POLITECNICO DI TORINO

Corso di laurea in Ingegneria Aerospaziale

Tesi di Laurea Magistrale

Development of a Code for Aeroelastic Optimization of Wings with Stress Constraints Aggregation

Relatori

Prof. Enrico Cestino

Prof. Giacomo Frulla

Candidato

Almerico Iacono

Supervisore esterno

ISAE SUPAREO

Prof. Joseph Morlier

Ottobre 2018

Dedica

A mia mamma

per il suo costante supporto.

Abstract

The development of an open-source code for the multidisciplinary design optimization related to the aero-structural optimization of wings is presented. This work is the result of a series of studies done thanks to the cooperation of ISAE and ONERA. The aim of this work is to develop an optimization process able to maximize the performance of a wing modifying its geometry out of respect of the constraints imposed, considering the aeroelastic coupling between aerodynamics loads and structural displacements. In order to manage a huge number of constraints, a constraint aggregation method is presented, based on the Kreisselmeier-Steinhauser function. It's also presented a reduced model in relation to the aeroelastic coupling in order to reduce the computational cost. The code is entirely written in Python, while external softwares are used in order to perform aerodynamic or structural analysis. Practical examples on common wings, like the Goland wing or NASA Common Research Wing CRM is presented.

Contents

List of Figures

Acknowledgements

I would like to say thank to my supervisors Joseph Morlier and Joan Mas Colomer for the extraordinary support, technical advise and for giving me the opportunity to join their project and to join their amazing research group. They was always ready to help with clarification, allowing me to acquire knowledge. Thanks to their support I was able to grow and improve my technical knowledge. I want to say thank also to the ISAE SUPAREO, and Institut Clément Ader to guest me in this period in Toulouse.

My thanks also go to professor Enrico Cestino and Giacomo Frulla and Ing. Claudia Bruni, to give me the possibility to participate to this project and for the support I've had every time I've asked for it.

I have to say thanks also to all the extraordinary people that i meet during the university experience, for the nice time that we spent together, for the mutual assistance throughout university period and for the friendship born from this.

Special thanks also go to the friends I met in Toulouse, for making this experience perfect and for the extraordinary friendship that was born and that did not stop when returned.

However, the biggest thanks is for my extraordinary family and for their endless support in all forms that may exist.

Chapter 1

Introduction to Aerostructural Optimization

1.1 Multidisciplinary Design Optimization MDO

The multidisciplinary design optimization **MDO** is a field of engineering that uses optimization method in design problems incorporating an high number of disciplines. Due to its multidisciplinary nature, aircraft design problems is one of the first application of the MDO. Aerospace vehicles are extremely complex systems, and their design needs the detailed consideration of disciplines such as aerodynamics, structural mechanics, materials, control, propulsion, etc ... and the interacting between this disciplines.

The MDO thus makes it possible to perform a design process (determinate the value of the design variables) joined to an optimization process (find the value of design variables to obtain the best value for the objective function under constraints) in view of the interaction of the different disciplines involved in the process.

«The main motivation for using MDO is that the performance of a multidisciplinary system is driven not only by the performance of the individual disciplines but also by their interactions.» [6]

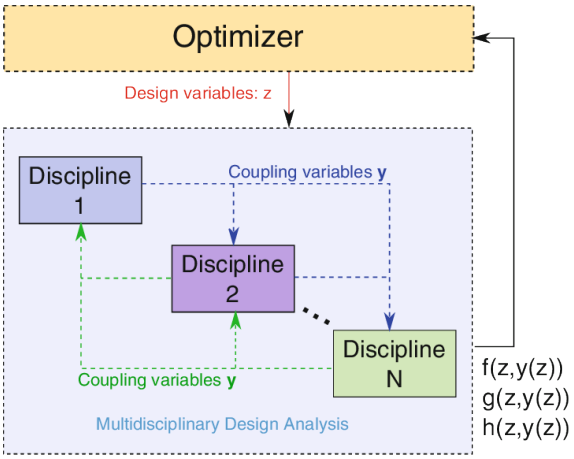


Figure 1.1: MDO scheme

Solving MDO problem in the early phase of the design process, by the support of advanced computational analysis tools, can improve the design and provide reduction of time and cost of the design cycle.

One of the first applications of the MDO was aircraft wing design, where there is a strong coupling between aerodynamics and structures, the aeroelastic coupling. After the application of MDO have been extended to the complete aircraft and to other field.

In MDO is important to define the architecture of the process,how organize the optimization software, the disciplines and the analysis of the model and the approximation model. So can be distinct two different architecture, the monolithic and the distributed. In the monolithic approach just one optimization problem will be solved, instead in the distributed approach there re multiple subproblem, which contain less variables and constraint, so the reaserch of the optimum of the main problem is split in the solving of little problems. So to solve the same optimal design problem there are many ways, and the choise of the architecture entails the computational cost and the final optimum design.

1.1.1 Optimization Methods

Computational design procedures are based on numerical analysis methods that evaluate the relative merit of a set of feasible designs. The merit of a design is based on the value of an objective function that is computed using numerical simulations

such as CFD and CSM programs. The choice of objective function is extremely important and requires a deep knowledge of the multidisciplinary design problem at hand. [5]
A typical example of a constrained optimization problem can be represent as:

$$\begin{aligned} &minimize && F(x_i) && \\ &w.r.t. && x_i && i = 1,2,..,N \\ &subject\ to && G_m(x_i) && m = 1,2,...,M \end{aligned}$$

where F is a non linear function of the N design variables x_i, and G_m are the M nonlinear inequality constraints to be respected. For a given design problem , a number of parameters x_i are allowed to change. Optimization algorithms is based to finding the design variables that yield the optimum. In our project we use the MDO to find the best configuration for a given wing taking account of the aeroelastic coupling, respecting a set of constraint.

There are a lot of optimization algorithms, but they can be classified in two categories:

- **gradient free**: the research of the optimum value is based just on the value of the objective function
- **gradient based**: the algorithm use over the value of the objective function also its gradient with respect to the design parameters

Gradient Free Methods

Between the gradient free methods there is the grid searching, where the design space is surveyed by evaluating each point in a multidimensional grid;the problem of this method is that the required number of evaluation of the objective function grow exponential with the number of design variables, so it's not worth to use this method when the design variables is more than a few. An alternative is the random

search, where is not needed an high number of evaluation, but is not guarantee that the optimum value will be found, also for an high number of evaluation. The most used methods is the non linear simplex; to create a simplex is necessary to evaluate $N + 1$ points in a $N-$dimensional space; the simplex evolves exploring the design space searching a better point, but also this method became useless if the design parameters is more than half dozen.
Ultimately the gradient free method is a powerful instruments to set an fast optimization but they became inefficient when the number of the design parameters is high.

Gradient Based Methods

In the second category we have the gradient based methods; this methods is characterized by the knowledge of the gradient of the objective function respect the design variables. That methods need a first and sometimes second order sensitivity analysis, with that information it can move into the design space with criteria, that will lead to the optimum. The great advantage of these methods is that they converge to the optimum with a significantly smaller number of objective function evaluation. On the other hand these methods work well only when the objective function changes smoothly with the design variables, and the convergence is guarantee just for a local minimum. The simple example of a gradient based method is the steepest descend, where the optimization step are choose in the direction of the gradient vector. Instead the Newton method require the Hessian Matrix, so a second order sensitivity information, in addition to the first derivatives, but it show an higher rate of convergence. A middle step is the Quasi-Newton, where the Hessian Matrix is approximated using conjugate gradient. Overall all of these method use the sensitivity analysis to identify the right direction in the design space and than perform a one-dimensional optimization in that direction before search a new direction.

Both of these methods are used currently, and the choice depends on the problem, for problems with small set of variables but with multiple local minima or

discontinues the gradient free methods are more suitable, instead for problem where the number of variables is pretty high, like high fidelity aerodynamics shape optimization a gradient based method is the best option.

1.1.2 Sensitivity Analysis

A possible definition of sensitivity analysis is the following: "The study of how the uncertainty in the output of a model (numerical or otherwise) can be apportioned to different sources of uncertainty in the model input."[7]
In our case sensitivity analysis consist in determining derivatives of one or more quantities, the objective functions, compared to the independent variables, the design parameters. Know these derivatives it's necessary for the gradient based algorithm. The determination of the gradient it's the most expansive operation in the optimization process, so it's important to use efficient methods to do the sensitivity analysis, to obtain high accurate gradients with the minimum computational cost.
There are different sensitivity analysis methods, with pros and cons, and the correct choice depends on the problem (number of independent variables and output, and how it affects the computational expense and scalability of the method), th importance of the computational efficiency and the amount of the human support. Let's see the most common method used for the sensitivity analysis:

Finite Differences

One of the common method to estimate the gradient is the finite differences method. This method is not particularly accurate and computationally efficient, but his implementation is really easy, that's why it found a large use in the design process with a huge number of variables.
All the finite differences formulas derive from the Taylor series expansion, by truncating it at the order of interest. The first order approximation, usinf the *foward difference* is given by:

$$\frac{df(x_i)}{dx_i} = \frac{f(x_i + h) - f(x_i)}{h} + \mathcal{O}(h)$$

where h is the finite difference step, x_i is the point where the derivate is evaluate and f is the function which we want to compute the gradient.

Complex-Step Derivate Approximation

The complex-step derivative approximation is a relatively new method that unlike finite differences is extremely robust to changes in the step size [5]. The approximation of the first derivative can be obtained from complex calculus and represented by the formula :

$$\frac{df(x_i)}{dx_i} = \frac{Im[f(x_i + jh)]}{h} + \mathcal{O}(h^2)$$

where the imaginary part of the function evaluation is obtained by a perturbation with a pure imaginary step, and dividing by h a second order approximation is reached.
The computational cost, as the finite difference method, is proportional to the number of variables N, but a complex arithmetic is required, so generally the cost is twice than the cost for finite differences.

Analytic Methods

The analytics methods are the most accurate and efficient methods for sensitivity analysis. But they are also the most expansive methods, because to determinate the gradients of a model by analytic approach it's required to know the governing equations of the model and how to resolve it. Usually it's hard to implement an algorithm capable to calculate the gradient, so it's required the human support to determinate it, alternatively is possible to use algorithms that solves the corresponding sensitivity equation, like the adjoint methods. That kind of method is appreciated because the cost to computing gradient is independent of the number of design variables, so it's a right chose for problem where a large number of design variables is engaged.

1.2 Aeroelasticity

Aeroelasticity is the science which studies the interaction among inertial, elastic and aerodynamic forces acting upon a flexible structure exposed to a fluid flow. It was defined by Arthur Roderick Collar in 1947 as "the study of the mutual interaction that takes place within the triangle of the inertial, elastic, and aerodynamic forces acting on structural members exposed to an airstream, and the influence of this study on design.".
This interaction is described by the Collar aeroelastic triangle Fig.1.2

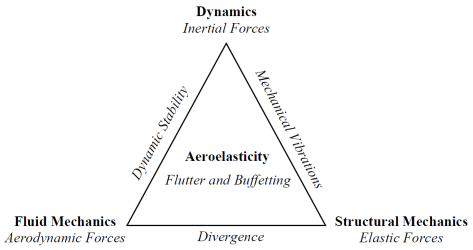


Figure 1.2: Collar Triangle

The interaction between these three forces can cause several undesirable phenomena like divergence (static aeroelastic phenomenon), flutter (dynamic aeroelastic phenomenon), limit cycle oscillations (nonlinear aeroelastic phenomenon), vortex shedding, buffeting, galloping (unsteady aerodynamic phenomena) .

1.2.1 Static aeroelasticity

The interaction between the aerodynamic forces ans the elastic forces determine the static aeroelasticity phenomena. In this work we are principally interested to the aeroelastic coupling, the aerodynamic force induce on the wing a structural deformation, which modify the geometry of the wing, and then the aerodynamic forces. So to determinate correctly the aerodynamic forces it's necessary to determinate it whit an iterative process, where step by step the forces and the deformation are updated, until the convergence is reached.
A typical aeroelastic problem can be described by the following matrix equation:

$$[M]\ddot{\mathbf{q}} + [C]\dot{\mathbf{q}} + [K]\mathbf{q} = \mathbf{F}(bs, \mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \mathbf{V}, t, \omega)$$

where $[M]$ is the mass matrix, $[C]$ is the damping matrix, $[K]$ is the stiffness matrix, $\ddot{\mathbf{q}}$, $\dot{\mathbf{q}}$, $\mathbf{q}$ are the degree of freedom(and the first and second time derivatives), $\mathbf{F}$ is the aerodynamic force, $\mathbf{V}$ is the air speed, t is the time, ω is the oscillation frequency and bs indicate bodyshape, and define the join between the aerodynamic forces and the shape of the structure.
In a static model the equation became:

$$[K]\mathbf{q} = \mathbf{F}(bs, \mathbf{q}, \mathbf{V})$$

To understand how the problem involve the study of the typical aeroelastic section can be approached.

