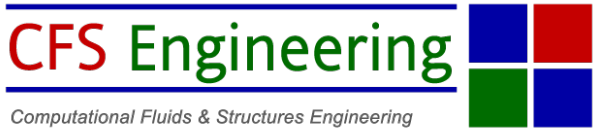
Imperial College London



MSc Major Individual Project

Imperial College London

Department of Aeronautics

Aero-elastic tool implemented in a preliminary aircraft design program suite

Author:
Jean-Philippe Kuntzer

Supervisor:
Dr. Jan Vos
Mr. Aidan Jungo

Second Marker:
Prof. Rafael Palacios

September 23, 2020

Abstract

The objective of this project was to build a versatile static aeroelastic tool that can be easily integrated into the software suite CEASIOMpy used for preliminary aircraft design. To meet this objective the partitioned loosely coupled approach is used, which means that separated solvers are used to solve the aerodynamics and the structural mechanics parts of the problem. However, this approach leads to the the problem of mismatched meshes. This problem has been solved during this master thesis by building the solver Aeroframe 2.0 that is based on the theory of virtual work.

The aeroelastic (also called Fluid Structure Interaction (FSI)) simulations were made using state of the art validated solvers for both the fluid and the structure problem. The computational fluid dynamics (CFD) solvers used are the well known high fidelity SU2 solver and the vortex lattice method (VLM) solver PyTornado. Concerning the low fidelity three dimensional beam model, the finite element solver FramAT was chosen.

Aeroframe 2.0 is part of a software suite, developed in the framework of the European project AGILE 4.0. AGILE regrouping many aeronautics actors that are using each other strengths to push science forward. Working with many actors opened the opportunity to use an external structural finite element solver. This led to a function into Aeroframe 2.0 capable of linking input files generated from a NASTRAN solver used by DLR, which is one of the actors of the AGILE 4.0 project.

Coupling two CFD solvers and two finite element structural solvers, all having different mesh types, led to a complicated task. The SU2 solver works with a three dimensional grid, PyTornado with a surface of panels, FramAT uses three dimensional beam elements, and finally the external solver uses three dimensional structural elements. As mentioned above, to overcome this challenge the principle of virtual work was used.

Finally the first steps towards validation were made. Comparison of the results obtained from the analytical solution for a straight wing and the solution obtained by Aeroframe 2.0 was done. Both SU2 and PyTornado gave satisfactory results with the finite element solver FramAT. No validation was made for the combination of SU2 with the external solver and PyTornado with the external solver. Only functionality tests were made and are not presented in this report.

Acknowledgements

The work presented here was carried out at CFS Engineering in Lausanne, Switzerland, in the period from May to September 2020.

I would like to thank my parents for the unconditional support that they gave me during all these years, I will never say it enough. I would like also to thank Dr. Jan Vos and Mr. Aidan Jungo for opening up the doors of the knowledge and science marvel that CFS Engineering is. Mr. Alain Gehri for sharing his many years of experience with aeroelasticity. To Mr. Fancesco Torregionni for his simulations at DLR. Professor Rafael Palacios from Imperial College London for his pastoral care all along the project. Finally, last but not least I would like to thank Professor Flavio Noca and Mr. Sergio Marques, for opening once more the doors of that little marvel of technology that are HEPIA, the ChaChaCha lab and Windshape.

The work presented in this report has been performed in the framework of the AGILE 4.0 project (TowardsCyber-physical Collaborative Aircraft Development) and has received funding from the European Union Horizon 2020 Programme under grant agreement nr°815122.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

With today's ever growing computer power, combined with the continuous increase in network speed and bandwidth, aircraft design is seeing a new paradigm: Industry 4.0. Knowledge sharing and close collaborations between different companies located in different countries is more than ever a reality.

These changes open the door to new design methods, relying increasingly on numerical simulation and data transfer. The tool developed in this thesis embraces this paradigm by being versatile enough to be used with high and low fidelity solvers, offering the possibility to be part of aeroelastic simulations where the aerodynamic and structural calculations are made at physically different locations.

The work carried out in this thesis was made in the frame of the EU funded H2020 project AGILE4.0 [1] [2] that is focused on Cyber-physical Collaborative Aircraft Development, and the aeroelastic tool developed in this thesis will become part of the CEASIOMpy software suite for aircraft design. Note that the AGILE4.0 project employs the common language for aircraft design, abbreviated in the rest of the document by CPACS [3]. This file format encapsulates in a single file all main aircraft features.

Flying vehicles must be optimised from many point of views. Weight, flight controls, structure, manufacturing and maintenance constraints, aerodynamic loads, noise, propulsion. All these parameters matter with their own importance depending on what the aircraft is designed for. For this reason, deformation during flight due to the simple fact of flying must be taken into account when designing an aircraft. This science is called aeroelasticity.

Aeroelasticity [4] shows in the first chapter the engineering approach of aircraft design, using Collar's triangle to show this optimum/trade-off idea. Taking all these forces and constraints into account leads to a flexible mechanical assembly which then leads to different phenomena depending on the flying state of the vehicle.

Knowledge of the state at which unwanted phenomena including divergence, flutter, or control reversal appear is essential for a safe aircraft flight. Simulation has become a necessity to design an aircraft that will avoid these situations in all flying conditions. Furthermore, certification norms have clear directives on the safety factors required to avoid these phenomena, and that an aircraft should comply with in order to be allowed to fly. Finally, knowledge of the deformation of the aircraft can help the engineers to take the best design decisions in order to optimise any parameter of their choice related to aeroelasticity.

1.2 Objectives

The objective of this project is to develop a python based aeroelastic tool for conceptual aircraft design.

The first requirement is that the tool should be able to make static aeroelastic simulations in order for the user to be able to determine the deformed geometry of the aircraft at a given flight state.

The second requirement is to use existing CFD solvers SU2 [5] and vortex lattice method PyTornado [6]. Regarding the structure part of the simulation, the software should link the CFD solvers to structure solver FramAT [7] and be able to take a .csv deformation file being the output file of a high fidelity structural simulation software. This should permit to link the CFD solvers to NASTRAN simulations computed at the Deutches Zentrum für luft-und Raumfahrt (DLR).

The third requirement is to stay as close as possible to the information given by the input file while using the FramAT solver. To achieve this objective, a structure discretisation function, called mesher or meshing tool in the rest of the document, needs to be developed. Furthermore principle of virtual work permit to achieve energy conservation and at the same time information transfer between aerodynamics and structure following a physical conservative law.

The second requirements implies that different solvers are used for the CFD and for the structure, and this might lead to a geometrical mismatch between the grids used for the CFD and for the structural solver. This leads to the requirement that the tool should be able to handle this for all combinations of the above mentioned solvers.

Chapter 2

Theoretical background

As mentioned in the introduction, this project does not require to build a vortex lattice method (VLM) solver nor a higher fidelity CFD solver, but theoretical knowledge of them is essential. Mapping mesh points between the computational fluid dynamics (CFD) solver on one side, and the computational structural dynamics (CSD) solver on the other side requires a general understanding of the underlying theories.

2.1 Computational fluid dynamics

2.1.1 Background

As the reader certainly knows, fluid dynamics is governed by the Navier-Stokes equations. The following will be a humble summary to get the big picture of how to go from the Navier-Stokes equations up to a Vortex lattice method.

If the reader feels that he needs a deeper dive into the proofs I strongly recommend to read Flight Vehicle Aerodynamics by Mark Drela [8] or/and Low speed aerodynamics by Joseph Katz and Allen Plotkin [9]. Both books tend to complement each other. Drela [8] being a little more visual and easy to read, while Katz and Plotkin [9] is more suitable for someone that enjoys the beauty and the rigorousness of mathematics. The following is inspired by both books.

Furthermore there is an excellent course on edx [10] taught by Michael Burton, Mark Drela, Alejandra Uranga that is based on Drela's book [8]

2.1.2 The singularity method

Starting from the Reynolds transport theorem, one can derive the Navier-Stokes equation as it is shown in [9]. By non-dimensionalizing the equation with the free stream values the following equation is obtained:

$$\Omega\frac{\partial u^*}{\partial t^*}+u^*\frac{\partial u^*}{\partial x}+u^*\frac{\partial u^*}{\partial y}+u^*\frac{\partial u^*}{\partial z}=\left(\frac{1}{Fr^2}\right)f_x^*-Eu\frac{\partial p^*}{\partial x^*}+\left(\frac{1}{Re}\right)\left(\frac{\partial^2 u^*}{\partial x^{*2}}+\frac{\partial^2 u^*}{\partial y^{*2}}+\frac{\partial^2 u^*}{\partial z^{*2}}\right) \quad (2.1)$$

The Strouhal(Ω), Froude (Fr), Euler (Eu) and Reynolds (Re) numbers have appeared and act as coefficients in front of each term of the Navier-Stokes equation. In the context of this project this equation is solved to simulate the behaviour of flying vehicles. This means that the Reynolds number is easily around a million or more, and as a first approximation we assume that the last term of (2.1) can be neglected for the calculations discussed in this thesis, and the Navier Stokes equation reduces to the Euler equation.

With this simplification, the flow field is now described by the incompressible continuity equation and the Euler equations. Notice that when the SU2 solver is used for the aeroelastic simulations discussed in this master thesis, equations (2.2) and (2.3) are solved. The rest of the demonstration concerns only the in-house solver PyTornado.

$$\nabla \boldsymbol{v} = 0 \tag{2.2}$$

$$\frac{\partial \boldsymbol{v}}{\partial t} + \boldsymbol{v} \cdot \nabla \boldsymbol{v} = \boldsymbol{f} - \frac{\nabla p}{\rho} \tag{2.3}$$

For the following demonstration, the dot product rule and the cross product rule will be needed.

$$\nabla(\boldsymbol{a} \cdot \boldsymbol{b}) = \boldsymbol{a} \cdot \nabla \boldsymbol{b} + \boldsymbol{b} \cdot \nabla \boldsymbol{a} + \boldsymbol{a} \times (\nabla \times \boldsymbol{b}) + \boldsymbol{b} \times (\nabla \times \boldsymbol{a}) \tag{2.4}$$

$$\nabla \times (\boldsymbol{a} \times \boldsymbol{b}) = \boldsymbol{a} \nabla \cdot \boldsymbol{b} - \boldsymbol{b} \nabla \cdot \boldsymbol{a} - \boldsymbol{a} \cdot \nabla \boldsymbol{b} \tag{2.5}$$

By setting $\boldsymbol{a} = \boldsymbol{v}$ and $\boldsymbol{b} = \boldsymbol{v}$ in equation (2.4), the following result is obtained:

$$\frac{1}{2} \nabla(\boldsymbol{v} \cdot \boldsymbol{v}) = \boldsymbol{v} \cdot \boldsymbol{v} + \boldsymbol{v} \times \boldsymbol{\omega} \tag{2.6}$$

Knowing that the definition of the vorticity in a flow is:

$$\boldsymbol{\omega} = \nabla \times \boldsymbol{v} \tag{2.7}$$

and substitution of equation (2.6) into (2.3) one obtains:

$$\frac{\partial \boldsymbol{v}}{\partial t} + \frac{1}{2} \nabla(\boldsymbol{v} \cdot \boldsymbol{v}) - \boldsymbol{v} \times \boldsymbol{\omega} = \boldsymbol{f} - \frac{\nabla p}{\rho} \tag{2.8}$$

Taking the curl of (2.8) leads to:

$$\frac{\partial \boldsymbol{w}}{\partial t} - \nabla \times (\boldsymbol{v} \times \boldsymbol{\omega}) = -\nabla \left(\frac{1}{\rho} \right) \times \nabla p \tag{2.9}$$

In the case of an aircraft or a flying vehicle, the only external force acting on the body is the gravity. The gravitational field is assumed to be constant, and computing the curl of this field is equivalent to delete it.

Concerning the left hand side (LHS) term, we have to keep in mind that the hypothesis of incompressible adiabatic flow was made in the beginning. This implies that the pressure gradient is aligned with the density gradient, and this argument forces the LHS term to be zero.

Applying the cross rule (2.5) to (2.9) simplifies the RHS term. This result can be transformed by using the scalar transport equation. At this point equation (2.6), becomes the vorticity transport equation, without the viscous term which was already neglected beforehand.

$$\frac{D\boldsymbol{\omega}}{Dt} = \boldsymbol{\omega} \cdot \nabla \boldsymbol{v} \tag{2.10}$$

Drela [8], see page 18, goes a bit deeper in the explanation, but the basic intuition is that the hypothesis of an irrotational incoming flow sets to zero the contribution of the RHS term of (2.10):

$$\frac{D\boldsymbol{\omega}}{Dt} = 0 \tag{2.11}$$

The system of equations described by (2.2) and (2.3) has now become:

$$\nabla \cdot \boldsymbol{v} = 0 \tag{2.12}$$

$$\nabla \times \boldsymbol{v} = 0 \tag{2.13}$$

Which represents the potential flow equations. It is now possible to start the derivation of the equations for the vortex lattice method.

