symmetry plane and wing surfaces

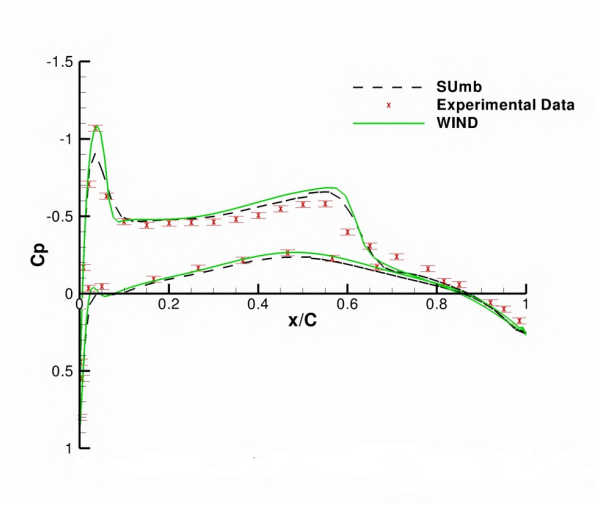
Figure 4.7: Cp and Mach zones obtained with *SUmb*.

Verification of *Tripan* Flow Solver The next step was the verification of *Tripan* using *SUmb*. As mentioned in Sub-section 3.7.2, this solver is for incompressible, inviscid external flows. So to verify this code, a flow that met these requirements was used. Its properties, based on the 1976 standard atmosphere up to 230,000 ft, are summarized in Table 4.4.

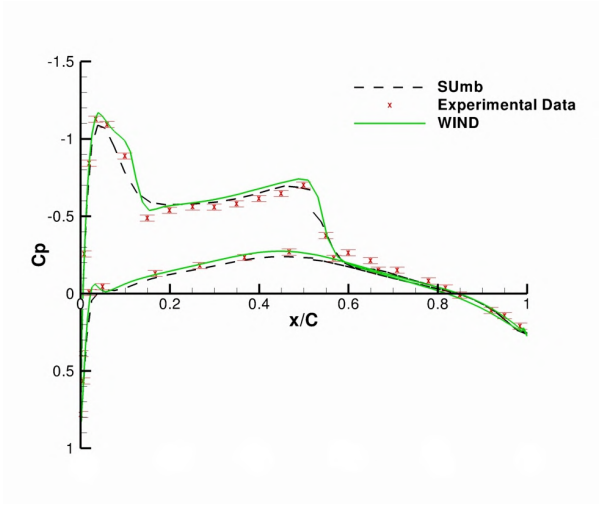
Table 4.4: Free-stream conditions for *Tripan* verification.

Design Parameter		
Mach	0.2	
Angle of Attack	0	$^\circ$
Pressure	315980	Pa
Temperature	255.6	K
Density	1.367	Kg/m^3
Speed of Sound	322	m/s

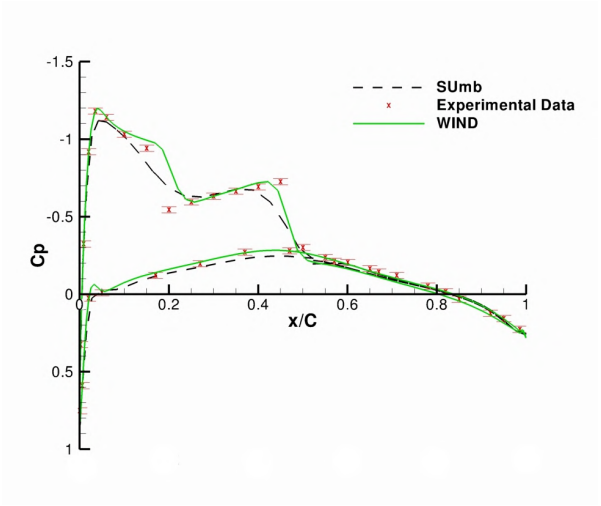
The case study used to perform the comparison analysis was the ONERA M6 wing. The grid used for *SUmb* was the same as in its validation, however for *Tripan*, a highly refined mesh was created with the geometry module. These meshes are presented in Fig. 4.9. The script used to run the simulation with *Tripan* was similar to the presented in Section 3.7. To illustrate the flow around the wing, the Cp distribution over the wing surface, for the two codes, is shown in Fig. 4.10. Figure 4.11 presents Cp distribution for three sections whose locations are at 20%, 50% and 85% of the wing span. Results show that the two numerical flow solvers are very close to one another. Only in the trailing-edge, a slightly difference is noted. In sum, the good agreement between solvers verify that *Tripan* provides accurate results when simulating incompressible inviscid flows.



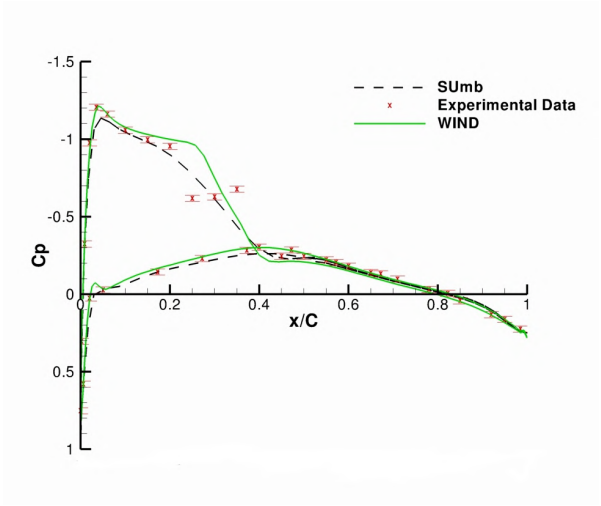
(a) Cp values for the section at z/span = 0.2



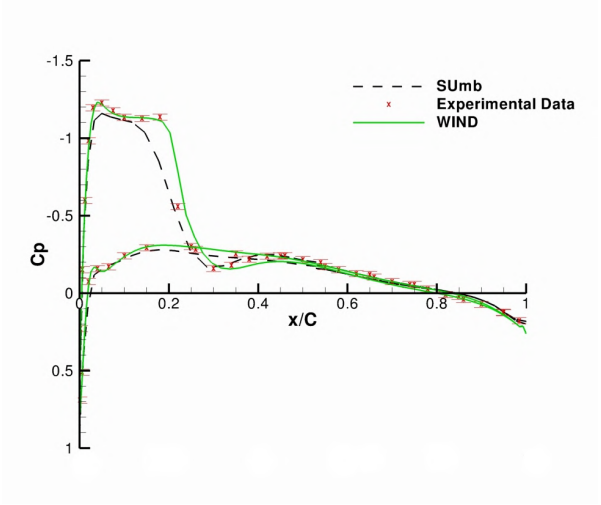
(b) Cp values for the section at z/span = 0.44



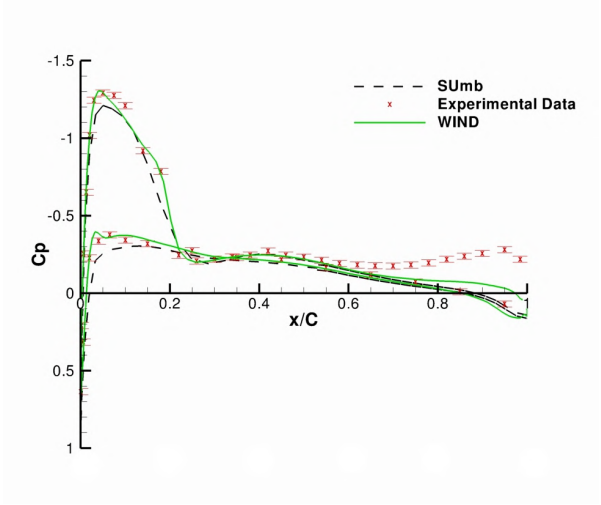
(c) Cp values for the section at z/span = 0.65



(d) Cp values for the section at z/span = 0.8

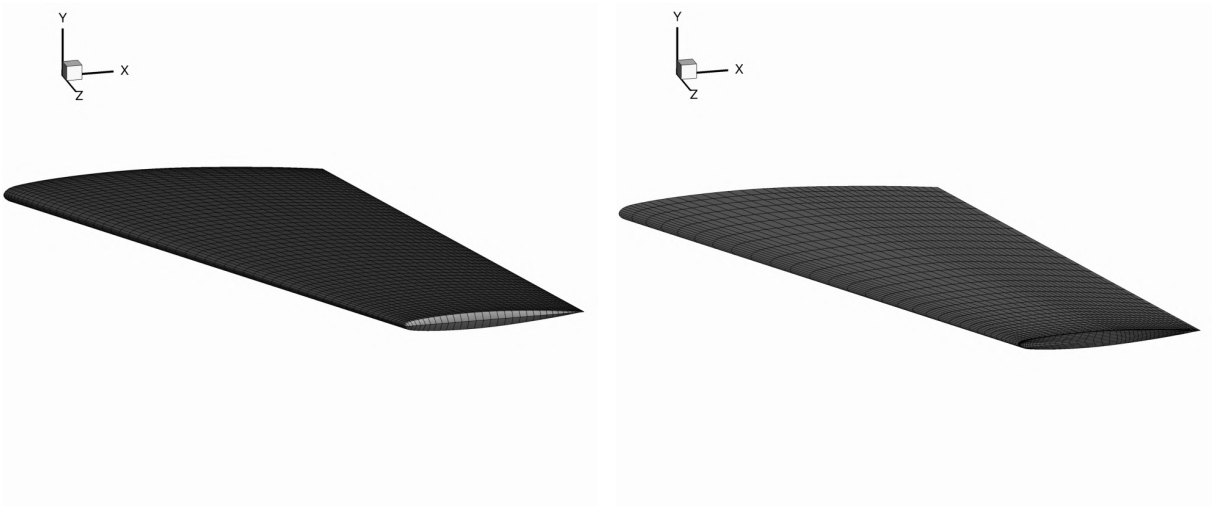


(e) Cp values for the section at z/span = 0.95



(f) Cp values for the section at z/span = 0.99

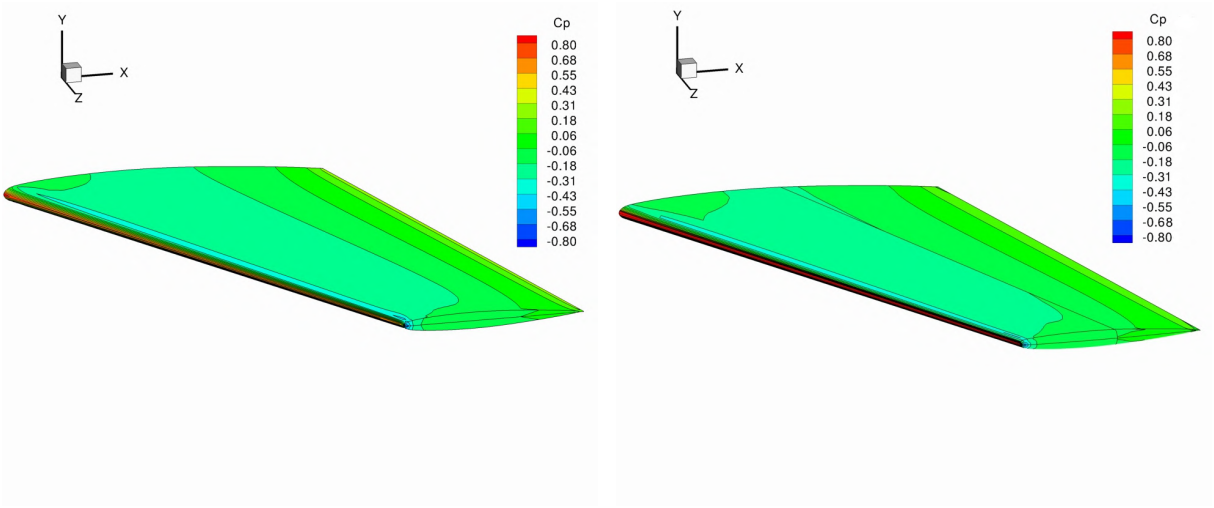
Figure 4.8: Comparison between the results for Cp with *SUmb*, *WIND* and experimental data. (The accuracy of the Cp measurements for the experimental data was determined to be +/- 0.02.)



(a) Surfaces grids for *Tripan*.

(b) Surfaces grids for *SUmb*.

Figure 4.9: Comparison between the surfaces grids for *Tripan* and *SUmb*.



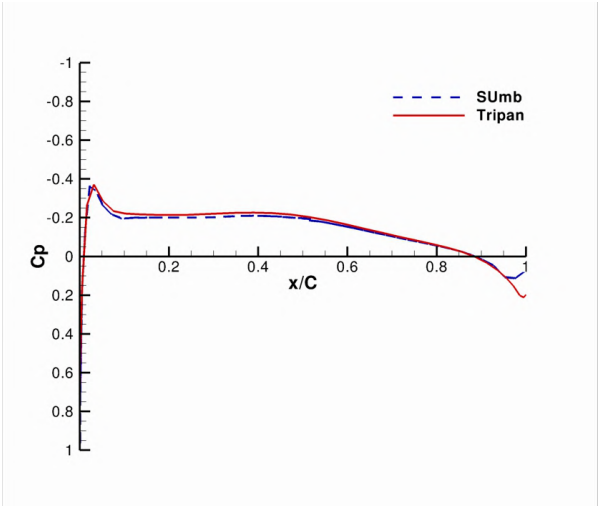
(a) Results for Cp with *Tripan*.

(b) Results for Cp with *SUmb*.

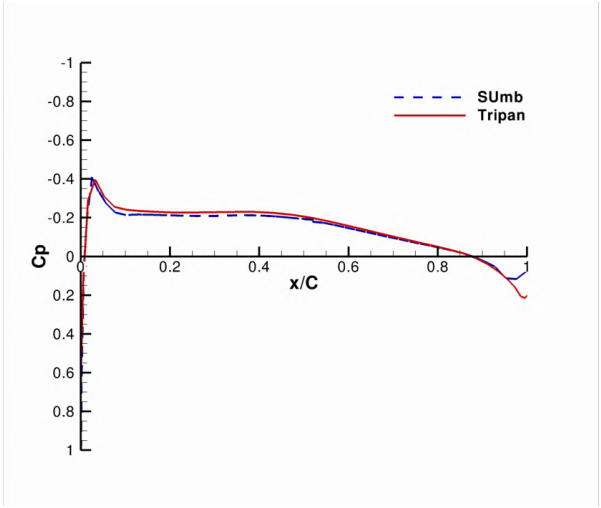
Figure 4.10: Comparison between the surface results for Cp with *SUmb* and *Tripan*.

4.3.2 Aerodynamic Analysis

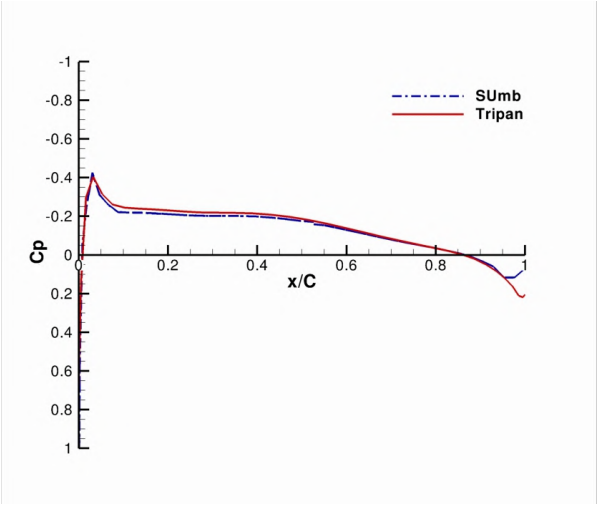
Once verified the code used in the aerodynamics module, aerodynamic analysis on the case studies were run. This section presents and discuss the results from that analysis. The geometry objects used in this analysis were those presented in Sub-section 4.2.1. As an academic exercise, there are no imposing requirements for the simulations performed. However, as the case studies are sailplane wings, the conditions chosen were those from a cross-country soaring flight at 1000 m, with a velocity of 25m/s. Table 4.5 summarizes the free-stream conditions defined in the simulation, based on the 1976 standard atmosphere up to 230,000 ft.



(a) Comparison between the results for Cp for a section at 20% of wing span.



(b) Comparison between the results for Cp for a section at 50% of wing span.



(c) Comparison between the results for Cp for a section at 85% of wing span.

Figure 4.11: Comparison between the sectional results for Cp with *SUmb* and *Tripan*.

Table 4.5: Free-stream conditions for the aerodynamic analysis.

Design Parameter		
Mach	0.074	
Angle of Attack	3	$^{\circ}$
Density	1.112	Kg/m^3
Speed of Sound	336	m/s

Convergence Study Sub-section 4.2.2 presented the geometry objects created. However, the exact number of panels used for the discretization the aerodynamic mesh was not presented. The reason was because a convergence study was performed to determine the proper number of panels. This section presents the results of the convergence study for the aerodynamic mesh.

The free-stream conditions are kept constant and equal to those presented in Table 4.5. The case study geometry chosen was the simplest of the two cases, the semi-tapered wing. Using this case study as reference, a range of *Tripan* objects was created from a simple mesh with 150 panels, to a highly refined mesh with 12,150 panels. Then, an aerodynamic analysis was performed on each one of these meshes. To assess the results obtained, a relative error evaluation was performed. The absolute error is a measure of the deviation from the value calculated for a real value. It can be calculated from the computed value, x and the exact value, X, using

$$\Delta = x - X. \tag{4.1}$$

However, in some cases, the difference between the value measured and the real value may be imperceptible. So another usual method of measuring the accuracy is to compute the relative error. This error allows a better assessment of the accuracy and is calculated from the absolute value of the absolute error divided by the exact value. Mathematically, it can be represented as

$$\delta = \frac{|\Delta|}{X}, \tag{4.2}$$

where the $|\Delta|$is the absolute error of x and X is the exact value. In this study, the real value of the aerodynamic quantities is not known, so the value computed for the most refined mesh is used as the best approximation to the real value. Thus, from the various aerodynamic analysis performed, a graphic with the convergence of the results was compiled and presented in Fig. 4.12. As *Tripan* uses a

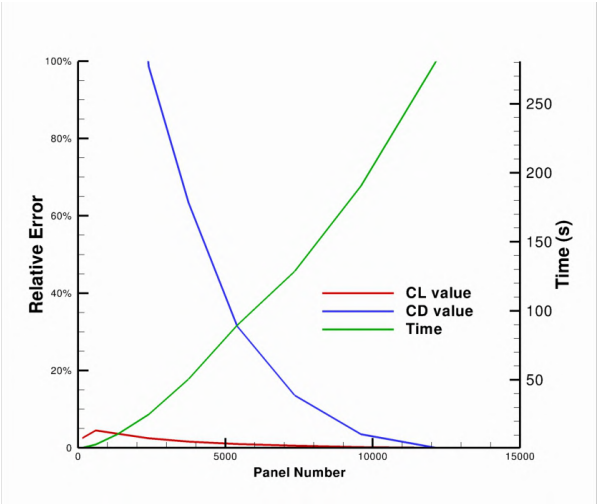
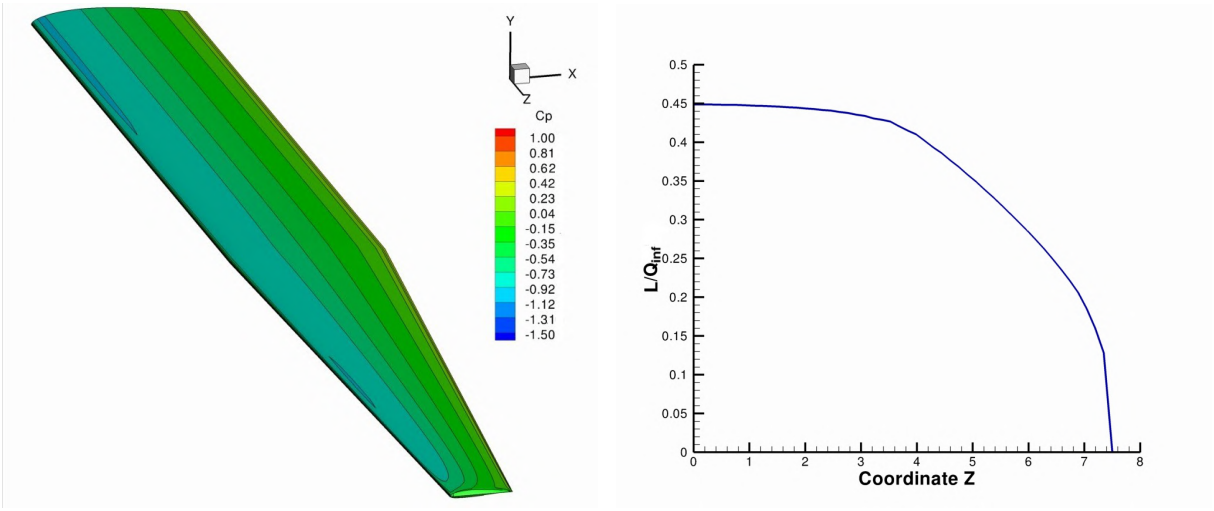


Figure 4.12: Results for the convergence study on the mesh resolution with *Tripan*.

panel code, the error for the lift coefficient converges much faster that the drag coefficient. Although the computed drag value is not accurate, as the code can not compute the total drag, it was important to

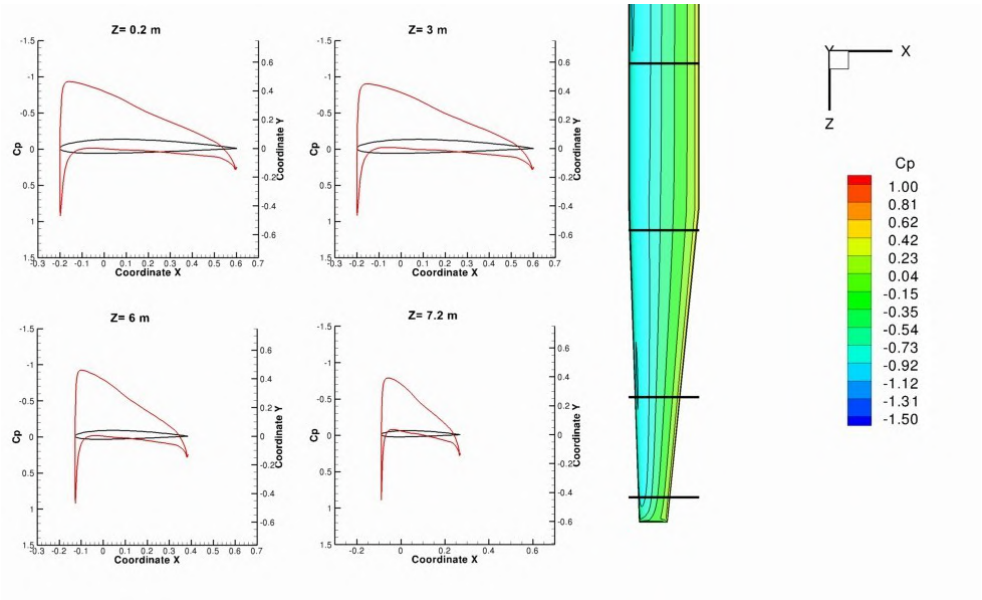
assess its convergence. Also, the time required to perform the aerodynamic analysis was measured. It is observable that time grows exponentially with the number of panels used. From the results seen, a panel number near 7,000, was considered to give the best relation between accuracy error (approximately 10%) and time to perform the analysis (approximately two minutes).