Typical Aeroelastic Section

The typical aeroelastic section consist in a model where the wing is assumed as a 2DoF system:

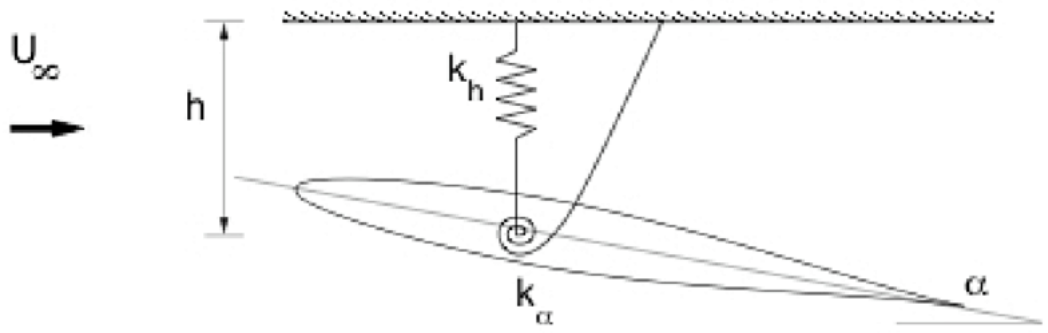


Figure 1.3: Typical aeroelastic section

The two DoF are the vertical translation h and the rotation α. The stiffness is concentrated in the elastic axes, and is represented by two spring, one linear and one torsional, with respectively K_h and $K\alpha$. Writing the equation of the rotation at the shear center we obtain:

$$Wd + Mc.a. + Le - K_\alpha \alpha = 0$$

where W is the weight, $d4$ the distance of the center of gravity and shear center, e the distance of the aerodynamic center and shear center. Collecting the terms that

depends of the rotation α, we obtain:

$$(K_\theta - qSC_{p,\alpha}e)\theta = M_0$$

where

$$M_0 = Wd + Mc.a. + qSC_{p,\alpha_0}e$$

The term $K_{ae} = K_\theta - qSC_{p,\alpha}e$ is the aerodynamic stiffness. So in the moment when we consider the deformation of the wing the stiffness of the structure change, while the aerodynamic stiffness is coupled with the aerodynamic forces and structural displacement, so it's not possible to compute its value at the start of the process, from this an iterative method is used.

The divergence problem is caused from this pattern, in fact the aerodynamic stiffness reduce the stiffness of the structure, when the total stiffness reach the value of 0 the structure became instable.

In our work we don't consider the divergence problem, but just the aerodynamic coupling, solved by the MDA loop.

Chapter 2

Source Code Description

The main goal of this project is to develop an open-source software to perform an aeroelastic optimization on an established wing model.
To perform aeroelastic optimization it is necessary to have an instrument capable of carrying out a structural analysis, in order to determine the extent of the structural deformations as well as the tensions generated in the structure under the action of the aerodynamic loads, of an instrument capable of to carry out an aerodynamic analysis, which allows to determine the entity of the aerodynamic forces as a function of the flight variables, of an instrument able to perform a dynamic analysis and therefore to determine factors such as flutter velocity, and frequency trends and of the damping according to the flight speed and finally of an instrument able to optimize the problem, going to modify the design variables according to the effect that the latter induce in the process and therefore determine the optimal value for these variables for which the best performances are obtained, but respecting the limits imposed.
To perform a multidisciplinary analysis like this, various software are required, which must be launched successively over and over again. For this reason we have chosen as the programming language the python language, through which it is possible to automate this process as much as possible, it is possible to configure the program in such a way as to compile the input files necessary for the different programs to run, launch these programs, and obtain the values of the desired variables from the output files.

So let's see the tools that make up our code:

- **Python**: open source coding language based on class and methods
- **OpenMDAO**: open source library for python containing method specialised for the multidisciplinary design optimization
- **Panair**: open source software developed by NASA for aerodynamic analysis
- **Panin**: precompiler for Panair
- **Nastran95**: open source software developed by NASA for static and dynamic structural analysis
- **Gmsh**: open source software for generation of meshes

2.1 Structure of the Code

The best way to see how the code is structured is via the XDSM diagram (eXtended Design Structure Matrix), a tool developed by Lambe and Martins (citation) that aims to represent MDO process.
In Fig.2.1 the XDSM diagram related to our project is shown. In the diagram each rectangular box represents an analysis (which can be a function or a computational code), whose input variables are represented by the white boxes on the left, while the output variables from the white boxes on the top, the gray lines represent the date dependencies, on the contrary the black lines represent process connections. All components are numbered in relation to the order in which they are executed.

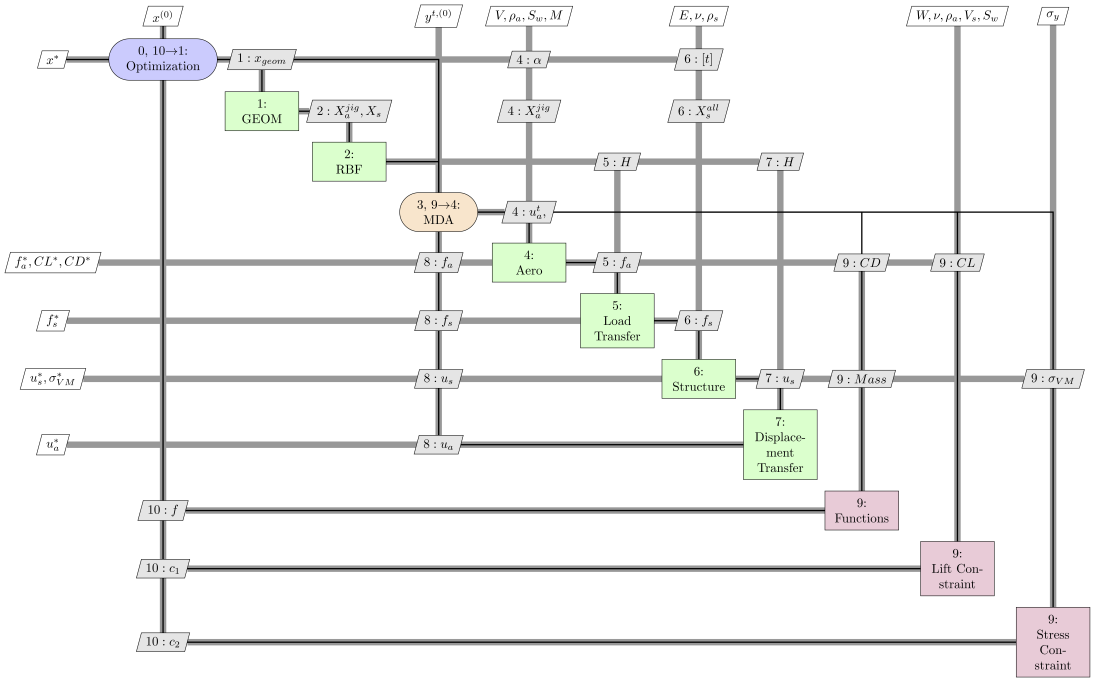


Figure 2.1: *XDSM* Diagram of the multidisciplinary analysis and optimization process

The steps that define the optimization process are the following:

0. Initiate the optimization process.
1. The initial geometry is created.
2. The interpolation matrix to couple aerodynamic and structure is created.
3. Initiate the coupling process between aerodynamic forces and structural displacements.
4. Determination of the aerodynamic loads by aerodynamic analysis.
5. Transfer the aerodynamic forces from aerodynamic mesh to structural mesh.
6. Determination of the structural displacements and stresses.
7. Transfer the structural displacements from structural mesh to aerodynamic mesh.
8. Determination of the characteristics of the wing for this configuration.

9. Computation of the constraints.

10. Based on the objective and constraints value, decide the design variables value for the next optimization iteration.

Steps 1 to 10 are repeated until the convergence of the optimization is achieved.

2.2 Component

In this section is explained how each component is structured. Each component is written on a different *.py* file, and it will be called in the main script. Using the openMDAO structure all the variables are connected between the components by the command *promotes = [*]*, so each time a component changes the value of a variable, its value will be updated for all the component.

2.2.1 Geometry

The first component it's the geometry component, this component will create the initial geometry from a bounch of parameters specified in the main code, or in an appropriate *.py* file. The user can specify the chord factor, the scale factor, the twist angle, the mid spar position and the sweep angle. From this parameters the component create and *.igs* file with the parametric geometry. After it will run the program *gmsh* to mesh this geometry and that meshed geometry will stored in an *.bdf* file.
At the end of the process the component stores in a dedicated python dictionary the coordinates of the aerodynamic points, the nodes of the outer surface and the nodes and them coordinates of the finite element model, in order to be used later by other components.

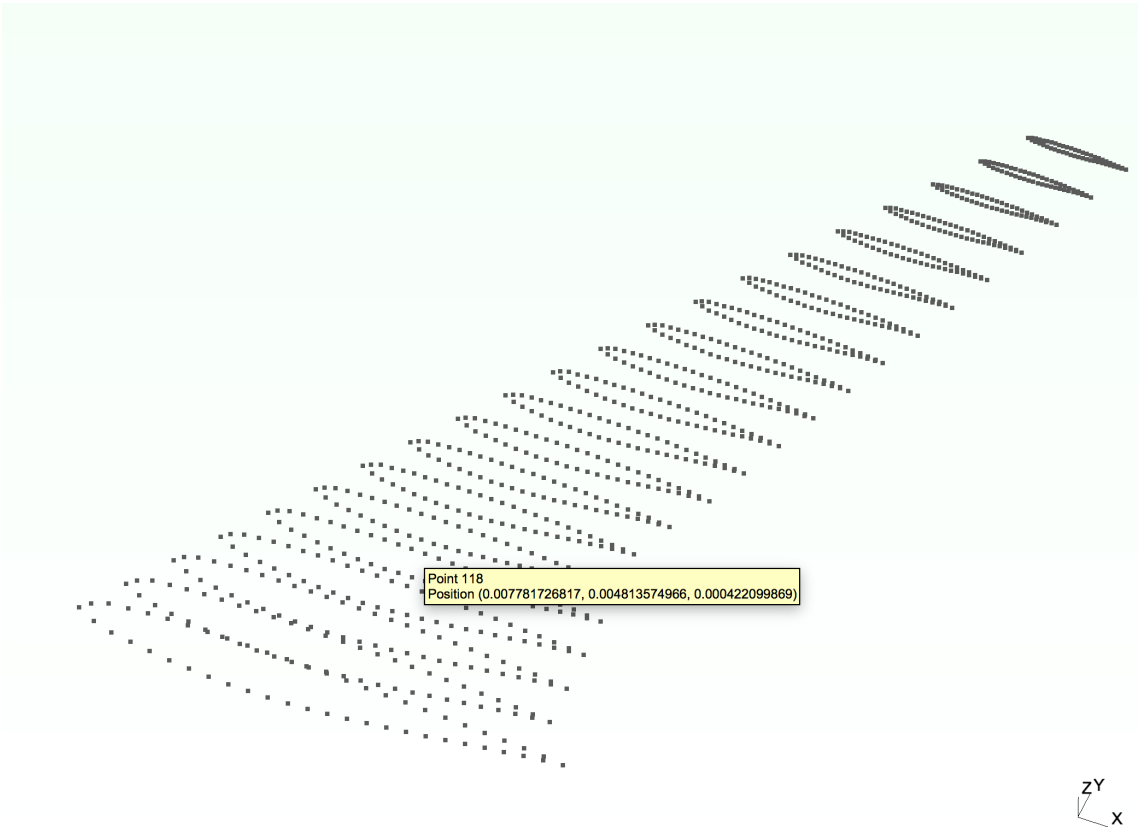


Figure 2.2: Example of .igs file created from the geometry component

2.2.2 Raidial Basis Functions

The second component is the radial basis functions, this component creates the interpolation matrix to couple the aerodynamic and structure component, in fact in the general case these grids are not coincident.The interpolation matrix is created using a fluide-structure interpolation and mesh motion scheme based on the use of radial basis functions (RBF), as the method presented in the work of Rendall and Allen [24]. This displacement and load transfer techniqueis conservative in terms of total load and moment, as shown by Jakobsson and Amoignon. [15] One of the advantages of this type of interpolations is that no mesh connectivity is required between the two disciplines. This is particularly suitable for the cases where the aerodynamic and structural models do not represent the same geometries. Usually, the aerodynamic grid is based on the outer mold line, even though there may be cases where it is based on the mean camber surface only (e.g., the vortex lattice methods).[14]

Once the interpolation matrix **H** have been created, since the dimensions of the problem will not change until the process, it will hold and it will be used each time the aerodynamic component and the structure component run in order to transfer respectively the aerodynamic forces from the CFD mesh to the FEM mesh and the nodal displacement from the FEM mesh to the CFD mesh.

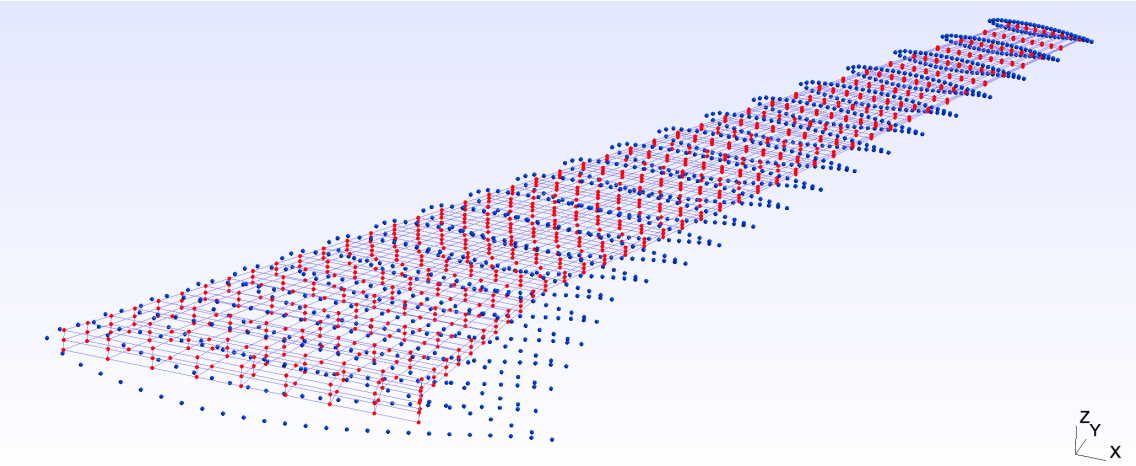


Figure 2.3: Example of structural mesh (red) and aerodynamic mesh (blue)

The input of this component are the outputs of the geometry component, in particular the vector containing the nodes of the CFD mesh X_a and the vector containing the nodes of the FEM mesh X_s. Instead the output will be the interpolation matrix H.