To solve the potential flow equations boundary conditions are needed. The far-field boundary condition corresponds to the speed of the aircraft and is written as:

$$\boldsymbol{v_\infty} = u_\infty \boldsymbol{e_x} + v_\infty \boldsymbol{e_y} + w_\infty \boldsymbol{e_z} \tag{2.14}$$

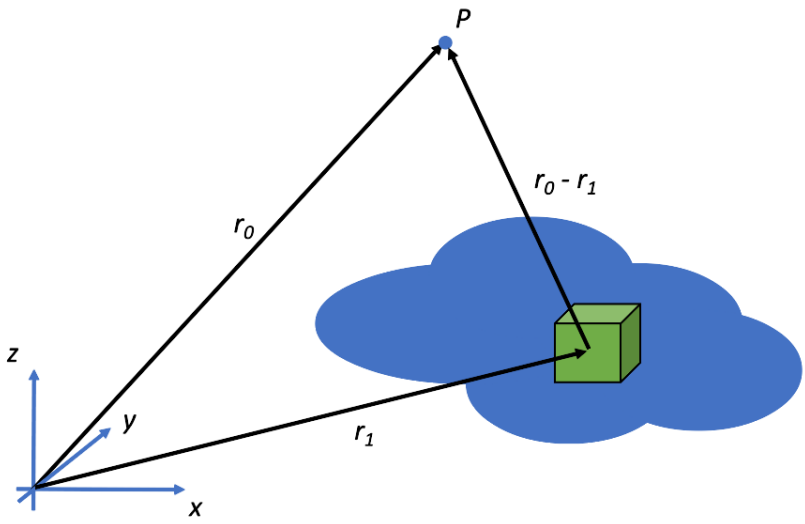


Figure 2.1: Velocity influence at point P

Where $\boldsymbol{e}_*$ is the unit vector in the $*$ direction and all three speed components are selected by the user. Since the user can choose what to impose at the boundary, it is possible to express the flow as the resulting field deformed by vortices. The user can then also select their intensity and their position in space. By doing so the velocity vector is defined as:

$$\boldsymbol{v} = \nabla \times \boldsymbol{B} \tag{2.15}$$

Where $\boldsymbol{B}$ represents a vector field. This leads towards a new definition of the vorticity.

$$\begin{aligned} \boldsymbol{w} &= \nabla \times \boldsymbol{v} \\ &= \nabla \times (\nabla \times \boldsymbol{B}) \\ &= \nabla(\nabla \cdot \boldsymbol{B}) - \nabla^2 \boldsymbol{B} \end{aligned} \tag{2.16}$$

By definition we impose $\boldsymbol{B}$ to follow the subsequent condition:

$$\nabla \cdot \boldsymbol{B} = 0 \tag{2.17}$$

Which gives a Poisson type of equation for the vorticity.

$$\boldsymbol{w} = -\nabla^2 \boldsymbol{B} \tag{2.18}$$

With the help of Green theorem we can find a solution to the user imposed velocity field in relation to the vorticity. (2.18)

$$\boldsymbol{B} = \frac{1}{4\pi} \int_V \frac{\boldsymbol{w}}{|\boldsymbol{r_0} - \boldsymbol{r_1}|} dV \tag{2.19}$$

And injecting this result into the definition of the velocity potential (2.15) we get:

$$\boldsymbol{v} = \frac{1}{4\pi} \int_V \nabla \times \frac{\boldsymbol{w}}{|\boldsymbol{r_0} - \boldsymbol{r_1}|} dV \tag{2.20}$$

By developing the internal terms of equation (2.15), one can find an equation which is equivalent to the Biot-Savart law. A detailed explanation on the formal proof is found in [9], page 36-38.

$$\boldsymbol{v} = \frac{1}{4\pi} \int_V \boldsymbol{w} \times \frac{(\boldsymbol{r_0} - \boldsymbol{r_1})}{|\boldsymbol{r_0} - \boldsymbol{r_1}|^3} dV \tag{2.21}$$

At this point the reader can understand why there is a quadratic relation between the distance of a vortex and its influence over the flow field. The reader can also understand that the velocity at

a given point is the sum of all the contribution of vorticity infinitesimal volumes.

The sum of all infinitesimal volumes gives a vorticity volume. The vorticity volume can be lumped down to a vortex sheet by integrating the infinitesimal vortexes strength over the height of the volume. This process gives an equivalent sheet that is called a vortex sheet. The velocity potential is then given by:

$$\boldsymbol{v} = \frac{1}{4\pi} \iint \boldsymbol{\gamma} \times \frac{(\boldsymbol{r_0} - \boldsymbol{r_1})}{|\boldsymbol{r_0} - \boldsymbol{r_1}|^3} ds\, dl \tag{2.22}$$

with γ defined as:

$$\gamma = \int_{h_1}^{h_2} \boldsymbol{w}\, dh \tag{2.23}$$

To further simplify the equation, we can lump the sheet down to a filament by integrating over the width. This step leads towards the following result:

$$\boldsymbol{v} = \frac{1}{4\pi} \int \Gamma \times \frac{(\boldsymbol{r_0} - \boldsymbol{r_1})}{|\boldsymbol{r_0} - \boldsymbol{r_1}|^3} dl \tag{2.24}$$

with Γ defined as:

$$\Gamma = \int_{s_1}^{s_2} \boldsymbol{\gamma}\, ds \tag{2.25}$$

Let's now talk about the airplane speed in the Earth-reference frame. The airplane speed is the sum of its ground-speed $\boldsymbol{u}$, and its rotation speed $\boldsymbol{\Omega} \times \boldsymbol{r}$. This can be translated in maths by:

$$\boldsymbol{v_a} = \boldsymbol{u} + \boldsymbol{\Omega} \times \boldsymbol{r} \tag{2.26}$$

The aircraft is modifying the airflow where it is. Since we have argued that we can add some vortex filament as we wish in order to simulate the presence of the aircraft the new local speed of the flow takes the following expression:

$$\boldsymbol{v}(\boldsymbol{r}) = \boldsymbol{v}_\gamma - (\boldsymbol{u} + \boldsymbol{\Omega} \times \boldsymbol{r}) \tag{2.27}$$

To describe the flow we now need to impose $\boldsymbol{v_\gamma}$, which was defined by equation (2.24). Since the surface of an aircraft is not simple we need cut it in small parts, called panels. Each of these panels represent a horseshoe vortex (HSV). We will compute its strength Γ to simulate the aircraft effect on the flow-field. This is not obvious at this stage, but it will soon become clearer. The flow is now described by the equations below. The summation comes from the fact that the airplane was discretized in small panels, and the total effect of the airplane on the flowfield is the summation of all of them.

$$\begin{aligned} \boldsymbol{v}(\boldsymbol{r}) &= \sum_{HSV} \frac{\Gamma_i}{4\pi} \int \frac{(\boldsymbol{r_0} - \boldsymbol{r_1})}{|\boldsymbol{r_0} - \boldsymbol{r_1}|^3} dl - (\boldsymbol{u} + \boldsymbol{\Omega} \times \boldsymbol{r}) \\ &= \sum_{HSV} \Gamma \boldsymbol{v}_i(\boldsymbol{r}) - (\boldsymbol{u} + \boldsymbol{\Omega} \times \boldsymbol{r}) \end{aligned} \tag{2.28}$$

The $\boldsymbol{v_i(r)}$ term once evaluated becomes equation (6.33) in [8]. Now a subtlety from the implementation in PyTornado comes. The vortex legs in [8] are aligned with the x axis. In Pytornado the assumption of aligning the vortex legs with the incoming flow was done. This leads to a simplified version of equation (6.33) in [8] which is:

$$\boldsymbol{v_i(r)} = \frac{1}{4\pi} \frac{\boldsymbol{a} \times \boldsymbol{b}}{\|\boldsymbol{a}\|\|\boldsymbol{b}\|(1+\epsilon) + \boldsymbol{a} \cdot \boldsymbol{b}} \left(\frac{1}{\|a\|} + \frac{1}{\|b\|} \right) \tag{2.29}$$

The ϵ is added to avoid a division by zero. After some tests, it looks like this value is not necessary anymore, but in an older version of PyTornado it was needed, hence the addition in the equation. Note that the index i denotes the panel, and j will be the index of the collocation point a panel. It is the opposite in [8].

We now need to find the vortex strength Γ_i for each panel. This is done by imposing the velocity normal to the panel to 0. The normal vector of the panel will be oriented normal to the camber

line of the wing. It is important to notice that the wing is lumped on the chordline, but the vector is normal to the camberline of the wing. So for each panel we need to find:

$$\boldsymbol{V}(\boldsymbol{r_j^c}) \cdot \boldsymbol{n_{0j}} = \left(\sum_{i=1}^{N} \Gamma_i \boldsymbol{v_i}(\boldsymbol{r_j^c}) - (\boldsymbol{U} + \boldsymbol{\Omega} \times \boldsymbol{r_j^c}) \right) \cdot \boldsymbol{n_{0j}} = 0 \tag{2.30}$$

We can then separate the left part and the right part of the central term of (2.30) to obtain:

$$\sum_{i=1}^{N} \Gamma_i \boldsymbol{v_i}(\boldsymbol{r_j^c}) \cdot \boldsymbol{n_{0j}} = (\boldsymbol{U} + \boldsymbol{\Omega} \times \boldsymbol{r_j^c}) \cdot \boldsymbol{n_{0j}} \tag{2.31}$$

At this point the system of equations is a $N \times N$ matrix system that can be computed by using the following equation.

$$\boldsymbol{v_i}(\boldsymbol{r_j^c}) \cdot \boldsymbol{n_{0j}} \Gamma_i = \overline{U}_x(\boldsymbol{e_x} \cdot \boldsymbol{n_{0i}}) + \overline{U}_y(\boldsymbol{e_y} \cdot \boldsymbol{n_{0i}}) + \overline{U}_z(\boldsymbol{e_z} \cdot \boldsymbol{n_{0i}}) + \overline{\Omega}_x(\boldsymbol{e_x} \cdot \boldsymbol{n_{0i}}) + \overline{\Omega}_y(\boldsymbol{e_x} \cdot \boldsymbol{n_{0i}}) + \overline{\Omega}_z(\boldsymbol{e_x} \cdot \boldsymbol{n_{0i}}) \tag{2.32}$$

This system of equations can further be simplified to the famous $Ax = b$ system of equations with

$$A_{ij} = \boldsymbol{v_i}(\boldsymbol{r_j^c}) \cdot \boldsymbol{n_{0j}} \tag{2.33}$$

$$x_i = \Gamma_i \tag{2.34}$$

$$b_i = \overline{U}_x(\boldsymbol{e_x} \cdot \boldsymbol{n_{0i}}) + \overline{U}_y(\boldsymbol{e_y} \cdot \boldsymbol{n_{0i}}) + \overline{U}_z(\boldsymbol{e_z} \cdot \boldsymbol{n_{0i}}) + \overline{\Omega}_x(\boldsymbol{e_x} \cdot \boldsymbol{n_{0i}}) + \overline{\Omega}_y(\boldsymbol{e_x} \cdot \boldsymbol{n_{0i}}) + \overline{\Omega}_z(\boldsymbol{e_x} \cdot \boldsymbol{n_{0i}}) \tag{2.35}$$

At this point the expensive computation part is the inversion of an $N \times N$ system of a fully populated matrix. From this one can find the velocity at point j.

$$\overline{\boldsymbol{V}}_j = \sum_{i=1}^{N} \Gamma_i \boldsymbol{v_i}(\boldsymbol{r_j}) - \overline{\boldsymbol{U}} - \overline{\boldsymbol{\Omega}} \times \boldsymbol{r_i} \tag{2.36}$$

Having computed the velocity one can find the force acting on this panel by using the following equation:

$$\boldsymbol{F_j} = \rho \overline{\boldsymbol{V}}_j \times \boldsymbol{l_j} \Gamma_i \tag{2.37}$$

Where $\boldsymbol{l_j}$ is the leading edge leg length.

2.2 Analytical static aeroelasticity of straight wings

Static aeroelasticity is the science that computes an aircraft behaviour under static loads. These loads could represent the aircraft during a steady level flight or during a static maneuver, for example a turn or a dive. Analytical solutions for a wing flying in these states are obtained by assimilating the wing mechanical behaviour to the one of an equivalent beam. The beam is then subjected to the aerodynamic span-wise lift distribution and the aerodynamic span-wise torque distribution.

2.2.1 Bending of straight wings

An aircraft is always subject to four forces: lift, drag, propulsion and gravity. Since the drag is generally small, and that the propulsion is generated to overcome it, these two forces will be neglected in the following. Doing a local sum of the remaining forces acting on the wing yields to:

$$F_z(y) = L(y) - nm(y)g \tag{2.38}$$

With n the load factor:

$$n = \frac{L}{W} = \frac{g + a_z}{g} = 1 + \frac{a_z}{g} \tag{2.39}$$

Where a_z is the vertical acceleration of the aircraft, $W = mg$ is the total weight of the aircraft, and g the gravity constant equal to: $9.81 m/s^2$.

The Euler-Bernoulli beam theory states that the relationship between the displacement and the applied external forces is:

$$\frac{d^2}{dy^2}\left(EI(y)\frac{d^2w}{dy^2}\right) = q(y) \tag{2.40}$$

$$q_l(y) = L(y) - nm(y)g \tag{2.41}$$

The governing equation for the displacement is then:

$$\frac{d^2}{dy^2}\left(EI(y)\frac{d^2w}{dy^2}\right) = L(y) - nm(y)g \tag{2.42}$$

Common boundary conditions are, as stated in Imperiall College Aeroelasticity lecture nodes [11]:

1. Clamped end: $w(l) = 0$ and $\frac{dw(l)}{dy} = 0$
2. Free end: $\frac{d^2w(l)}{dy^2} = 0$ and $\frac{d}{dy}\left(EI(y)\frac{d^2w(l)}{dy^2}\right) = 0$
3. Elastic constraint: $EI\frac{d^2w(l)}{dy^2} + k_e\frac{dw}{dy} = 0$ and $\frac{d}{dy}\left(EI(y)\frac{d^2w(l)}{dy^2}\right) + k_e w(l) = 0$

Where k_e is the external linkage rigidity.

2.2.2 Torsion of straight wings

The following is inspired by Pr. Palacios lecture nodes [11] second part chapter 1, and the book by Hodge, Introduction to structural dynamics and aeroelasticity [12] page 140 to 142. In the following explanation, the span-wise direction is considered to be the positive y-axis, and the leading edge up pitch as a positive rotation.

As stated in last section, a wing is subjected to aerodynamics forces and moments. Since the wing is represented by a beam and the rotation angle of the wing can vary in the span-wise direction (wing twist), the generated aerodynamic moment has to be expressed locally and is:

$$M(y) = M_{ac}(y) + eL(y) - nm(y)gd \tag{2.43}$$

Where $L(y)$ the local generated lift, M_{ac} the local aerodynamic moment, and m the local linear wing mass. The distance between the elastic axis and the aerodynamic center is called e, and the distance between the mass axis and the elastic axis is called d in the above equation. The load factor was already defined (2.39).

As a reminder the aerodynamic lift and moment, and the dynamic pressure are respectively defined as:

$$L(y) = qcC_l(y) \tag{2.44}$$

$$M_{ac}(y) = qc^2C_{mac}(y) \tag{2.45}$$

$$q = \frac{1}{2}\rho_\infty U_\infty^2 \tag{2.46}$$

From the beam theory, the torsion along the beam can be expressed as:

$$\int_y^{y_{\max}} M(y)dy = GJ(y)\frac{d\theta}{dy} \tag{2.47}$$

Where $M(y)$ is given by equation 2.43, G is the torsion modulus, J the polar second moment of area, and θ the pitch angle on the wing. As a reminder J is defined as:

$$J(y) = I_x(y) + I_z(y) \tag{2.48}$$

Generally it is easier to find or compute I_x and I_z which are the planar second moments of area.