Aerodynamic Analysis of the Case Studies With every parameter defined, the aerodynamics module of the MDO tool was used to perform the aerodynamic analysis on the case studies. Figures 4.13 and 4.14 summarize the results obtained. For each case study, Cp distribution over the wing, Cp over four sections along the wing semi-span and lift distribution over the wing are presented.



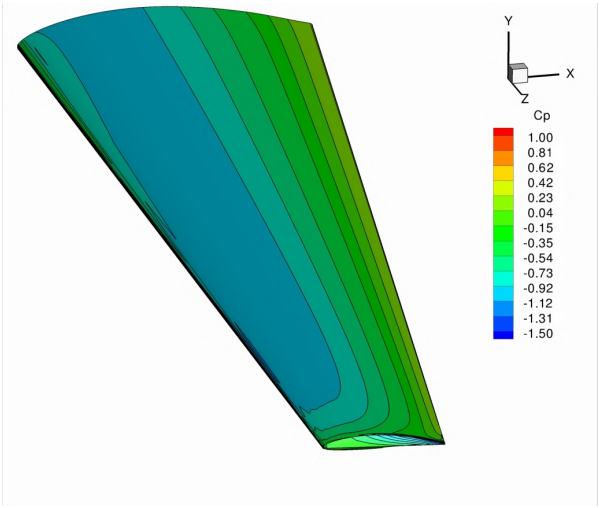
(a) Results for Cp distribution over the wing.

(b) Results for lift distribution over the wing.

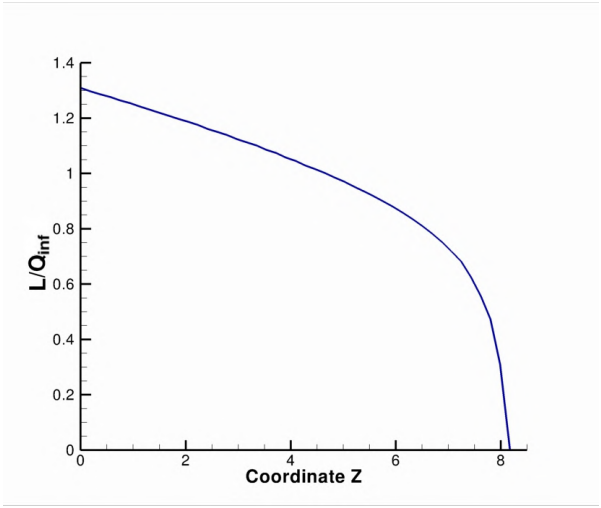


(c) Results for Cp distribution over four sections.

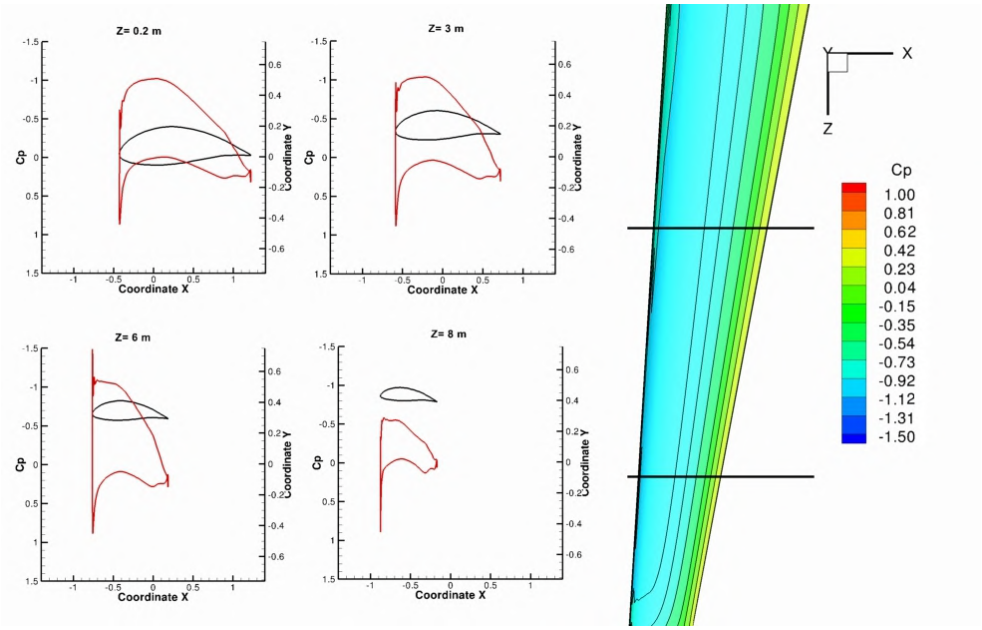
Figure 4.13: Aerodynamic analysis results for the semi-tapered sailplane wing.



(a) Results for Cp distribution over the wing.



(b) Results for lift distribution over the wing.



(c) Results for Cp distribution over four sections.

Figure 4.14: Aerodynamic analysis results for the L-23 sailplane wing.

As expected, although the free-stream conditions are the same, the results change with each case study. From the observation of the normalized lift distribution over the semi-span of the wing, one can see that the semi-tapered wing shows the nearest to an optimal distribution, approaching the elliptical distribution. On the other hand, the L-23 wing shows a near constant slope until 75% of the span. Then it rapidly increases towards the tip. These results are interesting as they show that the semi-tapered wing has a lift distribution closest to the optimum than the L-23. That could be expected since the first configures an academic case study. Also the taper ratio used was 0.4, which, from theory, is the optimal ratio for incompressible velocity ranges. However, the lift distribution of the L-23 wing was not as expected. As a real wing with proven performance, it was expected a more elliptical distribution. Yet, if one accounts for the fact that, although tapered, the L-23 wing also has a constant twist and sweep

angles, then the results seem more comprehensive. Also, the fact that this is a real wing geometry, that was already optimized by the manufacturer, taking into account more than aerodynamics, can justify these results. The Cp distribution over the wing shows clearly that some other factors were taken into account. The L-23 has more inboard lift, which results in a weaker bending moment at the wing root. This fact allowed the use of lighter and less strong structural components, which consequently reduced the total weight of the wing structure. As for the Cp distributions over wing sections, results also show some interesting differences. The semi-tapered wing presents high pressure gradients, between the upper and lower surfaces of the wing, after the leading edge zone. These gradually progress towards a negative value at the trailing edge. On the contrary, the Cp distributions over the L-23 wing sections show higher gradients through the major portion of the section. The increase and decrease of pressure on the upper and lower surfaces are also smoother. These differences come from the airfoils used for each case. In the semi-tapered, a NACA 2412 airfoil is used for all the wing and in the L-23 the airfoil morphs from a NACA 63_{2A}-615 at root to NACA 63_{2A}-612 at tip (both laminar airfoils). The differences in the values of the gradients are mostly due to the difference in the thickness of the airfoils. Also, the differences in the evolution of the Cp values at the upper and lower surfaces come from the differences slopes of the surface geometries. It is clear that the airfoils in the L-23 wing have been chosen to provide better results in gliding performance. In sum, for the same flow conditions, the L-23 wing shows higher Cp gradients over the wing than the semi-tapered, therefore generating higher lift values. From an MDO point of view, using a L-23 wing like design can make a better initial design point, as it already provides better results.

A note about the simulation process is that for the L-23 wing, near the leading-edge of some sections, there are some observable deviations that, though subtle, can denounce some numerical errors. This will be remarked for future works. Also, it is notable that the most time consuming analysis only took 4 minutes to perform.

In summary, the aerodynamic module provided good results performing aerodynamic analysis over the case studies, denoting the differences between the choices of the initial design properties for a sailplane wing design. These are important as they can reduce the time and effort in the initial preliminary stages of design.

4.3.3 Aerodynamic Optimization

One of the main objectives in sailplane performance is the maximization of the L/D ratio, to maximize the range of the flight. Naturally, the next step in the exercises run with the established MDO tool was an aerodynamic optimization of the case studies. The initial flight condition for the optimization is the same as that of the aerodynamic analysis, presented in Table 4.5. The objective function was the L/D ratio. As an academic exercise, there were no imposing constraints. However, some requirements were created to test the optimization process. So, in the two case studies, a lift constraint was applied. That constraint was set through C_L to equilibrate the weight of the sailplane ($W_{Sailplane}$). With the lift constrained, the range optimization problem is turned into a drag minimization problem. So, using the initial conditions

as stated and imposing a minimum lift as constrain, the optimizer had to adjust the geometry variables so that the required C_L is achieved with the minimum drag possible. The geometric design variables chosen are four twist angles ($\theta(z)$) and four chord scale factors ($c(z)$). The span is fixed to maximum of 15 m in the semi-tapered wing, as required by the standard sailplane class, and to 16.2 m in the L-23 wing. So the changes in section chord will reflect in the wing areas and, therefore, in the aspect ratio and C_L value. Mathematically, the optimization problem is represented as

$$\begin{aligned} \text{Minimize :} \quad & C_D, \\ \text{s.t.:} \quad & L = W_{Sailplane}, \\ \text{w.r.t.:} \quad & \alpha, \theta(z/b), c(z/b), \quad z/b = 0.3, 0.6, 0.9, 1. \end{aligned} \tag{4.3}$$

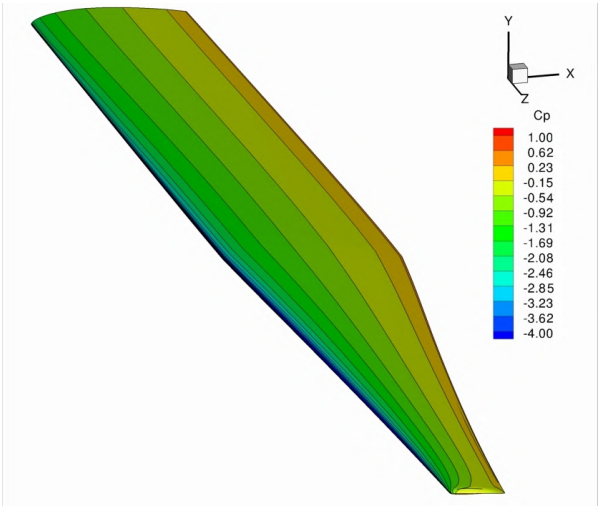
• **Case Study : Semi-Tapered Wing** The semi-tapered standard class sailplane wing configures a simple wing geometry. For this reason, its optimization is a study of much interest. If reaching an optimized wing design is the goal in the preliminary stages of the aircraft design, then this case study configures the good solid initial geometry. The results should confirm the theories and the reasons for the implementation of more complex geometries like tapered or swept wings. A summary of the initial and final parameters of the aerodynamic optimization for the present case study is shown in Table 4.6.

Table 4.6: Aerodynamic optimization parameters for the semi-tapered wing.

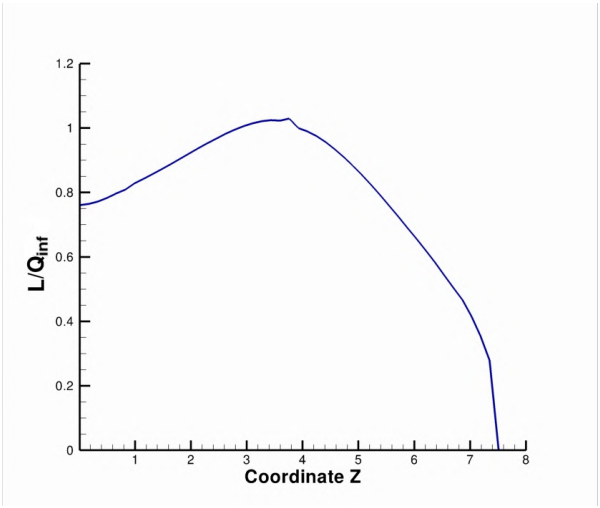
Constraint	Initial Value	Optimized Value	Lower Bound	Upper Bound	
C_L	0.54	1.188	1.188	1.188	
Parameter	Initial Value	Optimized Value	Lower Bound	Upper Bound	
Angle of Attack	3	5	-4	7	°
Twist (z/b=30%)	0	5	-10	10	°
Twist (z/b=60%)	0	5	-10	10	°
Twist (z/b=90%)	0	5	-10	10	°
Twist Tip	0	-1.45	-10	10	°
Chord Scale (z/b=30%)	1	1	0.5	2	
Chord Scale (z/b=60%)	1	1.19	0.5	2	
Chord Scale (z/b=90%)	1	0.99	0.5	2	
Chord Tip	1	0.84	0.5	2	
C_D	0.00157	0.00763			
Time	0	6040			$seconds$

The results verify that, although the initial C_L was much lower than the requested value, the constraint was fulfilled. As consequence of the lower initial C_L, the angle of attack had to be increased which result in the initial increase of C_D verified in Fig. 4.17(a). The values of twist and scale changed as well. The twist group of variables show that once reached a sufficient angle of attack, the optimizer chose to increase the twist angle of the sections. On the contrary, the closest section to the wing tip was changed to have negative twist. By increasing twist, the local angle of attack was also increased, allowing the wing to generate more lift. As for the scale group of variables, it shows a decrease as the sections approach the wing tip. That corresponds to inserting even more tapper to the already initially semi-tapered geometry. From an aerodynamic perspective, that was expected since introduction of taper

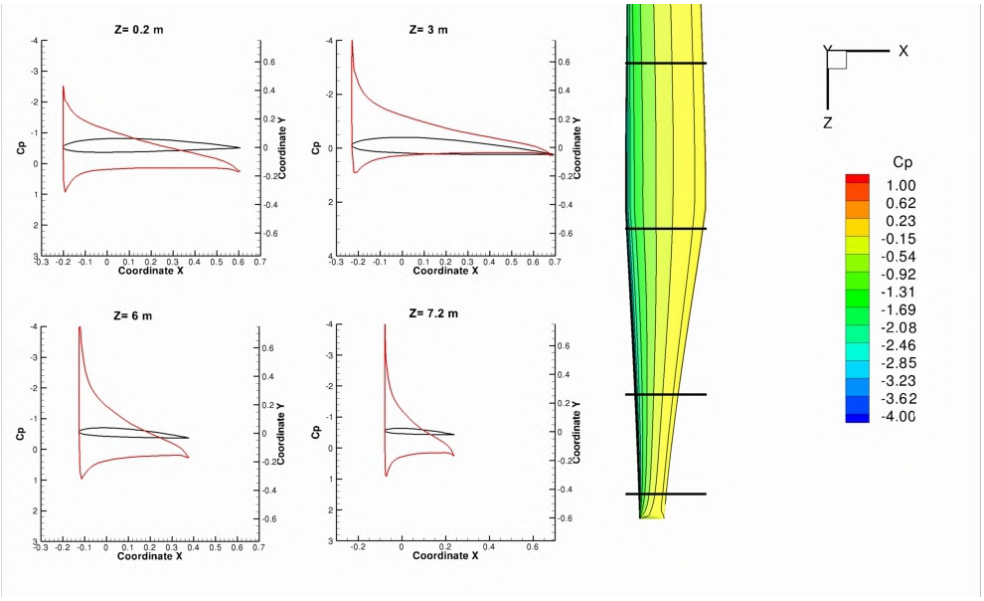
ratio leads to a lift distribution closest to the elliptical, therefore reducing the wing drag. Also as the span is fixed, reducing the wing chord, reduces the overall wing area, which increases the aspect ratio of the wing. This set of results shows that the optimizer tried to reduce drag as much as possible. Yet, the final C_D is much higher than the initial C_D. This shows a trade-off that had to be made by the optimizer in order to fulfill the C_L constraint. For a better visualization of the aerodynamic results, Fig. 4.15 shows the lift and Cp distribution over the wing surface and some sectional data of the optimized wing.



(a) Results for Cp distribution over the wing.



(b) Results for lift distribution over the wing.



(c) Results for Cp distribution over four sections.

Figure 4.15: Aerodynamic optimization results for the semi-tapered sailplane wing.

Comparing the optimization results to the analysis results, it is notable the difference in the lift distributions and Cp values. The lift distribution evidents the high taper ratio and very small chord at the tip. That is the reason for the peak around 50% of the span. Also, the sectional data reveals that the angle of attack is much higher than the initial. In sum, the presented results show that, to fulfill the C_L constraint, optimizer had to make changes to the geometry of the wing that had repercussions in its aerodynamic

performance. Yet, the optimization was performed successfully, since drag was reduced to the minimum possible when the wing is generating the requested lift, as shown by a 2.54% decrease in the C_D value of the final optimized design, when, compared to the first major iteration design (assumed as the first design to respect the C_L constraint). Figure 4.17(a) shows the history of the optimization objective with major iterations.

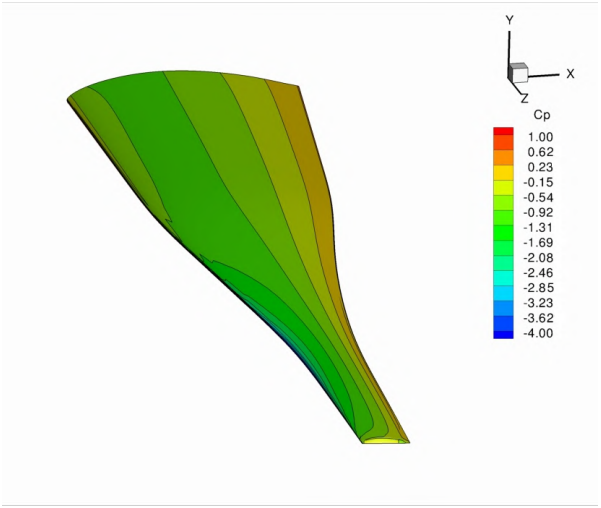
• **Case Study : L-23 Super Blanik Wing** This case study differs from the above as the initial geometry for the optimization is actually a real wing of a sailplane. Although this type of initial geometry would not be usually used as an initial design point for the preliminary design stage, it configures as a good test for the MDO tool. In this exercise, the objective is to change a real wing geometry and see how its performance can be or not to be improved, as fulfilling the defined constraints. It is important not to forget that, as this is a real wing with proven results (as the L-23 Super Blanik showed in its history), its design is already complex and optimized for the manufacturer's requirements. Comparing to the previous case study, the optimization of the L-23 wing should be faster or lead to unexpected results. A note for the fact that the initial twist variables, although present a null value in the optimization problem, they are in fact negative, as the L-23 wing has a twist angle of $-3°$. So, the final optimized twist values should be added to the initial real value, to calculate the real optimized twist values. The summary of the parameters of the aerodynamic optimization of the L-23 sailplane wing is presented in Table 4.7.

Table 4.7: Aerodynamic optimization parameters for the L-23 wing.

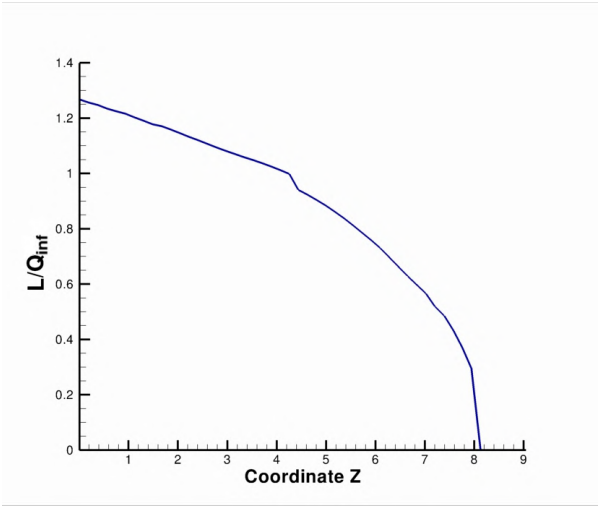
Constraint	Initial Value	Optimized Value	Lower Bound	Upper Bound	
C_L	0.981	0.779	0.779	0.779	
Parameter	Initial Value	Optimized Value	Lower Bound	Upper Bound	
Angle of Attack	3	3.15	-4	7	°
Twist (z/b=30%)	0	0	-10	10	°
Twist (z/b=60%)	0	5	-10	10	°
Twist (z/b=90%)	0	5	-10	10	°
Twist Tip	0	-5	-10	10	°
Chord Scale (z/b=30%)	1	1	0.5	2	
Chord Scale (z/b=60%)	1	0.5	0.5	2	
Chord Scale (z/b=90%)	1	0.5	0.5	2	
Chord Tip	1	0.5	0.5	2	
C_D	0.0143	0.00997			
Time	0	12680			$seconds$

The results from the optimization of the model of the real L-23 wing show some differences from the first case study. The most notorious is the angle of attack, which in this case study was decreased to a value little lower than the initial. This can be explained as the initial geometry and the airfoil of its sections provide better aerodynamic performance and therefore produce a C_L value over the C_L constraint. This was stated in the aerodynamic analysis to this case study. Thus, not to increase lift, the angle of attack was briefly maintained. Despite this difference, the other optimized variables confirm the trend shown by the previous case study. The taper ratio is increased from the root to the tip. This resulted in a smaller wing area and bigger aspect ratio. These changes to the taper ratio are also reflected in the decrease

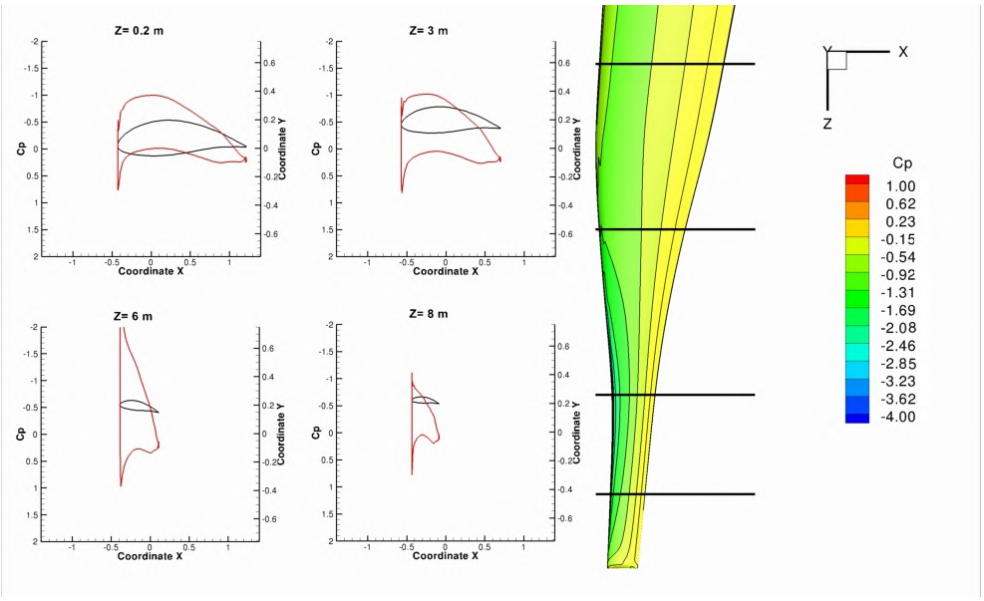
of C_D, that is verified against the initial value. Although the section at tip shows a negative twist of -5° , all the other sections show positive twist, confirming that the optimizer introduced some wash-in to the wing (positive twist), to nullify the initial negative twist of the L-23 wing. Unlike what happened in the semi-tapered case study, in this one the C_L constraint was fulfilled without a major change in the angle of attack. This allowed space for the optimizer to make small refinements, like the twist angles, to improve the wing aerodynamic performance. As with the first case study, the lift and Cp distribution over the wing surface and some sectional data of the optimized wing are illustrated in Fig.4.16.