This matrix is able to transform the displacements of the structural nodes into displacements of aerodynamic nodes using the formula :

$$u_a = H \cdot u_s$$

Furthermore, the transpose of this matrix is used to convert the forces of the aerodynamic nodes to the forces of the structural ones with the corresponding formula:

$$f_s = H^T \cdot f_a$$

2.2.3 Aerodynamics

The aerodynamics component as first creates the shape of the deformed wing by adding the deformation on the initial jig shape, after creates the input file to run an

aerodynamic analysis using the external software Panair, than launch Panair, and extracts the aerodynamic characteristics of the wing and stores it into an appropriated dictionary; for each iteration.
Once the jig shape is updated, the component creates an auxiliary file that Panin, the input file compiler for Panair, than run Panin in order to get the input file for Panair in a .wgs format. The aerodynamics loads are computed by means a potential flow panel code, Panair/A502, wich, from an aerodynamic mesh,the value of angle of attack and the Mach, determines the pressure coefficient C_p at the control points of the panel. In order to use the RBF interpolation we need the aerodynamics load on the grid points, so first we use numerical integration of the C_p over the aerodynamic panels and then evenly distributing the total panel forces among the four panel vertexes. A symmetric flow is assumed throughout this work.
At the end of process we have a python dictionary which contains the aerodynamics load for each grid point for that iteration.

2.2.4 Structure

The structure component uses the external software Nastran95 to perform the static, and when is request the modal and/or dynamic, analysis of the wing. As first the component takes as input a nastran template file, where is declared the BEGIN BULK section, a sample for each nastran cards used(GRID for nodes, QUAD for shell elements, FORCE1 for the nodal forces, MAT1 for material proprieties,BAR for rod elements [26]), the mechanical proprieties, the section proprieties, the dictionary containing the coordinates of the structural nodes. Using this parameters the component writes the input files for nastran, where the coordinates of the node from the template file are filled with the dictionary of all structural nodes, and the same for the other nastran card featured in the template file Fig. 2.4. Until the thickness of the shell elements is assumed as design variables also the thickness vector modified from the optimizer is included in the input variables. Whenever the dynamic analysis is requested it's essential define also the data required for the analysis.
When the input file is ready the component launch the nastran software.
As for the static structural analysis to compute the displacements of the wing and

the stresses of the elements a linear finite element model have been used. The equation that must be solved for the finite element analysis is :

$$Ku_s = f_s$$

which is linear with respect to the structural displacement, and the stiffness matric K depends only of the material properties and of the underformed geometry, contained in the jig shape. Nastran uses an LU decomposition of the stiffness matrix K to solve the linear system. So seen as structured the problem the LU decomposition of K doesn't change until the optimization process, than it will be stored, and using a $DMAP$ alteration of the FORTRAN code, it will used at the beginning of each MDA loop, to award a computational cost reduction.

After the analysis is finished the component takes from the output file (.out for displacement, .pnh for stresses, modal shape, frequencies,$V-g$ and $f-g$ diagrams) the value of the required variables and stores it into an dedicated python dictionary. Nastran can be used also to determinate the mass of the wing, that is an important information, especially when mass is used as objective function. As we will see more detailed in the next chapter, this component can be modified also to extract the mass and inertia properties of a section of the wing, using the nastran weight generator instrument and a unitary load ad nodal forces.

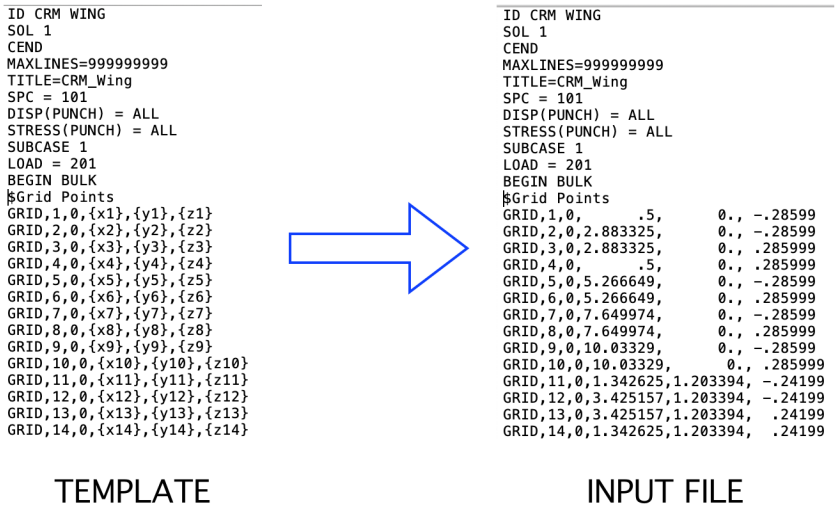


Figure 2.4: Example of nastran template file and input file generated from it

2.2.5 Load and Displacements Transfer

In the MDA loop as we said there is coupling between aerodynamic mesh point displacements u_a, structural node displacement u_s, aerodynamic forces referred on the structural nodes f_s and aerodynamic forces referred on the aerodynamic mesh points.

If the aerodynamic mesh is different from the structural mesh it's necessary to interpolate the structural displacement into the aerodynamic mesh points. The displacement interpolation scheme is based on the work by Rendall and Allen. [24] In that method, each component of the displacement vector $\mathbf{u}$ is interpolated as follows (Eq. 2.1 is written for the x component, but the same holds for y and z):

$$u_x = \sum_{i=1}^{N_s} \alpha_i^x \Phi\left(||\mathbf{x} - \mathbf{x_i}||\right) + \gamma_0^x + \gamma_x^x x + \gamma_y^x y + \gamma_z^x z \tag{2.1}$$

where $\Phi(r)$ is the form of function adopted. In that case, we choose $\Phi(r) = r^2 \ln r$, known as the Thin Plate Spline function (TPS).[14] According to Lombardi et al. [18] who performed a comparison between several available functions, the use of TPS functions is the best and safest option in terms of accuracy of the interpolation. The terms α_i^x are the coefficients of the radial basis functions. Each structural node is the center of an $RBF(\mathbf{x_i})$ and the γ terms are the coefficients of the linear polynomial part. By imposing the interpolating condition on these coefficients (the interpolation function evaluated at the structural nodes must be equal to their known displacements) and by evaluating this same function on the aerodynamic grid points, the transformation matrix between the displacements of the structural and aerodynamic points can be expressed as:

$$u_a = H u_s \tag{2.2}$$

where H is a matrix which depends only on the coordinates of the structural and aerodynamic grid points and the type of RBF chosen.

As detailed by Rendall and Allen,[24] and by virtue of the principle of virtual work to ensure the conservation of energy, we can determine the transformation matrix between the aerodynamic forces on the aerodynamic f_a and structural f_s points. The virtual work can be written as:

$$\delta W = \delta u_s^T \cdot f_s = \delta u_a^T \cdot f_a \tag{2.3}$$

where δu_s and δu_a are the virtual displacements of the structural and aerodynamic grids respectively. Through the displacement interpolation matrix H, we can express the virtual displacements of the aerodynamic grid:

$$\delta u_a = H \delta u_s \tag{2.4}$$

as function of δu_s. By substituting equation 2.4 into equation 2.3 we get that:

$$f_s = H^T f_a \tag{2.5}$$

In the case where gradient-based optimization techniques are used for optimization problems that use the aerostructural coupling presented herein, it can be useful to compute the partial derivatives of the coordinates of the deformed aerodynamic mesh X_a with respect to the structural displacements u_S, as well as the partial derivatives of the aerodynamic forces on the structural nodes f_s with respect to the forces on the aerodynamic grid points f_a.[14]
The deformed aerodynamic mesh is obtained by adding the interpolated displacements (given by equation 2.3) to the jig shape aerodynamic mesh:

$$X_a = X_a^0 + u_a = X_0 + H u_s \tag{2.6}$$

2.3 Driver

In this section is explained how the driver of the problem, the optimization and the aeroelastic coupling driver, is structured. The entirely code is based on the openMDAO structure, so we use the integrated optimization driver for the optimization loop, and also the integrated equation solver for the aeroelastic coupling loop. Let's see in details how each driver works.

2.3.1 Global Optimizer

The global optimizer controls all the process, it will change the design variables in order to minimize the objective function, each time the design variables change it will launch the mda loop until convergence to determinate the correct aerodynamics

loads and structural displacement, and consequently the right stresses, after that it will check if the constraint is respected and the effect of the changes on the objective function, than it will repeat this process until convergence.
To define the optimization driver first thing is to choose the optimizer and set the optimizer option, after need to define the objective function, the constraint and the design variables.

Optimizer

The optimizer chosen for this problem are two:

- **COBYLA**: a gradient free optimizer
- **SLSQP**: a gradient based optimizer (in this case gradient will be calculated using finite differences method)

In the examples chapter optimization using both method on the same model have done, to see the differences between the two optimization method.

Design Variables

In this code several design variables were implemented:

- The angle of attack α
- The twist angle θ
- The mid spar position
- The scale factor
- The area of stringer
- The sweep angle Λ
- The cord factor
- The thicknesses of the different section of the wing inputted as an array *

* The structural design variables consist of the thicknesses of the structural finite elements. The topology of structure remains unchanged. In the FEM model we divided the QUAD elements in groups, all the elements of the same group has the same thickness. How you can see in Fig. 2.5 there are different group for the upper surface and for the longerons, so we collect all the information about the thickness in an array. During the optimization process the thicknesses array can be set as design variable.

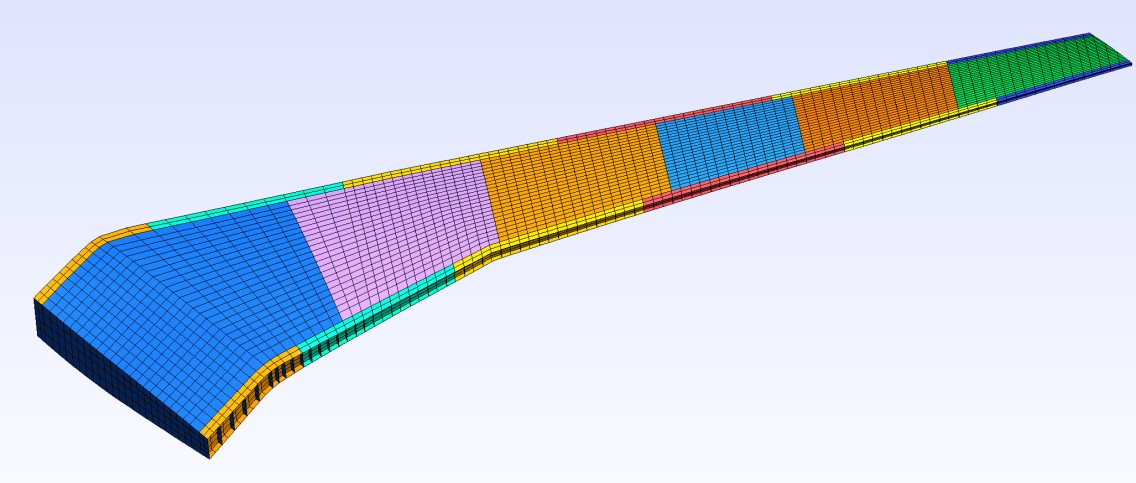


Figure 2.5: Different thickness sections in the CRM FEM model

For each design variable it's possible to set the limits and the initial value. It's possible also to choose just one, a group or all the design variables. In fact in the test cases presented in this work, where the relevant aspect is to test the aggregation component just the angle of attack and the thickness vector is selected as design variables, while all the variables related to the geometry are assumed as constant, so once the geometry is defined it will not change until the process.

Constraint

As for the constraints that were imposed, there is one constraint concerning the lift, where the lift must be at least equal to the weight of the aircraft during the cruise, we constrain the C_L by periodically adjusting of the angle of attack of the aircraft within the aero-structural solution until the desired lift is obtained; to set a constraint like this it's necessry to define an *Executable Component*, a component

of openMDAO where it's possibile to define a function that is updated for each iteration, and after set this function as constraint imposing the lower or the upper limit.

In this case the constraint will be:

$$C_L - \frac{2W}{\rho_a V^2 S_w} > 0$$

The stresses is also considered, so in addition to maintaining the C_L there is also a stress constraint that guarantee the stress of material is lower of the yield stress at the various load condition. The stresses is the result of the structural analysis performed with the finite elements method, so it's necessary to check that the stress of each element respect the constraint, to do this it's necessary to create an *Executable Component* for each element, but usually over than thousand od elements is necessary to describe the structure of the wing, and it becomes computationally very costly to treat these constraints separately. To avoid this problem in the next chapter is explained how we implemented the constraint aggregation.

In this case the constraint will be:

$$\sigma_i - \sigma_{yield} < 0 \quad for \quad i = 1,2,...,N_e$$

where N_e it's the number of finite elements.

Objective Function

In the typical aircraft optimization problem the objective is to find the good trade-off between aerodynamic drag and structural weight. So for our optimization problem we set as objective function the induced drag coefficient C_{D_i} and the structural mass W, or in the general case a function like:

$$F = \alpha C_{D_i} + \beta W$$

where α and β are scalar parameters that is indicative of the relative importance of the variables that we want to minimize.

Is important note that openMDAO is able to manage just minimization problem, so to manage maximization problem is necessary to create an objective function equal to the negative of the real objective function.