To go further in the understanding of the analytical study of the beam the rest of the demonstration can be found in [12] or [11], but it is not necessary in the context of this report. In the specific case of this thesis, lift and moment functions are extrapolated form the CFD analysis using an order seven polynomial fit. Hence the following equation is of interest:

$$\int_{y}^{y_{\max}} (M_{ac}(y) + eL(y) - Nm(y)gd)dy = GJ(y)\frac{d\theta}{dy} \tag{2.49}$$

This equations can be solved after defining suitable boundary conditions. For the aircraft wing, possible boundary conditions are given by [11]:

1. Clamped end: $\theta(l) = 0$
2. Free end: $\frac{d\theta(l)}{dy} = 0$
3. Elastic constraint: $GJ\frac{d\theta(l)}{dy} + k_{ext}\theta(l) = 0$

Where k_{ext} is the stiffness of the external component linked to the wing.

For ease of solving equation (2.49), the assumption of a straight wing, under a constant torque can be made. This permits to have a second order ODE with constants coefficients. To obtain this equation, one should use (2.44), (2.45) and (2.46).

$$\frac{d^2\theta}{dy^2} + \frac{qcae}{GJ}\theta = -\frac{1}{GJ}(qc^2C_{mac} + qcae\alpha_r - nmgd) \tag{2.50}$$

With α_r being the root angle of attack. Using a clamped root, and a free end boundary condition, the rotation distribution becomes:

$$\theta(y) = (\alpha_r + \overline{\alpha}_r)[\tan(\lambda l)\sin(\lambda y) + \cos(\lambda y) - 1] \tag{2.51}$$

With

$$\lambda^2 = \frac{qcae}{GJ} \tag{2.52}$$

and

$$\overline{\alpha}_r = \frac{cC_{mac}}{ae} + \frac{nmgd}{qcae} \tag{2.53}$$

The local lift can be expressed as:

$$L(y) = q_l ca(\alpha_r + \theta(y)) \tag{2.54}$$

Recall that the local lift coefficient is defined as:

$$c_l(y) = a(\alpha_r + \theta(y)) \tag{2.55}$$

A solution and an example of this problem is given in section 4.2.5.

2.2.3 Bending and Torsion of swept wings

First let's remind ourselves the definition of the dynamic pressure.

$$q_\infty = \frac{1}{2}\rho v_\infty^2 \tag{2.56}$$

This time the wing is swept and hence there is an angle between the y-axis which is perpendicular to the free-stream velocity direction and the elastic axis of the wing. The angle is called the sweep angle and is generally assigned to Λ.

The lift being computed for a profile perpendicular to the wing main axis, the speed needs to be tilted by Λ yielding the following result:

$$q = \frac{1}{2}\rho v_\infty^2 \cos^2(\Lambda) \tag{2.57}$$

Or as used in [11] part 2 chapter 3:

$$q = q_\infty \cos^2(\Lambda) \tag{2.58}$$

This leads to the local load definition:

$$q_l(y) = q_\infty \cos^2(\Lambda) c C_l(y) - nm(y)g \tag{2.59}$$

and the local torque is then defied as:

$$M'(y) = q_\infty \cos^2(\Lambda) e c C_l(y) + q_\infty \cos^2(\Lambda) c^2 C_{MAC} + nm(y)gd \tag{2.60}$$

In order to find and analytical solution for the wing deformation, both equations for $q(y)$ and $M'(y)$, need to be fed respectively to (2.40) and (2.47). These newly found equation are linked together by $\theta(y)$ which is inside the $C_l(y)$ coefficient. In our case both equations stay uncoupled since the rotation angle is found by a Taylor approximation of the CFD result. Furthermore, we can recognise that $p_\infty \cos^2(\Lambda) c C_l(y) = L(y)$. Taking a Taylor expansion of the lift found by the CFD computation simplifies the equation by a non negligible amount as explained before.

2.3 Radial basis functions

The choice was made to populate the transformation matrices linking the computational fluid dynamics and the structural mechanics solver by using radial basis functions. Radial basis functions are functions that have the following property:

$$\phi(\boldsymbol{x}) = \phi(\|\boldsymbol{x}\|) \tag{2.61}$$

The interesting property of this function is that its value only depends on the distance. In the definition (2.61) the function is centered at the origin. We can also impose the function to be centered on any point in space $\boldsymbol{c}$ by imposing that the function is defined as:

$$\phi(\boldsymbol{x}) = \phi(\|\boldsymbol{x} - \boldsymbol{c}\|) \tag{2.62}$$

There are many functions that satisfy this condition, some of the most used are summarized in the table 2.1. In this table r is the euclidean norm $r = \sqrt{x^2 + y^2 + z^2}$ of the input vector x, and ϵ is a tuning parameter defined by the user.

Function name	Definition
Polyharmonic spline	$\phi(r) = r^k$ for $k = 1, 3, 5, ..$
Polyharmonic spline	$\phi(r) = r^k \ln(r)$, for $k = 2, 4, 6, ...$
Gaussian	$\phi(r) = \exp -(\epsilon r)^2$
Multiquadric	$\phi(r) = \sqrt{1 + (\epsilon r)^2}$
Inverse multiquadric	$\phi(r) = \frac{1}{\sqrt{1+(\epsilon r)^2}}$
Wendland C0	$\phi(r) = (1 - r)^2$
Wendland C2	$\phi(r) = (1 - r)^4(4r + 1)$

Table 2.1: Commonly used radial basis functions

Some of the above functions are shown in figure 2.2.

2.4 Principle of virtual work for non matching meshes

In loosely coupled fluid-structure interaction simulations, there is a solver which takes care of the fluid solution and another solver taking care of the structural solution. Both solvers are independent, the input for the fluid solver is the fluid state and the geometry of the aircraft, and the input for the structural solver are the external forces and moments, and also the aircraft geometry.

In both cases the aircraft geometry is given in a mesh format. Care needs to be taken when trying to pass the calculated fluid forces to the structure solver and the calculated displacements back to

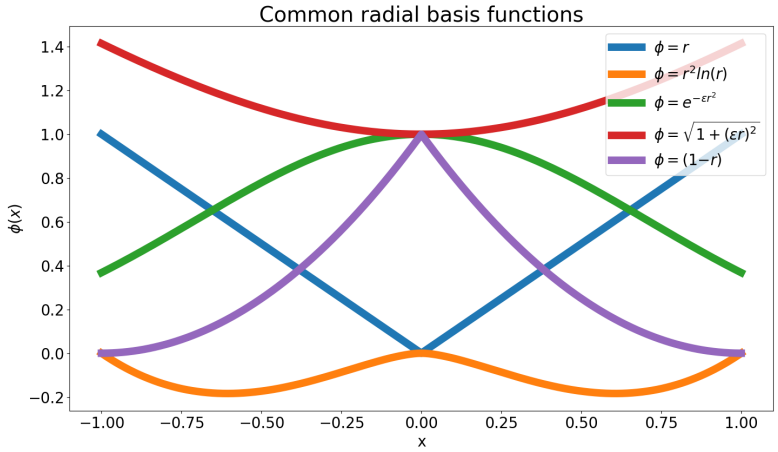


Figure 2.2: First order polyharmonic, etc.

the fluid solver. As explained before, both solvers do not know the presence of the other and hence have separated non matching meshes. To get a physical quantity conserved during the transfer process between the solvers one can choose to conserve virtual work, or geometry [13].

All the different available information exchange techniques have their pros and cons. Geometry conservation leads to better CFD mesh having smoother surfaces. This is done because the points of the NURBS (Non-Uniform Rational B-Spline) definition of the geometry in the CAD program leads to surfaces defined by functions and not by a mesh as a collection of point as in an .stl file for example. On the contrary, conserving virtual work will lead to the conservation of a physical quantity as is it shown by [14] [13] [15] and many others.

The principle of virtual work implies that there is a mapping function between the CFD mesh and the CSD mesh. This mapping function can be a Finite Interpolation Method (FIM) function, or a Radial Basis Function (RBF). Radial basis functions have the benefit of being of growing or decaying type. The decaying ones are of interest here, since they are exact on the mapping point but decay with distance. Furthermore they are not dependent of the input mesh type. The only thing that they need to be found is a set of points. Let's now dive into the theory of the implementation of the virtual work method for aeroelastictiy.

The radial basis function interpolation is generally defined as follows.

$$s(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i \phi(\|\boldsymbol{x} - \boldsymbol{x_i}\|) + p(\boldsymbol{x}) \tag{2.63}$$

Where $p(\boldsymbol{x})$ is a polynomial, $\phi(x)$ is a RBF, $\boldsymbol{x}$ is a point in space that we would like to interpolate and $\boldsymbol{x_i}$ is the i-th point that is taken as reference for the interpolation. The reader can understand x_i as an aerodynamic mesh point and x as a structural mesh point. In this case, for simplicity the polynomial is dropped. This means that the interpolation is now defined by:

$$s(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i \phi(\|\boldsymbol{x} - \boldsymbol{x_i}\|) \tag{2.64}$$

Principle of virtual work for a beam is defined as such:

$$\delta W_a = \Delta \boldsymbol{x_a^T} \cdot \boldsymbol{f_a} + \Delta \boldsymbol{\theta_a^T} \cdot \boldsymbol{m_a} = \Delta \boldsymbol{x_s^T} \cdot \boldsymbol{f_s} + \Delta \boldsymbol{\theta_s^T} \cdot \boldsymbol{m_s} = \delta W_s \tag{2.65}$$

Where $\Delta \boldsymbol{x_a^T}$ is the displacement vector of all the aerodynamic mesh points a part of the airplane. $\boldsymbol{f_a}$ is the force vector for all the aerodynamic mesh points of the same part. $\Delta \boldsymbol{\theta_a^T}$ is the rotation form the elastic axis of all the aerodynamic mesh points of the part of the airplane and $\boldsymbol{M_a}$ is the moment developed by each force of each point of the aerodynamic mesh on the elastic axis of the

airplane part. The s subscript stands for the structural mechanics mesh of the beam describing the structural mechanics behaviour of this aircraft part.

2.4.1 Forces conservation

Let's compute the influence of the force and the influence of the torque on a beam.
The displacement due to the forces is mapped by having

$$\Delta \boldsymbol{x_s} = \sum \alpha_j \phi(\|\boldsymbol{x_s} - \boldsymbol{x_{s_j}}\|) \tag{2.66}$$

that can be written in matrix form as:

$$\begin{pmatrix} \Delta x_{s1} \\ \vdots \\ \Delta x_{sN} \end{pmatrix} = \begin{pmatrix} \phi_{s1s1} & \cdots & \phi_{s1sN} \\ \vdots & \ddots & \vdots \\ \phi_{s1sN} & \cdots & \phi_{sNsN} \end{pmatrix} \begin{pmatrix} \Delta \alpha_{s1} \\ \vdots \\ \Delta \alpha_{sN} \end{pmatrix} \tag{2.67}$$

This can be transformed into the following system with $\boldsymbol{M}$ being an $N \times N$ matrix, N being the number of points into the structural mesh.

$$\Delta \boldsymbol{x_s} = \boldsymbol{M}\boldsymbol{\alpha} \tag{2.68}$$

From equation (2.68), one can find the $\boldsymbol{\alpha}$ vector.

$$\boldsymbol{\alpha} = \boldsymbol{M}^{-1}\boldsymbol{\Delta x} \tag{2.69}$$

Since the displacement of the structural mesh is assumed to be the same for the aerodynamic mesh, due to the condition of conservation of the virtual work, we can define the aerodynamic mesh displacement by:

$$\Delta \boldsymbol{x_a} = \sum \alpha_j \phi(\|\boldsymbol{x_a} - \boldsymbol{x_{s_j}}\|) \tag{2.70}$$

The α coefficients were computed using the structural mesh. Note that the structural mesh has generally fewer points than the aerodynamic mesh. Furthermore we only need the displacement of the external points of the structure mesh, in order to have a good mapping. If it is not too complicated to get only those, the size of the $\boldsymbol{M}$ matrix could be significantly reduced. This is interesting since the limiting factor of virtual work conservation by a mapping with radial basis functions is bottle-necked by the inversion of this matrix. Today's computer having always bigger computational power and it is less of an issue, but the user have to keep this in mind if he wants to implement high fidelity aeroelasticity.

Let's now introduce the A matrix.

$$\boldsymbol{A} = \begin{pmatrix} \phi_{a1s1} & \cdots & \phi_{a1sN} \\ \vdots & \ddots & \vdots \\ \phi_{a1sN} & \cdots & \phi_{aMsN} \end{pmatrix} \tag{2.71}$$

By the conservation of the virtual work due to forces we get:

$$\delta W_{a_f} = \Delta \boldsymbol{x_a}^T \boldsymbol{f_a} = \Delta \boldsymbol{x_s}^T \boldsymbol{f_s} = \delta W_{s_f} \tag{2.72}$$

This leads to the transformation equation.

$$\Delta \boldsymbol{x_a} = \boldsymbol{A}\boldsymbol{M}^{-1}\Delta \boldsymbol{x_s} \tag{2.73}$$

That can be further simplified to

$$\Delta \boldsymbol{x_a} = \boldsymbol{H}\Delta \boldsymbol{x_s} \tag{2.74}$$

Where H is transformation matrix. The H matrix is of size $N \times M$. N being the number of nodes in the structural mesh and M being the number of surface nodes on the CFD mesh. We can see that only the $N \times N$ matrix M needs to be inverted. Furthermore this inversion is only needed once in static aeroelasticity.

2.4.2 Moments conservation

As it was started with the demonstration of the force conservation, the moments conservation on the structural side is:

$$\Delta\theta_s = \sum \beta_j \phi(\|\boldsymbol{x_s} - \boldsymbol{x_{s_j}}\|) \tag{2.75}$$

This equation is then transformed into the following.

$$\begin{pmatrix} \Delta\theta_{s1} \\ \vdots \\ \Delta\theta_{sN} \end{pmatrix} = \begin{pmatrix} \phi_{s1s1} & \cdots & \phi_{s1sN} \\ \vdots & \ddots & \vdots \\ \phi_{s1sN} & \cdots & \phi_{sNsN} \end{pmatrix} \begin{pmatrix} \Delta\beta_{s1} \\ \vdots \\ \Delta\beta_{sN} \end{pmatrix} \tag{2.76}$$

As it was done for the forces conservation, β is found by inverting the $\boldsymbol{M}$ matrix. Since this was already done for the forces conservation, there is no need to do it here. The program will just use the already computed matrix.