(a) Results for Cp distribution over the wing.



(b) Results for lift distribution over the wing.

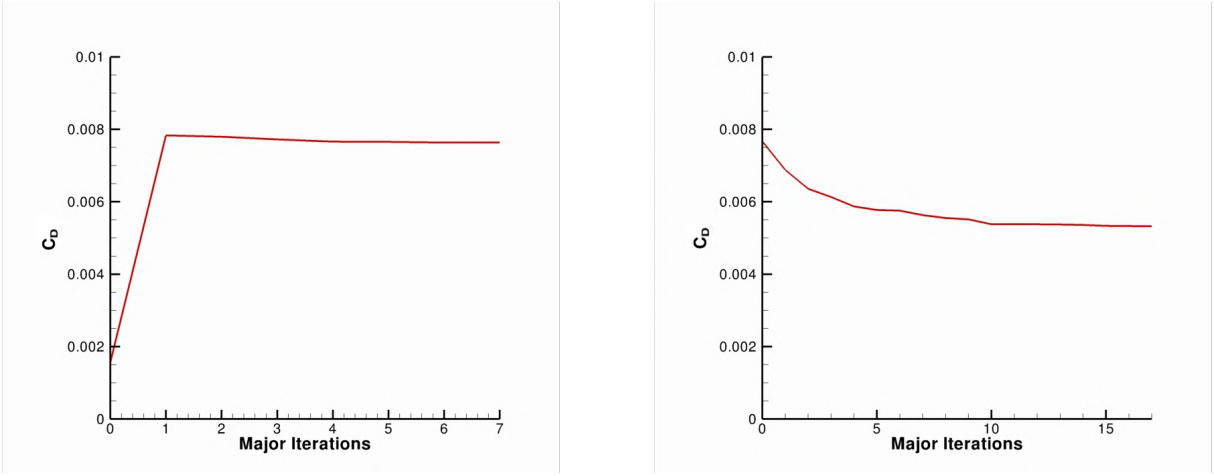


(c) Results for Cp distribution over four sections.

Figure 4.16: Aerodynamic optimization results for the L-23 sailplane wing.

From the top view, it is possible to observe that the initial negative sweep angle is negated by the change in the scale of the last sections of the wing, from approximately 35% span. This change affects the incident velocity of that airflow which, for instance, makes the next sections (towards the tip) generate more lift. The optimized wing lift distribution shows some differences from the initial, being closest to the

optimum (elliptical) distribution. The overall results show that using the L-23 wing as the stating point for the optimization allowed more space to the optimizer to enhance the aerodynamic performance of the wing geometry. The C_L constraint was easily achieved and then the remaining parameters were modified by the optimizer to improve the wing performance. Ultimately, the optimization objective was achieved, as drag was reduced to a even lower value than the initial, while the overall aerodynamic performance was increased. As shown in the convergence history of the optimization objective, presented in Fig.4.17(b), the C_D value decreased 30.6% in the final optimized design.



(a) Convergence history for the semi-tapered case study.

(b) Convergence history for the L-23 case study.

Figure 4.17: Convergence history for the aerodynamic optimization of the case studies.

This concludes Section 4.3, dedicated to the presentation of the results obtained with the MDO framework modules for the analysis and optimization in the discipline of Aerodynamics.

4.4 Structures

This section shows the results obtained with the structures module of the MDO tool. It documents the structural analysis of the case studies presented at Section 4.2.

Similar to the aerodynamic analysis, there were no imposing requirements for the structural simulations performed. Thus, the conditions chosen are merely academic. The exercise performed consisted in the study of the stresses and deformations of the wing-box structures of the case studies, when subjected to a single vertical wing tip nodal load of 500 N.

The mechanical properties used for all the wing structures were based on Aluminum 7075, a reference in the aeronautic industry, whose properties are listed in Table 4.8.

The finite-elements used for the structural mesh are based on mixed interpolation of tensorial components approach (MITC) shell elements (Chapelle et al., 2003) and the internal structural layouts of the case studies were defined in Sub-section 4.2.1. The only information needed to define all the parameters for the structural models of the case study wings is the level of mesh refinement.

Table 4.8: Mechanical properties of Aluminum 7075.

Mechanical properties		
Density	2810	Kg/m^3
Young's Modulus	71.7	GPa
Poisson's Ratio	0.33	
Yield Strength	434	MPa

4.4.1 Convergence Study

The first step to assess the results of the structural analysis is to determine how many elements are needed to have a reliable structural mesh discretization. Therefore, a convergence study for the structural mesh was performed and the results are documented in this section.

The number of structural components and the layout chosen for this study were the same as presented in Sub-section 4.2.1 for the case study of the semi-tapered wing. The exercise performed in the study was the same as the one used for the structural analysis, described in the beginning of Section 4.4. So, a single load was applied to the node located in the lower middle zone of the rib component nearest the wing tip. An illustration of the process is shown in Fig. 4.18.

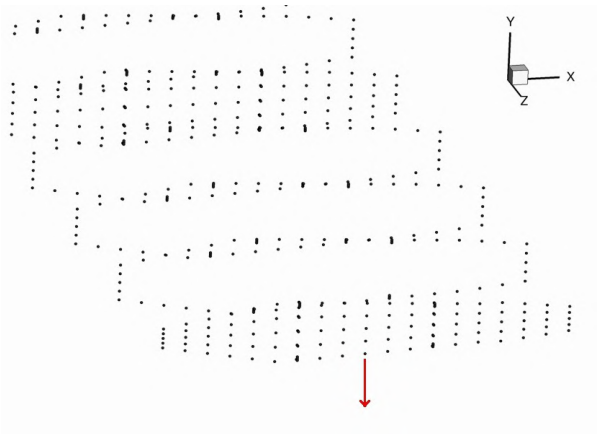


Figure 4.18: Example of a point load applied to the middle node of the wing-box tip (rib component).

Using the stated exercise, the result studied was the vertical displacement. This was computed for a set of structural meshes, ranging from 6,000 to 22,000 elements. Figure 4.19 shows the results of the convergence study.

To assess the accuracy, the relative error was used. As the real value of the deformation was unknown, a reference value given by a mesh refined with 22,000 elements was used. As expected, a convergence in the value for the maximum vertical deformation in the structure is verified as the number of elements is increased. Also the time required to perform the analysis was measured. As observable from Fig. 4.19, the time grows almost linearly with the number of elements used. From the results seen, a total number of elements above 7,500 was considered to give the best relation between accuracy error (approximately 10%) and time to perform the analysis (approximately 13 seconds). This number correspond to a span, chord and vertical spacing division of 9, 12 and 10 elements, respectively.

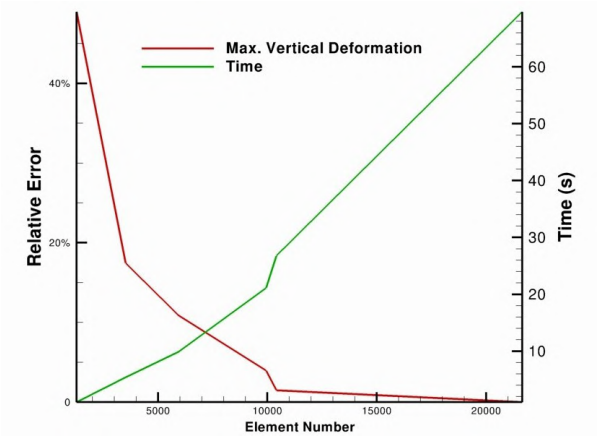
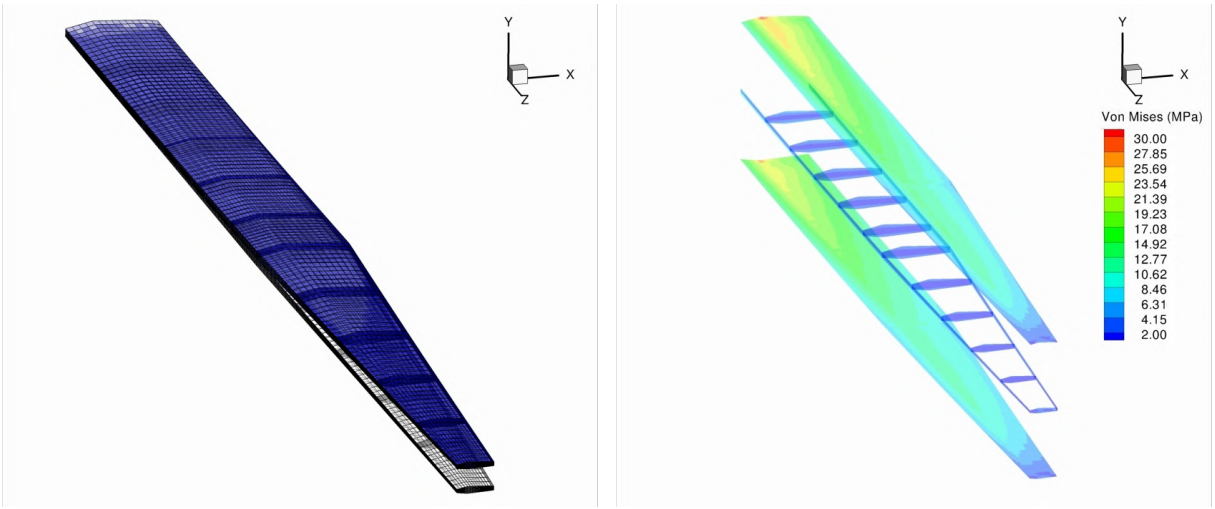


Figure 4.19: Results for the convergence study between accuracy and element number with *TACS*.

4.4.2 Structural Analysis

Once defined the structure of the mesh to use with *TACS*, all parameters needed to perform structural analysis to the wing-boxes of the case studies were defined. Using the methodology previously described, the structural analysis of the case studies were performed. Figures 4.20 and 4.21 summarize the results for each case study.



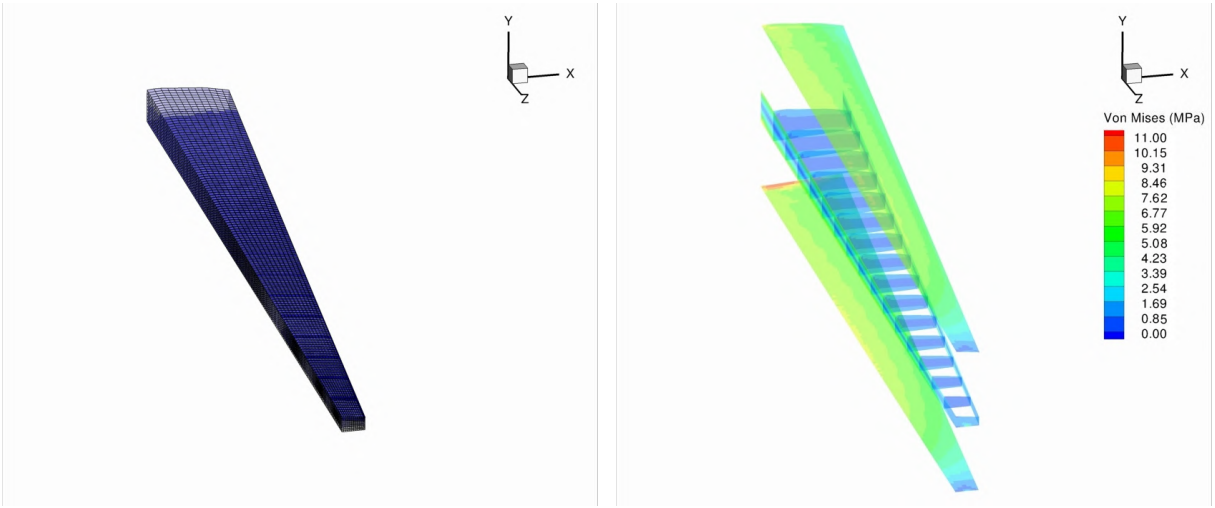
(a) Results for deformation of the wing structural layout (outlined is the undisplaced structure).

(b) Results for the Von Mises stress of the wing structural layout.

Figure 4.20: Structural analysis results for the semi-tapered sailplane wing.

The results show that the two structures have different stiffness, being the L-23 wing-box the stronger. That was already expected since the structure layouts of the two wing-boxes are different. Although both have the same number of spars, the number of ribs is higher in the L-23 wing-box. Also the span of the L-23 wing-box is greater than the one of the semi-tapered wing-box. The former fact should result in a higher deformation of the L-23 wing-box. Yet, that was not verified proving that the difference in the number of ribs is more important for the structure stiffness than its dimension. That was probably a reason why the manufacturer of the L-23 designed its wing-box with such number of rib components.

The other results shown present the Von Mises stresses in the structure components. These give



(a) Results for deformation of the wing structural layout (outlined is the undisplaced structure).

(b) Results for the Von Mises stress of the wing structural layout.

Figure 4.21: Structural analysis results for the L-23 sailplane wing.

information about how much effort are the components sustaining. The results show important differences between the structures of the two case studies. In the semi-tapered wing, the higher stresses are concentrated near the middle zone of the wing root skin panels. Then, they rapidly decrease towards the wing-box tip zone. On the other hand, in the L-23 wing-box, the values of the Von Mises stresses are much lower. The maximum values are verified in the lower skin panels at the wing-box root. So, sweep, twist and dihedral angles applied to the wing box increase the effort made in the bottom skin panels near the root. The higher height of the L-23 wing-box also allows the observation of zones that are sustaining higher stresses within the ribs. From a structural perspective, the structure of the L-23 showed better results, thus, a wing-box structure with higher height and higher number of ribs presents a better starting point for an MDO of a sailplane wing. In summary, the structures module provided consistent results for the exercised structural analysis over the two case studies. These highlighted the differences between the structure layouts of wing-boxes, mainly due to the difference in the number of rib components. Although simple, these observations are important in the preliminary design stage, as they can allow the early choice of the better overall structure layout for the main components of the sailplane wing. Then, in later design stages, refinements to the structure can be made with less effort.

4.5 Multi-Disciplinary Analysis and Optimization

The aero-structural optimization results for the presented case studies are documented in this section. The previous steps were performed and documented in order to assess the performance of the disciplinary modules of the MDO tool. However, the objective of the established MDO tool was to run the aero-structural optimization of the case studies. So a first sub-section presents a multi-disciplinar analysis on the case studies. The results obtained were used as reference for the initial design geometries for the MDO. Then, a second sub-section presents the aero-structural optimization of the case studies. The results from this optimization are discussed and the important differences and trade-offs in relation

to the results, either the MDA performed and the aerodynamic optimization, are highlighted.

4.5.1 Aero-Structural Analysis

This section focuses on the results of the aero-structural analysis of the initial geometries and structures of the two case studies. The new aspect of this exercise is that the transfer scheme referred in Section 3.9 was employed within the MDO tool, to pass the loads from the aerodynamic mesh to the structural model and the displacement from the former to the first.

The objects studied were those used in the previous disciplinary exercises. The conditions used to simulate the initial flight condition was the same used for the aerodynamic analysis of Sub-section 4.3.2. As for the structural model, the layouts and specifications used, were already presented in Section 4.2 and Section 4.4. To recap the aerodynamic and structural parameters of the initial case studies, a summary is presented in Table 4.9. Using the methodology presented in Chapter 3, an aero-structural analysis was performed with the established MDO tool. Figures 4.22 and 4.23 summarize the results obtained for the two case studies. For each, the pressure coefficient distribution over the wing, the lift distribution over the span, the deformation of the wing-box and the Von Mises stresses are presented. The results of the aero-structural analysis are consistent with the disciplinary analysis. With the stated

Table 4.9: Overall parameters for the aero-structural analysis.

Aerodynamic Parameters	Value	
Mach	0.0743	
Angle of Attack	3	$^{\circ}$
Density	1.1117	Kg/m^3
Speed of Sound	336.4346	m/s
Structural Parameters	**Value**	
Material Density	2810	Kg/m^3
Material Young Module	71.7	GPa
Material Poisson Ratio	0.33	
Material Correlation Factor	0.8333	
Material Yield Strength	434	MPa
Top Skin Thickness	5	mm
Bottom Skin Thickness	5	mm
Spar Thickness	10	mm
Rib Thickness	8	mm

initial conditions, the wing that generates more lift is clearly the L-23 sailplane wing. That was expected from the results verified in the aerodynamic analysis of this case study. Also, the differences in the airfoil shapes were once again evidenced, with the L-23 laminar airfoils having smother slopes in the Cp evolution and higher Cp gradients between surfaces. In the lift distribution results, the trend was the same as in the aerodynamic analysis, with the semi-tapered wing showing the closest to optimal distribution.

In relation to the structural results, some differences were noted, with the L-23 wing-box showing lightly higher maximum Von Mises stresses than the semi-tapered wing-box, in the lower skin panels

near the wing-box root. Although showing more deformation, the structure in the semi-tapered wing is sustaining less effort. On one hand, the L-23 higher stiffness guarantees that the wing aerodynamic shape is maintained in flight condition, providing better aerodynamic performance. On the other hand, it shows less flexibility than the semi-tapered, thus requiring that its components support higher stresses.

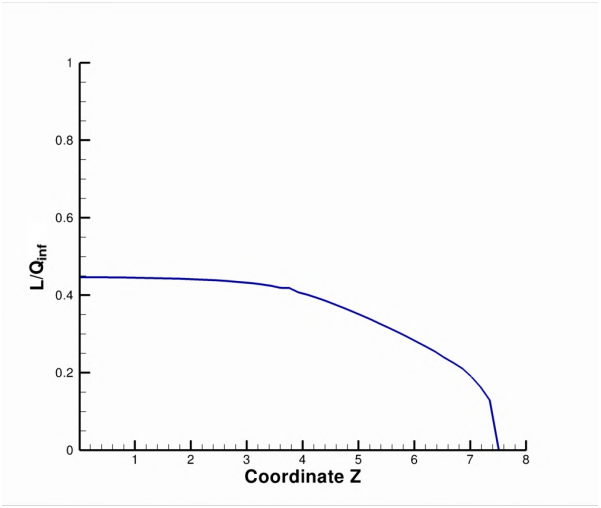
In the simulation, it is important to refer that the loads and displacements seem to have been successfully transfered between the aerodynamic and structural meshes, as it was possible to verify that the deformation of the wing-boxes was consistent with the deformation of the OML of the wing surfaces.

Overall, the results corroborate the idea that some compromises where made in the design of the L-23 wing, between aerodynamic performance and structural performance. Being the semi-tapered case study a simpler initial geometry.

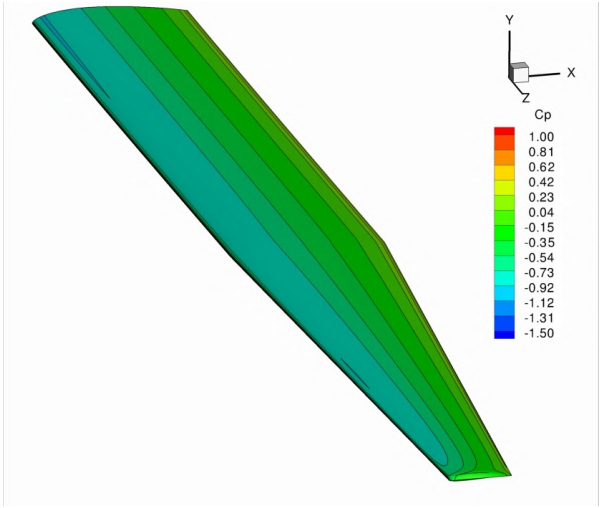
4.5.2 Aero-Structural Optimization

The last exercise performed in the scope of this thesis was an aero-structural optimization. This was also its main objective. So the other exercises were steps towards this objective, beginning with the exercise of generating the case study geometries, with the geometry module, to the exercise of performing an aero-structural analysis using the disciplinary modules of the established MDO tool. Similar to what was described in the previous sub-section, for the MDA, the parameters used to simulate the initial flight condition were the same used for the aerodynamic analysis of Sub-section 4.3.2. As for the structural model, the layouts and specifications used were already presented in Section 4.2 and Section 4.4. The methodology was also discussed in Chapter 3. So, the MDO tool established in this thesis was used to perform an aero-structural optimization on the sailplane wing of the case studies.