2.3.2 MDA Driver

To solve the coupling problem between aerodynamics loads and structural displacements it's necessary an internal loop that through an iterative process which update iteration for iteration the value of the variables implicitly linked reach the convergence. That process is tasked to a non linear equation solver, in this case we choose one of the solver included in the openMDAO library, the non linear Gauss Seidel Solver *NLGaussSeidel* with the aitken accelaration [13].
To do this the MDA driver call the components that we discussed above in order to reach the convergence. In the Fig. 2.6 it's showed the XDSM concerning the MDA loop:

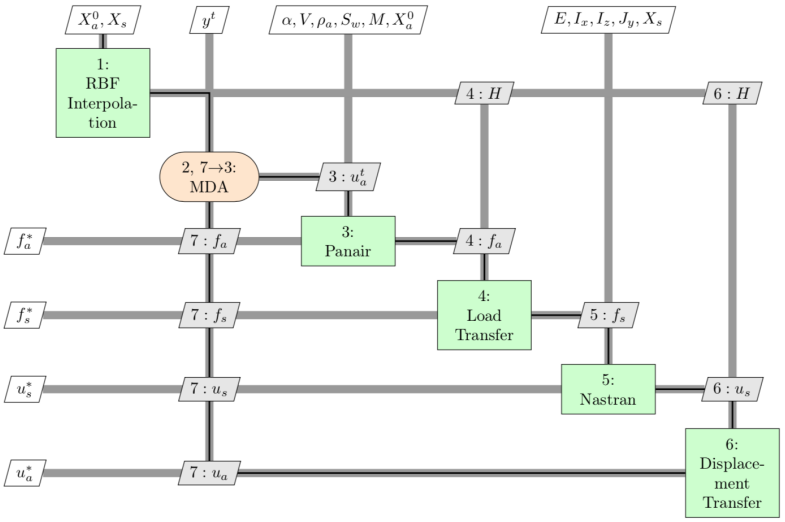


Figure 2.6: XDSM of the MDA Driver

The required parameters for each component is the following, fro:

- RBF interpolation: the aerodynamic mesh X_a^0 and the structural mesh X_s;
- Aerodynamic analysis: the angle of attack α,the flight speed V, the density of the air ρ_a, the wing surface S_w, the Mach M;
- Structural analysis: the Young module E,the barycentric moment of inertia I_x, I_y *and* I_z;
- Interpolation: the interpolation matrix H.

while the coupling variables are u_a, u_s, f_a *and* f_s. The steps to reach the coupling convergence is the following:

1. Compute the interpolation matrix H.
2. Initiate the MDA loop.
3. Compute the aerodynamic forces on the aerodynamic grid points.
4. Compute the aerodynamic forces on the structural grid points.
5. Compute the structural displacements.
6. Compute the aerodynamic grid point displacements

Repeat 3 $\rightarrow$ 7 until convergence. In Fig. 2.7 it's showed how the vertical displacement of the wingtip nodes change until the process, and how the MDA loop reach the convergence.

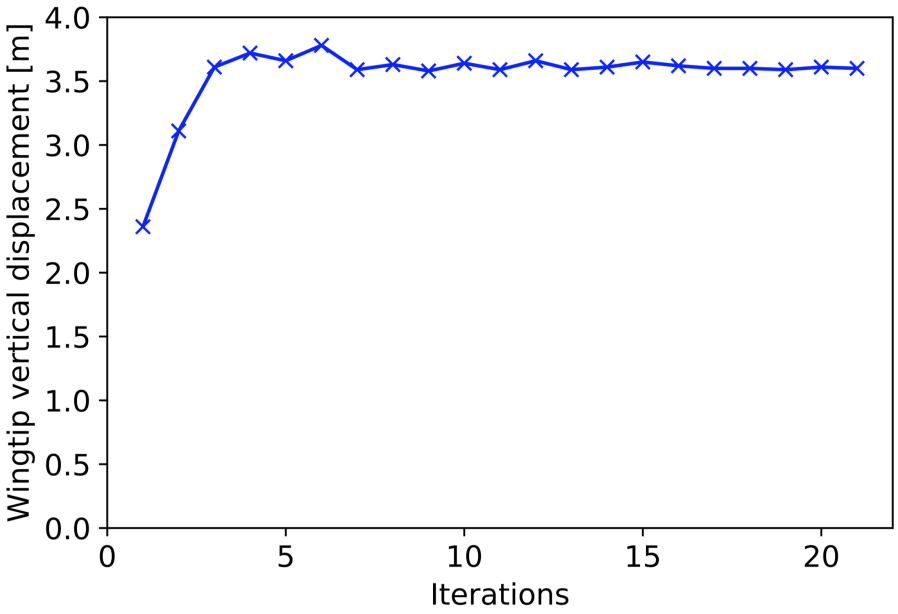


Figure 2.7: Results of the MDA loop refered to the wingtip vertical displacement

For each optimization iteration, the MDA loop determinate the displacements and the aerodynamic loading considering the aeroelastic coupling. Normally to reach the convergence needs 6 or 7 iteration, for each iteration one aerodynamic analysis and

one static structural analysis is performed, so for high fidelity model, aerodynamic and/or structural, the computational cost can be huge, that's the first reason to introduce a reduce model for the MDA loop, showed in the chapter 4.

Chapter 3

Constraint Aggregation

3.1 Aggregation

For aircraft wing design, currently, there is a trend to employ higher-fidelity (more expensive) and multidisciplinary numerical simulations, such as the coupling of CFD code and CSD code. In addition, optimization problems become more and more complex, with many design variables, tightly coupled objectives, and a large number of constraints. Handling large-scale constraints presents a challenge for wing structural and aero-structural design, since the refined finite element model with multiple freedoms incurs millions of structural failure constraints. [30]

In our optimization problem we are also interested to minimize the structural mass subject to constraint on structural failure, in particular we are interested in failure based on yield stress. In the design checking the failure criteria in the optimization process is desirable, because allow to verify that the optimized structure is suitable for the prescribed load conditions.

The primary issue with including failure constraints directly in the structural optimization problem is the resulting size of the optimization problem. Conceptually, for a continuum structure, failure constraints need to be enforced throughout the material domain, leading to an infinite number of constraints. More practically, failure constraints may be enforced element-wise in the finite element model. For detailed, high-fidelity structural models, this can lead to an optimization problem with many thousands or millions of failure constraints. These constraints are costly to enforce

because they can only be checked by completing the structural analysis.[17]
Aggregation methods allow to manage an huge number of constraints, in fact that methods lump a large number of constraint into one global constraint, so the computational cost of the multidisciplinary optimization drastically decrease. These methods consist in the use of an aggregation function, which transform a set of local function values into a scalar function. This scalar function converge in the limit to the maximum local function value:

$$\lim_{P\to\infty} G(\mathbf{f}, P) = \max(f_1, f_2, ..., f_N)$$

where $\mathbf{f} = (f_1, f_2, ..., f_N)^T$ is a vector in which the entries are the local function values, G is the scalar aggregation function and P is the draw-down factor that manage the aggregation.
There are different aggregation function, some of them approximate the maximum of the local function from above, and other from belove. So depending on this characteristic behavior aggregation function forms an upper or lower bound to the maximum local function value.
Related to our problem aggregation function became necessary for the stress failure constraints, in fact when the structural model is detailed, so the number of elements became huge, we need to check the stress criteria on each single element; this entails that need one constraint for each element in the optimization process, with the result that the computational cost became prohibitive, and openMDAO find problem in the setup phase to creates this huge number of constraint, and sometimes it can crash in this phase.Additionally using a gradient-based optimizer the aggregation fucntion can avoid the problem of the not differentiable maximum function, increasing the curvature of the function in the region where there is this problem modifying the draw-down factor. That's one of the reason because we implemented a new component in the optimization process, the aggregation component, so the XDSM of the optimization process is modified as follow:

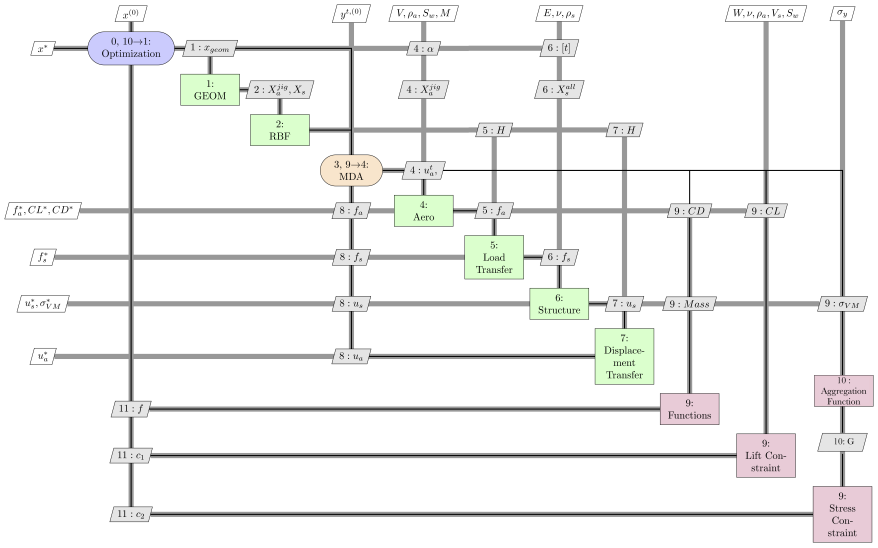


Figure 3.1: XDSM after constraints aggregation

As we can see after the implementation of the aggregation component a new box function have been added, the box function 11. So after the extraction of the Von Mises stresses vector by the structural component the aggregation component take this vector as input and give as output the scalar G. Now the constraint function is connected to$G$ and to σ_{yeild}, so the constraint will be determinate using these variables.

3.1.1 Aggregation Method

Let's see how the aggregation implementation modifies the approach of the optimization design problem, and in particular referred to our problem.
Suppose the original problem is to optimize the mass of a structure under the failure criteria constraint, the problem in this case can be written as:

$$\begin{aligned} &minimize && m \\ &subject\ to && \sigma_{VM_i} < \sigma_{Yield} \qquad \forall\ i = 1,2,...,N \end{aligned}$$

where m is the mass of the structure, $\sigma_{\mathbf{VM}} = [\sigma_{VM_1}, \sigma_{VM_2}, ..., \sigma_{VM_N}]$ is the vector containing the Von Mises stresses of each finite element used for the structural analysis, σ_{Yield} is the yield stress and N is the number of the finite elements.
Let's see how the problem is modified using the maximum constraint approach, the

simplest aggregation method. Using this method we have that the most violated constraint is selected, while the rest are ignored.
In this case the problem can be written as:

$$\begin{aligned} &minimize && m \\ &subject\ to && \max(\sigma_{VM_i}) < \sigma_{Yield} \qquad i = 1,2,...,N \end{aligned}$$

However, because the constraints is not smooth it's not possible to use a gradient-based algorithms in the optimization process, and the search direction is determined by considering only the Lagrange multipliers of the most violated constraint, usually leading to the violation of another constraint in the next iteration.
Using an constraint aggregation function the optimization is subject to the value of the aggregation function G, so the problem can be written as:

$$\begin{aligned} &minimize && m \\ &subject\ to && G < 0 \end{aligned}$$

where G is a scalar, given from the aggregation function and it's representative of the maximum value of the Von Mises stresses vector. G depends, as we said, also to the aggregation function choose, so in the next paragraph we will introduce the most common aggregation function, and theirs properties.

3.2 Aggregation Functions

In literature several aggregation function have been used. The choice of the function depends of the nature of the problem, of the size of the problem and of the properties required to perform the analysis. For the failure criteria, where the constraint impose the stresses lower than yield stress, the natural choice for these constraint aggregates is a maximum- or minimum-value function. But the problem it's that this kind of functions are not differentiable, so cannot be used efficiently in gradient-based optimization. Another choice can be the **p-mean** and **p-norm** function, or the **Kreisselmeier-Steinhauser KS** function, kind of smooth estimator. These estimators do not precisely reproduce the true feasible design space provided by the

original constraints, so the final design determined by the optimizer will be different depending on the aggregation scheme. [17]

3.2.1 Function

First, we briefly discuss aggregation function used in the literature.
About the nomenclature we use G to indicate the aggregation function, the superscript $_L$ and $_U$ to denote respectively an lower- or upper-bounded aggregation function, and P to indicate the draw-down factor.
The first two aggregation function that we analize is the **p-norm** and the **p-mean** function. This kind of aggregation function can be use when the local function value **f** are non-negative, as in our case where **f** is the Von Mises stresses vector, so for definition we have that all the component of the vector, the Von Mises stress of each finite element, is positive.

P-norm

$$G_{PN}^{U} = \left(\sum_{i=1}^{N} f_i \right)^{1/P} \tag{3.1}$$

P-mean

$$G_{PM}^{L} = \left(\frac{1}{N} \sum_{i=1}^{N} f_i^P \right)^{1/P} \tag{3.2}$$

The difference between these two aggregation functions is that the P-norm is an upper bound, and the P-mean is a lower bound to the maximum local function value:

$$G_{PM}^{L} \leq \max(f_1, f_2, ..., f_N) \leq G_{PN}^{U}$$

For this propriety it's important to note, in the implementation in an optimization process, that the p-norm function is conservative, backwards the p-mean is not conservative.