By defining $\Delta\theta_a$ as it was done for the displacement we get:

$$\Delta\boldsymbol{\theta_a} = \sum \beta_j \phi(\|\boldsymbol{x_a} - \boldsymbol{x_{s_j}}\|) \tag{2.77}$$

Finally the radial basis function part of the above equation, is the A matrix previously computed. This means that transformation equation for the rotations is:

$$\Delta\boldsymbol{\theta_a} = \boldsymbol{H}\Delta\boldsymbol{\theta_s} \tag{2.78}$$

With the hypothesis made during this project, we can use the same matrix for translation and rotations. This is a non negligible advantage when the structural mesh becomes big. Fully populated matrices tend to scale memory usage in $O(n^2)$. 10^3 structural nodes will then lead to a 1GB memory usage.

Chapter 3

Implementation

3.1 CPACS file format

CPACS stands for Common Parametric Aircraft Configuration Schema [3], and is a common language to describe airplanes that can be used in aircraft design. CPACS is developed and supported by DLR for more than 10 years. A CPACS file is a file written in the XML format which contains all type of information about an aircraft. For example, wing profiles, wing shapes, fuselage shape, mass, engines, etc. . The idea behind the CPACS file is to have a portable, easy to use file format that can be exchanged between all the project actors. This permits to have multiple teams working at the same time on the project and exchanging information using the same file. As one can now imagine, the file size and hence the amount of information that a CPACS files contains varies with time. During this project the only information that was used from the CPACS file were the fuselage length and shape, the wing size and shapes, and the take-off mass.

The main advantage of working with a CPACS file is that it comes with open-source libraries that permit to read, add, modify etc. the file. During this project two CPACS related libraries were used: TiXI which is a CPACS reading and writing tool and TiGL [16] which is a geometry library. Both libraries complement each-other and are generally used simultaneously. Both programs have Python interfaces, which made them particularly easy to use during this project.

3.2 Structural mechanics mesh

3.2.1 User input

As of today the CPACS format does not permit to store general airplane structural properties, but it can store the spars, ribs, stringers and many other mechanical components of an airplane. In the frame of this thesis, the main goal is to develop a low fidelity aeroelastic tool. Instead of putting the structural information into the CPCAS file, since it is not yet possible, the user will give the parameters summarized in table 3.1 as an input into the **.JSON** input file of the tool. All the properties in 3.1 have to be given for each wing and fuselage of the airplane.

3.2.2 Reading a CPACS file

To understand how the meshing process works, we need to first introduce the CPACS file structure. The most important feature of a CPACS file is the UID or unique identifier. This variable permits to access with ease each instance of the file. For instance, a wing has a UID, the fuselage has also one, and each section defining the wing also has its UID.

The fuselage is defined by a collection of sections. Each section has its height, width, center and the vector normal to the surface it spans. This permits to place and orient the sections. All the sections put one after the other define the aircraft fuselage. Between each center section, the CPACS defines a segment. A point on the segment is defined by the η variable, $\eta = 0$ being the coordinate of the center of the first defined section and $\eta = 1$ being the coordinate of the section

Property name	Definition
Wing cross section properties	
A	Beam cross section area $[m^2]$
Iy	Second moment area (local y-axis) $[m^4]$
Iz	Second moment area (local z-axis) $[m^4]$
J	Torsional constant $[m^4]$
Wing material properties	
E	Young's modulus $[N/m^2]$
G	Shear modulus $[N/m^2]$
ρ	Density $[kg/m^3]$
Finite Element properties	
Mechanical Interpolation type	Second moment of area variation with wing area
nodes number	Beam node numbers
rigid nodes	Connecting point rigid nodes number

Table 3.1: Aeroframe required beams properties

defined right after. Again, each section is defined by its UID and is accessed by calling it with the TiGL library.

A wing is defined in a similar way. The user needs to specify the center, the chord length, the profile type and the orientation. In the same fashion as for the fuselage, a wing section orientation is given by the normal vector to the surface of the section. Last but not least, if the wing is symmetric, only a tag is added to the definition. The tag defines which plane is used for the symmetry.

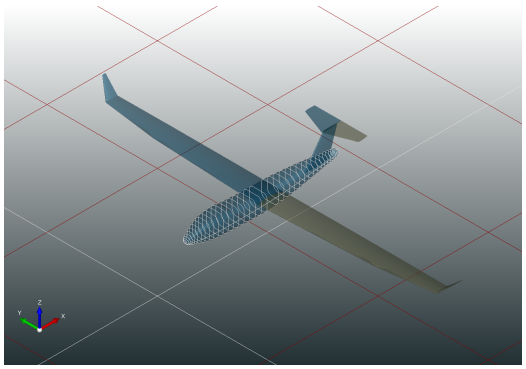


Figure 3.1: Fuselage sections

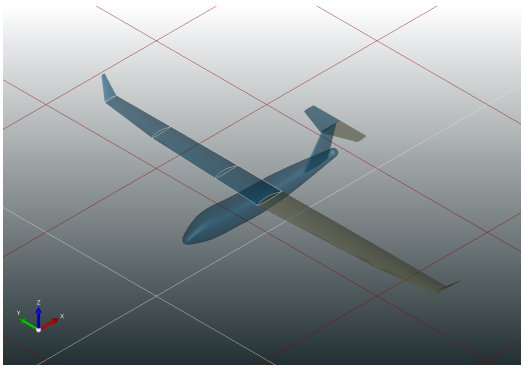


Figure 3.2: Wing sections

As mentioned in table 3.1 the user can select how many nodes are attributed to each beam representing each part (fuselage, or wings) of the aircraft.

To decompose the fuselage in even pieces, the meshing program first creates a line of the same length as all the fuselage segments placed one after the other. This line is then decomposed in bits of even length. After that, each bit is assigned to its corresponding segment and start and end positions are converted into *eta* coordinates. Then the program calls the corresponding TiGL function to find the actual point in space. The same procedure is used to discretize the wing. At this point the reader has enough knowledge to understand how aeroframe functions. The workflow will be described in next section.

3.3 Aeroframe 2.0

3.3.1 Workflow

Aeroframe 2.0 is the program developed during this master thesis. The figure below, gives a rough idea on how the program is thought. The workflow follows closely the architecture.

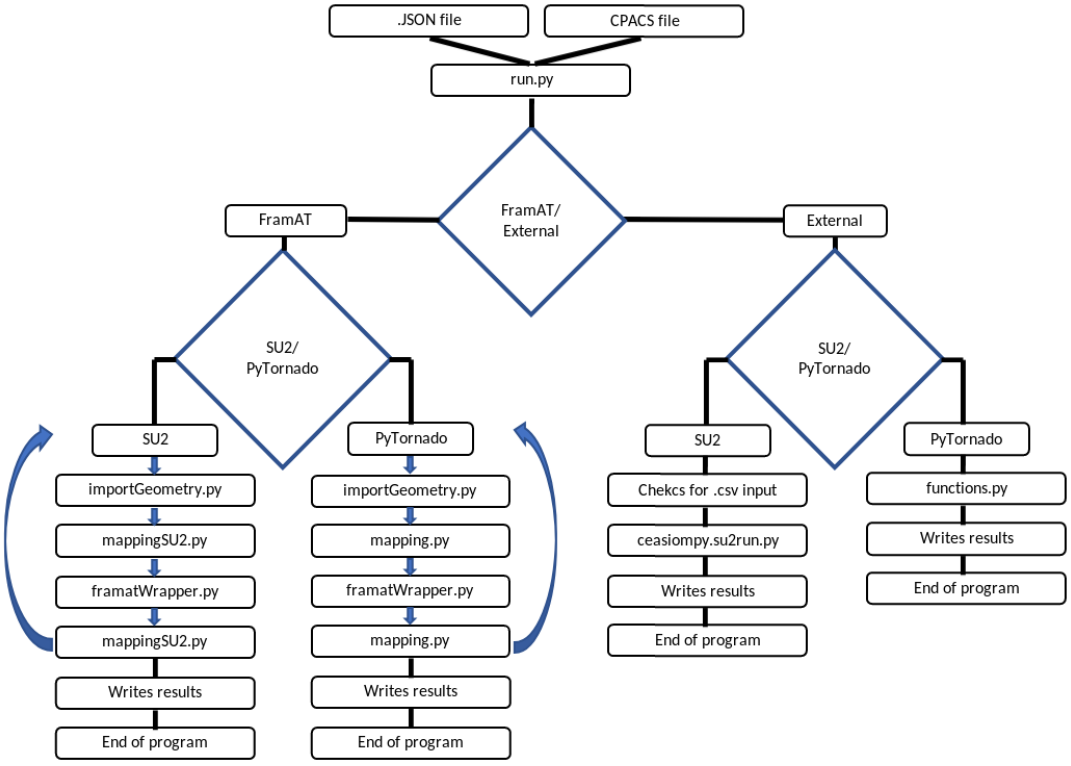


Figure 3.3: Aeroframe2.0 workflow

The command line input will call the run.py file. This file contains all the functions managing the calls of the sub-classes. "run.py" will check which path the user want to go for. The implemented combinations of CFD and CSD solvers are the following four:

1. VLM solver: Pytornado - Structural FEM solver: FramAT
2. VLM solver: Pytornado - External solver
3. CFD Euler solver: SU2 - Structural FEM solver: FramAt
4. CFD Euler solver: SU2 - External solver

3.3.2 Function files

importGeometry.py

This class will first be initialized, by taking as an input the path of the CPACS file and the path to the user input JSON file. Then the initialisation function reads the CPACS file and stores the information into a TiGL handler.

The initialisation function will then test if the CPACS file has a fuselage. If not there is no problem, the program will take it into account for the rest of the simulation. If there is more than one fuselage, this is a problem and the program will signal an error and stop.

Then the initialisation function reads the wings stored into the handler. If no wing is found the program throws an error and then stop. If more than one wing is found, the program will check if there is a fuselage. If not, the program stop. Otherwise it can continue the initialisation process, an it will read the fuselage UID and store it into a class variable.

At this point the function will test what is inside the user input file. If one of the CPACS instances (fuselage of wing) is not described, the program will signal that there is a mismatch between the

CPACS file and the JSON input file and will stop. If the JSON file matches the required information from the CPACS files the program will continue.

When FramAT is used, the aeroelasticity director function will then call the following sequence of methods of the importGeometry class:

1. **getOrigin()**
2. **computesWingsMeshPoints()**
3. **computesFuselageMeshPoints()**
4. **readsMaterials()**
5. **assembleMatrices()**
6. **computesWingConnexions()**

The **getOrigin()** will simply read the CPACS file and get the point used as origin in this file. This point is crucial for the rest of the simulation since it will be used as the center of mass of the aircraft.

computesWingsMeshPoints() is a not a trivial function. It first checks if the user has not been too original for his structural node request. For the reader that would like to use Aeroframe 2.0, the condition is clear: a minimum of 3 mesh points for each half of a wing. If this constraint is not satisfied the program stops with an error.

Now it is important to understand what ξ and η are for a TiGL handler. ξ is a number between 0 and 1 which describes the position of a point on the wing, between two sections and in the chord-wise direction. $\xi = 0$ being the leading edge of the wing and $\xi = 1$ is a point on the trailing edge. Anything between 0 and 1 will be a point in between in function of the chord length at this point in the span-wise direction. η is also a number that varies between 0 and 1 and describes a point on the wing in the span-wise direction. Those points are on a segment line between two sections. See figure 3.4 below form the TiGL documentation [17].

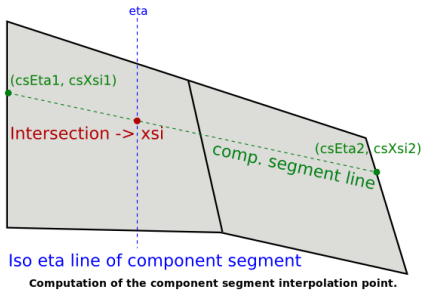


Figure 3.4: Position on a wing for a CPACS file

In the input JSON file the user needs to specify the elastic axis in terms of chord length. This permits to have the exact ξ that needs to be fed into the TiGL function that is needed to compute the point.

Let's go back to the wing meshing function. The function now needs to know how many segments are defining the wing. Furthermore the function will also read the number of sections. This will permit to tell the user if he is asking for less nodes than the number of sections defining the current wing in the CPACS file. At first it was thought that the program could better capture the true shape of the airfoil by moving all the points of the mesh closest to a section. This led to bad results in SU2 simulations, where the radial basis function started to do non-smooth interpolations for some CFD nodes. This is the reason for the deactivation of this function in the code. If the reader feels confident enough he cans still activate it for PyTornado CFD computations, but he will loose the opportunity of a comparison with SU2.

Knowing each segment position the program needs to compute the structural mesh nodes. The first step is to put each segment one after the other and by doing so computing the true length of the half of the wing. Now the wing structural nodes are computed by setting the number of user asked nodes on the virtual line. The program can now know which node belongs to which segment and can start computing the η for each point. Once this is done the program computes the actual structural mesh point on the wing by calling the correct segment UID, η and ξ, and then stores all the information into an array.

At this point half of the wing mesh is computed and the other half needs to be computed if the wing has a symmetry. First the program checks if there is a symmetry plane. If yes, gets which one it is. For the record, the nomenclature is the following:

- TIGL_NO_SYMMETRY = 0
- TIGL_X_Y_PLANE = 1
- TIGL_X_Z_PLANE = 2
- TIGL_Y_Z_PLANE = 3

The program can now compute the symmetric points and add them at the end of the first array. Note that the last point of the first array is the symmetric point. For this reason the last point of the first array for each wing is not inverted. This implies that a wing will always have an odd number of points, independently of what the user asks for.

In order to stay concise, only the elastic mesh points computation is explained here. At the exact same time, the same process is done for the computation of the mass points, and the area of the section at each of these points is computed. The mass points are the points on which the mass of the segment will be lumped. These points are used only when the acceleration effects are included into the computation, but are computed anyway.