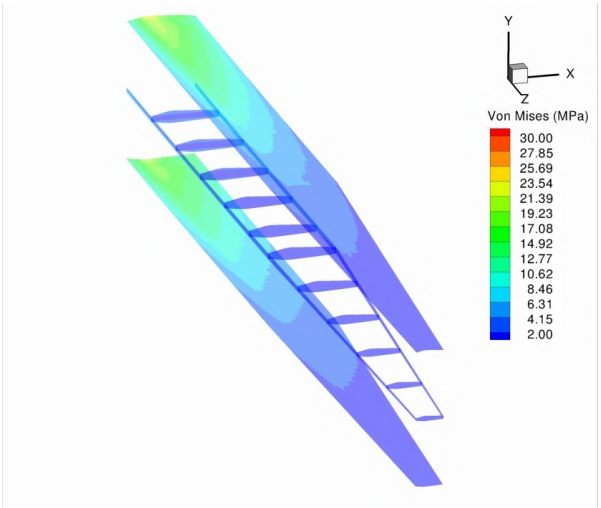
In addition to the maximization of the L/D ratio, to maximize the range of flight, the weight minimization is one of the main objectives in sailplane performance enhancement. The weight reduction can make the flight last longer or, in other words, maximize the flight endurance. So, the established optimization problem, used to perform the MDO on the case studies, was a drag minimization with a weight constraint, enforcing the weight reduction. As this was an academic study, the requirements for the optimization study were not imposed, so two types of constraints were set. An aerodynamic constraint set to the L/W ratio and a set of structural constraints for the maximum Von Mises stresses. The aerodynamic constraint implied that the lift generated by the wing had to be equal to the sailplane weight. This was not fixed, since reducing the weight of the wing structure was one of the objectives of the optimization. So a percentage of the initial weight of the sailplane was fixed, allowing the remaining percentage to change according to the wing structures weight. The Von Mises stress constraints were done indirectly through KS function constraints. KS functions are used to aggregate all stresses into a single constraint, for the skin, spar and rib group elements. These were set to the range of 0.3 to 2, which can be interpreted as the minimum safety before failure. So the variation of the structural weight was possible due to the variations on the component thicknesses, (structural) variables in the optimization problem and the variation of lift was possible due to the variation of the aerodynamic parameters: angle of attack, twist and scale.



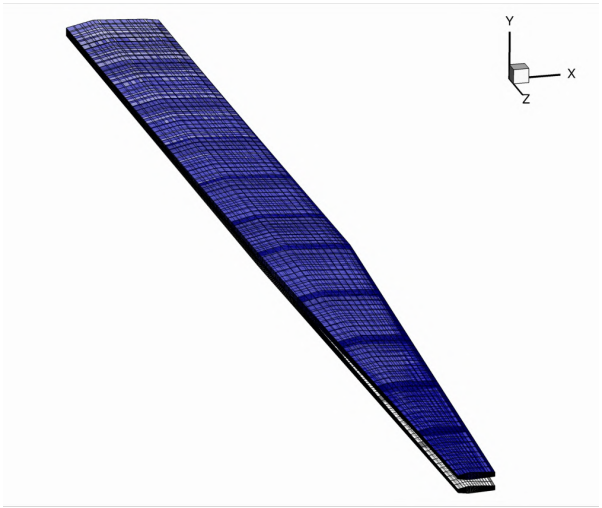
(a) Results for lift distribution over the wing.



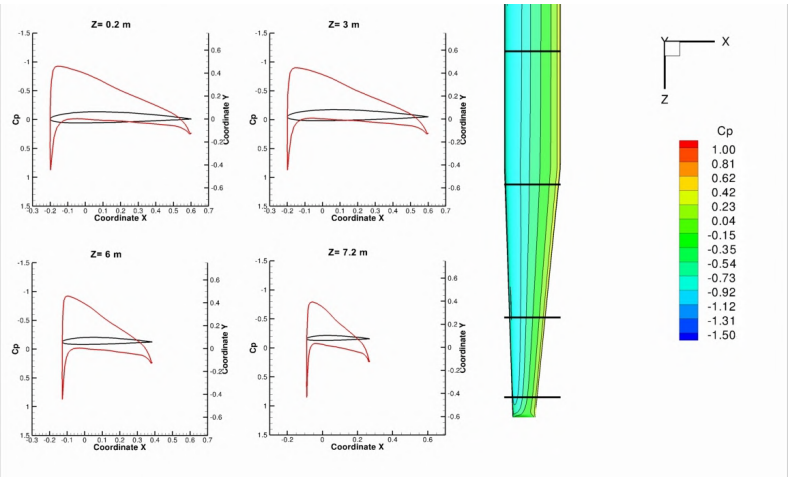
(b) Results for Cp distribution over the wing.



(c) Results for the Von Mises stress of the wing structural layout.

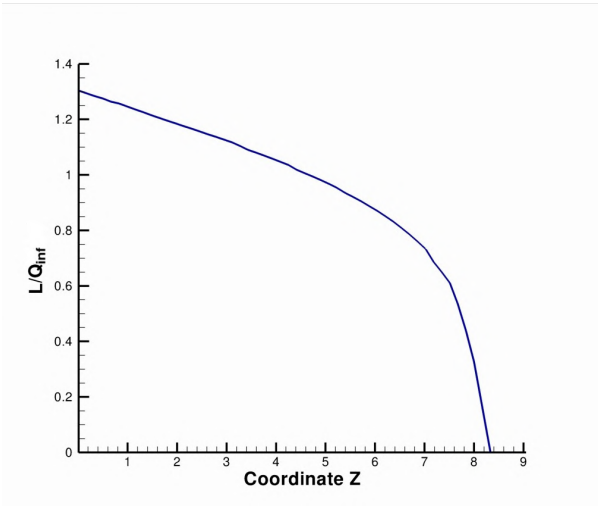


(d) Results for deformation of the wing structural layout (outlined is the undisplaced structure).

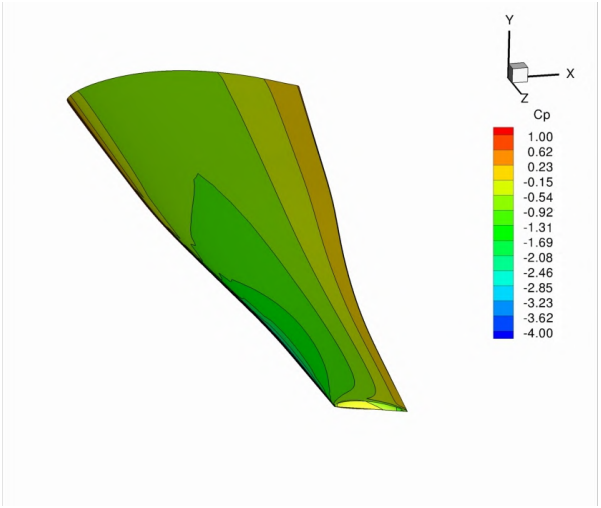


(e) Results for Cp distribution over four sections.

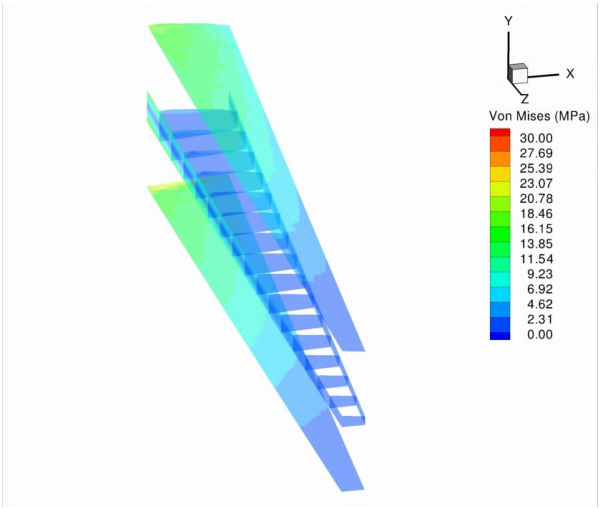
Figure 4.22: Aero-structural analysis results for the semi-tapered sailplane wing.



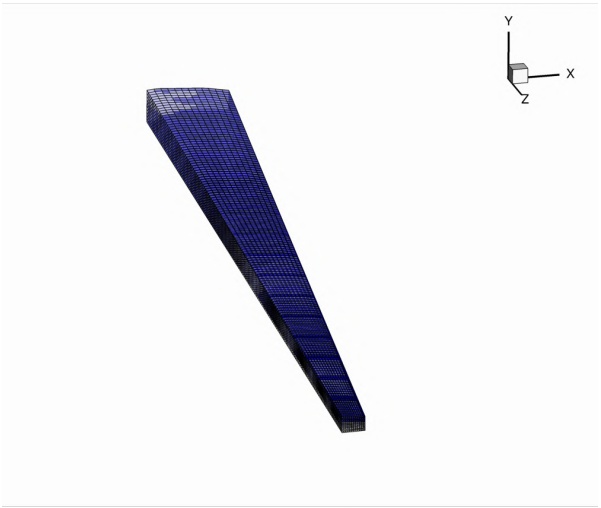
(a) Results for lift distribution over the wing.



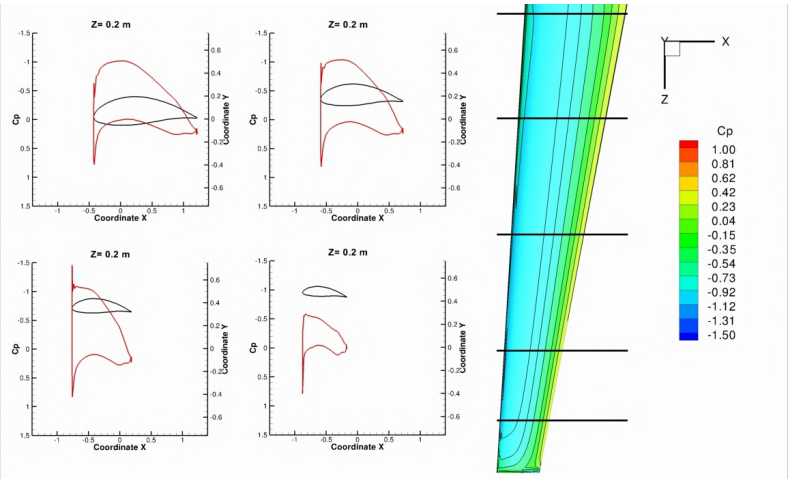
(b) Results for Cp distribution over the wing.



(c) Results for the Von Mises stress of the wing structural layout.



(d) Results for deformation of the wing structural layout (outlined is the undisplaced structure).



(e) Results for Cp distribution over four sections.

Figure 4.23: Aero-structural analysis results for the L-23 sailplane wing.

Mathematically, the aero-structural optimization problem is represented as

$$\begin{aligned} \text{Minimize :} \quad & C_D, \\ \text{s.t.:} \quad & L = W_{Sailplane}, \\ & 0.3 < KS < 2, \\ \text{w.r.t.:} \quad & \alpha, \theta(z/b), c(z/b), \qquad z/b = 0.3, 0.6, 0.9, 1, \\ & Top\ Skin, Bottom\ Skin, Spar, Rib\ Thicknesses. \end{aligned} \tag{4.4}$$

As the script for the aero-structural optimization is extensive, it is remitted to Appendix D. The next topics within this sub-section will address the results of each case study.

• **Case Study : Semi-tapered Wing** To summarize the results of the aero-structural optimization, a comparison of the initial and optimized design variables and constraints is given in Table 4.10. As the number of thickness variables was too long, a median was made for each group of components.

Table 4.10: Aero-structural optimization parameters for the semi-tapered wing.

Constraint	Initial Value	Optimized Value	Lower Bound	Upper Bound	
Total Mass	430	382.3	0	525	Kg
Vertical resultant force	-	0	0	0.3	N
KS top skin group	-	0.345	0.3	2	
KS bottom skin group	-	0.436	0.3	2	
KS spar group	-	0.458	0.3	2	
KS rib group	-	2	0	2	
Parameter	**Initial Value**	**Optimized Value**	**Lower Bound**	**Upper Bound**	
Angle of Attack	3	7	-4	7	$^\circ$
Twist (z/b=30%)	0	5	-10	10	$^\circ$
Twist (z/b=60%)	0	5	-10	10	$^\circ$
Twist (z/b=90%)	0	5	-10	10	$^\circ$
Twist Tip	0	5	-10	10	$^\circ$
Chord Scale (z/b=30%)	1	0.769	0.5	2	
Chord Scale (z/b=60%)	1	0.769	0.5	2	
Chord Scale (z/b=90%)	1	0.769	0.5	2	
Chord Tip	1	0.769	0.5	2	
Median Top Skin Thickness	5	1.5	1.5	10	mm
Median Bottom Skin Thickness	5	1.5	1.5	10	mm
Median Spar Thickness	10	5	5	10	mm
Median Rib Thickness	8	5	1.5	10	mm
C_D	0.00158	0.00515			
Time	0	32827			$seconds$

The results of the aero-structural optimization of the semi-tapered case study show that its objective was achieved, which was the weight reduction of the wing. Looking to the optimization constraints, all of them have been fulfilled, so a feasible design was generated in the optimization. The aerodynamic parameters show some important aspects of the optimization. First, the optimized angle of attack is equal to the defined upper bound. This shows that the optimizer had to increase the angle of attack to

its maximum to generate an higher value for the lift. Therefore, the drag was also increased significantly in the starting iterations of the optimization, as visible in Fig.4.24(f). Also, when that limit was reached, the optimizer increased the twist angle in all the sections of the wing, so a higher local angle of attack could be reached and, therefore, more lift could be generated. The scale of the sections along the span was also decreased, introducing a higher taper ratio to the wing. Yet, the scale values were not lowered to the minimum allowed, as was seen in the aerodynamic optimization. This should be expected since that would probably decrease the drag of the wing. However, as the objective was weight reduction, some trade-offs had to be made between the aerodynamic and structural performance. This fact explains why the optimized drag value was higher than the initial. Also, decreasing the chord of the wing sections to lower values, would have implications in the structural stiffness of its wing box and, therefore, in the values of the KS constraints. The structural parameters are consistent with the constraint values. The median thickness of the top skin, bottom skin and spar groups were lowered to the minimum possible, therefore reducing the weight of the wing-box structure from the first iterations, as shown in Fig.4.24(f). Therefore, the value of their KS functions was also lowered. The optimizer chosen to lower these thicknesses so that the lift, generated by the wing surface, could balance the weight of the aircraft. However, the median rib thickness was not lowered to its minimum possible. This justifies the higher value of the KS function in this group of components. This indicates that this value could not be lowered due to the structural constraints. For example, reducing the rib thickness could increase the stresses that had to be sustained by other structural components, which, for instance, would decrease the value of their KS functions to a point where they would not be within the constraint bounds. So a trade-off has made by the optimizer between lowering the structural weight and fulfilling the structural constraints. Figure 4.24 shows some relevant results of the optimized wing geometry. Looking to the figures, one can see the higher deformation of the wing-box. Also the Von Mises stresses comproved that a higher effort is being made by the structural components, with the root skin panels of the upper surface showing the higher stresses. These results are consistent with the minimum KS function value, which corresponds to the top skin group. The Cp distributions over the presented sections and the lift distribution over the wing comproved the higher angle of attack present in the optimized parameters. Yet, the optimization problem was successfully accomplished, as the objective of reducing the weight and drag of the sailplane was achieved while fulfilling all the structural and aerodynamic constraints. Using the MDO technique a final optimized design, respecting all the constraints, was obtained with an increase of 225% in the C_D and a decrease of 11% in the total weight.

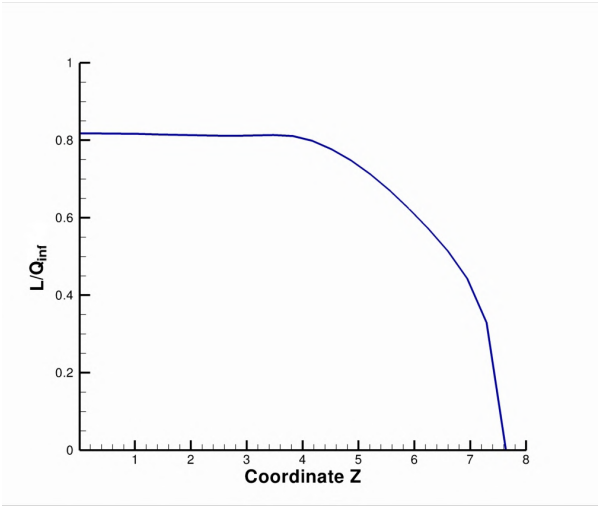
• **Case study : L-23 Super Blanik Wing** As in the first case study, a summary of the results for the aero-structural optimization of the L-23 case study is given in Table 4.11. As with the semi-tapered case study, the results of the aero-structural optimization of the L-23 case study show that its objective was achieved, which was the weight reduction of the wing. Looking to the optimization constraints, all of them have been fulfilled, so a feasible design was generated in the optimization. The aerodynamic parameters show some important aspects of the optimization that highlight the difference between the case studies. The optimized angle of attack is lower than the initial. This shows that the optimizer had

Table 4.11: Aero-structural optimization parameters for the L-23 wing.

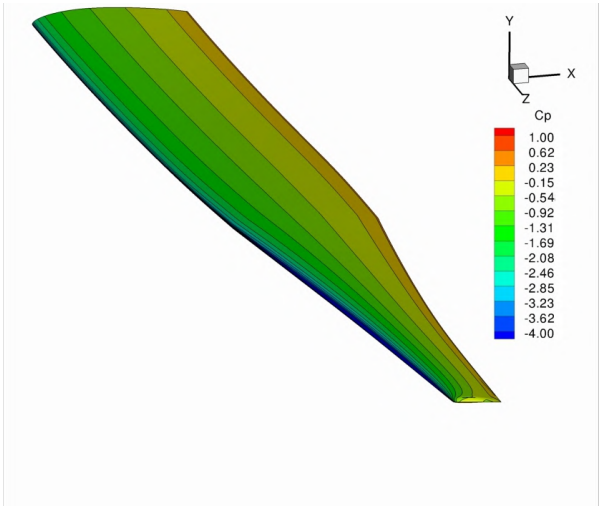
Constraint	Initial Value	Optimized Value	Lower Bound	Upper Bound	
Total Mass	530	525	0	525	Kg
Vertical resultant force	-	0	0	0	N
KS top skin group	-	0.340	0.3	2	
KS bottom skin group	-	0.358	0.3	2	
KS spar group	-	0.353	0.3	2	
KS rib group	-	2	0	2	
Parameter	**Initial Value**	**Optimized Value**	**Lower Bound**	**Upper Bound**	
Angle of Attack	3	1.24	-4	7	$^\circ$
Twist (z/b=30%)	0	5	-10	10	$^\circ$
Twist (z/b=60%)	0	5	-10	10	$^\circ$
Twist (z/b=90%)	0	5	-10	10	$^\circ$
Twist Tip	0	5	-10	10	$^\circ$
Chord Scale (z/b=30%)	1	0.768	0.5	2	
Chord Scale (z/b=60%)	1	0.768	0.5	2	
Chord Scale (z/b=90%)	1	0.768	0.5	2	
Chord Tip	1	0.768	0.5	2	
Median Top Skin Thickness	5	1.5	1.5	10	mm
Median Bottom Skin Thickness	5	1.5	1.5	10	mm
Median Spar Thickness	5	5	5	10	mm
Median Rib Thickness	8	10	1.5	10	mm
C_D	0.00774	0.00687			
Time	0	17635			$seconds$

to decrease the angle of attack to decrease the lift generated. Thus, the initial decrease in drag visible in the starting iterations of the optimization process (Fig.4.25(f)). However, the twist angles in all the sections of the wing were increased, so that more lift could be generated with less drag increase. The scale of the sections along the span was decreased, introducing a higher taper ratio to the wing. Yet, the scale values were not lowered to the minimum allowed, as was seen in the aerodynamic optimization. These results are consistent with those of the first case study. Yet, in this case study, the overall drag was minimized in relation to the initial geometry. If the scale values were lowered even more, there could have been implications in the structural results.

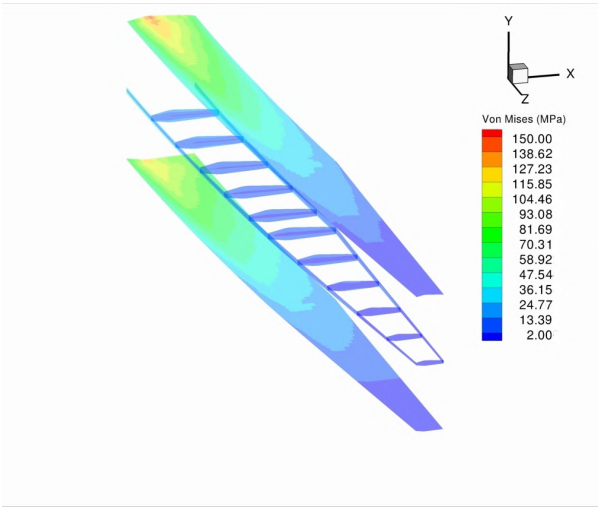
The structural parameters are consistent with the constraint values. The median thickness of the top skin, bottom skin and spar groups, were lowered to the minimum possible, therefore reducing the value of their KS functions (lowering the associated safety factor). The optimizer chose to lower these thicknesses so that the lift, generated by the wing surface, could balance the weight of the aircraft. Again, the median rib thickness showed different results, being in the upper bound value. This explains why the KS function value of the rib group is at the higher bound, and why the weight has been optimized to the higher bound value as visible in Fig.4.25(f). The optimizer chose not to reduce the rib thickness more, as it could increase the stresses that had to be sustained by the other structural components, which, for instance, would decrease the value of their KS functions. The values of the KS functions for the other component groups are near the lower bound so having them sustaining more effort was not an option. Again, a trade-off has been made by the optimizer between lowering the structural weight and fulfilling the structural constraints.