Kreisselmeier-Steinhsauser function

The Kreisselmeier-Steinhsauser aggregation function have been presented for the first time by G. Kreisselmeier and R. Steinhsauser [16].The function constains a "draw-down" factor or aggregation parameter, P, which is analogous to the penalty factor in penalty methods used to perform constrained optimization [19].
This function have been used first to aggregate multiple objectives and constraints into single functions, but in the time became popular to be used in the direct constrained optimization problem. This function has few important proprieties, about we will discuss after, but ne of the more important property is the following: "The function produces an envelope surface that is $C1$ continuous and represents a conservative estimate of the maximum among the set of functions" [28].
The KS function can be used also to aggregate only the constraints into a single continuous function, in that way is defined as:

$$G_{KS}^{U} = \frac{1}{P} \ln \left(\sum_{i=1}^{N} e^{P f_i} \right) \tag{3.3}$$

This function, as the superscript denote, is an upper bounded function, for each $P > 0$, so it overestimates the maximum local function value. Let's see the properties of this function:

$$\begin{aligned}
G_{KS}(x, P) &\geq \max(f_i(x)) && for && P > 0 \\
\lim_{P \to \infty} G_{KS}(x, P) &= \max(f_i(x)) && for && j = 1,2, ..., N \\
G_{KS}(x, P_1) &\geq G_{KS}(x, P_2) && for && P_1 > P_2 > 0 \\
G_{KS}(x, P) & && is\ convex,\ if\ and\ only\ all\ constraint\ are\ convex
\end{aligned}$$

These properties is really important for the KS function to be considered an valid aggregation function. The first three properties indicate, as we said, that the KS function is upper-bounded, it overestimates the maximum of the constraint defined as $G_i \leq 0$, a positive value returned indicate that a constraint is violated or close to being violated. The conservative nature is determined from the aggregation parameter P, id P increases the aggregation function value became more closer to the maximum value. The effect of the aggregation parameter will be investigate in

the next paragraph, but in Fig. 3.2 we can see that effect from the investigation done by J. R. R. A. Martins, for a two constraints problem, g_1 and g_2: [19]:

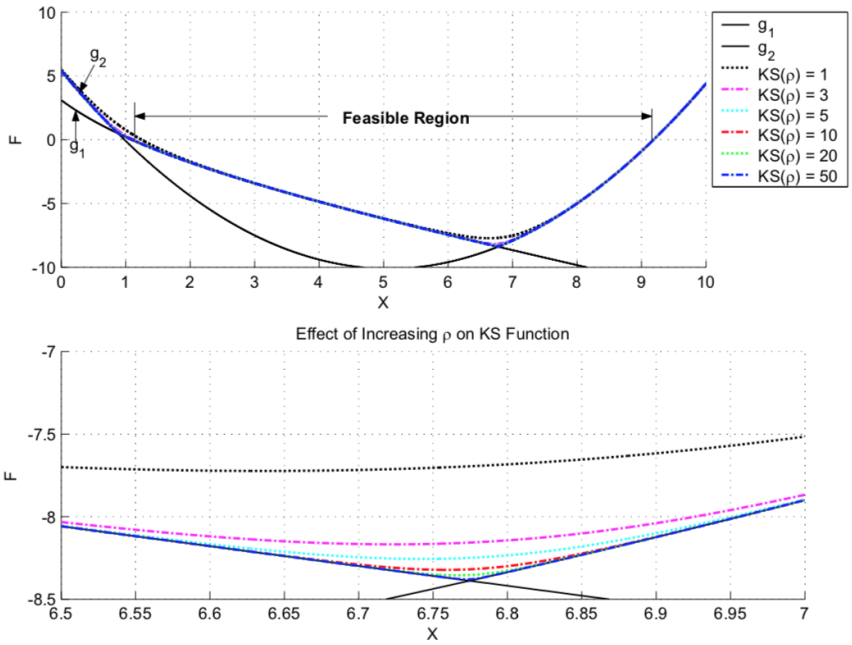


Figure 3.2: Effects of the aggregation parameter P on the KS function

An alternative formulation can be used, to avoid numerical difficulties caused by numerical overflow, the KS function can be written as:

$$G_{KS}^{U} = f_{max} + \frac{1}{P}\ln\Big[\sum_{i=1}^{N} e^{P(f_i - f_{max})}\Big] \tag{3.4}$$

where f_{max} is the max value of the local function.

The maximum difference between the maximum value of the local function and the value of the aggregation function is determinate by P, and it's value its:

$$\frac{1}{P}\ln\left(Ne^{Pf_{ma}}\right) - f_{max} = \frac{1}{P}ln(N) \tag{3.5}$$

that's mean:

$$f_{max} < G_{KS} < f_{max} + \frac{1}{P}ln(N) \tag{3.6}$$

From this property can be obtained an alternative formulation of the KS function, a lower bounded KS function. To obtain it we need to subtract the maximum difference between the maximum of the local function to the KS upper bounded

function, obtaining :

$$G_{KS}^{L} = G_{KS}^{U} - \frac{1}{P} ln(N) = \frac{1}{P} ln \left(\frac{1}{N} \sum_{i=1}^{N} e^{P f_i} \right) \tag{3.7}$$

that became the following using the alternative formulation:

$$G_{KS}^{L} = f_{max} + \frac{1}{P} \ln [\sum_{i=1}^{N} e^{P(f_i - f_{max})}] - \frac{1}{P} ln(N) \tag{3.8}$$

3.2.2 Effects of the Aggregation Parameter P

The aggregation parameter or draw-down factor P has several effects on the aggregation function. Choose it correctly is really important to have low relative error of aggregation function relatively at the real maximum of the local function, and also a low computational cost. The right choice of the aggregation parameter depends as first of the problem size, but also from the dispersion of the value of the local function and their absolute value. In addiction is important to choose the draw-down factor to smooth the aggregation function in the region where the constraints intersect, to be possible to compute the derivatives.

As first we can see in Fig. 3.3 how the size of the problem influence the relative error for a given draw-down factor. We use the aggregation function to aggregate the Von Mises stresses vector, so the value of the aggregation function is representative of the maximum stress for the finite elements of the structure. In this case we choose a Kreisselmeier-Steinhsauser function lower bounded, we set the draw-down factor as 50, then we compute the relative error as:

$$\epsilon = \left| \frac{G_{KS}^{L} - f_{max}}{f_{max}} \right|$$

Than we compute the relative error for different size of the stresses vector. Locking the draw-down factor we can see how the relative error grow with the size of the problem, or analogously with the number of the constraints that we want to aggregate.

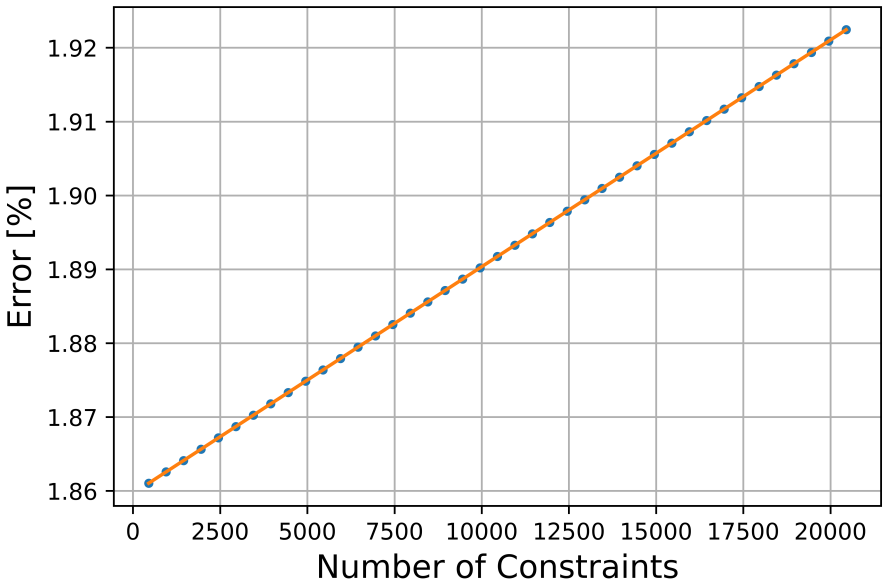


Figure 3.3: Effects of the number of constraint on the relative error of the aggregation function

The choice of aggregation parameter is also constrained to the aggregation function and to the maximum relative error. In Fig. 3.4 there are showed how the relative error changes with the aggregation parameter for the different aggregation function implemented in the code. To obtain this graph we take one Von Mises stresses vector as example, and we determinate the aggregation function value for different value of the aggregation parameter, then we compare the aggregation function value with the maximum of the stress vector obtained with the maximum-value function:

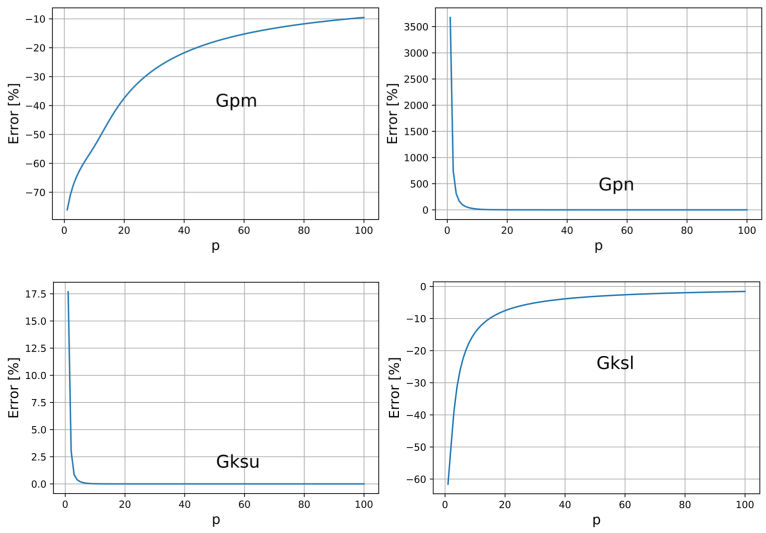


Figure 3.4: Effects of the aggregation parameter P on the relative error

From these graphics we can saw the difference between a lower bounded function (G^L_{KS} and G^L_{PM}) or an upper bounded function (G^U_{KS} and G^U_{PN}), in fact as we can see in the first case the relative error converge to 0 from the bottom, and in the second case from the top. As we can see, as definition, for $p \to \infty$ the relative error go to 0, and that error depends from the aggregation function, the relative error is acceptable for $P \approx 50$ for the upper bounded function, while for the lower bounded the error is still not acceptable. "$P = 50$ is usually a reasonable value that has a maximum relative error of ≈ 0.03 for two constraints, and is often used"[20].
In Fig. 3.5 it's showed an zoom-out to see also the upper-bounded function reach the convergence.

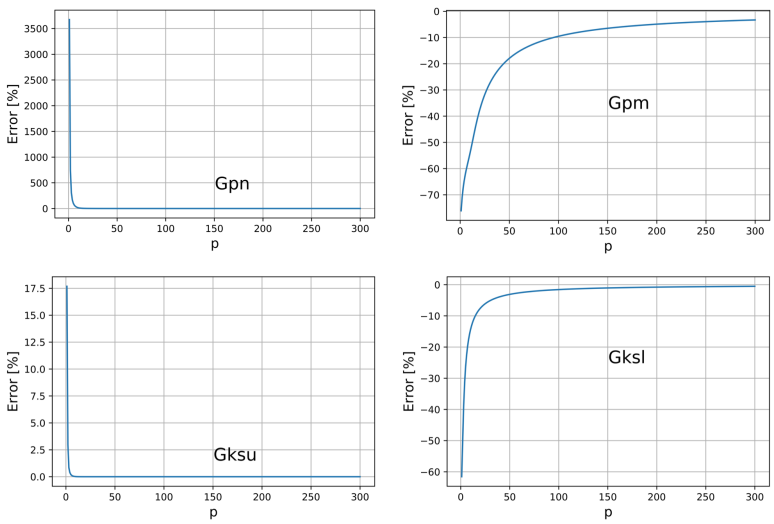


Figure 3.5: Zoom-out of effects of the aggregation parameter P on the relative error

3.3 Aggregation Component

To implement the constraint aggregation in the optimization code needs a component that take as input the Von Mises stresses vector and give as output the value of the aggregation function. To be implemented in our script the component should be written in the openMDAO structure. In Fig. 3.6 is represented the flow chart of the component.

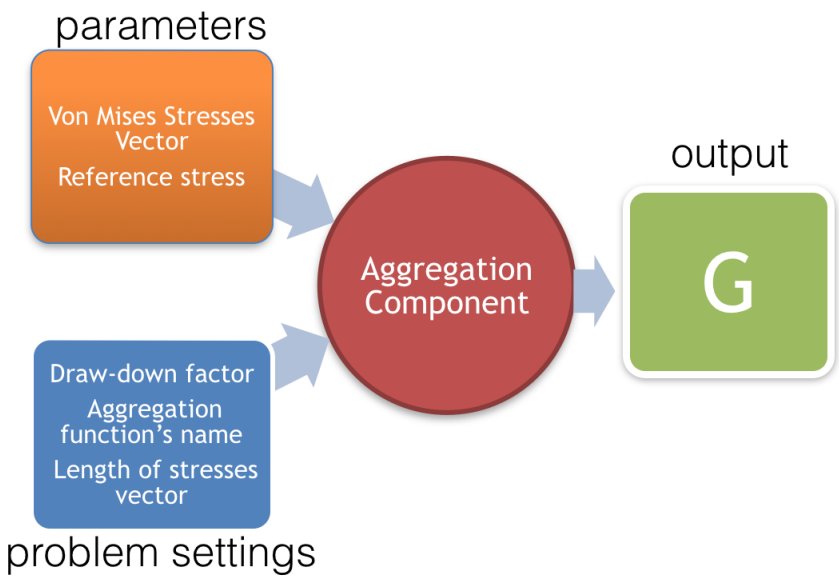


Figure 3.6: Flow chart of the aggregation component

The aggregation component takes as input parameters the Von Mises stresses vector $\sigma_{\mathbf{VM}}$ and the reference stress σ_0 used for the nondimensionalization, connected to the relative variables of the main code between the command *promotes=[*]*; the settings of the aggregation are declared in the main script and are input for the component, there are the value choose for the draw-down factor, the aggregation function selected and the dimension of the vector. Than the component compute the aggregation function and give as output it's value, joined to the constraints functions to be used during the optimization process.