Lastly each point of the wing is given a new UID. This UID will be useful into the FramAT wrapper to associate the correct mechanical properties to each structural mesh node.
Once the wing points are computed, the fuselage points are computed. Again, the function **computesFuselageMeshPoints()** will check if the user asks for a realistic number of points. Since a point for the center of mass is needed in addition to the nose and tail points, the minimum number of four points is required. It is not recommended to run a simulation with a fuselage described by less than 5 nodes. Ideally 10 to 15 nodes is the bare minimum, especially if SU2 is used for the CFD simulation. The reason being that some non physical clamping will happen in order to prevent a gap during the mesh displacement process (SU2_DEF function) that will lead to divergence of the CFD computation.

At this point the function needs to know how many section and segments are defined in the CAPCS file. Again, the same procedure of lining up all the segment is done. Then the virtual line is divided into the number of sections asked by the user. The program will search for each point which segment it belongs to, then will compute the η distance for the segment and then will call the TiGL function in order to get the point on the center of the section for the asked η on the segment of the correct UID. It was noted that when a segment is very long, the distribution of points is not linear. This will not lead to any error and is not a problem.

Since the assumption that the mass of the fuselage is lumped on the central point, there is no need to compute the mass line for the fuselage. Nevertheless the section area for each structure mesh point is computed and stored. The area is needed to compute the mass of the segment. This is also valid for the wing points.

readsMaterials is a function that does exactly what it is named after. The functions reads the input JSON file and allocates the materials properties to the correct beam.
assemblesMatrices will assembles the matrices in such a way that it will be easier to call each beam instance into the FramAT wrapper.

mapping.py

mapping.py is the class that will translate the forces from the CFD solution to the structural mesh. It first starts by getting the CFD mesh points and the CSD mesh points. The mesh being of different shape for the VLM solver and the Grid solver, two versions of this class doing mostly the same work were necessary.

computesTransformationsMatrices will compute the $\boldsymbol{A}$, $\boldsymbol{M}$ and $\boldsymbol{H}$ transformation matrix. The mathematical side of this function is explained in section 2.4. As explained in the theoretical part, the matrices need to be filled in by the radial basis function. In the results section, it is shown that the best choice is the Wendland C0 function, and is then implemented into the code.
phi is the function containing all the radial basis functions. If the user wishes to change the default function and add one of his own, it can easily do it by adding the function and the necessary information into the code.
aeroToStructure will compute the loads that will be mapped onto the structural mesh. The first step is to call the subfunction **computeMoments()** which will first compute the distances between the structural mesh and the CFD mesh. This is not a trivial task to program.

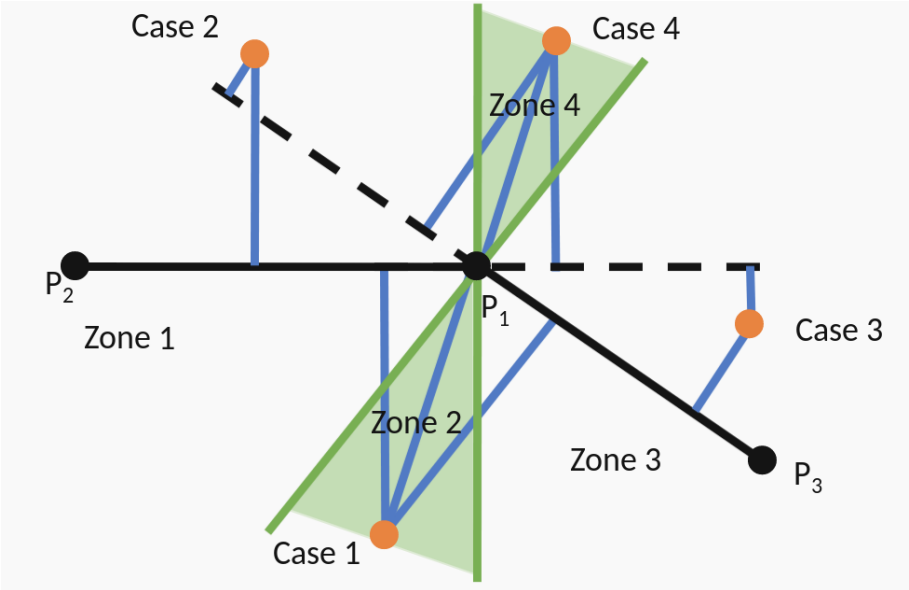


Figure 3.5: Distance computation cases, for moment computation

When the wing is straight, there are no such problems but when the wing is swept, there is mainly a problem at the symmetric axis. When there is an angle on the structural beam there are mainly four cases as it is possible to see on figure 3.5.

A point, in orange on the figure, has three closest neighbours. The moment has to be projected on one of these points, and the torque is given by the force at the point multiplied by the distance between the point and the elastic axis. The projected point on the elastic axis could be on mesh, in black on the figure, or on what the program could think is the mesh, but is outside the mesh, the dashed lines.

In case one, the point could be mapped on the line P_2P_1, P_3P_1, or on the point. In this case the closest distance condition is applied.

In case 2 the situation is similar than case 3, at the difference that the line on which the point needs to be projected not the same one. In this case, the program has a routine that test which one is the correct line to project the point on.

Finally, in case 4, the point could be projected on any physical elastic lines. In this case the routine will select the closest point and will use the distance between the CFD point and the closest

structural point.

The routine computing the moments having finished its work, the forces and moments are multiplied by the transformation matrix $\boldsymbol{H}^T$ in order to get the forces and torques for the structural computation. The mapped forces are saved into a .csv file in order to keep track of what was computed during the aeroelastic simulation. Structural forces are stored into arrays that will be used into the FramAT wrapper.

The next step is to compute the mass of the wing. The areas computed in the **importGeometry.py** class are red, and multiplied by each segments length. Since the program has the knowledge of all CFD nodes forces, the total lift is computed. Since the mass is also computed, the program can now compute the G loads that are acting on the aircraft for this aeroelastic iteration. There is the possibility to impose the G load to 1G for comparison with wind-tunnel data.

structureToAero is the inverse function of **aeroToStructure**. It computes the structural displacement of the aerodynamic mesh form the displacement resulting from the FramAT solver. To do so the exact same $\boldsymbol{H}$ matrix multiplies the displacements computed by FramAT. The aerodynamic mesh displacements are then stored into arrays that can be used in the respective solver deformation tool.

mappingSU2.py

The main difference between this class and the **mapping.py** class, is that the CFD mesh used for the computation is of a different type. Only the initialization phase is different, the rest of the code structure is identical than in **mapping.py**.

framatWrapper

Initialisation of the class is done by generating a FramAT model, by importing the beam nodes point. Into the **run.py** function file, the **mesh()** function is called.

mesh() will first read the material and mechanical properties for each beam. These properties were stored by the **importGeometry** class. It will then start to add the nodes to the model by calling the **computesCsdMesh** method. This method will first generate a beam instance, then add all the nodes, assign the materials, and orientation of the vertical axis.

Once the mesh is computed **run.py** has to compute the CFD solution and do the transformation. once this is done, the **run()** method from the **framatWrapper** class is called.

run() will first impose the boundary conditions. The boundary condition are of two types in FramAT: clamped and rigid link. The clamped node will impose no translation and no rotations. This is imposed at the center of mass of the aircraft, or a the symmetric point of the wing, if there is no fuselage.

Rigid links are links that force the displacement of two nodes to be identical. For instance this boundary condition is imposed between one node of the fuselage mesh and the symmetric node of the main wing. These links are also used to over stiffen all the connections between the different beams, in order to have a rigid motion and prevent mesh holes at interfaces between the fuselage and the main wing for example.

ceasiompy.su2run.py

When the function is used with and external solver, it first needs to know which aeroelastic iteration it will be doing. This is a choice of the user that is read from the .JSON input file. It must start at 0, otherwise the program will search for a folder and it will crash since it will not find one. This is necessary because the software stores previous aeroelastic steps results in order to better analyse the results.

At this point the program will create the current iteration folder, read the initial SU2 configuration file and modify it accordingly to for the current iteration. By doing so the file will know where to look for the mesh, where are the previous iteration files if there are some, and where is the deformation file located.

Once the deformation is finished by SU2_DEF, if it is not the first iteration, the program will call the SU2_CFD function. A small detail about SU2_DEF: the best results were obtained while setting the parameter DEFORM_STIFFNESS_TYPE = WALL_DISTANCE. During this project it was experienced that this step leads to better mesh quality and faster deformation times.

Chapter 4

Case studies

4.1 Choice of the interpolation function

4.1.1 FramAT General testing

Using an external structural solver requires a minimal validation. For this reason, two toy test cases were done. The first one constitutes of running Aeroframe 2.0 with no aerodynamic loads and no wing loads. Basically testing if the program is introducing forces on its own. The second toy case consists of looking at the wing displacement under its own weight. This test permits to see two separate results. One is to show if the displacements under a constant weight is mapped correctly. The second results that is tested is if aeroframe computes the weight of the wing correctly.

No loads test

This test was executed from 5 to 160 nodes, doubling the mesh size each time. The results are as expected and shown below. The wing used for the validation is always the same one and its properties are given in table 4.2. For this cases the G load is set to 1. As the reader can note, if the G loads are activated into the solver, the error on the wing displacement can only come from the aerodynamic load mapping procedure. As the reader can note, the result is as expected with no displacement.

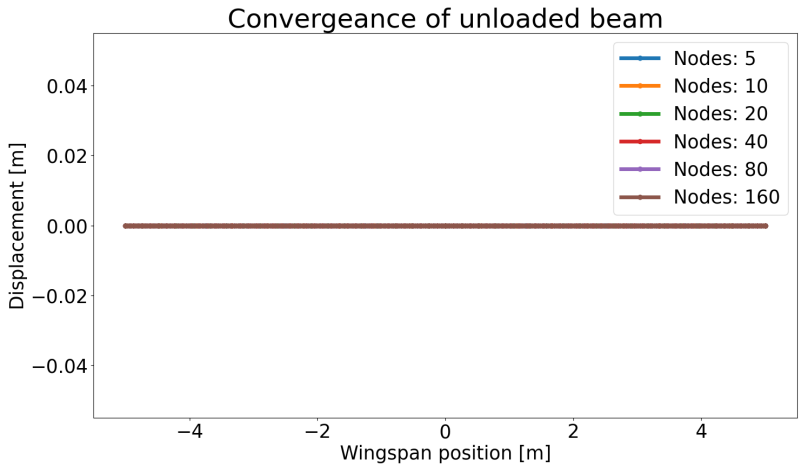


Figure 4.1: Unloaded beam

Side note, the error is computed on the maximal displacement. The Finite element tends to have more displacement than the analytical solution.

Weighted wing

As it was done for the unloaded case, a structural mesh containing 5 to 160 nodes doubling size each time was used. The G load was imposed at 1. The results are shown in the figure below.

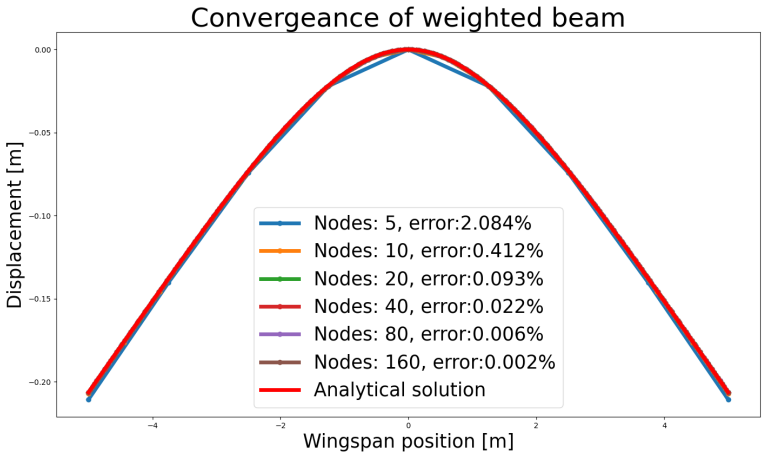


Figure 4.2: Unloaded beam

4.1.2 External structural solver

As explained in section 3.3, the information transfer applet overcomes the problem of transfer of forces and moments between mismatching meshes. While solving an aeroelastic problem with the external solver NASTRAN, aeroframe receives a set of .csv file containing the data points. Each point has its original place on the VLM mesh and its displacement associated to it. There is no constraint on the .csv file, the points could be structure points. Ideally it should be the VLM mesh original points, but it can also represent a curved surface which the VLM mesh should now match. A constraint for this to work is that surface contained in the .csv file should be as big as the airplane external shape. That is for a wing, from the leading edge to the trailing edge and from root to tip. No need of points between let's say the wing and the horizontal tail, but the horizontal tail must be defined.

With such a problem, Radial basis functions are ideal, since they are exact on the data-set points and yield a continuous function over the whole surface. This implies that there will not be a mesh mismatch and that all mesh points will have a displacement assigned to them. The important part is to use the best function for this task. Rendall and Allen [14] cites Smith et al. [18] on the function choice. These documents are at the foundation of this thesis.

As stated in [14] Radial basis functions are of two types: growing and decaying as the reader can see in Figure 2.2. Growing functions are not ideal in our case since they increase with distance and hence could lead to a force increase in the mapping, but Thin plate spline function and Euclid Hat are nevertheless commonly used for 2D and 3D problems. Furthermore it will be shown that for the beam case it is impossible to get a meaningful result without using increasing functions.

For ease of use the external function interpolation uses the python class **scipy.interpolate.rbf**. The use of the python built-in functions was done as much as possible since theses libraries are generally compiled FORTRAN code like Intel MKL or LAPACK libraries.
Choosing the best radial function is, even with a reduced choice, not an easy task. All of them tend to have pros and cons, but during this project, most of the time decaying functions led to poor or diverging results. The set of working functions while using an external solver are the Hardy's multiquadric function with $\epsilon < 10^{-1}$, the linear, cubic, quintic function and the thin plate spline function.

Cubic and quintic functions are avoided because they lead to poor results if the surface input is not well defined. Hardy's multiquadric led to poor wingtip results as it is shown on figure 4.7, blue color indicating a adverse pressure, which was not present on the undeformed state, nor when linear, cubic, and quintic function were used. As it was observed during the flat wing test case, linear functions tend to have a rounding effect on the wing chord direction and are hence avoided.