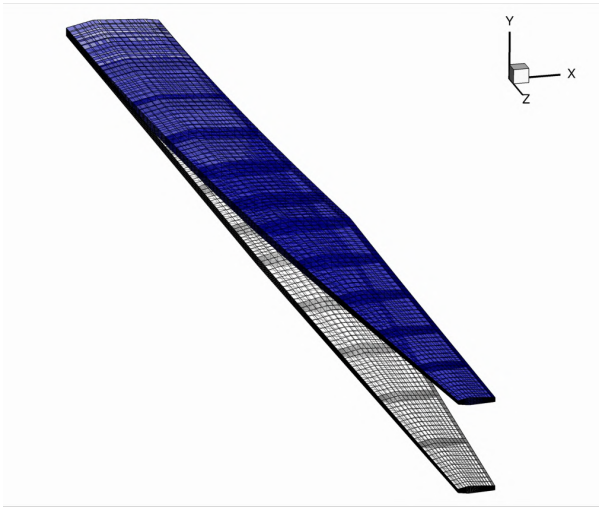
(a) Results for lift distribution over the wing.



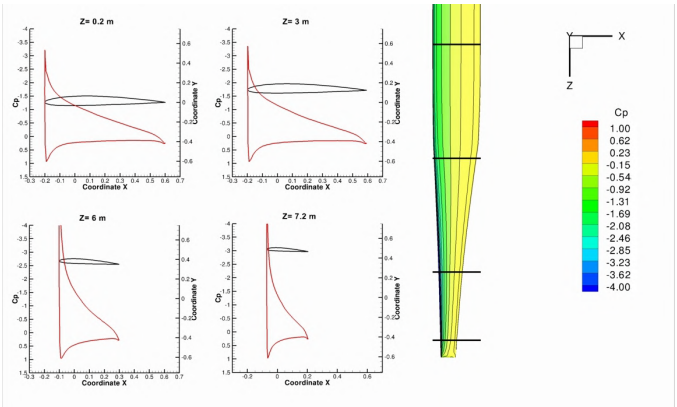
(b) Results for Cp distribution over the wing.



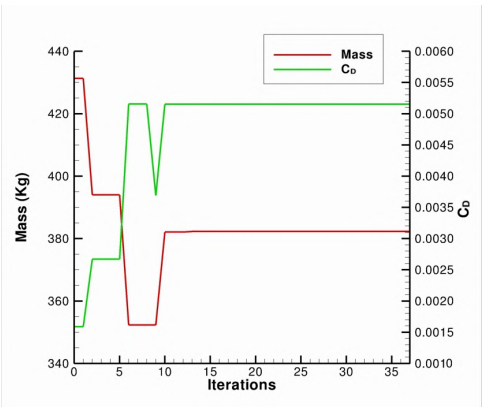
(c) Results for the Von Mises stress of the wing structural layout.



(d) Results for deformation of the wing structural layout, (outlined in the undisplaced structure).



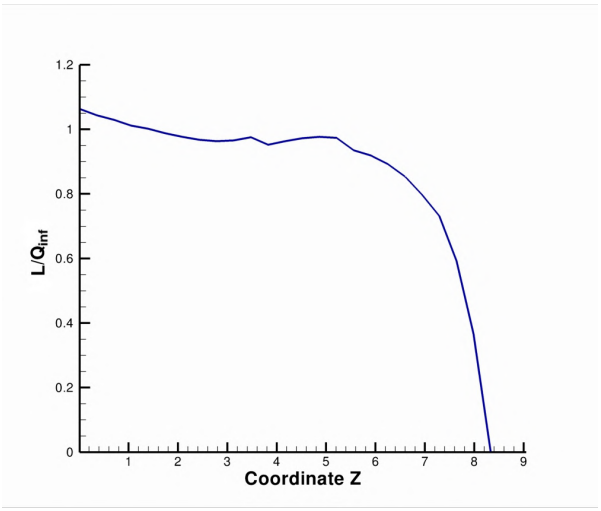
(e) Results for Cp distribution over four sections.



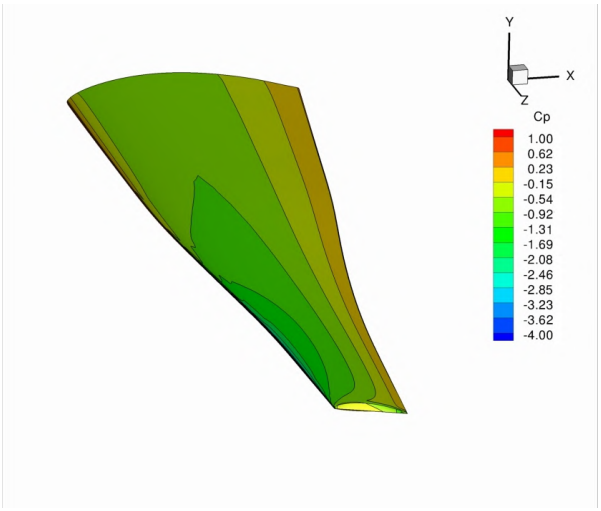
(f) Convergence history.

Figure 4.24: Aero-structural optimization results for the semi-tapered sailplane wing.

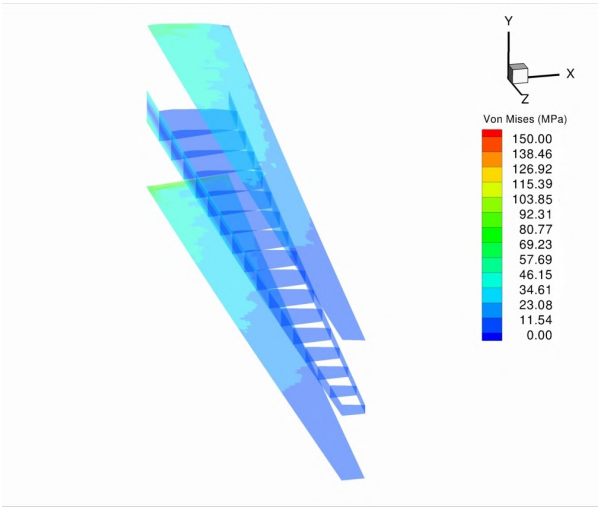
For a better visualization of the results, Fig. 4.25 shows the relevant aerodynamic and structural results.



(a) Results for lift distribution over the wing.

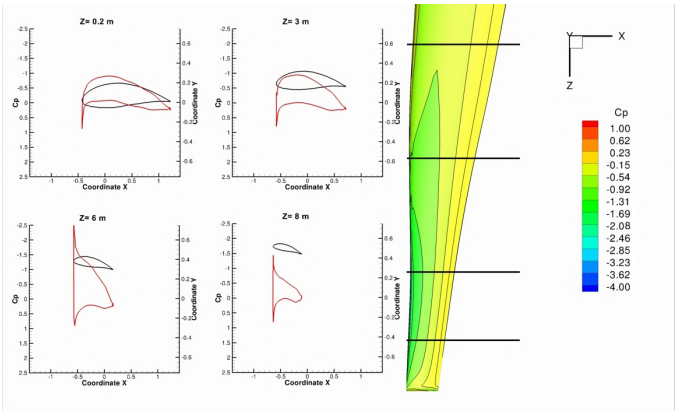


(b) Results for Cp distribution over the wing.

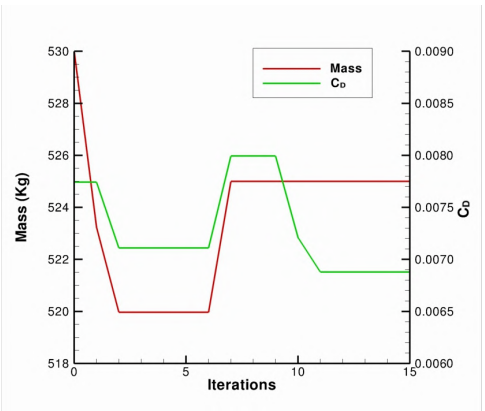


(c) Results for the Von Mises stress of the wing structural layout.

(d) Results for deformation of the wing structural layout, (outlined in the undisplaced structure).



(e) Results for Cp distribution over four sections.



(f) Convergence history for the L-23 case study.

Figure 4.25: Aero-structural optimization results for the L-23 sailplane wing.

The deformation of the wing-box was higher that the one verified in the MDA. Also the Von Mises stresses comproved that a higher effort was made by the structural components, with the root skin panels of the upper and lower surfaces showing the higher stresses. These results are consistent with the minimum KS function values of these groups. The Cp distributions over the presented sections and the lift distribution over the wing, comproved the higher overall angle of attack due to the increase in the twist angles. Indeed, the sections near the wing tip show some higher gradients in the Cp evolution over the airfoils, which is also consistent with higher local angles of attack. In general, the optimization problem was successfully accomplished, as the objective of reducing the drag and weight of the sailplane was achieved while fulfilling all the aerodynamic and structural constraints. Using the MDO technique a final optimized design, respecting all the constraints, was obtained with a decrease of 11% in the C_D and a decrease of 0.009% in the total weight. Also the differences from the simple disciplinary optimization were evident, proving that some trade-offs had to be made between the structural performance and the aerodynamic performance.

4.6 Summary

This chapter presented the exercises run with the MDO tool, from the generation of the case study surfaces to the results of the MDO. The results proved that adding aerodynamics and structures to the optimization process, from the same initial design, lead to different optimized results. Making these compromises earlier in the preliminary design stages of an aircraft design can reduce the efforts and resources of the following stages, as some trade-offs have already been made. In sum, this section assessed the MDO tool established in this thesis for the MDO of sailplane wings.

Chapter 5

Conclusions and Future Work

5.1 Achievements and Acquired Knowledge

The motivation for this thesis was to develop knowledge in the field of multi-disciplinary design of aircraft wing configurations. Two core disciplines were considered, aerodynamics and structures, for the objective of this thesis, which was running an aero-structural optimization of sailplane wings.

A literature review was made in order to the acquisition of knowledge in the field of MDO. From this review, which lasted until the end of the thesis, it was clear the importance that MDO approaches can have in preliminary design stages of aircraft design. Its usage can save important resources and help engineers to enhance their view. This review also provided the contact with the tools and architecture that would be used in the MDO framework. Based on that review, an multi-disciplinary feasible architecture was chosen for the framework. For the disciplinary analysis a panel code was chosen and proved to be well suited for the analysis of incompressible flows and a finite-element code was employed for the analysis of stresses and deformations. The implementation of the tools required a deeper knowledge of programming languages, which was important to complement the knowledge acquired during the aeronautical engineering course. The optimization algorithm employed was a sequential quadratic programming with adjoint sensitivities evaluation, which is very used in aero-structural optimization problems. This implementation stage was confirmed as the longest step of the realization of this thesis, as the task of coupling and testing the modules often revealed tough. This, however, was important as it allowed the increase of knowledge of the code, methods and processes behind the MDO framework and in some cases, the detection of bugs. With the MDO tool established, tests were created and used to assess the framework performance. These tests provided deeper knowledge of the behavior of wing geometries and structures in a cruise flight condition. Also, worth of mention is the fact that, though, the exercises performed were simple, through the course of this thesis, the formulation of these aero-structural problems was the hardest task, as transforming multiple disciplinary problems into one multi-disciplinary problem is complex and time consuming. And if this happen with a simple aero-structural MDO exercise, in a real aircraft project, were the number of either disciplines and variables is much higher, the time and effort needed for such task can be unbearable. Maybe this is why MDO is not yet a standard practice in

aircraft industry. Still, after the problems have been formulated, the aero-structural optimization run with the established MDO tool performed rapidly and smoothly. This fact is very important if its taken into account that the work for this thesis was all made using a normal notebook. So, if used with a proper workstation with more computational power, multi-disciplinary problems with higher complexity can be solved by using MDO, in a faster pace, saving time when compared with a discipline iterative approach.

Finally, from the results obtained with the aerodynamic only and aero-structural optimizations, it was possible to capture the multi-disciplinary trade-offs between what was best in terms of aerodynamics and what was feasible in terms of aero-structural requirements. So, using MDO in preliminary design stage exercises have shown that taking into account more than one discipline can lead to better optimized designs. Having this multi-disciplinary perspective right from the beginning of the aircraft design process can reduce the feasible design space, allowing resources to be saved from later re-designs. With this verification, the objective of this thesis was accomplished, as an aero-structural optimization was run and proven that MDO can really make a difference in the early design stages, by extending the engineer view of the design multi-disciplinary problems and allowing earlier multi-disciplinary optimum points to be found. Thus, I personally think that the base skills acquired during the realization of this thesis, in the field of MDO, will be a real asset in future projects.

5.2 Directives for Future Work

Future work in the development of the aero-structural MDO framework established in this thesis is expected, to explore the full capabilities of each of the modules employed, so that more realistic aircraft design problems can be solved. A specialized study can be made about each module, by performing more complex exercises. This is important as some aerodynamic results have shown that there are some numerical deviation when computing the sectional Cp. Also, different flight conditions should be considered, as its implementation is already possible within the current framework. These could be associated with the performance of sailplanes as, for example, to minimize the sinking speed or an maneuver flight condition. From a structural point of view, critical aerodynamic load cases associated with the structural stress constraints could be also considered.

Regarding the modules, it would be interesting to improve the geometry module so different aircraft wings could be employed in the MDO framework, as for example wings with lift-enhancement devices. The aerodynamic module *Tripan* could also be improved so that a method more complex method could be used for high-fidelity analysis. This would allow the modeling of compressible flows, which would extend the range of possible flight conditions. The structural module *TACS* was the least explored in the exercises performed with the framework. Despite this, future works could use its full potential, for example, in terms of the use of composite materials. Also, an interesting work could consist of a validation of the TACS with experimental tests, for example with the Portuguese Air Force sailplanes.

In sum, the work developed within the present thesis has established a base MDO framework that have only been explored to its minor portion and, if desired, it can be extended and used in more complex multi-disciplinary studies in the future.

Appendix A

Script for the Geometry Module

Listing A.1: Full script for wing geometry generation.

```
#!/usr/bin/python
# ==============================================================================
# Geometry of a Wing
# ==============================================================================
# A generic semi-tappered wing is used for this example.
# ==============================================================================
# Geometry Parametrization and generation for Tripan and TACS
# ==============================================================================
# Part 1 : Importing Standard Python Modules
import os, sys, string, pdb, copy, time, numpy, datetime
# Set the beginning of the timer for code
t0 = datetime.datetime.now()

# ------------------------------------------------------------------------------
# Part 1.1: Importing External Python Modules
from numpy import linspace, cos, sin, zeros, ones, intc, tan, pi
from mpi4py import MPI

# ------------------------------------------------------------------------------
# Part 1.2: Importing Extension modules
from mdo_import_helper import *
exec(import_modules('pyGeo','pySpline', 'pyLayout'))

# ==============================================================================
# Part 2: Defining The Geometric Parameters of the Wing
# ------------------------------------------------------------------------------
# Part 2.1: Selection the number of sections and the airfoil shape
# Defining the Wing Span, Tapper and the chord dimensions at root and tip
span = 15.0 # m
tapper = 0.56
cr   = 0.8016  # m
ct   = cr*tapper  # m
# Create the airfoil list
nsections = 3
sec_list  = ['geo/naca2412.dat']*(2*nsections-1)
sec_list[0]  = 'geo/naca2412.dat'
sec_list[-1] = 'geo/naca2412.dat'
# Initializing variables
xm  = zeros(2*nsections-1)
ym  = zeros(2*nsections-1)
zm  = zeros(2*nsections-1)
rxm = zeros(2*nsections-1)
rym = zeros(2*nsections-1)
rzm = zeros(2*nsections-1)
scale = ones(2*nsections-1)
offset = zeros((2*nsections-1,2))
# When to switch to the last
pspan = [ 0.0, 0.97, 1.0 ]
sspan = [ cr, ct, ct ]
# Calculate the chord in the middle sections with tapper
u_one = (pspan[1] - pspan[0])/(pspan[1] - pspan[0])
sspan[1] = ct*u_one + (1.0-u_one)*cr
# Defining the values for xm,ym,zm,rotx,roty and rotz, to add Sweep, Diehdral
# or Twist Angles to the wing
for i in xrange(nsections):
    zm[i]              = -pspan[-(i+1)] * 0.5*span
    zm[i+nsections-1] =  pspan[i] * 0.5*span
    scale[i]           = sspan[-(i+1)]
```

```
    scale[i+nsections-1] = sspan[i]
    offset[i,0]              = 0.25
    offset[i+nsections-1,0] = 0.25
# If the wing as Sweep Angle (example: -5 degrees)
#X_delta_one = numpy.tan(numpy.pi*-5/180)*(pspan[1]*span/2)
#X_delta = numpy.tan(numpy.pi*-5/180)*(span/2)
#xm[0]= X_delta
#xm[1]= X_delta_one
#xm[2]= 0
#xm[3]= X_delta_one
#xm[4]= X_delta
#If the wing as Dihedral Angle (example: 3 degrees)
#Y_delta_one = numpy.tan(numpy.pi*3/180)*(pspan[1]*span/2)
#Y_delta = numpy.tan(numpy.pi*3/180)*(span/2)
#ym[0]= Y_delta
#ym[1]= Y_delta_one
#ym[2]= 0
#ym[3]= Y_delta_one
#ym[4]= Y_delta
#If the wing as Twist Angle (example: -3 degrees)
#twist_angle = -3.0
#rzm[0]=twist_angle
#rzm[1]=twist_angle
#rzm[2]=0
#rzm[3]=twist_angle
#rzm[4]=twist_angle

# ----------------------------------------------------------------------
# Part 2.2: Selection the number of spars, ribs and stringers
#Number of Ribs and Spars
nribs = 10
nspars = 2
# If there are holes in the ribs
#rib_hole_spec = [[] for i in xrange(nribs)]
#for irib in xrange(nribs):
#    rib_hole_spec[irib] = [[0,2],[2,3]]
# If there are holes in the skins
#skin_hole_spec = [[] for i in xrange(nribs-1)]
#for iskin in xrange(nribs-1):
#    skin_hole_spec[iskin] = [[5,8]]
# Blanking: turn on or off spars, ribs or skins
#Choose not to blank spars
spar_blank = numpy.ones((nspars, nribs-1), numpy.intc)
# or to blank the middle spars
#spar_blank = numpy.zeros((nspars, nribs-1), 'intc')
#spar_blank[0,:] = 1                    # LE Spar
#spar_blank[-1,:] = 1                   # TE Spar
#spar_blank[nspars/2,0:5] = 1           # Middle Spars
# Blanking for ribs
#Choose not to blank ribs
rib_blank = numpy.ones((nribs, nspars-1), numpy.intc) # None
# Turn off the root rib
rib_blank[0,:] = 0
# or to blank the middle ribs
#rib_blank = numpy.zeros((nspars, nribs-1), 'intc')
#rib_blank[0,:] = 1                     # Root rib
#rib_blank[-1,:] = 1                    # Tip rib
# Set the position for the LE Spar and the TE Spar to form the Wing Box
# x = 0 corresponds to the 1/4 chord location
xle_root = -0.1*cr
xte_root = 0.5*cr  # This is the 3/4 chord location
xle_tip = -0.1*ct
xte_tip = 0.5*ct   # This is the 3/4 chord location
# Set the finite-element order
elem_order = 2
# Sets the level of refining for the structural mesh
refine_level= 1
# Sets the number of control points for the pyGeo geometry object
Nctl = 27

# ======================================================================
# Part 3: Defining The Code for the functions that generate Tripan Input Files
def write_tripan_file(geo, trifile, wakefile, edgefile=None,
                      nu=40, nv=50, spacing='cosine', beta=2.0):
    U = zeros((nu+1,nv+1))
    V = zeros((nu+1,nv+1))
    if spacing == 'cosine':
        u = 0.5*(1.0 - cos(linspace(0.0,pi,nu+1)))
        v = 0.5 + 0.25*(1.0 - cos(linspace(0.0,pi,nv+1)))
    elif spacing == 'hyperbolic':
        x = linspace(0.0,1.0,nu+1)
        u = x - beta * x * (x-1.0) * (x-0.5)

        x = linspace(0.0,1.0,nv+1)
        v = 0.5*(1.0 + x - beta * x * (x-1.0) * (x-0.5))
    else:
        u = linspace(0.0,1.0,nu+1)
```

```
        v = linspace(0.5,1.0,nv+1)
    # end
    # Assume that you only want half of the wing!
    for j in xrange(nv+1):
        for i in xrange(nu+1):
            U[i,j] = u[i]
            V[i,j] = v[j]
    # Evaluate the top surface second
    ntop = zeros((nu+1,nv+1),intc)
    Xtop = geo.surfs[0](U,V)
    # Evaluate the bottom surface
    nbot = zeros((nu+1,nv+1),intc)
    Xbot = geo.surfs[1](U,V)
    # Write the TriPan file ...
    fp = open(trifile,'w')
    npts = 2*nu*(nv+1)
    npanels = 2*nu*nv
    fp.write('%d %d\n'%(npts,npanels))
    # Write out the top surface
    for j in xrange(nv+1):
        for i in xrange(nu+1):
            ntop[i,j] = i + j*(nu+1)
            fp.write('%.8g %.8g %.8g \n'%(Xtop[i,j,0],
                                          Xtop[i,j,1], Xtop[i,j,2]))
    # Write out the bottom surface
    for j in xrange(nv+1):
        for i in xrange(nu+1):
            if (i == 0 or i == nu):
                nbot[i,j] = ntop[i,j]
            else:
                nbot[i,j] = i-1 + j*(nu-1) + (nu+1)*(nv+1)
                fp.write('%.8g %.8g %.8g \n'%(Xbot[i,j,0],
                                              Xbot[i,j,1], Xbot[i,j,2]))
    # Write out the connectivity information for the top surface
    for j in xrange(nv):
        for i in xrange(nu):
            fp.write('%d %d %d %d \n'%(ntop[i,j], ntop[i+1,j],
                                       ntop[i+1,j+1], ntop[i,j+1]))
    # Write out the connectivity information for the bottom surface
    for j in xrange(nv):
        for i in xrange(nu):
            fp.write('%d %d %d %d \n'%(nbot[i,j], nbot[i,j+1],
                                       nbot[i+1,j+1], nbot[i+1,j]))
    fp.close()
    # Create the wake file along the trailing edge
    fp = open(wakefile,'w')
    nseg = 1
    fp.write('%d\n'%(nseg))
    fp.write('%d\n'%(nv+1))
    for j in xrange(nv,-1,-1):
        fp.write('%d %d\n'%(ntop[0,j],3))
    fp.close()
    if edgefile != None:
        fp = open(edgefile,'w')
        fp.write('%d\n'%(nv+1))
        for j in xrange(nv+1):
            fp.write('%d %d\n'%(ntop[-1,j],ntop[0,j]))
        fp.close()
    return

# ==========================================================================
# Part 4: Defining The Code for the generation of pyGeo Geometry object
#Names for the Output Files
geo_name = 'geo/wing.igs'
geo_tec_name = 'geo/wing_geo.dat'
bdf_name = 'mesh/wing.bdf'
bdf_tec_name = 'mesh/wing_struc.dat'
#Generate a wing file with Y = 0
if not os.path.isfile('geo/zero.dat'):
    fp = open('geo/ONERA_airfoil.dat','r')
    lines = fp.read()
    xs = string.split(lines)
    fp.close()
    fp = open('geo/zero.dat','w')
    for i in xrange(0,len(xs),2):
        fp.write('%s 0.0 \n'%(xs[i]))
    fp.close()
k = 2
geo = pyGeo.pyGeo('lifting_surface',
                  xsections=sec_list,
                  scale=scale, offset=offset,
                  x=xm, y=ym, z=zm,
                  rot_x=rxm, rot_y=rym, rot_z=rzm,
                  Nctl=Nctl,
                  k_span=k,
                  con_file='geo/wing.con',
                  tip = 'rounded' )
```

```
geo.writeIGES(geo_name)
final_time = datetime.datetime.now() - t0
print 'Total time spent in the generation of .igs file:', final_time

# ==============================================================================
# Part 5: Defining The Code for the generation of Tripan and TACS Input Files
if os.path.isfile('geo/wing.igs'):
    #-----------------------------------------------------------------------
    # Part 5.1: Script for the generation of TACS Input files

    geo = pyGeo.pyGeo('iges','geo/wing.igs')
    geo.doConnectivity()#'wing.con')
    geo.writeTecplot(geo_tec_name, edge_labels=True)
    # Now, generate aerodynamic analysis files
    write_tripan_file(geo, 'geo/wing_15x20.tripan', 'geo/wing_15x20.wake',
                      edgefile='geo/wing_15x20.edge',
                      nu=15, nv=20, spacing='hyperbolic', beta=1.8)
    write_tripan_file(geo, 'geo/wing_35x60.tripan', 'geo/wing_35x60.wake',
                      edgefile='geo/wing_35x60.edge',
                      nu=35, nv=60, spacing='hyperbolic', beta=1.8)
    tripan_time = datetime.datetime.now() - t0
    print 'Total time spent in the generation of tripan and tecplot files:', tripan_time

    #-----------------------------------------------------------------------
    # Part 5.2: Script for the generation of TACS Input files
    # Set up an array of constitutive properties
    X = numpy.zeros((nribs, nspars, 3))
    #Semi-span
    semi_span = span/2.0 # m
    z_tip = semi_span*0.98
    zloc = numpy.linspace(1e-4, z_tip, nribs)
    for j in xrange(nribs):
        v = (1.0*j)/(nribs-1.0)
        for k in xrange(nspars):
            u = (1.0*k)/(nspars-1.0)
            x_tip = (1.0-u)*xle_tip + u*xte_tip
            x_root = (1.0-u)*xle_root + u*xte_root
            X[j,k,2] = zloc[j]
            X[j,k,0] = (1.0-v)*x_root + v*x_tip
    # Set up the blanking for the top/bottom skins -
    # no leading or trailing edges
    top_blank = numpy.ones((nribs-1, nspars+1), numpy.int)
    bot_blank = numpy.ones((nribs-1, nspars+1), numpy.int)
    top_blank[:,[0,-1]] = 0
    bot_blank[:,[0,-1]] = 0
    # Set the spacing in the different directions
    span_spacing = refine_level * 5
    chord_spacing = refine_level * 8
    vertical_spacing = refine_level * 6
    # Set the number of elements along the skin between ribs
    span_space = span_spacing*numpy.ones(nribs-1, numpy.intc)
    # Set the number of elements along the skin between spars
    chord_space = chord_spacing*numpy.ones(nspars+1, numpy.intc)
    # Set the number of elements along the stringers between ribs
    #stringer_space = 3
    #rib_stiffner_space = 3
    # Set the number of elements along the holes in ribs
    #rib_O_space = 1
    # Set the number of elements along the holes in skins
    #skin_O_space = 2
    # Set up pyLayout
    te_list = []
    # Now, generate aerodynamic structural files
    layout = pyLayout.Layout(geo, te_list, nribs, nspars, X=X,
                             element_order=elem_order,
                             rib_space=chord_space,
                             span_space=span_space,
                             v_space=vertical_spacing,
                             top_blank=top_blank,
                             bot_blank=bot_blank,
                             spar_blank=spar_blank,
                             rib_blank=rib_blank)
    layout.finalize(bdf_name,bdf_tec_name)
    tacs_time = datetime.datetime.now() - t0
    print 'Total time spent in the generation of tripan,tecplot and TACS files:', tacs_time

# ==============================================================================
```