In Fig. 3.7 it's showed how the component is made, in the openMDAO structure and the links between the variables:

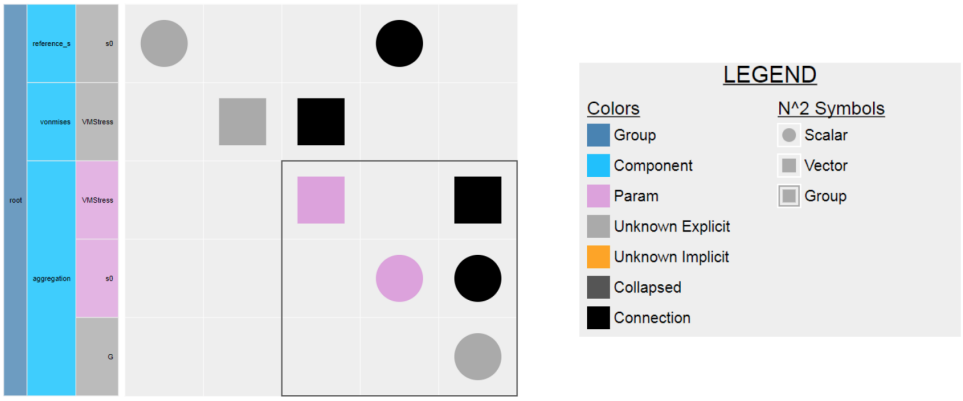


Figure 3.7: openMDAO structure of the aggregation component

Chapter 4

Reduced Model for MDA Loop

4.1 Introduction to the Reduced Model

The aerolastic analysis is one of the costly operation in the optimization process, in fact many computation must be performed, with different software; to perform an aerolastic analysis a static structural analysis, an aerodynamic analysis and a dynamic structural analysis, for each iteration is required. So to contain the cost of the analysis models with a low computational cost are needed. In parallel with the multidisciplinary design optimization process, Josè Serralta and Dimitrios Glenis, work to the project which the objective is to perform a model reduction from a complex 3D geometry to a simpler one with a computational cost compatible with the one required for the optimization process and yet a sufficient accuracy for a satisfactory preliminary design, as is illustrated in their report [25]. A modal analysis will be performed on both the complete and the reduced model, to check that their dynamic are indeed similar. Then, the reduced model will be used for the static, dynamic and aeroelastic analysis in the optimization loop.

In the preliminary design phase simplified beams models, knowns as stick models, are used to perform static and dynamic aeroelastic analysis, to have some preliminary results without high cost. During the last stages of the design, high fidelity detailed 3D FEM models are used for design validation and optimization, but this kind of models are too expensive for the MDO process. Thus, condensed models are created for dynamic simulations, using reduction techniques to develop models which are

sufficiently simple but sophisticated enough to predict the dynamic behavior [22]. Guyan reduction technique was one of the first to appear, it is one of the most popular condensation methods and it is included in many commercial FEM codes [29].

4.1.1 Approach for the Reduced Model

In Fig. 4.1 are showed the different modeling levels for a wing. As we can see the structure of the wing can be represented with a box-like model, where the semi-monocoque structure, composed by discrete stiffeners, like stringer, and thin walled panels, like the cover skin and ribs, is modeled and meshed using FEM elements, like beam and shell elements. Another approach to model the wing is using a beam-like model, where each beam element represent the stiffness properties of an associated wing strip.
In literature two approaches are considered:

- Reduction techniques, such as guyan or IRS [10], can be used to reduce the original 3D FEM model with N degree of freedom to one which a much smaller number of nodal points $M << N$. The DoF of the reduced model are referred as masters, and the deleted ones as slaves. The selection of the master nodes is very important, although there exist iterative methods which make their selection much less critical [11]. The result of condensation is still a 3D model, but with less nodes.

- Generating a stick model by extracting the stiffness properties of the full 3D FEM model and applying them to a set of beam elements extending along the structure's elastic axis [4]. In this case the full model is not reduced, but a new equivalent FEM model is built.

The reduction techniques are usually used for dynamic analysis, while for the aeroelastic studies the condensation to stick models is more used, so for this project the second approach is adopted.
So the main idea for the reduced model is to start from the detailed FEM model

with the shell elements, perform an static analysis with exploration load, and from that analysis compute the stiffness and inertial properties of a section, then for each section one beam elements are associated to one section, and we set its stiffness and inertial properties the properties of the respective section.

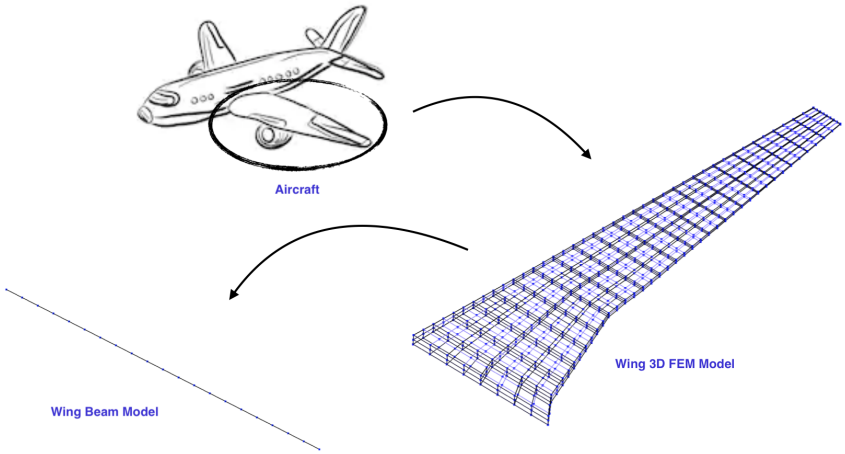


Figure 4.1: Different modeling levels

4.2 Structure of the Code

In this part we will see how the stick model is obtained from the full model, and how the validation of the stick model is performed. In Fig. 4.2 is represented the flow chart of the reduction model:

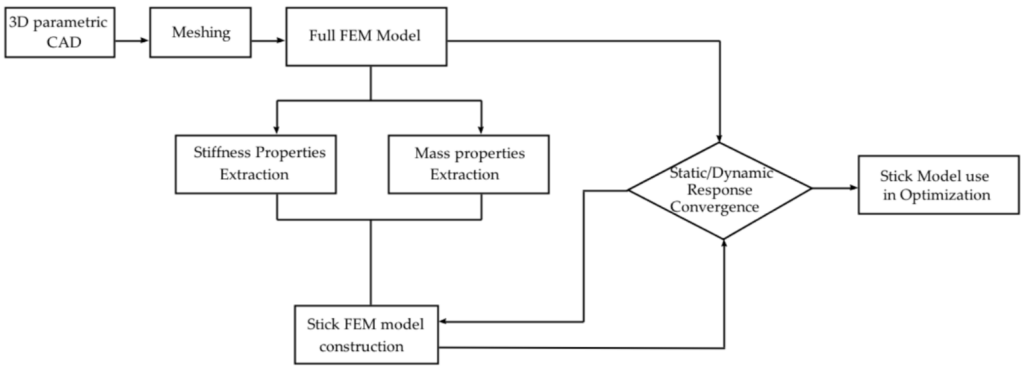


Figure 4.2: Flow chart of the reduced model code

The Full FEM model is created using the geometry component, that we discuss in the chapter 2. From the full FEM model the mass and stiffness properties of the section, which we split our model, are extracted using the Nastran Weight Generator instrument. Then the stick model be created, importing the mass and stiffness properties of each section in one beam element. The stick model need now to be validated, one static analysis and one modal analysis will be performed on the stick model, then the result will be compared to the result of the same analysis performed on the full model. While the results is not acceptable some corrective coefficient will be changed until the convergence of the relative error is guaranteed. When the stick model is validated, it can be used in the optimization code, in particular in the MDA loop, to obtain a gain on the computational cost of this operation.

4.2.1 Creation of the FEM model

To create the FEM model of the stick model the procedure it's the same of the creation of the FEM model of the full model. As we saw in the chapter 2, after the geometry component creates the *.igs* file, and after the program gmsh create the FEM mesh, the structure component write the bfd file, putting the properties of each elements in a NASTRAN card, from a template *.bdf* file. In this case we use another *.bdf* template, where there are just BAR elements (the beam elements in the NASTRAN95 language). For this part just the number of the elements, equal to the number of the section used to split the full model, and the position of the nodes are required. For each element also a lumped mass will be created, and for the moment it's value is 0 and it's located in the first node of the element, when we will have the information about the mass propierties, using the offset command we will change the effective position of the lumped mass. At the end of this process we will have a *.bdf file* of a stick model, that present N BAR elements and N lumped mass, but without information about the material and inertia properties. In Fig. 4.3 are schematized what this component do:

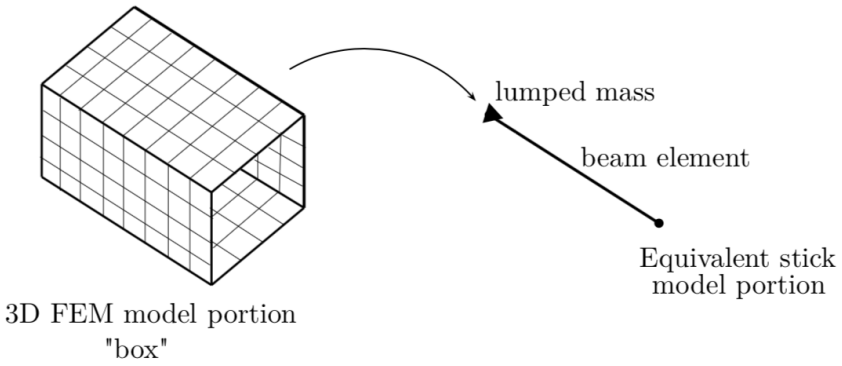


Figure 4.3: Schematization of the creation of the stick model

4.2.2 Extraction of Stiffness and Inertia Properties

Starting from the 3D FEM model of the structure, we split it into section called boxes, which could be delimited by the ribs of the wing. The object is to build a 1D model in which each box is represented from an equivalent beam element. To do this as first we need to determinate the stiffness properties (A, I_y, I_z, J) and the inertia properties (m, x_G, I_G) to give to the beam element so it's representative of the box.

Stiffness Properties

The flexibility matrix **C** of each box is obtained from the displacements due to a set of unitary loads applied to one of its sections. [25].

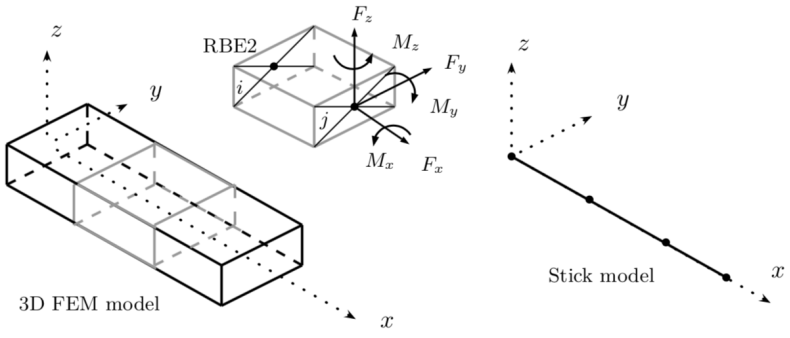


Figure 4.4: Detailed model subdivisions and unitary loads for the extraction

In Fig.4.4 is showed a box with the 2 associated nodes of the reduced model (nodes i and j). To compute the flexibility matrix $\mathbf{C}$ a set of six unitary loads are applied at node j and the consequent displacements are measured at nodes i and j.

Let $\mathbf{C_i}$ and $\mathbf{C_j}$ be the flexibility matrices relating loads at j with displacements at i and j respectively. If a unitary load vector $\mathbf{F}$ is applied, then from $C\dot{F} = U$ the associated displacements u provides a column of matrix $\mathbf{C}$. In this way, with the displacements from the six unitary loads, the six columns of matrices $\mathbf{C_i}$ and $\mathbf{C_j}$ are built. [25]

The $\mathbf{C_i}$ and $\mathbf{C_j}$ are the flexibility matrices relating loads at node j with displacement at nodes i and j respectively. We can obtain the flexibility matrix of the box from the forces and strains. Strains are approximated obtained from these relations:

$$\epsilon_x = \frac{u_{x_j} - u_{x_i}}{\Delta x}; \qquad \gamma_y = \frac{u_{y_j} - u_{y_i}}{\Delta x}; \qquad \gamma_z = \frac{u_{z_j} - u_{z_i}}{\Delta x}$$

$$\kappa_x = \frac{\theta_{x_j} - \theta_{x_i}}{\Delta x}; \qquad \gamma_y = \frac{\theta_{y_j} - \theta_{y_i}}{\Delta x}; \qquad \gamma_z = \frac{\theta_{z_j} - \theta_{z_i}}{\Delta x}$$

where u_{x_j} is the displacement in x-direction at node j, θ_{x_j} is the rotation in x-direction at node j, and the same for the others, Δx is the length of the box. So the flexibility matrix of the box will be:

$$\mathbf{C} = \frac{\mathbf{C_j} - \mathbf{C_i}}{\Delta x}$$

from that we can obtain the stiffness matrix of the box from:

$$\mathbf{K} = \mathbf{C^{-1}}$$

From the stiffness matric we can obtain the section proprieties, from the following relation:

$$\begin{pmatrix} F_x \\ F_y \\ F_z \\ M_x \\ M_y \\ M_z \end{pmatrix} = \begin{bmatrix} A_{11} & 0 & 0 & 0 & A_{12} & A_{13} \\ 0 & S_{11} & S_{12} & S_{13} & 0 & 0 \\ 0 & S_{21} & S_{22} & S_{23} & 0 & 0 \\ 0 & S_{31} & S_{32} & S_{33} & 0 & 0 \\ A_{21} & 0 & 0 & 0 & A_{22} & A_{23} \\ A_{31} & 0 & 0 & 0 & A_{32} & A_{33} \end{bmatrix} \begin{pmatrix} \epsilon_x \\ \gamma_y \\ \gamma_z \\ \kappa_x \\ \kappa_y \\ \kappa_z \end{pmatrix}$$

Where A_{ij} and S_{ij} are the stiffness terms associated to axial and shear sets. So from the constitutive equation it's possible to determinate the section properties:

$$\mathbf{A} = \begin{bmatrix} A_{11} & 0 & 0 \\ & A_{22} & 0 \\ sym & & A_{33} \end{bmatrix} = \begin{bmatrix} EA & 0 & 0 \\ & EI_y & 0 \\ sym & & EI_z \end{bmatrix}$$

$$\mathbf{S} = \begin{bmatrix} S_{11} & 0 & 0 \\ & S_{22} & 0 \\ sym & & S_{33} \end{bmatrix} = \begin{bmatrix} GA_y & 0 & 0 \\ & GA_z & 0 \\ sym & & GJ \end{bmatrix}$$

From this it's possible to extract the section properties A, I_y, I_z, J to give to the beam element to be representative of the box section.