The last one that gave the best results is the thin plate spline. Furthermore, pressure symmetry was best kept by this function. On the horizontal tale one can observe a slight difference between figure **??** and **??**, **??**, **??**, (Appendix). Nevertheless having a flat center helps. Experience during the debugging phase showed that flat plates tend to have an sudden error on the lift distribution, in the center of the airplane when the wing is dihedral or anhedral since it flattens the wing while getting interpolation from points on the other side of the symmetric plane.

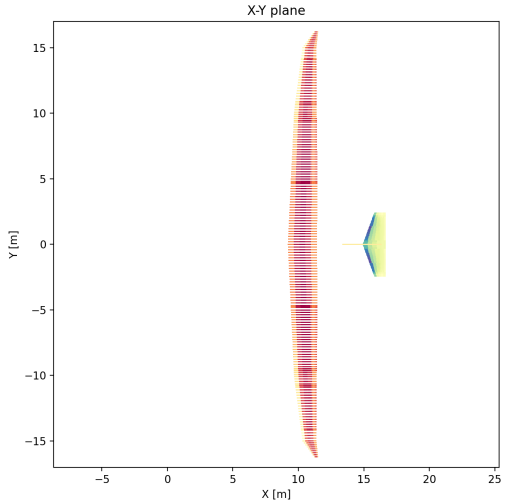


Figure 4.3: Undeformed wing

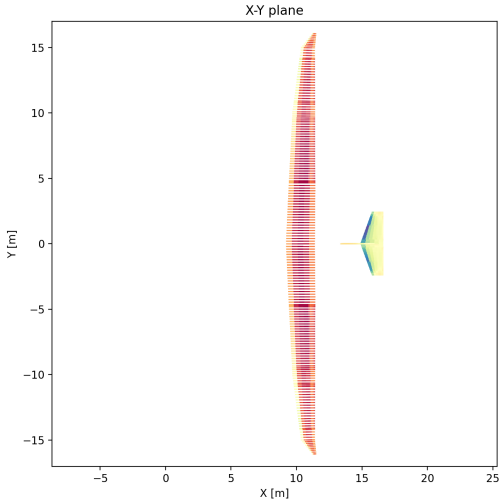


Figure 4.4: Deformed wing (TPS function)

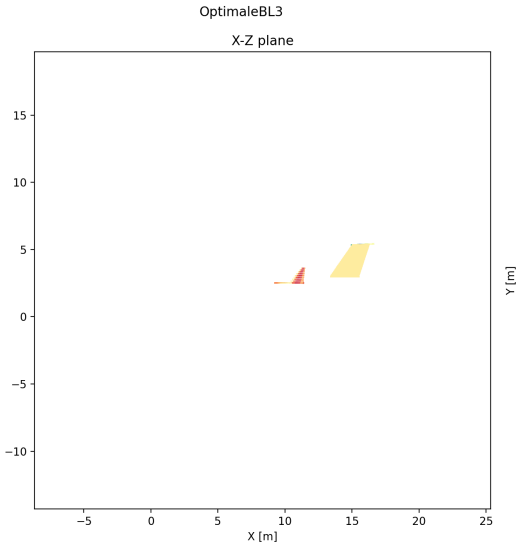


Figure 4.5: Undeformed wing

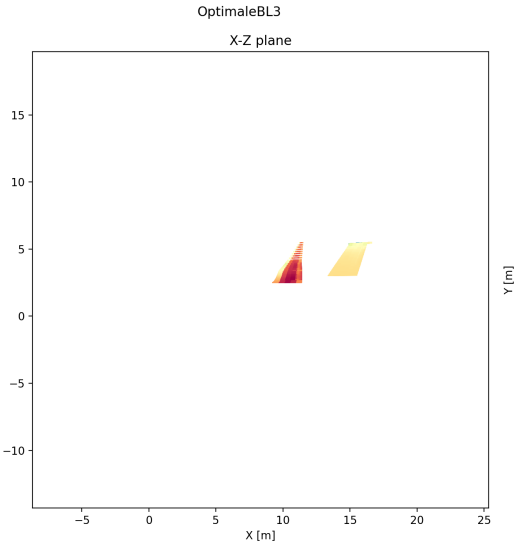


Figure 4.6: Deformed wing (TPS function)

Figure 4.7: Multiquadric wing tip induced error

4.2 Straight wing

In order to verify the simulation results a comparison with analytical results is necessary. The theory used for this test case is developed in sections 2.2.1 and 2.2.2. Staying simple is the best way to get useful and accurate results. The selected case is build on this principle. The selected airfoil is the NACA 2315 of constant shape, flat, with no sweep. Table 4.2 gives a summary of the wing properties necessary to perform the simulation and the analytical analysis.

Property name	Value	Units
airspeed	100	m/s
AoA (alpha)	10	deg
AoS (beta)	0	deg
density	1.225	kg/m^3
rate P	0	rad/s
rate Q	0	rad/s
rate R	0	rad/s

Table 4.1: Aerodynamic state

Property name	Value	Units
Wing geometry		
external chord: c	2.0	m
internal chord: c	1.96	m
external thickness: t	0.240	m
internal thickness: t	0.235	m
span: s	5.0	m
Wing cross section properties		
A	1 E-2	m^2
Iy	4 E-5	m^4
Iz	4 E-5	m^4
J	8 E-5	m^4
Wing material properties		
E	70 E9	N/m^2
G	27 E 9	N/m^2
ρ	2100	kg/m^3
Axis in function of chord length		
Aerodynamic axis	Computed by CFD	-
Elastic axis	0.6	chord-length
Mass axis	0.1	chord-length
e	0.2	m
d	0.4	m

Table 4.2: Wing properties

The bending inertia I_y is set so the wing experience a meaningful displacement but stays in the Hypothesis of the Euler-Bernouilli beam theory. I_z term is only useful for the FEM analysis, the analytical solution will not depend on it.

4.2.1 FramAT solver

Aeroelasticity using a loosely coupled paradigm faces mesh mismatch. As it was explained before, the conservation of virtual work was chosen for support in the information transfer between the CFD mesh and the CSD mesh. This is easier to implement with radial basis functions. All these aspects together give a solution that depends on the number of points used for both meshes, as it is always the case with finite element, finite volume or panel methods, but also to the choice of the mapping function.

A summary tables of the displacements errors as function of the number of nodes used for the structural mesh and the radial basis function used can be found below. The wing mechanical properties are given in table 3.1. The VLM solver Pytornado was used for this study for its speed. The mesh has 10 chord-wise panels and 40 span-wise panel for half a wing. For the first test, gravitational led effect were neglected (0G). A second test was made this time taking into account gravitational effects simulating a steady level flight (1G).

4.2.2 Straight wing 0G

The error is computed between the displacement found by the Finite element method for the structure, and the displacement found with the analytical solution. During the testing phase no viable results were found with the following functions:

- Gaussian
- Thin plate spline
- Multiquadric
- Inverse multiquadric
- Wendland's C2, C4, C6 and C8

C2 function experienced oscillation at the tips, so even if results seems to be meaningful, successive steps will have a non physical geometry on the wing leading divergence.

Nodes	5	10	20	40	80
RBF	Displacement error %				
L	34.5	5.7	-0.4	-6.5	-7.6
C0	43.8	10.3	0.4	-3.0	-4.2
EH	34.5	5.7	-3.3	-6.5	-7.6

Table 4.3: Percentile displacement error for a 10x40 VLM mesh, 0G

Nodes	5	10	20	40	80
RBF	Displacement error %				
L	34.5	5.7	-3.3	-6.5	-7.7
C0	43.8	10.3	0.4	-3.0	-4.3
EH	34.5	5.7	-3.3	-6.5	-7.6

Table 4.4: Percentile displacement error for a 10x80 VLM mesh, 0G

As the reader can notice, Wendland C0 function is the best choice, leading to a displacement error of 0.4%. Increasing the number of nodes on the structural mesh will insert integration errors since the mapping procedure always underestimates the force. This can be seen during the test case 1.

4.2.3 Straight wing 1G

For this investigation, the G load is set to one. The acceleration is set to earth's gravitational acceleration i. e. ($9.81 m/s$). Results being really far from what the analytical solution gives, there

is something wrong with this part of the code. Acceleration loads are visibly not implemented correctly and only the 20 nodes structural mesh is investigated. The reader will appreciate the analytical development done to get these result in a later subsection.

Nodes	5	10	20	40	80
RBF	Displacement error %				
L	140.6	23.1	-14.5	-27.7	-32.5
C0	180.3	42.9	1.5	-12.8	-18.0
EH	140.6	23.1	-14.4	-27.6	-32.4

Table 4.5: Percentile displacement error for a 10x80 VLM mesh, 1G

Nodes	5	10	20	40	80
RBF	Displacement error %				
L	140.8	23.1	-14.5	-27.7	-32.8
C0	180.6	42.9	1.5	-12.8	-18.2
EH	140.8	23.1	-14.5	-27.7	-32.6

Table 4.6: Percentile displacement error for a 10x80 VLM mesh, 1G

An interesting fact is that the last step was compared to the analytical solution and the error stayed the same.

4.2.4 General consideration

Many tests were made using a flat straight wing . The mesh size influence was investigated with PyTornado and FramAT in the section above. For this analysis a convergence criteria of a displacement less than $0.001m$ between two iteration was imposed. This is the reason for the flat shape on figure 4.12.

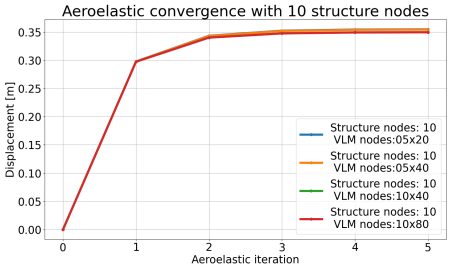


Figure 4.8: Convergence analysis 10 structural nodes

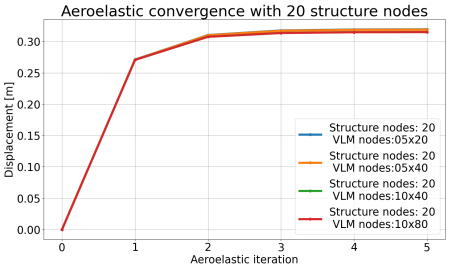


Figure 4.9: Convergence analysis 10 structural nodes

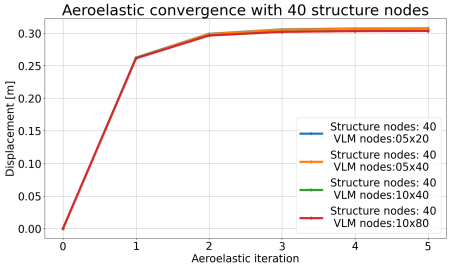


Figure 4.10: Convergence analysis 20 structural nodes

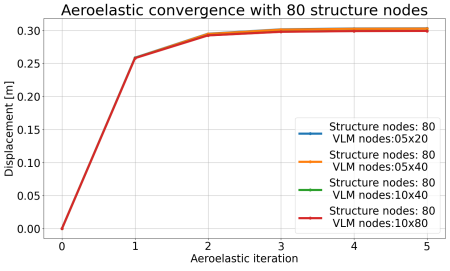


Figure 4.11: Convergence analysis 40 structural nodes

As a general result, the overall solution tends towards the same displacement with an identical structural node number and a converging radial basis function, even when the VLM mesh size

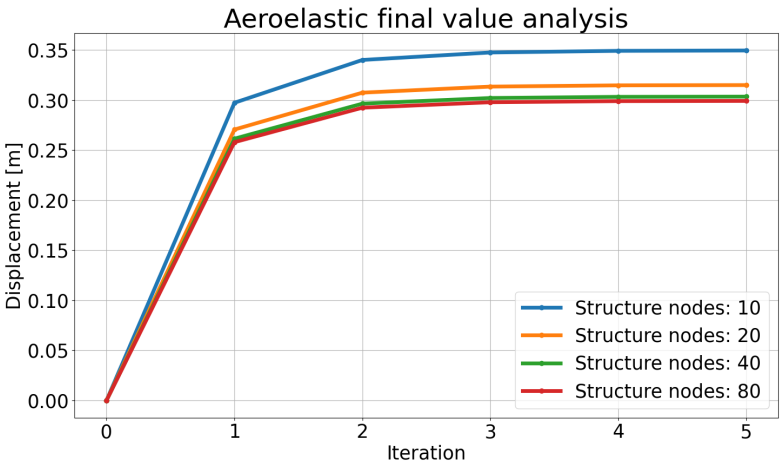


Figure 4.12: Structural mesh total displacement convergence analysis

varies.

During the radial basis function choice test, results showed clearly that for a $10m$ wingspan, 20 structural nodes with a 10x80 VLM mesh give the best results, and are the best trade-off between simulation time and quality of the result.