Appendix B

Script for the Aerodynamics Module

Listing B.1: Full script for an aerodynamic analysis.

```
# ==============================================================================
# Aerodynamic Analysis of a Wing
# ==============================================================================
# A generic semi-tappered wing is used for this example.
# ==============================================================================
# Aerodynamic Analysis With Tripan Flow Solver
# ==============================================================================
# Part 1 : Importing Standard Python Modules
import os, sys, string, pdb, copy, time, string, re, numpy, datetime
# Set the beginning of the timer for code
t0 = datetime.datetime.now()

# ------------------------------------------------------------------------------
# Part 1.1: Importing External Python Modules and setting of broadcasting variable
from mpi4py import MPI
comm = MPI.COMM_WORLD

# ------------------------------------------------------------------------------
# Part 1.2: Importing Extension modules and initializing 'comm' variables
from mdo_import_helper import *
exec(import_modules('pySpline','tripan','functions'))

# ------------------------------------------------------------------------------
# Part 1.3: Defining the folder for the results output
prefix = './results/'
for arg in sys.argv:
    # Find the prefix from the command line arguments
    m = re.match('(prefix=)(.*)', arg)
    if m:
        prefix = m.group(2)
# create a new directory and broadcast it to everything
if os.path.isdir(prefix):
    i = 1
    while os.path.isdir(os.path.join(prefix,'Aero_Analysis_Num%d'%(i))):
        i = i+1
    prefix = os.path.join(prefix,'Aero_Analysis_Num%d'%(i))
    os.mkdir(prefix)
    prefix = prefix + os.sep
else:
    print 'Prefix is not a directory!'
prefix = MPI.COMM_WORLD.bcast(prefix, root=0)
print 'Using prefix  = %s'%(prefix)

# ==============================================================================
# Part 2: Defining The  Functions to set up the Tripan Object
def setUpTriPanWing(comm, trifile='geo/wing.tripan', wakefile='geo/wing.edge'):
    # Set up TriPan using the files
    ndownstream = 100
    sym_direction = 2 # Use symmetry about the z-axis
    down_dist = 150.0
    time_dependent = 0 # A steady state simulation
    a_wake_dir = numpy.zeros(3)
    b_wake_dir = numpy.zeros(3)
    a_wake_dir[1] = 1.0
    b_wake_dir[2] = 1.0
    # Stretch the wake downstream
    wake_history = tripan.WakeHistory(ndownstream, down_dist,
                                      tripan.WakeHistory.STRETCHED)
    triPan = tripan.TriPanel(comm,
                             trifile, wakefile, wake_history,
                             time_dependent, ndownstream,
                             a_wake_dir, b_wake_dir, sym_direction)
```

```
    npanels = triPan.getNumPanels()
    triPan.setPCSizes(1.5, 150*npanels)
    print 'TriPan panels ', npanels
    return triPan

# ========================================================================
# Part 3: Core of the Script for Aerodynamic Analysis With Tripan Flow Solver
# ------------------------------------------------------------------------
# Part 3.1: Setting Up the Tripan Flow Solver
#Defining The Names for the Tripan Input Files
trifile='geo/wing_50x100.tripan'; wakefile='geo/wing_50x100.wake'
edgefile='geo/wing_50x100.edge';
#Set Up Tripan Object
triPan = setUpTriPanWing(comm, trifile=trifile, wakefile=wakefile)
edgeinfo = tripan.TriPanEdgeInfo( edgefile )
#Set Up Tripan Solver
n_flight_cons = 1
triOpt = tripan.TriPanOpt(triPan, n_flight_cons)

# ------------------------------------------------------------------------
# Part 3.2: Defining The Design Parameters for the Atmosphere properties
Semi_Span = 15/2
# Generic atmospheric conditions for 1000m , 25m/s with MAC as reference lenght
Minf = 0.0743                                   # Imcompressible Mach number
rho  = 1.11164                                  # Air density  kg/m^3
ainf = 336.4379                                 # Speed of sound m/s
alpha = (4.0/180.0)*numpy.pi                    # Angle of attack
Vinf = Minf*ainf                                # Air Speed
Qinf = 0.5*rho*Vinf**2                          # Dynamic Pressure

# ------------------------------------------------------------------------
# Part 3.3: Solving the Aerodynamic System
# Function to get the Wing Area to compute coefficients
area_func = tripan.TriPanProjectedArea()
Area_ref = area_func.evalFunction(triPan)
print 'Area_Tripan = ', Area_ref
# Setting the Load Case, the Wing Angle of Attack and Flight Condition
load_case = 0
alpha_num = 0
fcon = tripan.FlightCondition(rho, Minf, Vinf, alpha,
                              alpha_num, load_case)
triOpt.addFlightCondition(0, fcon)
triOpt.setFlightCondition(0)
#Setting the options for the  Generalized Minimal Residual Method (GMRES) Solver
gmres_iters = 60
max_iters = 5*gmres_iters
triOpt.setGMRESIters(gmres_iters, max_iters)
triOpt.monitor()
# Solve the aerodynamic problem
triOpt.solve()

# ------------------------------------------------------------------------
# Part 3.4: Setting Up the Solution Output Files
# Setting the names for the Output Files
obj_aero_name = prefix+'wing_obj_aero.dat'
tecplot_sol_name = prefix+'wing_tripan_solution.dat'
wake_sol_name = prefix+'wing_wake_file.dat'
load_file_name = 'load_data/wing_aero_load.dat'
lift_dist_name = prefix+'wing_lift_dist.dat'
# Generating Surface solution output
triOpt.writeAeroFile(obj_aero_name)
# Generating .dat Tacplot Visualization files
out_type = 1
triPan.writeSeqTecplotFile(tecplot_sol_name, -1.0, out_type)
triPan.writeWakeFile(wake_sol_name)
# Generating Load data
triPan.writeAeroForceFile(Qinf, load_file_name)
# Generating lift distribution graph
Zloc = numpy.linspace(0.01, Semi_Span)
triPan.writeLiftDistribution(lift_dist_name ,Zloc)
# Evaluation of aerodynamic functions
lift_func = tripan.TriPanLift()
drag_func = tripan.TriPanDrag()
lift = triOpt.evalAeroFunc(lift_func)
drag = triOpt.evalAeroFunc(drag_func)
# Writing some outputs for user visualization
print 'Lift = ', lift*Qinf*2
print 'Drag = ', drag*Qinf*2
print 'CL = ', (lift/Area_ref)*2
print 'CD = ', (drag/Area_ref)*2
final_time = datetime.datetime.now() - t0
print 'Total time spent in the aerodynamic analysis:', final_time

# ========================================================================
```

Appendix C

Script for the Structures Module

Listing C.1: Full script for an structural analysis.

```
# ==============================================================================
# Structural Analysis of a Wing
# ==============================================================================
# A generic semi-tappered wing is used for this example.
# ==============================================================================
# Structural Analysis With TACS
# ==============================================================================
# Part 1 : Importing Standard Python Modules
import os, sys, string, re, numpy, datetime
# Set the beginning of the timer for code
t0 = datetime.datetime.now()

# ------------------------------------------------------------------------------
# Part 1.1: Importing External Python Modules
from mpi4py import MPI
comm = MPI.COMM_WORLD

# ------------------------------------------------------------------------------
# Part 1.2: Importing Extension modules and initializing 'comm' variables
from mdo_import_helper import *
exec(import_modules('TACS', 'elements', 'constitutive', 'functions'))

# ------------------------------------------------------------------------------
# Part 1.3: Defining the folder for the results output
prefix = './results/'
for arg in sys.argv:
    # Find the prefix from the command line arguments
    m = re.match('(prefix=)(.*)', arg)
    if m:
        prefix = m.group(2)
# create a new directory and broadcast it to everything
if os.path.isdir(prefix):
    i = 1
    while os.path.isdir(os.path.join(prefix,'Structural_Analysis_Num%d'%(i))):
        i = i+1
    prefix = os.path.join(prefix,'Structural_Analysis_Num%d'%(i))
    os.mkdir(prefix)
    prefix = prefix + os.sep
else:
    print 'Prefix is not a directory!'
prefix = MPI.COMM_WORLD.bcast(prefix, root=0)
print 'Using prefix  = %s'%(prefix)

# ==============================================================================
# Part 2: Defining a Function to write a tecplot file with the node numbering of structural mesh
def writeNodeInfo(comm):
    if os.path.isfile(prefix + 'tecplot_node_number.dat'):
        print ''
        print 'There is already a Tecplot File with structural mesh nodes.'
        print ''
        return
    else :
        # Create a .dat tecplot file with the nodes localization for the strctural mesh
        nnodes = tacs.getNumNodes()
        Xpts = numpy.zeros(3*nnodes)
        tacs.getNodes(Xpts)
        print numpy.size(Xpts)
        fileName = prefix + 'tecplot_node_number.dat'
        fp = open(fileName, 'w')
        print >>fp, 'TITLE = \"Structural Mesh Nodes"'
        print >>fp, 'VARIABLES = X, Y, Z, Node'
        print >>fp, 'ZONE  T = \"Node numbering\" I = ', nnodes
```

```
        print >>fp, 'DATAPACKING=POINT'
        for i in range(nnodes):
            print >>fp, Xpts[3*i], Xpts[3*i+1], Xpts[3*i+2], i    # i+1
        fp.close()
        print ''
        print 'Tecplot File with structural mesh nodes done!'
        print ''
        return
    return

# ==============================================================================
# Part 3: Core of the Script for Structural Analysis With TACS Structural Solver
# ------------------------------------------------------------------------------
# Part 3.1: Loading the structural mesh
# The name of the structural file to open
bdf_fname = 'mesh/wing.bdf'
# Create the mesh loader class
mesh = TACS.TACSMeshLoader(comm)
mesh.scanBdfFile(bdf_fname)
# Get the number of ribs and spars - relies on a pyLayout-generated file
nribs = mesh.getNumRibs()
nspars = mesh.getNumSpars()
ncomponents = mesh.getNumComponents()
# Setting some outputs for visualization and control
print ''
print 'From .bdf file:'
print 'Number of Components: %d'%(ncomponents)
print 'Number of Ribs: %d'%(nribs)
print 'Number of Spars: %d'%(nspars)
print ''

# ------------------------------------------------------------------------------
# Part 3.2: Setting the domains for KS functions,
#defining the material properties and finite-element type
# Set up the lists that will contain the domains of the KS functions
ks_func_domains = []
for k in xrange(4):
    ks_func_domains.append([])
# The current design variable number
dv_num = 0
# Setting an outputs to list components inported from structural mesh
print ''
print 'List of Components:'
print ''
for k in range(mesh.getNumComponents()):
    # Get the element and component descriptions
    elem_descript = mesh.getElementDescript(k)
    comp_descript = mesh.getComponentDescript(k)
    print 'Component %s is a element of type %s'%(comp_descript, elem_descript)
    # The reference axis used to define the local coordinate system in
    # the shell elements.
        ref_axis = numpy.array([0.0, 0.0, 1.0]) # Reference axis for the spars and skin (z direction)
    # Determine where the element will be located
    if comp_descript[0:3] == 'Top':
        elem_id = 0
        ks_func_domains[0].append(k)
    elif comp_descript[0:3] == 'Bot':
        elem_id = 1
        ks_func_domains[1].append(k)
    elif comp_descript[0:4] == 'Spar':
        elem_id = 2
        ks_func_domains[2].append(k)
    else:
        # The ribs are in the x-y plane - perpendicular to the z-axis
        elem_id = 3
        ks_func_domains[3].append(k)
        ref_axis = numpy.array([1.0, 0.0, 0.0]) # Reference axis for the ribs (x direction)
    # Defining the Design Material Properties - Example uses Aluminium as reference
    # Set up the constitutive relationship (Material Properties)
    rho = 2810.0 # kg/m^3
    E = 70e9      # 70 GPa
    nu = 0.33
    kcorr = 0.8333 # the shear correction factor
    ys = 434.0e6 # 434 MPa
    t = 0.005 # 5 mm skin thickness everywhere
    t_num = dv_num
    t_min = 0.001
    t_max = 0.01
    #Increase the design variable number
    dv_num += 1
    # Create an isotropic material class for the structure.
    # Setting the constitutive class For DShell elements
    con = constitutive.isoFSDTStiffness(rho, E, nu, kcorr,
                                         ys, t, t_num, t_min, t_max)
    # MITC (Mixed interpolation of tensorial components)-based shell element
    elem = elements.MITCShell2(con, elements.MITCShell2.LINEAR, elem_id) #
    #Setting the finite-element
```

```
    mesh.setElement(k, elem)
print ''

# ------------------------------------------------------------------------
# Part 3.3: Creating TACS object and KS functions
# Create the TACSAssembler object on all processors
vars_per_node = 6
nload_cases = 1
tacs = mesh.createTACS(vars_per_node, nload_cases)
# Create the KS functions
load_case = 0
ks_weight = 30.0
ks_funcs = []
for domain in ks_func_domains:
    print domain
    domain = numpy.array(domain, dtype=numpy.intc)
    #Setting SimpleKSFailure KS function failure-load aggregation
    ks = functions.SimpleKSFailure(tacs, ks_weight, load_case)
    # Make the correspondence of the KS domain with the KS function
    mesh.setFunctionDomain(ks, domain)
    ks_funcs.append(ks)

# ------------------------------------------------------------------------
# Part 3.4: Setting design parameters for the load case (structural analysis)
# Create the vectors and matrices that will be used
mat = tacs.createFEMat()
rhs = tacs.createVec()
ans = tacs.createVec()
# Create a load case:
# Point Load:
# Set the force load
force_load = numpy.array([0.0, 500.0, 0.0, 0.0, 0.0, 0.0])
node_load = 1128 #1128 ou 7601
tacs.addPointLoad(load_case, node_load, force_load)
print 'Point load force of %d applied at node %d!' %(force_load[1],node_load)
# Create a set of nodal point-loads - only in the y-direction
#forces = tacs.createVec()
#forces.setMod(1.0, 6, 1) # Set foces[i] = 1.0 if (i%6 == 1)
#forces.applyBCs()

# ------------------------------------------------------------------------
# Part 3.5: Setting up the solver parameters for TACS Structural Solver
# Create the preconditioner - in this case a direct solve
lev = 4500
fill = 10.0
reorder_schur = 1
pc = TACS.PcScMat(mat, lev, fill, reorder_schur)
pc.setMonitorFactorFlag(1)
pc.setAlltoallAssemblyFlag(1)
# Set up the GMRES solver
gmres_iters = 15
nrestart = 5
sflexible = 0
gmres = TACS.GMRES(mat, pc, gmres_iters, nrestart, isflexible)
rank = MPI.COMM_WORLD.rank
freq = 1
gmres.setMonitor(TACS.KSMPrintStdout('GMRES', rank, freq))
# Set the convergence tolerances for GMRES. The test is to see if:
# |R| < rtol*|R_init| or |R| < atol
rtol = 1e-10
atol = 1e-30
gmres.setTolerances(1e-10, 1e-30)

# ------------------------------------------------------------------------
# Part 3.6: Solving the Structural System
# Assemble the stiffness matrix and the residual.
tacs.assembleMat(load_case, mat, rhs)
# Factor the preconditioner
pc.factor()
# Solve the system of equations using GMRES
gmres.solve(rhs, ans)
# Assemble the residual to check the residual norm
tacs.assembleRes(load_case, rhs)
ans.scale(-1.0)
rnorm = rhs.norm()
# Set the variables into TACS
tacs.setVariables(load_case, ans)

# ------------------------------------------------------------------------
# Part 3.7: Setting Up the Solution Output Files
# Create a outout with the numbering of the nodes for the structural mesh
writeNodeInfo(comm)
# Write the vector ans to a binary file
answer_file = prefix + 'answer.bin'
ans.writeToFile(answer_file)
# Write things to an HDF5 file directly
h5 = TACS.TACSToHDF5(tacs)
```

```
for i in xrange(4):
    h5.addGroup('Zone %d'%(i), i, elements.SHELL,
                h5.WRITE_CON |
                h5.WRITE_NODES |
                h5.WRITE_DISPLACEMENTS |
                h5.WRITE_STRESSES |
                h5.WRITE_STRAINS |
                h5.WRITE_EXTRAS)
h5.writeToFile(load_case, prefix+'Wing_Struct_sol%d.h5'%(comm.size),
               'The structural solution')
# ==============================================================================
```