Inertia Properties

The inertia properties of each box are extracted using the *NASTRAN grid point weight generator* [26], which generate an output file containing the information relative to the position of the center of gravity X_G, the mass m and inertias I of the box. In Fig. 4.5 is showed an example of the output file. From this information we will change the position and the properties of the lumped mass that we create in the first phase. Since the mass of the boxes are represented from the lumped mass, the density of the material is set to 0.

```
                O U T P U T   F R O M   G R I D   P O I N T   W E I G H T   G E N E R A T O R
                                      REFERENCE POINT =      7000

                          MO - RIGID BODY MASS MATRIX IN BASIC COORDINATE SYSTEM
***                                                                                                  ***
*  1.64255236D+02 -4.39940442D-17 -1.83849163D-19 -2.31803818D-18  3.29209999D-04 -3.98045207D+03 *
*  1.23931932D-16  1.64255236D+02 -3.61993559D-17 -3.29209999D-04  6.93383328D-18  2.77461977D+01 *
*  1.85051884D-19 -2.81999503D-17  1.64255236D+02  3.98045207D+03 -2.77461977D+01 -8.12128098D-18 *
*  6.30551518D-18 -3.29209999D-04  3.98045207D+03  9.64954788D+04 -6.69318595D+02 -6.25941537D-05 *
*  3.29209999D-04  1.58992725D-18 -2.77461977D+01 -6.69318595D+02  6.76956221D+01 -7.89638253D-03 *
* -3.98045207D+03  2.77461977D+01 -4.38995655D-18 -6.25941537D-05 -7.89638253D-03  9.65629663D+04 *
***                                                                                                  ***

                                S - TRANSFORMATION MATRIX FOR SCALAR MASS PARTITION
                                ***                                                  ***
                                *  1.00000000D+00  0.00000000D+00  0.00000000D+00 *
                                *  0.00000000D+00  1.00000000D+00  0.00000000D+00 *
                                *  0.00000000D+00  0.00000000D+00  1.00000000D+00 *
                                ***                                                  ***

      DIRECTION
MASS AXIS SYSTEM (S)       MASS              X-C.G.          Y-C.G.          Z-C.G.
        X            1.642552364D+02     -1.411241572D-20  2.423333443D+01  2.004258774D-06
        Y            1.642552364D+02      1.689212372D-01  4.221377314D-20  2.004258774D-06
        Z            1.642552364D+02      1.689212372D-01  2.423333443D+01 -4.944305677D-20

                                I(S) - INERTIAS RELATIVE TO C.G.
                             ***                                                  ***
                             * 3.585248512D+01 -3.064294303D+00  6.983593468D-06 *
                             *-3.064294303D+00  6.300870002D+01 -8.147346814D-05 *
                             * 6.983593468D-06 -8.147346814D-05  9.865314930D+01 *
                             ***                                                  ***
                                          I(Q) - PRINCIPAL INERTIAS
                             ***                                                  ***
                             *  6.335017954D+01                                    *
                             *                  3.551100559D+01                    *
                             *                                  9.865314930D+01  *
                             ***                                                  ***
                                   Q - TRANSFORMATION MATRIX - I(Q) = QT*I(S)*Q
                             ***                                                  ***
                             *  1.107526588D-01  9.938480007D-01  0.000000000D+00  *
                             * -9.938480007D-01  1.107526588D-01  0.000000000D+00  *
                             *  0.000000000D+00  0.000000000D+00  1.000000000D+00  *
                             ***                                                  ***
```

Figure 4.5: NASTRAN grid point weight generator output file

At the end of the process the beam element has the stiffness and inertia properties of the box, as the Fig. 4.6 shows:

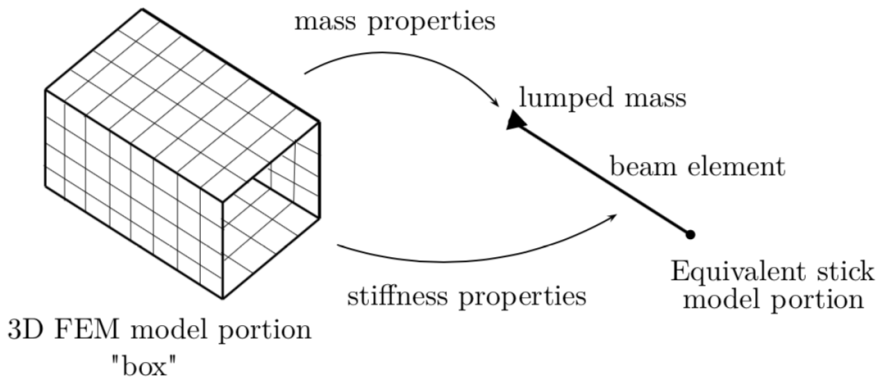


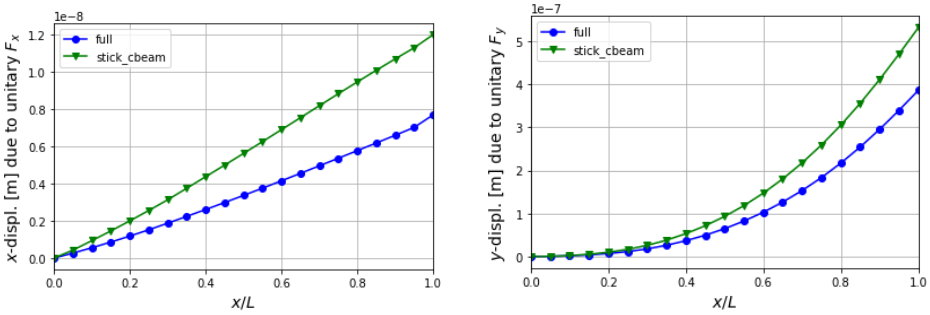
Figure 4.6: Schematization of the creation of the stick model and importing of the properties

4.3 Validation of the Stick Model

For the validation of the stick model two comparison have been done, one on the static response results, in terms of displacement and rotation of the nodes, and one on the modal analysis, in terms of natural frequencies and modal shapes. In both the cases to allow the comparison, since the number of nodes and their position is different, a series of nodes have been added in the full model, in particular we add in the full model the same nodes used for the stick model in the same relative position, then we connected these nodes to the structural nodes of the full model between rigid connection. In that way we can comparison the displacement, the eigenvector and the eigenvalue of the two different models.

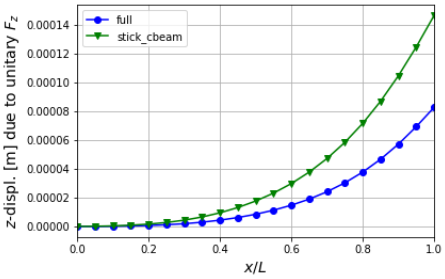
4.3.1 Comparison of the Static Response

To do the comparison of the static response an exploration load have been applied at the wing tip. To determinate the displacements and the rotation in all the direction 6 different load have been applied, 3 force of 1 N applied on the x, y and z direction, F_x, F_y and F_z, and 3 moments of 1 $N\ mm$, ; M_x, M_y and M_z.
In the Fig. 4.7 there are the results obtained from the comparison.
As we can see the errors on the displacements are $\approx 15\%$ at the wingtip, while the errors on the rotation are smaller, $\approx 10\%$, and $\approx 1\%$ on the horizontal rotation θ_y.

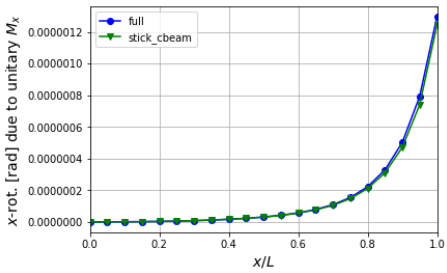


(a) Longitudinal displacement u_x

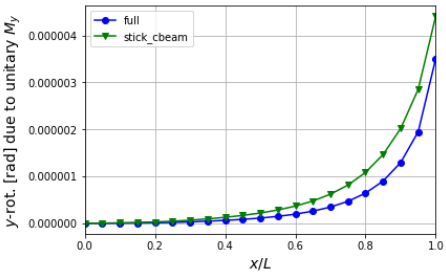
(b) Torsional rotation θ_x



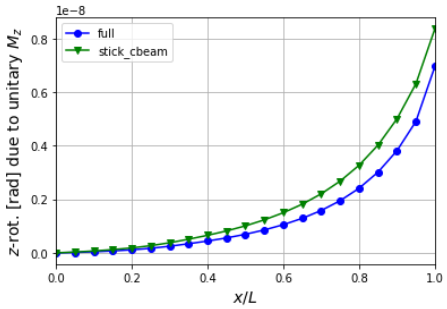
(c) Horizontal displacement u_y



(d) Horizontal rotation θ_y



(e) Vertical displacement u_z



(f) Vertical rotation θ_z

Figure 4.7: Comparison of the static response

To get better results it's possible to adopt a correction on the Young module E and on the shear module G, using virtual modules obtained from the relative error on the displacements and rotation:

$$E_{corr} = \frac{u_{z|stick}}{u_{z|full}} E; \qquad G_{corr} = \frac{\theta_{x|stick}}{\theta_{x|full}} G$$

4.3.2 Comparison of the Modal Properties

To do the comparison of the modal properties we compare as first the modal shapes of the two model, and after the natural frequencies. Also in this case the extra nodes

are used in the full model to have a compatibility for the two modal shapes. To compare the modal shape the Modal Assurance Criterion **MAC** have been used.

Modal Assurance Criterion MAC

The function of the modal assurance criterion (MAC) is to provide a measure of consistency between estimates of a modal vector. [8] The Modal Assurance Criterion is a vector correlation index frequently used in experimental dynamics to quantify the similarity of mode shapes[9].
This criterion is based on the computation of a normalized scalar product of the eigenvector of the system given by:

$$MAC(\mathbf{\Phi_1}, \mathbf{\Phi_2}) = \frac{(\mathbf{\Phi_1^T} \cdot \mathbf{\Phi_2}^T)}{||\mathbf{\Phi_1}||^2 \cdot ||\mathbf{\Phi_2}||^2} \qquad 0 \leq MAC \leq 1$$

where $\mathbf{\Phi_1}$ and $\mathbf{\Phi_2}$ are the eigenvector related to the modal shape that we want to compare. A value of 1 means that the modal shapes are identical, while a value of 0 means that the modal shapes are not similar at all. In our case we are interested to check the similarity of the first ten modal shapes, to do this we use the eigenvector output of the modal analysis performed with NASTRAN. For each node 6 DoF are present, so we put the vector related to each singol Dof in a single expanded vector to compare them:

$$\mathbf{\Phi} = [\mathbf{\Phi_{TX}}, \mathbf{\Phi_{TY}}, \mathbf{\Phi_{TZ}}, \mathbf{\Phi_{RX}}, \mathbf{\Phi_{RY}}, \mathbf{\Phi_{RZ}}]$$

Then 100 comparison of the modal shapes have been done, and the results can be visualized using a matrix with a colour scale, as is showed in Fig. 4.8.

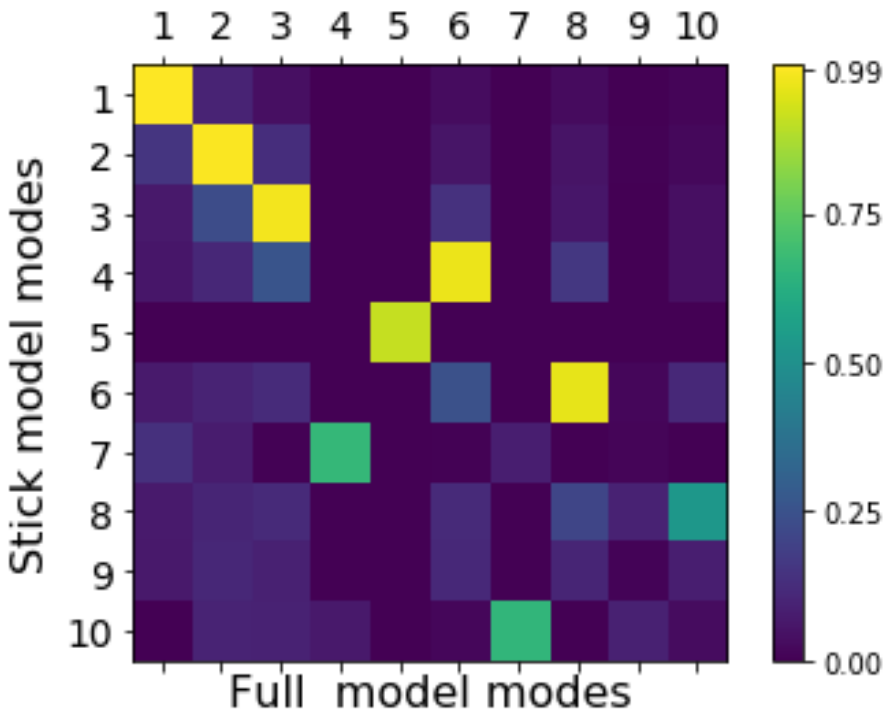


Figure 4.8: MAC colour matrix for the first ten modal shapes

A value of MAC ≈ 1 means the modal shape of the i mode of the full structure $\mathbf{\Phi_i}$ is similar to the modal shape of the j mode of the stick structure $\mathbf{\Phi_j}$.
As we can see there is a large compatibility on the full bending modes, while the compatability of the torsional and coupled modes is lower.