4.2.5 Case 1

In order to test all the coded options, the first simulation will not take into account gravitational effects. The equations that give respectively the vertical displacement and the torsion angle of the beam are:

$$\frac{d^2}{dy^2}\left(EI(y)\frac{d^2w}{dy^2}\right) = L(y) - nm(y)g \tag{4.1}$$

$$\int_0^y (M_{ac}(y) + eL(y) - Nm(y)gd)dy = GJ(y)\frac{d\theta}{dy} \tag{4.2}$$

Displacements

With gravity or acceleration effects neglected, and for a constant cross section, equations (4.1) and (4.2) become:

$$EI\frac{d^4w}{dy^4} = L(y) \tag{4.3}$$

$$\int_y^{y_{max}} (M_{ac}(y) + eL(y))dy = GJ\frac{d\theta}{dy} \tag{4.4}$$

Assembling equations (2.55) and (2.51), one can compute the lift distribution over a wing using the hypothesis of a constant torque.

$$L(y) = qca\{\alpha_r + (\alpha_r + \overline{\alpha}_r)[\tan(\lambda l)\sin(\lambda y) + \cos(\lambda y) - 1]\} \tag{4.5}$$

Taking the first four terms of the Taylor expansion of sin and cos in y gives:

$$L(y) = qca\{\alpha_r + (\alpha_r + \overline{\alpha}_r)[\tan(\lambda l)\left((\lambda y) - \frac{(\lambda y)^3}{3!} + \frac{(\lambda y)^5}{5!} - \frac{(\lambda y)^7}{7!}\right) + \left(1 - \frac{(\lambda y)^2}{2!} + \frac{(\lambda y)^4}{4!} - \frac{(\lambda y)^6}{6!}\right) - 1]\} \tag{4.6}$$

Equation (4.6) can be further simplified into the following form:

$$L(y) = c_0 + c_1 y - c_2 y^2 + c_3 y^3 - c_4 y^4 + c_5 y^5 + c_6 y^6 + c_7 y^7 \tag{4.7}$$

This equations shows that an excellent approximation of the lift distribution can be found using an order 7 polynomial. The next step is to find the numerical approximation of the CFD generated lift using an order seven polynomial. This is easily done in a python program

$$L(y) = -0.4174 \cdot y^6 + 5.376 \cdot y^4 - 152.9 \cdot y^2 + 1263 \tag{4.8}$$

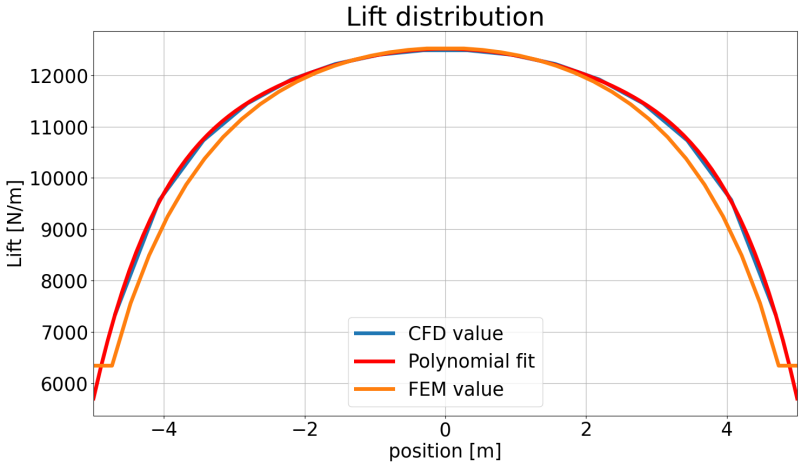


Figure 4.13: Lift distribution, CFD result in blue, polynomial fitting in red, and value found by the radial basis function interpolation matrix.

As stated before, the vertical displacement follows a fourth order ordinary differential equation:

$$EI\frac{d^4w}{dy^4} = L(y) \tag{4.9}$$

With a clamped boundary condition at $y = 0$ and free end boundary condition at $y = l$ we need to apply the following equations.

$$w(0) = 0 \tag{4.10}$$

$$\frac{dw(0)}{dy} = 0 \tag{4.11}$$

$$\frac{d^2w(L)}{dy^2} = 0 \tag{4.12}$$

$$\frac{d^3w(L)}{dy^3} = 0 \tag{4.13}$$

Which has as solution:

$$w(z) = \frac{1}{EI}\left(-0.4174 \cdot \frac{y^{10}}{5040} + 5.376 \cdot \frac{y^8}{1680} - 152.9 \cdot \frac{y^6}{360} + 1263 \cdot \frac{y^4}{24} + c_1\frac{y^3}{6} + c_2\frac{y^2}{2}\right) \tag{4.14}$$

$$c_1 = -\left(-0.4174\frac{l^7}{7} + 5.376\frac{l^5}{5} - 152.9\frac{l^3}{3} + 1263l\right) \tag{4.15}$$

$$c_2 = -\left(-0.4174\frac{l^8}{56} + 5.376\frac{l^6}{30} - 152.9\frac{l^4}{12} + 1263\frac{l^2}{2} + c_1 \cdot l\right) \tag{4.16}$$

Where l is the half wingspan.
The analytical results and the theoretical results are both plotted on Figure 4.14. As we can see, the finite element solutions matches perfectly the analytical solution. One should be very careful with such a result. It happens that for 24 structure nodes, the mapping error in the center is compensated at the tips. This should generally be the case for every wings, but odd shapes were not tested.

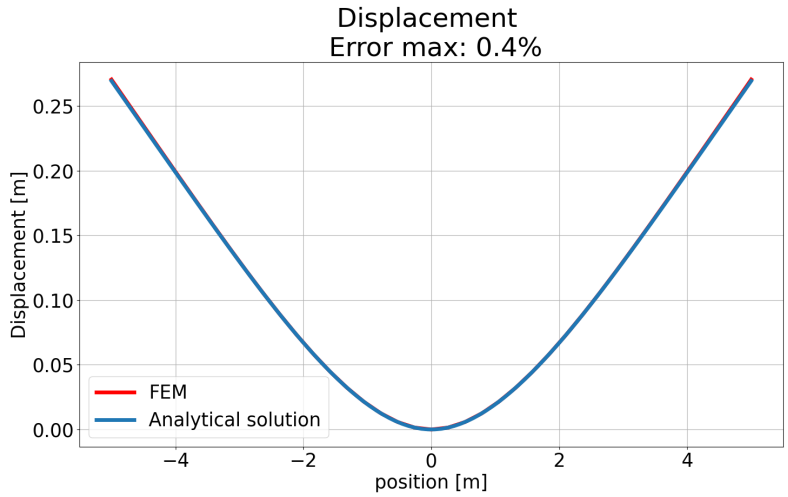


Figure 4.14: Displacement distribution, a comparison between the analytical solution and the finite element beam theory.

Rotations

By making the assumption (as it is done in the code), that the aerodynamic center is at a constant distance a from the elastic axis, which itself is at a distance e from the leading edge. The torque contribution of each point is:

$$M(y) = M_{ac}(y) + eL(y) \tag{4.17}$$

Where $M_{ac}(y)$ finds its source in the lift. Taking all this information into account the hypothesis of a torque distribution over the span of the wing of the same type of the lift is done.

$$M(y) = c_0 + c_1 y - c_2 y^2 + c_3 y^3 - c_4 y^4 + c_5 y^5 + c_6 y^6 + c_7 y^7 \tag{4.18}$$

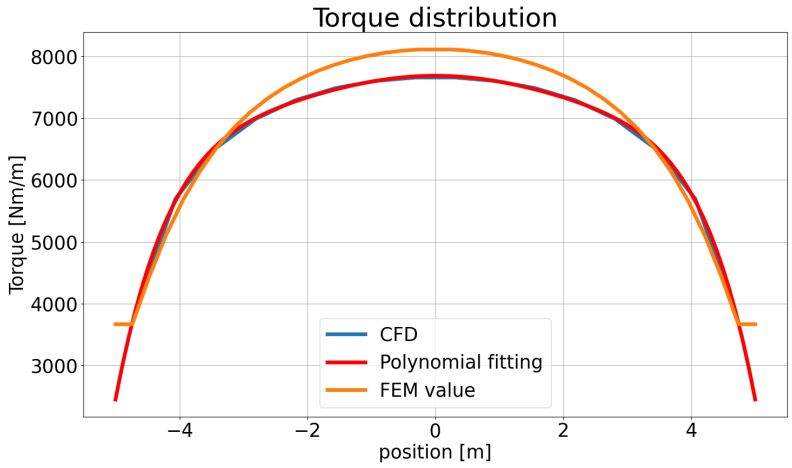


Figure 4.15: Moment distribution, CFD result in blue, polynomial fitting in red

The y-axis units are written so to keep in mind that this is a distributed moment and not a force. As we can see the polynomial fitting is in good agreement with the CFD computation of the torque on the elastic axis. The plotted torque fitting is the following equation:

$$M'(y) = -0.4266 \cdot y^6 + 7.077 \cdot y^4 - 119.9 \cdot y^2 + 7581 \tag{4.19}$$

With all the above hypothesis, the LHS of equation (4.2) becomes:

$$\begin{aligned} M(y) &= \int_{y}^{y_{max}} M'(y)dy \\ &= \left(-0.4266 \cdot \frac{y_{max}^7}{7} + 7.077 \cdot \frac{y_{max}^5}{5} - 119.9 \cdot \frac{y_{max}^3}{3} + 7581 \cdot y_{max}\right) - \\ &\left(-0.4266 \cdot \frac{y^7}{7} + 7.077 \cdot \frac{y^5}{5} - 119.9 \cdot \frac{y^3}{3} + 7581 \cdot y\right) \end{aligned} \tag{4.20}$$

The angle varies along the y axis following equation this equation.

$$\begin{aligned} \theta(y) &= \frac{1}{GJ} \int_{0}^{y_{max}} M(y)dy \\ &= \frac{1}{GJ}\left(-0.4266 \cdot \frac{y_{max}^7}{7} + 7.077 \cdot \frac{y_{max}^5}{5} - 119.9 \cdot \frac{y_{max}^3}{3} + 7581 \cdot y_{max}\right) \cdot y \\ &- \frac{1}{GJ}\left(-0.4266 \cdot \frac{y^8}{56} + 7.077 \cdot \frac{y^6}{30} - 119.9 \cdot \frac{y^4}{12} + 7581 \cdot \frac{y^2}{2}\right) \end{aligned} \tag{4.21}$$

The result is injected into equation (4.2) and gives the analytical solution for the rotation angle at each point on the wing, which is plotted in Figure 4.16.

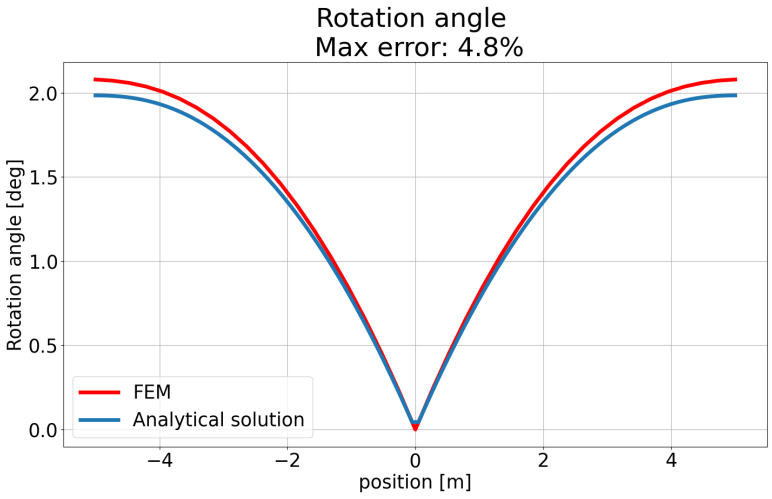


Figure 4.16: Rotation in function of the position

The same comments as for the displacement plot distribution apply here. This might be a lucky strike, and the reader should be aware of it. There is no formal proof, since adding more than 24 nodes will probably lead to an error convergence of 20% with this radial basis function (Wendland C0).

Finally the aeroelastic converged VLM solution in comparison with the undeformed state.

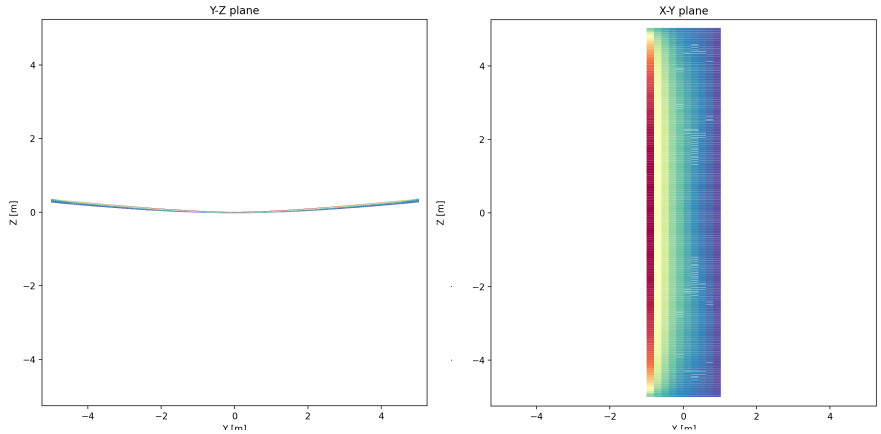


Figure 4.17: Converged aeroelastic solution with VLM solver PyTornado and structural finite element solver FramAT

Case 2

The objective of this test case is to compare the results found in case 1 from the CFD solver PyTornado with the results obtained with the higher fidelity solver SU2. The equations for the analytical solutions are exactly the same as for the first test case. However, Lift, Torque, displacements, and rotations, are different since they are computed from a grid based solver and not a panel solver. 1000 iteration were done for each aeroelastic iteration. No results were used form a previous aeroelastic iteration. The total mesh size is of 982'408 cells. Euler equations were solved for this case. The wing mechanical properties are obviously the same as for case one and can be found in table 3.1.