Appendix D

Script for the MDO Framework

Listing D.1: Full script for the Aero-structual Optimization.

```
# ==============================================================================
# Aero-structural Optimization of a Standard Class Competition Glider
# ==============================================================================
# Restricted to a maximum wing-span of 15 metres and fixed wing sections
# (flaps or other lift-enhancing devices not allowed), maximum all-up mass 525 kg.
# ==============================================================================
# Aero-structural Optimization With MDO tool
# ==============================================================================
# Part 1 : Importing Standard Python Modules
import os, sys, string, pdb, copy, time, string, re, numpy, datetime
# Set the beginning of the timer for code
t0 = datetime.datetime.now()

# ------------------------------------------------------------------------------
# Part 1.1: Importing External Python Modules and setting of broadcasting variable
from mpi4py import MPI
try:
    from petsc4py import PETSc
except:
    pass

# ------------------------------------------------------------------------------
# Part 1.2: Importing Extension modules and initializing 'comm' variables
from mdo_import_helper import *
exec(import_modules('pyGeo', 'pySpline', 'pyLayout'))
exec(import_modules('tripan', 'TACS', 'elements', 'constitutive', 'functions'))
exec(import_modules('pyOpt_optimization', 'pySNOPT'))

# ------------------------------------------------------------------------------
# Part 1.3: Defining the folder for the results output
prefix = './results/'
for arg in sys.argv:
    # Find the prefix from the command line arguments
    m = re.match('(prefix=)(.*)', arg)
    if m:
        prefix = m.group(2)
# create a new directory and broadcast it to everything
if os.path.isdir(prefix):
    i = 1
    while os.path.isdir(os.path.join(prefix,'AS_Optimization_Num%d'%(i))):
        i = i+1
    prefix = os.path.join(prefix,'AS_Optimization_Num%d'%(i))
    os.mkdir(prefix)
    prefix = prefix + os.sep
else:
    print 'Prefix is not a directory!'
prefix = MPI.COMM_WORLD.bcast(prefix, root=0)
print 'Using prefix  = %s'%(prefix)

# ==============================================================================
# Part 2: Defining The Auxiliary Functions and Classes
# ------------------------------------------------------------------------------
# Part 2.1: Set up the comparing variables class
class compare_design_vars:
    def __init__(self, dvdict):
        self.dvdict = dvdict
        return
    def compare(self, x, y):
        xarg = numpy.atleast_1d(self.dvdict[x])
        yarg = numpy.atleast_1d(self.dvdict[y])
        return int(xarg[0] - yarg[0])
```

```
# ----------------------------------------------------------------------
# Part 2.2: Set up the function to get the velocity vectors
def getknots(nu, ku, low=0.0, high=1.0):
    tu = numpy.zeros(nu+ku)
    for i in xrange(ku):
        tu[i]          = low
        tu[nu+ku-1-i] = high
    tu[ku-1:nu+1] = numpy.linspace(low,high,nu-ku+2)
    return tu

# ----------------------------------------------------------------------
# Part 2.3: Set up the reference axis for the geometry functions
def set_up_aero_ref_axis(n_fixed=3, n_ref=8, dv_start=0, p_chord=0.25, spanNum=-1):
    twist = True
    scale = True
    # Now read/create the remaining points
    ffd_volumes = []
    nx = 4
    ny = 2
    nz = 16
    Xffd = numpy.zeros((nx, ny, nz))
    Yffd = numpy.zeros((nx, ny, nz))
    Zffd = numpy.zeros((nx, ny, nz))
    for k in xrange(nz):
        for j in xrange(ny):
            for i in xrange(nx):
                Xffd[i,j,k] = -1.0 + (4.0*i)/(nx-1.0)
                Yffd[i,j,k] = -0.5 + (1.0*j)/(ny-1.0)
                Zffd[i,j,k] = (15.0*k)/(nz-1.0)
    # Set up the knot vectors
    kx = numpy.min([4, nx])
    ky = numpy.min([4, ny])
    kz = numpy.min([4, nz])
    Tx = getknots(nx, kx)
    Ty = getknots(ny, ky)
    Tz = getknots(nz, kz)
    # Create the FFD volume for this block
    ffd_volume = elements.R3SplineFFD(int(kx), int(ky), int(kz),
                                      Tx, Ty, Tz,
                                      Xffd, Yffd, Zffd)
    # Append it to the list
    ffd_volumes.append(ffd_volume)
    # The provided coordinates follow the scheme:
    # Y == out the wing
    # X == streamwise
    # Z == up
    Xaxis = numpy.zeros(3*(n_fixed+n_ref))
    Xaxis[2::3] = numpy.linspace(0, 15.0, n_fixed+n_ref)
    # Twist/scale
    twist_nums = -numpy.ones(n_fixed+n_ref, numpy.intc)
    twist_nums[n_fixed:] = numpy.arange(dv_start, dv_start+n_ref,
                                        dtype=numpy.intc)
    dv_start +=n_ref
    scale_nums = -numpy.ones(n_fixed+n_ref, numpy.intc)
    scale_nums[n_fixed:] = numpy.arange(dv_start, dv_start+n_ref,
                                        dtype=numpy.intc)
    span_num = -1    #Fixing Span
    axis = numpy.zeros(3)
    axis[2] = -1.0
    ffd_twist = elements.FFDTwistScale(twist_nums, scale_nums, span_num,
                                       Xaxis, axis)
    if twist:
        # Set the bounds on the twist
        ub_twist = (10.0*numpy.pi)/180.0
        lb_twist = -ub_twist
        lower = lb_twist*numpy.ones(n_fixed+n_ref)
        upper = ub_twist*numpy.ones(n_fixed+n_ref)
        ffd_twist.setTwistBounds(lower, upper)
    if scale:
        # Set the bounds on the scale
        lb_scale=0.2
        ub_scale=2.0
        lower = lb_scale*numpy.ones(n_fixed+n_ref)
        upper = ub_scale*numpy.ones(n_fixed+n_ref)
        ffd_twist.setScaleBounds(lower, upper)
    if span_num >= 0:
        # Set the bounds on the span
        lb_span=0.5
        ub_span=1.5
        ffd_twist.setSpanBounds(lbspan, ubspan)
    for ffd_volume in ffd_volumes:
        ffd_volume.addFFDObject(ffd_twist)
    return ffd_volumes

# ----------------------------------------------------------------------
# Part 2.4:Defining The Function to set up the material class
def set_up_wing_tacs_iso(comm, bdf_name,
```

```
                        dv_num=0, dv_dict={}, ys=370e6,
                        nload_cases=1):
    mesh = TACS.TACSMeshLoader(comm)
    mesh.scanBdfFile(bdf_name)
    nribs = mesh.getNumRibs()
    nspars = mesh.getNumSpars()
    # The empty data for the ribs/spars/skins
    top_skin_con = numpy.empty((nribs-1, nspars-1), numpy.object)
    bot_skin_con = numpy.empty((nribs-1, nspars-1), numpy.object)
    spar_con = numpy.empty((nspars, nribs-1), numpy.object)
    rib_con  = numpy.empty((nribs, nspars-1), numpy.object)
    # The reference positions
    y_axis = numpy.array([0.0, 1.0, 0.0])
    z_axis_top = numpy.array([0.0, 0.0, 1.0])
    z_axis_bot = -z_axis_top
    # Create the ribs
    dcon = create_iso_stiffness(dvnum=-1, t=0.01)
    dcon.setRefAxis(y_axis)
    rib_con[:,:] = dcon
    # Create constitutive classes for the top and bottom skin
    tskin = 0.005
    con_vals = []
    con_dvs  = []
    con_allow = []
    delta_t = 0.010 # 1 mm
    for j in xrange(nspars-1):
        for i in xrange(nribs-1):
            top_skin_con[i,j] = create_iso_stiffness(t=tskin,
                                                     dvnum=dv_num,
                                                     t_min=0.0015, ys=ys)
            top_skin_con[i,j].setRefAxis(z_axis_top)
            dv_dict['Top skin thickness (%d,%d)'%(i,j)] = dv_num
            # Set the constraint up
            if i > 0:
                con_vals.append([1.0, -1.0])
                con_dvs.append([dv_num-1, dv_num])
                con_allow.append(delta_t)
            if j < nspars-1:
                con_vals.append([1.0, -1.0])
                con_dvs.append([dv_num, dv_num + nribs-1])
                con_allow.append(delta_t)
            # Increment the design variable number
            dv_num += 1
    for j in xrange(nspars-1):
        for i in xrange(nribs-1):
            bot_skin_con[i,j] = create_iso_stiffness(t=tskin,
                                                     dvnum=dv_num,
                                                     t_min=0.0015, ys=ys)
            bot_skin_con[i,j].setRefAxis(z_axis_bot)
            dv_dict['Bottom skin thickness (%d,%d)'%(i,j)] = dv_num
            # Set the constraint up
            if i > 0:
                con_vals.append([1.0, -1.0])
                con_dvs.append([dv_num-1, dv_num])
                con_allow.append(delta_t)
            if j < nspars-1:
                con_vals.append([1.0, -1.0])
                con_dvs.append([dv_num, dv_num + nribs-1])
                con_allow.append(delta_t)
            # Increment the design variable number
            dv_num += 1
    # Create new constitutive classes for the spars
    tspar = 0.01
    for i in xrange(nspars):
        for j in xrange(nribs-1):
            spar_con[i,j] = create_iso_stiffness(t=tspar,
                                                 dvnum=dv_num,
                                                 t_min=0.005, ys=ys)
            spar_con[i,j].setRefAxis(z_axis_top)
            dv_dict['Spar thickness (%d,%d)'%(i,j)] = dv_num
            # Set the constraint up
            if j > 0:
                con_vals.append([1.0, -1.0])
                con_dvs.append([dv_num-1, dv_num])
                con_allow.append(delta_t)
            # Increment the design variable number
            dv_num += 1
    # Set the rib stifnesses
    trib =0.008
    for i in xrange(1, nribs):
        rib_stiff = create_iso_stiffness(t=trib,
                                         dvnum=dv_num,
                                         t_min=0.005, ys=ys)
        rib_stiff.setRefAxis(y_axis)
        rib_con[i,:] = rib_stiff
        dv_dict['Rib thickness %d'%(i)] = dv_num
        dv_num += 1
```

```
    ks_domains = []
    for k in xrange(4):
        ks_domains.append([])
    for k in xrange(mesh.getNumComponents()):
        elem_descript = mesh.getElementDescript(k)
        comp_descript = mesh.getComponentDescript(k)
        m = re.match(r'.*\(([0-9]+),([0-9]+)\)', comp_descript)
        if m:
            i = string.atoi(m.group(1))
            j = string.atoi(m.group(2))
        else:
            print 'Component description not understood'
        con = None
        if comp_descript[0:3] == 'Top':
            elem_id = 0
            con = top_skin_con[i,j]
            ks_domains[0].append(k)
        elif comp_descript[0:3] == 'Bot':
            elem_id = 1
            con = bot_skin_con[i,j]
            ks_domains[1].append(k)
        elif comp_descript[0:4] == 'Spar':
            elem_id = 2
            con = spar_con[i,j]
            ks_domains[2].append(k)
        else:
            elem_id = 3
            con = rib_con[i,j]
            ks_domains[3].append(k)
        if elem_descript == 'CQUAD4':
            elem = elements.MITCShell2(con, elements.MITCShell2.LINEAR, elem_id)
        mesh.setElement(k, elem)
    vars_per_node = 6
    num_load_cases = 2
    tacs = mesh.createTACS(vars_per_node, nload_cases)
    # Create the KS functions and set the domains
    ks_list = []
    for load_case in xrange(nload_cases):
        ks_list.append([])
        for domain in ks_domains:
            d = numpy.array(domain, dtype=numpy.intc)
            ks_weight = 30.0
            ks = functions.SimpleKSFailure(tacs, ks_weight, load_case)
            load_factor =  1.0 #1.5
            ks.setLoadFactor(load_factor)
            mesh.setFunctionDomain(ks, domain)
            ks_list[load_case].append(ks)
    # Create the h5 viewer
    h5 = TACS.TACSToHDF5(tacs)
    zone_title = ['top skin', 'bottom skin', 'spars', 'ribs']
    for i in xrange(len(zone_title)):
        h5.addGroup(zone_title[i], i, elements.SHELL,
                    h5.WRITE_CON |
                    h5.WRITE_NODES |
                    h5.WRITE_DISPLACEMENTS |
                    h5.WRITE_STRESSES |
                    h5.WRITE_STRAINS |
                    h5.WRITE_EXTRAS)
    return tacs, h5, ks_list, dv_num, \
        con_vals, con_dvs, con_allow

#-------------------------------------------------------------------------
# Part 2.5:Defining The classes for the aerostructural-optimization
class :
class aerostructure_case:
    dv_num = 0
    dv_dict = {}
    aero_member = False
    structure_member = False
    solver = None
    tacs = None
    triPan = None
    h5_viewer = None
    ks_list = None
    n_flight_cons = 1
    flight_cons = []
    nt_con = 0
    con_vals = None
    con_dvs = None
    con_allow = None
    tacsMap = None
    triMap = None
    Qinf = 0.0
    return

class aerostructure_opt:
    def __init__(self, as_case, fixed_mass, nks):
```

```
        self.as_case = as_case
        self.fixed_mass = fixed_mass
        self.num_design_vars = as_case.dv_num
        self.nks = nks
        self.output_freq = 5
        # Set up the functions
        self.mass_func = None
        self.mass_func = functions.StructuralMass(self.as_case.tacs)
        # Set up the TriPan functions
        self.ks_list = self.as_case.ks_list
        if self.ks_list == None:
            self.ks_list = [None]*self.nks # Create a dummy list
        self.lift_func = tripan.TriPanLift()
        self.drag_func = tripan.TriPanDrag()
        # 1 lift, 1 mass and nks stress KS functions constraints,
        # nt_con structural thickness constraints
        self.ncon = 2 + self.nks + as_case.nt_con
        self.fc_num = 0
        self.g = 9.81 # m/s^2
        self.load_factor = 1.0
        self.con_mat = None
        # Assemble the constraint matrix
        self.con_mat = numpy.zeros((as_case.nt_con, self.num_design_vars))
        # Go through and set the values
        for k in xrange(as_case.nt_con):
            for j in xrange(len(as_case.con_dvs[k])):
                self.con_mat[k, as_case.con_dvs[k][j]] = \
                as_case.con_vals[k][j]
        sroot = self.as_case.structure_root
        self.con_mat = self.as_case.global_comm.bcast(self.con_mat, root=sroot)
        self.write_init_files()
        self.iteration_count = 0
        return

#-------------------------------------------------------------------------
# Part 2.6:Defining The function to write the initial geometry file
    def write_init_files(self):
        file_name = prefix+'obj_aero_init.dat'
        self.as_case.solver.writeAeroFile(file_name)
        file_name = prefix+'obj_tacs_init.h5'
        self.as_case.h5_viewer.writeToFile(0, file_name,
                                           'objective case')
        return
#-------------------------------------------------------------------------
# Part 2.7:Defining The function to compute the objective and constraints
    def obj_con(self, x):
        obj = 0.0
        con = numpy.zeros(self.ncon)
        solver = self.as_case.solver
        solver.setDesignVars(x)
        solver.zeroFlightCondition(self.fc_num)
        solver.setFlightCondition(self.fc_num)
        solver.solveKrylov()
        obj = solver.evalAeroFunc(self.drag_func)
        # Evaluate the lift constraint
        lift = 2.0*solver.evalAeroFunc(self.lift_func)
        wing_mass = 2.0*solver.evalStructureFunc(self.mass_func)
        mass = self.fixed_mass + \
            2.0*solver.evalStructureFunc(self.mass_func)
        con[0] = lift - \
            self.load_factor*((self.g*mass)/self.as_case.Qinf)
        con[1] = mass
        # Evaluate the ks constraints
        for k in xrange(self.nks):
            con[k+2] = \
                solver.evalStructureFunc(self.ks_list[k])
        con[2+self.nks:] = numpy.dot(self.con_mat, x)
        if self.iteration_count % self.output_freq == 0:
            it = self.iteration_count
            file_name = prefix+'obj_aero%03d.dat'%(it)
            solver.writeAeroFile(file_name)
            file_name = prefix+'obj_lift_dist%03d.dat'%(it)
            Zloc = numpy.linspace(0.01, 8.0)
            self.as_case.triPan.writeLiftDistribution(file_name,
                                                      Zloc)
            file_name = prefix+'obj_tacs%03d.h5'%(it)
            self.as_case.h5_viewer.writeToFile(0, file_name,
                                               'objective case')
            # Write the design variables to a file
            fp = open(prefix+'design_vars.dat', 'w')
            for xval in x:
                fp.write('%18.10e\n'%(xval))
            fp.close()
        self.iteration_count += 1
        itnum = self.iteration_count
        print 'Iteration Number = ',itnum
        print''
```

```
        print 'Design variables ', x
        print''
        print 'Total Mass ', mass
        print 'Wing Mass ', wing_mass
        print 'Objective (Drag) ', obj
        print 'Vertical R Constraint ', con[0]
        print 'KS Constraint ', con[1:5]
        print''
        delta_time = datetime.datetime.now() - t0
        print 'Time spent since start:', delta_time
        print 'Evaluation of  objective and constraint : Done!'
        fail = 0
        return obj, con, fail

# ----------------------------------------------------------------------
# Part 2.8:Defining The function to compute the gradients of the objective and constraints
    def gobj_con(self, x, obj, con):
        gobj = numpy.zeros(self.num_design_vars)
        gcon = numpy.zeros((self.ncon, self.num_design_vars))
        solver = self.as_case.solver
        solver.setFlightCondition(self.fc_num)
        solver.setUpAdjoint()
        solver.solveAKAeroFunc(self.drag_func, gobj)
        # Evaluate the gradient of the lift
        glift = numpy.zeros(self.num_design_vars)
        solver.solveAKAeroFunc(self.lift_func, glift)
        glift *= 2.0
        # Evaluate the gradient of the structural mass
        gmass = numpy.zeros(self.num_design_vars)
        solver.evalPartialStructureFunc(self.mass_func, gmass)
        gmass *= 2.0
        gcon[0,:] = glift - \
            self.load_factor*(self.g/self.as_case.Qinf)*gmass
        gcon[1,:] = gmass
        # Evaluate the ks constraints
        for k in xrange(self.nks):
            solver.solveAKStructureFunc(self.ks_list[k], gcon[2+k,:])
        gcon[2+self.nks:,:] = self.con_mat
        gobj = numpy.mat(gobj)
        delta_time = datetime.datetime.now() - t0
        print 'Time spent since start:', delta_time
        print 'Evaluation of  objective and constraint gradients: Done!'
        fail = 0
        return gobj, gcon, fail

# ======================================================================
# Part 3: Core of the Script for Optimization
# ----------------------------------------------------------------------
# Part 3.1: Defining The Design Parameters for the Atmosphere properties (Soaring Circuit Situation)
# An auxiliary geometric parameter for the outputs calculation
Semi_Span = 15/2
MTOW = 430.0 #kg
# Atmosphere properties calculated with relation to the 1976 standard atmosphere up to 230,000 ft.
# Generic atmospheric conditions for 1000m , 25m/s with MAC as reference lenght
# Set the aerodynamic load case information
load_case = 0
Minf = 0.0743                                   # Imcompressible Mach number
rho  = 1.1117                                   # Air density  kg/m^3
ainf = 336.4346                                 # Speed of sound m/s
alpha = (3.0/180.0)*numpy.pi                    # Small Gliding Angle
Vinf = Minf*ainf                                # Air Speed
Qinf = 0.5*rho*Vinf**2                          # Dynamic Pressure

# ----------------------------------------------------------------------
# Part 3.2: Create the aerostructure_case object
as_case = aerostructure_case(MPI.COMM_WORLD)
as_case.Qinf = Qinf

# ----------------------------------------------------------------------
# Part 3.3:Set the aerodynamic load case information
# ----------------------------------
load_case = 0
alpha = 3.0/180.0 * numpy.pi
fcon = tripan.FlightCondition(rho, Minf, Vinf, alpha,
                              as_case.dv_num, load_case)
alpha_low  = -4.0/180.0 * numpy.pi
alpha_high = 7.0/180.0 * numpy.pi
fcon.setAlphaBounds(alpha_low, alpha_high)
as_case.dv_dict['Angle of attack'] = as_case.dv_num
as_case.dv_num += 1
as_case.flight_cons.append(fcon)

# ----------------------------------------------------------------------
# Part 3.4: Set up the FFD volume with twist
n_ref = 4   # Number of twist-variable ref-axis points
as_case.dv_dict['Twist angles'] = range(as_case.dv_num,
                                        as_case.dv_num+n_ref)
```

```
as_case.dv_num += n_ref
as_case.dv_dict['Scale Factor'] = range(as_case.dv_num,
                                        as_case.dv_num+n_ref)
ffd_volumes = set_up_wing_ref_axis(n_ref=n_ref,
                                   dv_start=as_case.dv_num-n_ref,
                                   p_chord=0.25)
as_case.dv_num += n_ref
for ffd in ffd_volumes:
    ffd.printTecplotFile('ffd_vol.dat')

# ------------------------------------------------------------------
# Part 3.5: Set up the structure
bdf_name = 'mesh/wing.bdf'
tacs, h5_viewer, ks_list_of_lists, dv_num, \
      con_vals, con_dvs, con_allow = \
       set_up_wing_tacs_iso(as_case.local_comm, bdf_name,
                            dv_num=as_case.dv_num,
                            dv_dict=as_case.dv_dict, ys=434e6,
                            nload_cases=1)
tacsMap = TACS.TACSParametricNodeMap(len(ffd_volumes))
for k in xrange(len(ffd_volumes)):
    tacsMap.addMap(k, ffd_volumes[k])
# Assign the appropriate values to the as_case
as_case.dv_num = dv_num
as_case.tacs = tacs
as_case.h5_viewer = h5_viewer
as_case.ks_list = ks_list_of_lists[0]
as_case.nt_con = len(con_dvs)
as_case.tacsMap = tacsMap
as_case.con_vals = con_vals
as_case.con_dvs = con_dvs
as_case.con_allow = con_allow
for ks in as_case.ks_list:
    ks.setLoadFactor(1.0)

# ------------------------------------------------------------------
# Part 3.6: Set up the aerodynamic problem
tripan_file = 'geo/wing_35x60.tripan'
wake_file   = 'geo/wing_35x60.wake'
# Set up TriPan using the files
ndownstream = 60
sym_direction = 2 # Use symmetry about the z-axis
down_dist = 150.0
time_dependent = 0
a_wake_dir = numpy.zeros(3)
b_wake_dir = numpy.zeros(3)
a_wake_dir[1] = 1.0
b_wake_dir[2] = 1.0
# Stretch the wake downstream
wake_history = tripan.WakeHistory(ndownstream, down_dist,
                                  tripan.WakeHistory.STRETCHED)
triPan = tripan.TriPanel(as_case.local_comm,
                         tripan_file, wake_file, wake_history,
                         time_dependent, ndownstream,
                         a_wake_dir, b_wake_dir, sym_direction)
npanels = triPan.getNumPanels()
triPan.setPCSizes(1.5, 150*npanels)
as_case.triPan = triPan

# ------------------------------------------------------------------
# Part 3.7: Set up the aero-structural solver
rank = as_case.global_comm.rank
transfer = tripan.setUpLDTransfer(as_case.global_comm,
                                  as_case.structure_root,
                                  as_case.aero_root,
                                  as_case.local_comm,
                                  as_case.triPan, as_case.tacs,
                                  prefix+'rigid_links%d.dat'%(rank))
# Create the aero-structural object
solver = tripan.TACSTriPan(as_case.global_comm,
                           as_case.triPan, as_case.tacs, transfer,
                           as_case.n_flight_cons)
for k in xrange(as_case.n_flight_cons):
    solver.addFlightCondition(k, as_case.flight_cons[k])
triMap = tripan.TriPanMap(as_case.triPan, len(ffd_volumes))
for k in xrange(len(ffd_volumes)):
    triMap.addMap(k, ffd_volumes[k])
triMap.projectPoints()
for k in xrange(as_case.n_flight_cons):
    solver.setTriPanMap(k, triMap)
as_case.triMap = triMap
for k in xrange(as_case.n_flight_cons):
solver.setTACSMap(k, as_case.tacsMap)
as_case.solver = solver

# ------------------------------------------------------------------
# Part 3.8:  Data for setting up the solver options
```

```
# Monitor the residuals and the Krylov residuals
resmonitor = 1
kmonitor   = 0
nkmonitor  = 0
solver.setMonitor(resmonitor, kmonitor, nkmonitor)
tmonitor = 1
solver.setTimeMonitor(tmonitor)
levFill = 25
fill    = 15.0
solver.setStructurePcFill(levFill, fill)
# Set the structure options
iters   = 50
inner   = 5
restart = 0
solver.setStructureGMRESIters(iters, inner, 0)
max_iters = 16
solver.setMaxGSIters(max_iters)
# Approximate the derivative of the residual w.r.t. the nodal locations
use_pc_approx = 1
solver.setNKUsePCApproximation(use_pc_approx)
# Set the aerodynamic parameters
max_iters = 15
solver.setAeroGMRESIters(max_iters)
# Set the Newton-Krylov parameters
formXptMat = 1
xptMatFreq = 8 # Recalculate the XptMat every n iterations
aeropconly = 0
tacspconly = 0
solver.setNKOptions(formXptMat, xptMatFreq, aeropconly, tacspconly)
iters    = 60
nrestart = 0
solver.setNKGMRESIters(iters, nrestart)
rtol = 1e-4
atol = 1e-30
solver.setNKGMRESTol(rtol, atol)
nks = 0
if as_case.ks_list != None:
    nks = len(as_case.ks_list)
nks = as_case.global_comm.bcast(nks, root=0)

# ------------------------------------------------------------------
# Part 3.9: Set up the optimization problem
# Create the optimization object
fixed_mass = 0.50*MTOW
as_opt = aerostructure_opt(as_case, fixed_mass, nks)
ndvs = as_case.dv_num
xvals = numpy.zeros(ndvs)
xlower = numpy.zeros(ndvs)
xupper = numpy.zeros(ndvs)
xlower[:] = -1e20
xupper[:] = 1e20
# Get the variable values
# Need to do something to uniquify the design variables
as_case.solver.getDesignVars(xvals)
as_case.solver.getDesignVarRange(xlower, xupper)
# Set up the optimization problem
opt_prob = Optimization('Drag minimization', as_opt.obj_con)
# Add the objective
opt_prob.addObj('Drag')
# Add the constraints
opt_prob.addCon('Lift con', lower=0.0, upper=0.0)
opt_prob.addCon('Mass con', lower=0.0, upper=525.0)
for i in xrange(nks):
    opt_prob.addCon('KS %d'%(i), lower=0.5, upper=1.0)
# Add the thickness constraints
for k in xrange(as_case.nt_con):
    opt_prob.addCon('t con %d'%(k), lower=-as_case.con_allow[k],
                    upper=as_case.con_allow[k])
keys = as_case.dv_dict.keys()
dvcmp = compare_design_vars(as_case.dv_dict)
keys.sort(cmp=dvcmp.compare)
# Add the variables
for key in keys:
    a = numpy.atleast_1d(as_case.dv_dict[key])
    if len(a) == 1:
        opt_prob.addVar(key, 'c', value=xvals[a[0]],
                        lower=xlower[a[0]], upper=xupper[a[0]])
    else:
        for i in xrange(len(a)):
            opt_prob.addVar(key + ' %3d'%(i+1), 'c', value=xvals[a[i]],
                            lower=xlower[a[i]], upper=xupper[a[i]])
# Set up the optimizer
opt = SNOPT()
opt_tol = 5e-5
con_tol = 5e-5
opt.setOption('Nonderivative linesearch')
opt.setOption('Major optimality tolerance', opt_tol)
```

```
opt.setOption('Major feasibility tolerance', con_tol)
opt.setOption('Major step limit', 0.1)
opt.setOption('Iterations limit', 100000)
opt.setOption('Major iterations limit', 10000)
opt.setOption('Minor iterations limit', 10000)
opt.setOption('Print file', prefix+'SNOPT_print.out')
opt.setOption('Summary file', prefix+'SNOPT_summary.out')
print opt_prob
opt(opt_prob, as_opt.gobj_con)

# ------------------------------------------------------------------------
# Part 3.10: Setting Up the Solution Output Files
# Setting the names for the Output Files
lift_dist_name = prefix+'wing_opt_lift_dist.dat'
opt_sol_name = prefix+'Wing_Opt_Solution.out'
tacs_sol_name = prefix+'wing_opt_tacs_solution.h5'
# Generating Surface solution output
opt_prob.write2file(outfile=opt_sol_name,disp_sols='True')
# Generating .dat Tacplot Visualization files
out_type = 1
triPan.writeSeqTecplotFile(tecplot_sol_name, -1.0, out_type)
# Generating lift distribution graph
Zloc = numpy.linspace(0.01, Semi_span)
triPan.writeLiftDistribution(lift_dist_name,Zloc)
as_case.h5_viewer.writeToFile(0, tacs_sol_name,'objective case')
final_time = datetime.datetime.now() - t0
print 'Total time spent in the aerodynamic optimization:', final_time
# ========================================================================
```