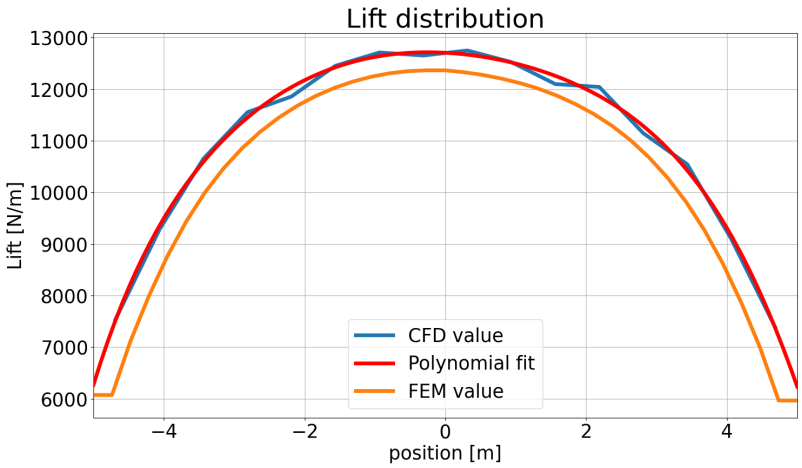


Figure 4.18: Lift distribution on the wing, polynomial fitting, and mapped lift on beam model

Summary and analysis of case 1 and 2

Let's now compare cases 1 and 2. As the reader can see in table 4.7, results obtained with the FEM solver FramAT, are comparable between the first and second case.
On case 1 the lift distribution is smoother, and this is due to the mesh geometry. In the first case, the VLM mesh is square and the wing is divided in the span-wise direction. This discretisation permits to have and equidistant, structured mesh, which is easy to lump in the chordwise direction, by taking the average panel span. As the reader can appreciate on figure 4.13 and 4.15. The only difference is that regarding the torque distribution, the distance from the bound leg midpoint to

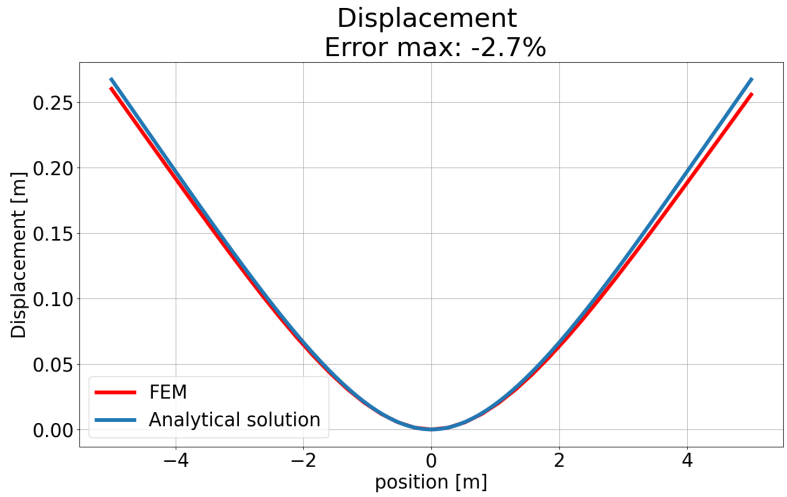


Figure 4.19: Vertical displacement given by FEM, in comparison with the analytical solution, error computed on maximal displacement

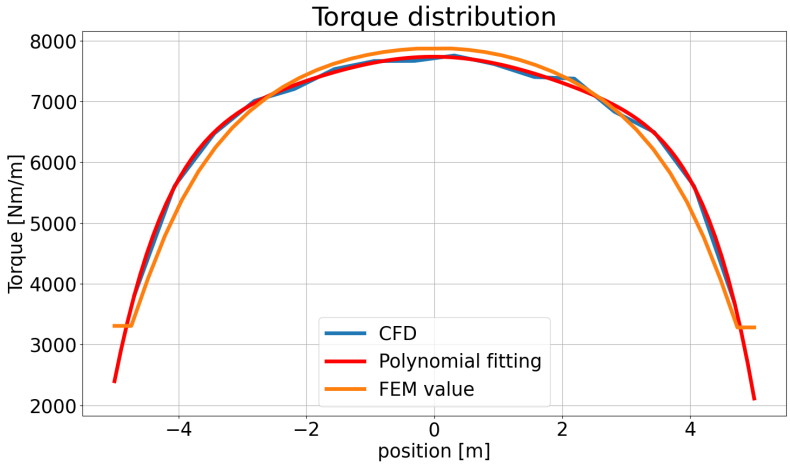


Figure 4.20: Torque distribution on the wing, CFD solution in blue, polynomial fit in red and mapped torque on beam model in orange

Propriety	PyTornado	SU2
CFD mesh	800 panels	4733 surface cells
FEM mesh	20 nodes	20 nodes
C_l	2.02	1.94
Tip displacement	0.315m	0.348 m
Max error step 1	0.4%	-2.7%
Tip rotation	4.7°	4.4°
Max error step 1	4.8%	0.6%

Table 4.7: Aeroelastic results of last converged step for case 1 and 2

the imposed elastic axis of the wing is multiplied by the force of the panel.

On the opposite the SU2 mesh is a tetrahedral mesh, There is no constraints on the spacing or the orientation of the cells. Effects of this pseudo random distribution is easy to spot on both figures 4.18 and 4.21 with the discontinuities on jumping over and under the polynomial fitting.

Regarding the displacements, the error between the analytical solution with PyTornado and SU2

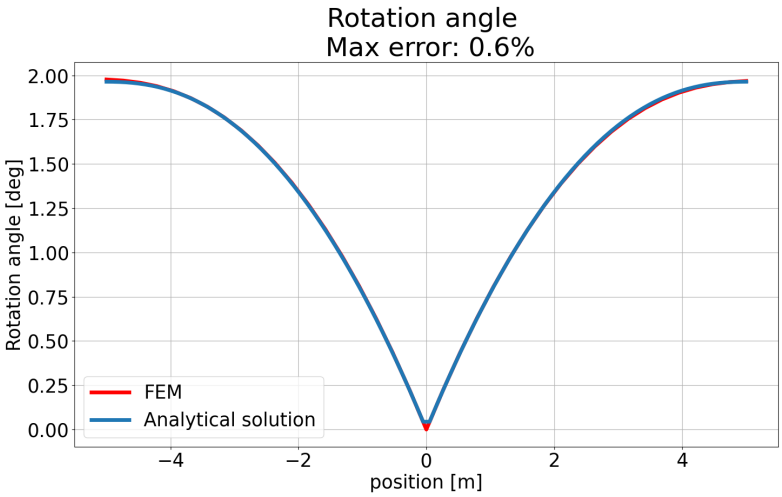


Figure 4.21: Comparison of the rotation angle computed by FEM and by the analytical solution, error computed on maximal displacement

is about the same and is acceptable for this case. However it was shown in section 4.2.4 that the CFD mesh size has less impact on the final solution, than the number of structural nodes. By saying so, it makes sense to compare both CFD solvers 20 structural nodes results.

Finally there is good agreement between the PyTorando results and the SU2 results on the final displacement with a difference of $0.023m$ for the final displacement between both methods. This could be explained by the fact that the NACA profile used is the 2315 which has a relatively thin thickness. This results could be less good for thicker wings, since volume effects on the flow can be captured by the SU2 simulation but not by the PyTornado one.

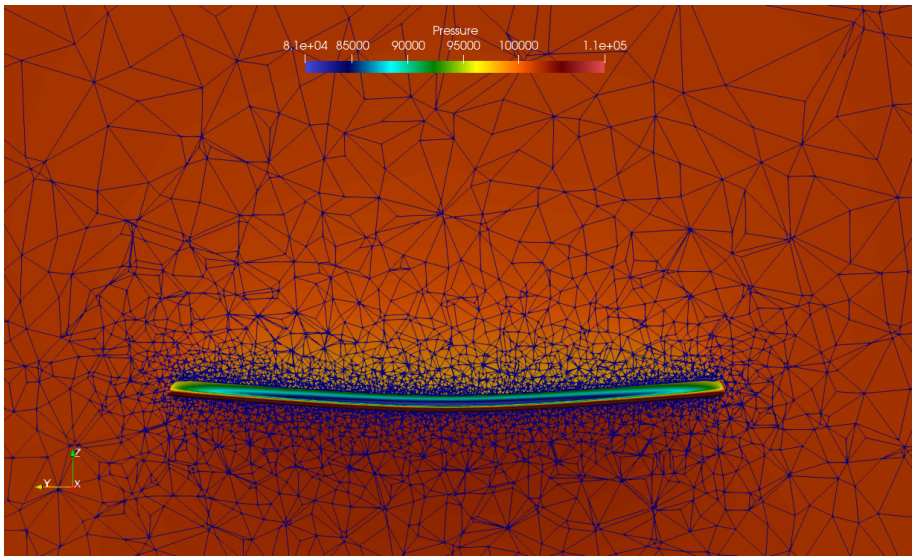


Figure 4.22: Converged aeroelastic solution with CFD solver Su2 and structural finite element solver FramAT

Chapter 5

Conclusions

5.1 Future Work

On all lift and moments plots there is the pretty visible zero order approximation for the first and last node of the structural mesh, finding a reliable way to compute the force and moment for these nodes could improve the stability of the solver and the solution quality.

Radial basis functions are of multiple types and not all of them were tested during this thesis. Doing a more in-depth analysis could also have a positive impact on the solution, especially for large deformations. The inverse isoparametric mapping method, also known as the finite interpolation method was not tested during this thesis but uses a similar theory than the one used in this theses. Minimal changes in the $\boldsymbol{H}$ matrix are required in order to apply this theory.

G loads are already implemented into the code, but results are not tested nor validated yet. Further development and validation of this part of the code could permit to validate the aeroelastic simulations done with the external solver.

More validation cases needs to be done in order to truly validate the results found during this thesis. Furthermore only two combinations were truly tested, the FramAT solver with the VLM solver PyTornado and the FramAT solver with the SU2 solver. No reference value was used for any simulation with Pytornado and an external solver and SU2 with an external solver.

Finally a proper experimental campaign with appropriate material quantification, wind-tunnel references and balance references calibrated would permit to have a complete validation of the aeroelastic tool.

5.2 Implementation

Aeroelasticity is not a trivial simulation to code. Many ways to couple the fluid dynamics part with the structural part were developed over the years. The non initiated engineer could easily be lost in all the choices. All methods have advantage and disadvantage that should be taken into account.

In addition, the number of parameter that needs to be taken into account is huge. Solver have their own mesh subtleties, and no standard is available yet.

5.3 Results

Virtual work proved to be a good fit for the first validation case. It showed to give good results compared to the analytical solution. The user should nevertheless be careful since this is only a simple case that was mainly made for debugging. The road towards a full validation is long.

Concerning the validation of the two CFD solvers with the external structural solver, no case were run. Only an interfacing test was achieved, but proved that the tool is usable. Again validation is extremely difficult to achieve since the knowledge is halved between the CFD specialist company and the structural mechanics specialist company. At this point no structural property data were exchanged besides the displacements for the CFD mesh. This is mainly due to time constraints and should not be an issue in the future.

Finally a difficult project is also be an enjoyable project. The learning outcomes, scientific and personal are countless. Project management, code architecture, coding skills, debugging skills, are all skills that I improved during this project.

Bibliography

[1] Ciampa PD, Nagel B, Rocca GL. A MBSE Approach to MDAO Systems for the Development of Complex Products. vol. 2020-3150. Virtual: AIAA Aviation; 2020. Available from: https://www.agile4.eu/cloud/index.php/s/47k2McBc8WKoefC.

[2] Aircraft generation MDO for innovative collaborative of heterogeneous teams of experts; 2020. Available from: https://www.agile4.eu/.

[3] DLR. Common Parametric Aircraft Configuration Schema; 2020. Available from: https://cpacs.de/.

[4] Raymond L Bisplinghoff RLH Holt Ashley. Aeroelasticity. Dover Publications, Inc.; 1983.

[5] Economon TD, Palacios F, Copeland SR, Lukaczyk TW, Alonso JJ. SU2: An open-source suite for multiphysics simulation and design. AIAA Journal. 2016;54(3):828–846.

[6] Airinnova. PyTornado documentation;. Available from: https://pytornado.readthedocs.io/en/latest/.

[7] Airinnova. FramAT documentation;. Available from: https://framat.readthedocs.io/en/latest/.

[8] Drela M. Flight Vehicule Aerodyn. The MIT Press; 2014.

[9] Joseph Katz AP. Low speed aerodynamics, second edition. Cambridge Aerospace Series. CAMBRIDGE UNIVERSITY PRESS; 2001.

[10] Mark Drela MB Alejandra Uranga. Flight Vehicle Aerodynamics; 2020. Available from: https://www.edx.org/course/flight-vehicle-aerodynamics.

[11] Palacios R. Aeroservoelasticity - class notes; 2020.

[12] Hodges DH. Introduction to Structural Dynamics and Aeroelasticity. Cambridge Aerospace Series. CAMBRIDGE UNIVERSITY PRESS; 2011.

[13] Liu W, Huang C, Yang G. Time efficient aeroelastic simulations based on radial basis functions. Journal of Computational Physics. 2017;330:810 – 827.

[14] T C S Rendall CBA. Unified fluid-structure interpolation and mesh motion using radial bais function. International journal for numerical methods in engineering. 2008;74(10):1519–1559.

[15] Lars Reimer ea. Development of a Modular Method for Computational Aero-structural Analysis of Aircraft. Berlin, Heidelberg: Springer Berlin Heidelberg; 2010.

[16] Siggel M, Kleinert J, Stollenwerk T, Maierl R. TiGL: An Open Source Computational Geometry Library for Parametric Aircraft Design;13(3):367–389. Available from: https://doi.org/10.1007/s11786-019-00401-y.

[17] DLR. TiGL Geometry Library; 2020. Available from: https://dlr-sc.github.io/tigl/doc/latest/group__WingFunctions.html#gad3e6018b0ee31cba109206a3515f73b4.

[18] Marilyn J Smith CESC Dewey H Hodges. An evaluation of computational algorithms to interface between CFD and CSD methodologies; 1995.

[19] Drela M. Area and bending Inertia of Airfoil Sections; 2005. Available from: https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-01-unified-engineering-i-ii-iii-iv-fall-2005-spring-2006/systems-labs-06/spl10b.pdf.

[20] Dettmann A. Loosely coupled, modular framework for linear static aeroelastic analysis; 2019.

[21] Hosters N, Helmig J, Stavrev A, Behr M, Elgeti S. Fluid–structure interaction with NURBS-based coupling. Computer Methods in Applied Mechanics and Engineering. 2018;332:520 – 539.

[22] Coulier P, Darve E. Efficient mesh deformation based on radial basis function interpolation by means of the inverse fast multipole method. Computer Methods in Applied Mechanics and Engineering. 2016;308:286 – 309.

[23] ALBANO E, RODDEN WP. A doublet-lattice method for calculating lift distributions on oscillating surfaces in subsonic flows. AIAA. 1968;68-73.

[24] Samareh J. Discrete Data Transfer Technique for Fluid-Structure Interaction. 2007 06;2.

[25] Aguerre HJ, Márquez Damián S, Nigro NM. Conservative interpolation on surface interfaces for transport problems in the Finite Volume Method. Journal of Computational Physics. 2019;395:144 – 165.

[26] Samareh J, Bhatia K. A Unified Approach To Modeling Multidisciplinary Interactions. 2000 10;.

[27] Neumann J, Krüger W. Coupling Strategies for Large Industrial Models. 2013;p. 207–222.

[28] Thomas D, Cerquaglia ML, Boman R, Economon TD, Alonso JJ, Dimitriadis G, et al. CUPyDO - An integrated Python environment for coupled fluid-structure simulations. Advances in Engineering Software. 2019;128:69 – 85.

[29] Maierl R, Gastaldi A, Walther JN, Jungo A. In: Aero-structural Optimization of a MALE Configuration in the AGILE MDO Framework; 2020. p. 169–187.

[30] Beckert A. Coupling fluid (CFD) and structural (FE) models using finite interpolation elements. Aerospace Science and Technology. 2000;4(1):13 – 22.

[31] Stickan B, Bleecke H, Schulze S. In: Kroll N, Radespiel R, Burg JW, Sørensen K, editors. NASTRAN Based Static CFD-CSM Coupling in FlowSimulator. Berlin, Heidelberg: Springer Berlin Heidelberg; 2013. p. 223–234.