Bibliography

Abbott, I. and Doenhoff, A. V. (1945). Characteristics of airfoil sections. Technical report, NACA Report 824.

Abbott, I. and Doenhoff, A. V. (1949). *Theory of Wing Sections*. Dover Publications.

Abdo, M. and Samareh, J. (2005). Optimization of a business jet. *Canadian Aeronautcs and Space Institute 52 Annual General Meeting and Conference.* Toronto.

Aerospace Design Laboratory of SU (2010). Official website for the aerospace design laboratory of Stanford University. `http://adl.stanford.edu/`. Accessed September 2011.

Ajmera, H., Mujumdar, P., and Sudhakar, K. (2004). MDO architectures for coupled aerodynamic and structural optimization of a flexible wing. In *45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, pages 1–2.

Alexandrov, N., Hussaini, M., for Computer Applications in Science, I., Engineering, and Center, L. R. (1997). *Multidisciplinary design optimization: state of the art*. Proceedings in Applied Mathematics Series ; No. 80. Society for Industrial and Applied Mathematics.

Alexandrov, N. and Kodiyalam, S. (1998). Initial results of an MDO method evaluation study. In Paper, A., editor, *Proceedings of the Seventh AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 98-4977, St. Louis MO. AIAA. pp. 1315-1327.

Alonso, J. (2011). AA241 aircraft design: Synthesis and analysis. `http://adg.stanford.edu/aa241/AircraftDesign.html`. Accessed December 2011.

Alonso, J. J., LeGresley, P., Van Der Weide, E., Martins, J. R. R. A. M., and Reuther, J. J. (2004). pyMDO: A framework for high-fidelity multi-disciplinary optimization. In *Proceedings of the 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, pages 2004–4480.

Anderson, J. D. (2001). *Fundamentals of Aerodynamics*. McGraw-Hill, third edition.

Balay, S., Buschelman, K., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., and Zhang, H. (2004). PETSc users manual. Technical Report ANL-95/11 - Revision 2.3.0, Argonne National Laboratory.

Balling, R. and Wilkinson, C. (1996). Execution of multidisciplinary design optimization approaches on common test problems. *AIAA Paper 96-4033*.

Barcelos, M., Bavestrello, H., and Maute, K. (2006). A Schur-Newton-Krylov solver for steady-state aeroelastic analysis and design sensitivity analysis. In *Computer Methods in Applied Mechanics and Engineering*, volume 195, pages 2050–2069.

Barcelos, M. and Maute, K. (2008). Aeroelastic design optimization for laminar and turbulent flows. In *Computer Methods in Applied Mechanics and Engineering*, volume 197, pages 1813–1832.

Biros, G. and Ghattas, O. (2005a). Parallel Lagrange–Newton–Krylov–Schur methods for PDE-constrained optimization. Part i: The krylov–schur solver. In *SIAM Journal on Scientific Computing*, volume 27, pages 687–713.

Biros, G. and Ghattas, O. (2005b). Parallel Lagrange–Newton–Krylov–Schur methods for PDE-constrained optimization. Part ii: The Lagrange–Newton solver and its application to optimal control of steady viscous flows. In *SIAM Journal on Scientific Computing*, volume 27, pages 714–739.

Braun, R. (1996). *Collaborative optimization: an architecture for large-scale distributed design*. Ph.d. thesis, Stanford University.

Braun, R., Gage, P., Kroo, I., and Sobieszczanski-Sobieski, J. (1996). Implementation and performance issues in collaborative optimization. In Paper, A., editor, *Proceedings 5th AIAA/USAF MDO symposium*, 96-4017, Bellevue WA. AIAA.

Brown, S. (1997). Displacement extrapolation for CFD+CSM aeroelastic analysis. *AIAA Paper 97-1090*.

Chapelle, D., Oliveira, D., and Bucalem, M. (2003). MITC elements for a classical shell model. *Comput. & Structures*, 81:523–533.

Chen, S., Zhang, F., and Khalid, M. (2002). Evaluation of three decomposition MDO algorithms. In *Proceedings of 23rd International Congress of Aerospace Sciences*, Toronto Canada. AIAA.

Chittick, I. R. and Martins, J. R. R. A. (2007). Aero-structural optimization using adjoint coupled post-optimality sensitivities. *Structures and Multidisciplinary Optimization*.

Corke, T. C. (2003). *Design of Aircraft*. Pearson Education, Inc.

Cramer (1994). Problem formulation for multidisciplinary optimization. In *SIAM Journal of Optimization*, 4, pages 754–776.

Cramer, E., Dennis, J., Frank, P., Lewis, R., and Shubin, G. (1993). Problem formulation for multidisciplinary optimization. Technical report, Center for Research on Parallel Computation Rice University.

Culick, F. E. C. and Jex, H. R. (1985). Aerodynamics, stability and control of the 1903 Wright Flyer. In *Proceedings of The Wright Flyer : An Engineering Perspective. AIAA Wright Flyer Project Report (WF 84/09-1)*.

Dennis, J. and Lewis, R. (1994). Problem formulations and other optimization issues in multidisciplinary optimization. In Paper, A., editor, *AIAA Symposium on Fluid Dynamics*, 94-2196, Colorado Springs. AIAA.

Federal Aviation Administration (2003). *Glider Flight Handbook*. Federal Aviation Administration, U.S. Department Of Transportation.

Felippa, C. A., Park, K. C., and Farhat, C. (2001). Partitioned analysis of coupled mechanical systems. In *Computer Methods in Applied Mechanics and Engineering*, volume 190, pages 3247–3270.

Floreano, D. and Mattiussi, C. (2008). *Bio-inspired artificial intelligence: theories, methods, and technologies*. Intelligent robotics and autonomous agents. MIT Press.

Gill, P. E. (2008). *User's Guide for SNOPT Version 7: Software for Large-Scale Nonlinear Programming*. Department of Mathematics of University of California, San Diego, La Jolla.

Gill, P. E., Murray, W., and Wright, M. H. (1982). *Practical Optimization*. Academic Press. ISBN: 0122839528.

Heil, M., Hazel, A., and Boyle, J. (2008). Solvers for large-displacement fluidstructure interaction problems: segregated versus monolithic approaches. In *Computational Mechanics*, volume 43, pages 91–101.

Hess, J. L. and Smith, A. (1967). Calculation of potential flow about arbitrary bodies. *Progress in Aerospace Sciences*, 8:1–138.

Hicken, J. and Zingg, D. (2010). A simplified and flexible variant of GCROT for solving nonsymmetric linear systems. *SIAM Journal on Scientific Computing*, 32(3):1672–1694.

Houard, G. and Peslin, Y. (1943). Jean-Marie Le Bris, marin français précurseur du vol à voile. *Pages d'Aviation, Société d'Édition Aéronautique*, pages 22–40.

Hulme, K. and Bloebaum, C. (1998). A comparison of solution strategies for simulation-based multidisciplinary design optimization. In Paper, A., editor, *Proceedings Seventh AIAA/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 98-4977, St. Louis MO. AIAA. pp. 2143-2153.

Isaacs, A., Sudhakar, K., and Mujumdar, P. M. (2003). Design and development of MDO framework. In *MSO-DMES 2003 Conference Paper*, 78.

Kafyeke, F., Abdo, M., Pepin, F., Piperni, P., and Laurendeau, E. (2002). Challenges of aircraft design integration. *RTO AVT Symposium on "Reduction of Military Vehicle Acquisition Time and Cost through Advanced Modelling and Virtual Simulation*.

Kennedy, G. J. (2011). Aerostructural design optimization of composite aircraft with stress and local buckling constraints using an implicit structural parametrization.

Kennedy, G. J. and Martins, J. R. R. A. (2010). Parallel solution methods for aerostructural analysis and design optimization. *13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*.

Kenway, G. K., Kennedy, G. J., and Martins, J. R. R. A. (2010). CAD-free approach to high-fidelity aerostructural optimization. *13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, pages 1–3.

Kim, J., Aluru, N., and Tortorelli, D. (2003). Improved multi-level newton solvers for fully coupled multi-physics problems. In *International Journal for Numerical Methods in Engineering*, volume 58, pages 463–480.

Kroo, I. (2008). AA222 introduction to multidisciplinary design optimization. `http://adg.stanford.edu/aa222`. Accessed December 2011.

LET Aircraft Industries (2011). *L-23 Super-Blanik Sailplane Maintenance Manual*. Portuguese Air Force.

Liebeck, R. H. (2004). Design of the blended wing body subsonic transport. *Journal of Aircraft*, 41(1):10–25.

Mader, C. A., Kenway, G. K., and Martins, J. R. R. A. (2008). Towards high-fidelity aerostructural optimization using a coupled ADjoint approach. In Paper, A., editor, *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2008-5968, Victoria British Columbia Canada. AIAA.

Maman, N. and Farhat, C. (1995). Matching fluid and structure meshes for aeroelastic computations: a parallel approach. In *Computers and Structures*, volume 54, pages 779–785.

Martins, J. R. R. A. (2002). *A coupled-adjoint method for high-fidelity aero-structural optimization*. Ph.d. thesis, Stanford University, Stanford.

Martins, J. R. R. A., Alonso, J. J., and Reuther, J. J. (2002). High-fidelity aero-structural design optimization of a supersonic business jet. *Journal of Aircraft*, 41:2004.

Martins, J. R. R. A., Alonso, J. J., and Reuther, J. J. (2005). A coupled–adjoint sensitivity analysis method for high–fidelity aero–structural design. In *Optimization and Engineering*, volume 6, pages 33–62.

Martins, J. R. R. A., Peterson, P., and Alonso, J. (2001). Fortran to Python interface generator with an application to aerospace engineering. In *Proceedings of the 9th International Python Conference*.

Martins, J. R. R. A., Sturdza, P., and Alonso, J. (2003). The Complex-Step derivative approximation. In *ACM Transactions on Mathematical Software*, volume 29, page 245–262.

Martins, J. R. R. A. and Tedford, N. P. (2006). On the common structure of MDO problems: A comparison of architectures. In *11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, AIAA 2006-7080.

Maute, K. and Allen, M. (2004). Conceptual design of aeroelastic structures by topology optimization. In *Structural and multi-disciplinary optimization*, volume 27, pages 27–42.

Maute, K., Nikbay, M., and C.Farhat (2001). Coupled analytical sensitivity analysis and optimization of three-dimensional nonlinear aeroelastic systems. In *AIAA Journal*, volume 39, pages 2051–2061.

MDO Laboratory of UoM (2010). Official website for the multidisciplinary design optimization laboratory of University of Michigan . `http://mdolab.engin.umich.edu/`. Accessed September 2011.

MDO Laboratory of UoT (2000). Official website for the multidisciplinary optimization laboratory of University of Toronto. http://mdolab.utias.utoronto.ca/. Accessed September 2011.

MDO Technical Commite, editor (1991). *Current State of the Art in Multidisciplinary Design Optimization*, volume White Paper. AIAA, Reston VA.

Megson, T. (2007). *Aircraft Structures for Engineering Students*. Butterworth-Heinemann.

M.G.Dodson (2005). An historical and applied aerodynamic study of the Wright brothers wind tunnel test program and application to successful manned flight. Technical report usna-334, US Naval Academy.

Morgado, J. A. and de Sousa, J. T. B. (2009). PITVANT. http://www.academiafa.edu.pt/conteudos/investigacao/O%20PROGRAMA%20DE%20INVEST%20E%20TECNOLOGIA%20EM%20VA.pdf. Accessed October 2011.

Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*. Springer.

Park, G.-J. (2007). *Analytic methods in design practice*. Springer.

Perez, R., Liu, H. H. T., and Behdinan, K. (2004). Evaluation of multidisciplinary optimization approaches for aircraft conceptual design. In *AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Albany, NY.

Perez, R. E., Jansen, P. W., and Martins, J. R. (2011). pyOpt: A Python-based object-oriented framework for nonlinear constrained optimization. *Structures and Multidisciplinary Optimization*.

Portuguese Air Force (2009). Official Portuguese Air Force website. http://www.emfa.pt/www/index.php?fsh=1. Accessed August 2011.

Portuguese Air Force (2010). PERSEUS. http://www.academiafa.edu.pt/index.php?bd0b6f49=011.005.002&lang=PT. Accessed October 2011.

Portuguese Air Force Academy (2011). *L-23 Super-Blanik Sailplane Flight Manual*. Portuguese Air Force.

Reuther, J. J., Alonso, J. J., Martins, J. R. R. A., and Smith, S. C. (1999). A coupled aero-structural optimization method for complete aircraft configurations. *AIAA paper 99-0187*.

Roake, J. (2005). Gliding membership report. Technical report, FAI Gliding Commission (IGC).

Saad, Y. and H.Schultz, M. (1986). GMRES: A generalized minimal residual algorithm for solving non-symmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869.

Samareh, J. A. (1999). A novel shape parameterization approach.

Samareh, J. A. (2001). Survey of shape parameterization techniques for high-fidelity multidisciplinary shape optimization. *AIAA Journal*, 39:877–884.

Schmitt, V. and Charpin, F. (1979). Pressure distributions on the ONERA-M6-wing at transonic mach numbers. *Experimental Data Base for Computer Program Assessment. Report of the Fluid Dynamics Panel Working Group 04, AGARD AR 138*.

Sederberg, T. W. and Parry, S. R. (1986). Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph.*, 20:151–160.

Shin, M. K. and G.J.Park (2005). Multidisciplinary design optimization based on independent subspaces. *Journal for Numerical Methods in Engineering*, 64, Issue 5:599–617.

Sitek, M. A. and Blunt, F. L. V. (1940). *Gliding and Soaring*. Alliance Press, first edition.

Slater, J. W. (2008). ONERA M6 wing: Study 1. http://www.grc.nasa.gov/WWW/wind/valid/m6wing/m6wing01/m6wing01.html. Accessed December 2011.

Smith, S. C. (1996). A computational and experimental study of nonlinear aspects of induced drag. Technical report, National Aeronautics and Space Administration.

Snyman, J. (2005). *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*. Applied optimization. Springer. ISBN: 9780387243481.

Soaring Society of America (2005-10). Official website for the Soaring Society of America. `http://www.ssa.org/`. Accessed November 2011.

Sobieszczanski-Sobieski, J. (1988). Optimization by decomposition: A step from hierarchic to non-hierarchic systems. Technical report, NASA Technical Report CP-3031.

Sobieszczanski-Sobieski, J. (1993). Multidisciplinary design optimization: An emerging new engineering discipline. In *Proceeding of the World Congress of Optimal of Structural Systems*.

Sobieszczanski-Sobieski, J. and Haftka, R. T. (1997). Multidiscpinary aerospace design optimization:Survery of recent developments. *Structural Optimization*, 14:1–23.

Sobieszczanski-Sobieski, J., Jeremy, S. A., and Robert, R. S. J. (1998). Bi-level integrated system synthesis (BLISS). In *National Aeronautics and Space Administration Technical Manual*, NASA/TM-1998-208715. AIAA.

Tedford, N. (2006). Comparison of MDO architectures within a universal framework. Master's thesis, University of Toronto.

Tedford, N. P. and Martins, J. R. R. A. (2009). Benchmarking multidisciplinary design optimization algorithms. *Optimization and Engineering*, 11(1):159–183.

Turner, M. J., Clough, R. W., Martin, H. C., and Topp, L. P. (1956). Stiffness and deflection analysis of complex structures. *J. Aeronautical Society*, 23:856–869.

Van Der Weide, E., Kalitzinand, G., J.Schliter, and Alonso, J. J. (2006). Unsteady turbomachinery computations using massively parallel platforms. In Paper, A., editor, *44th AIAA Aerospace Sciences Meeting and Exhibit*, 2006-421, Reno Nevada. AIAA.

Wakayama, S. and Kroo, I. (1998). The challenge and promise of blended wing body optimization. *AIAA Paper 98-4736*.

Weide, B. E. V. D., Kalitzin, G., and Schluter, J. (2005). On large scale turbomachinery computations. *CTR Annual Research*, (1981):139–150. Accessed December 2011.

Welch, A. (1980). *The Story of Gliding*. John Murray, second edition.

White, J. L. T. (1961). Eilmer of Malmesbury, an eleventh century aviator: A case study of technological innovation, its context and tradition. In *Technology and Culture*, volume 2, pages 97–111.

Wrenn, G. (1989). An indirect method for numerical optimization using the Kreisselmeier-Steinhauser function. Nasa technical report cr-4220, National Aeronautics and Space Administration.

Yi, S. I., Shin, J. K., and Park, G. J. (2007). Comparison of MDO methods with mathematical examples. *Structural and Multidisciplinary Optimization*.