Frequencies Comparison

As we said also a comparison on the first ten natural frequencies have been done. It's important to remark that the frequencies are ordered in ascending order by NASTRAN, so when the error on the frequencies is more than the step from two consecutive frequencies the frequency i of the full model cannot be associated at the same mode of the frequency i of the stick model. To avoid this problem it's required to order the frequencies using the modal shape order. To order the frequencies a visualization of the modal shapes is required, than we can recognize the type of the mode and associate the correct number to the mode to do the comparison.
In Fig. 4.9 there is the bar plot of the error between the full model natural frequencies and the stick model natural frequencies relative to the same modal shape:

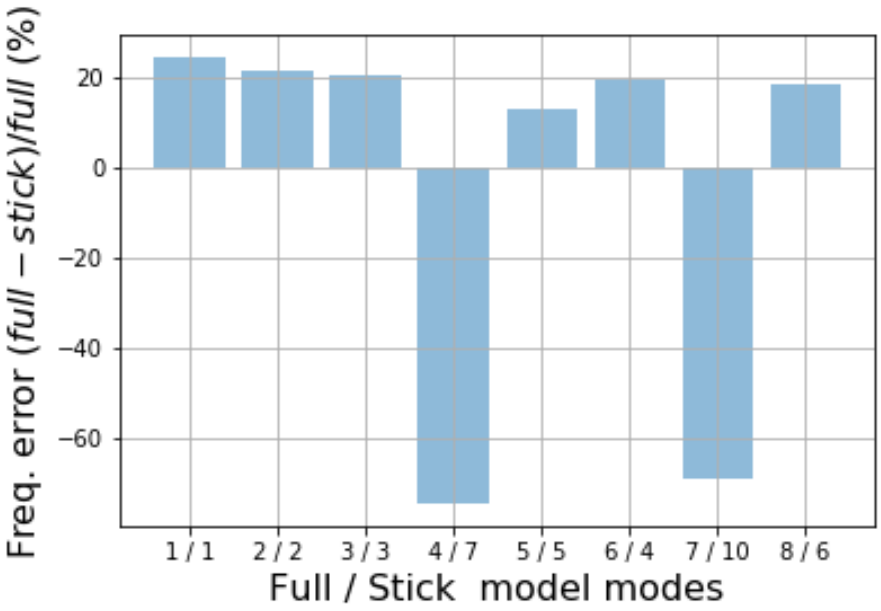


Figure 4.9: Bar plot of the error between the natural frequencies of the full and stick model

In Tab. 4.1 there are the value of the first ten frequencies, ordered for the same modal shape for both model, and the relative error computed as:

$$E = \frac{F_{i_{full}} - F_{i_{stick}}}{F_{i_{full}}} * 100$$

$Mode$	$Full\ model freq.[Hz]$	$Stick\ model freq.[Hz]$	$Relative\ error[\%]$
1	0.73553	0.55505	24.53683
2	2.46789	1.92782	21.88389
3	5.4666	4.33701	20.66340
4	7.66535	13.3805	−74.55837
5	9.40136	8.16675	13.13220
6	9.50287	7.64263	19.57560
7	13.2508	22.4217	−69.21020
8	14.4031	11.7250	18.5939
9	19.613	21.4223	−9.22500
10	20.0155	16.4092	18.01750

Table 4.1: Frequencies of full and stick model and relative error

4.4 Modification and Results

As we can see the validation od the stick model is passed for the static analysis, while for the modal analysis there are some incompatibility. The modal shape related to the torsional and coupled torsional-bending modes are not quite similar, and the results on the natural frequencies present for this mode an error really high, and without sense. A modification at the stick model have been done to avoid this problem. After investigation we understand that the problems are related to the mass distribution, in fact while until the span wing there is a good discretization of the mass of the wing, 20 lumped muss are used, until the cord there isn't a valid distribution, all the lumped mass are positioned until the elastic axis, so the torsional behavior is not correctly represented. A new mass distribution have been implemented to correct this errors.

4.4.1 New Mass Distribution

As we showed in the first section to import the inertia proprieties of the full model on the stick model a series of lumped mass have been used. These lumped mass are positioned in the center of gravity of each box, that's mean that all the mass is concentrated until the elastic axis, then the torsional inertia proprieties of the full model are not imported in the stick model.
The idea to avoid this problem is to use a different mass distribution, in particular which takes into account of the distribution of mass until the cord. While just beam elements and lumped mass are used the only way to avoid this problem is to add lumped mass for each section and delocalise that mass from the elastic axis.
The new mass distribution provides to create 3 lumped mass for each box, one located in the trailing edge, one located in the center of gravity and one located in the leading edge. The value of the mass is chosen with a linear distribution on the cord, the total mass of the box should be unchanged. To do this the structure component have been modified, as first to compute the value of the 3 lumped mass, then to create the grid point to allocate the mass and finally to modify the *.bdf* file.

In Fig. 4.10 it's showed how the *.dbf* file is built after the correction:

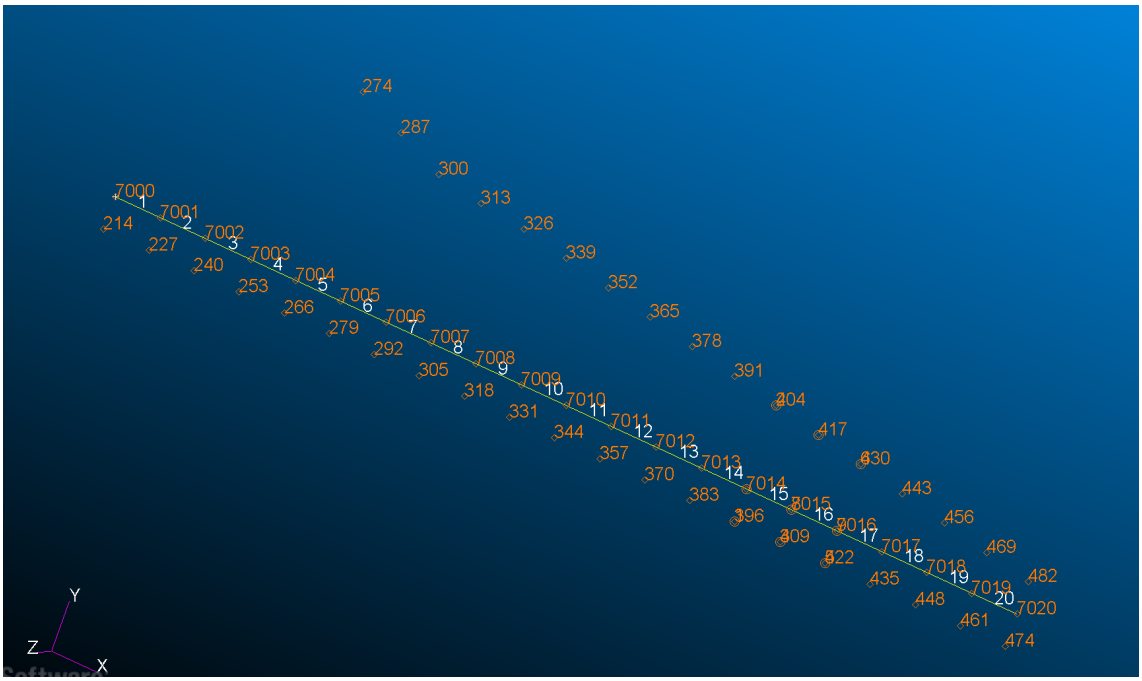


Figure 4.10: FEM model after new mass distribution implementation

In white there are the beam elements used for the stick model, in orange there are the lumped mass used, as u can see for each element, representative of one wing box section, there are 3 lumped mass, one localized on the center of gravity of the element, and two respectively localized on the trailing edge and on the leadng edge. Using this new mass distribution as first we solved the problem related to the torsional modes, then the modal shapes and the frequencies related to the torsional and coupled mode assumed sense value, but there are also improvement on the bending modes. The error on the frequencies for the torsional and coupled mode are now of the same order of magnitude of the errors on the bending mode frequencies, and the last see a reduction of $\approx 5\%$.

The new MAC matrix and the new comparison between the frequencies are showed respectively in Fig. 4.11 and in Fig. 4.12.

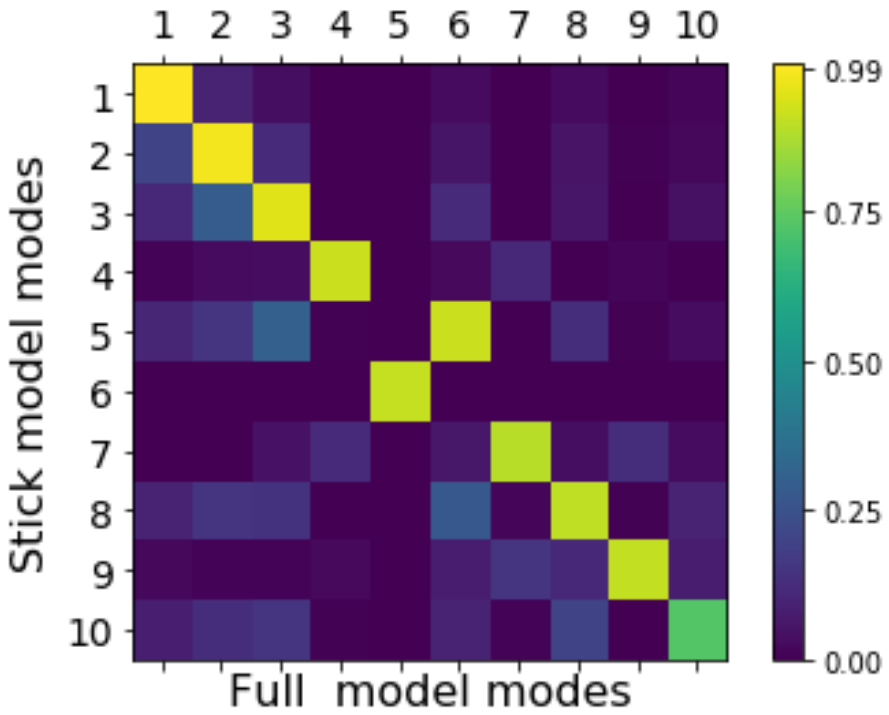


Figure 4.11: Improvements on the MAC matrix after correction

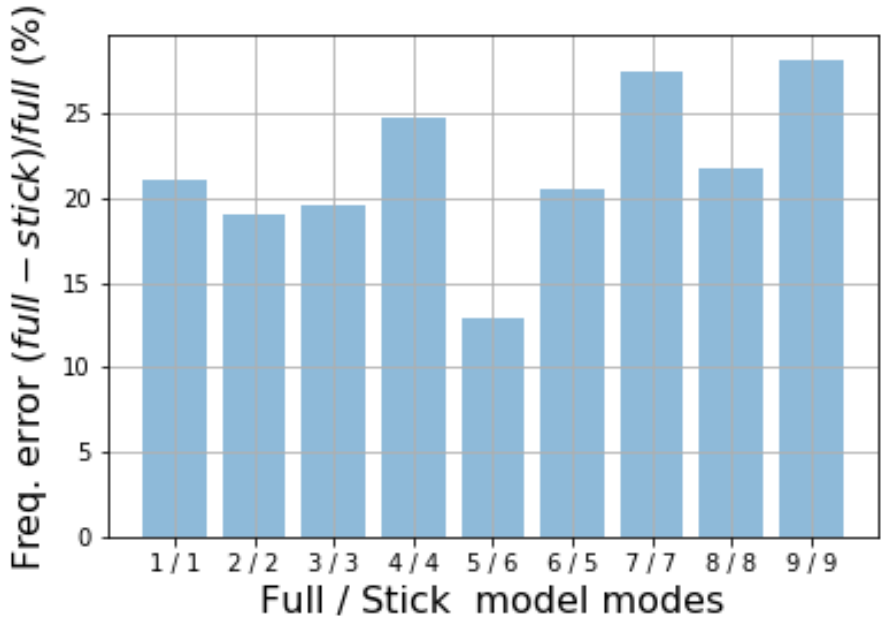


Figure 4.12: Improvements on the bar frequencies plot after correction

In Tab. 4.2 are showed the new frequencies for the stick model after correction and the relative errors:

$Mode$	$Full\ model freq.[Hz]$	$Stick\ model freq.[Hz]$	$Relative\ error[\%]$
1	0.73553	0.55505	21.0668
2	2.46789	1.92782	19.0021
3	5.4666	4.33701	19.6497
4	7.66535	13.3805	24.7905
5	9.40136	8.16675	12.9799
6	9.50287	7.64263	20.6029
7	13.2508	22.4217	27.5247
8	14.4031	11.7250	21.778
9	19.613	21.4223	28.2195
10	20.0155	16.4092	23.1296

Table 4.2: Frequencies of full and stick model and relative error after correction

4.4.2 Automatization of the Mode Pairing

Another problem of the reduction method was the non completely automatization of the process, that doesn't allow an implementation in the optimization process. In fact as we said, when the error on the frequencies is bigger than the step of two frequencies in the row, the order of the mode is not the same for the full and the stick model. Can happen that the 4th mode for the full model is the 1st torsional mode while for the stick model the 4th mode is the 3rd bending mode. Now for a correct evaluation of the errors on the frequencies and the relative correction the errors should be evaluate on the same mode, and the vector of frequencies need to be reorganized. This operation was did manually, by watching the modal shapes using a vizualization software for the output files.

To automatize the process we create an algorithm, based on the evaluation of the MAC matrix, to order the vector of frequencies related of the MAC value. In particular blocked the vector of the frequencies related to the full model, we put for each mode the frequencies related to the mode which have the maximum value of

MAC, by exploring the full MAC matrix.

Once this function was implemented the reduction model could be implemented in the optimization loop to reduce the computational cost of the aeroelastic coupling loop.

4.5 Results

The main objective of this chapter is to prove that the stick model is reducing the overall time of the procedure and can be used for complex and multidisciplinary computations. Thus, in each component a timer was placed in order to calculate the time reduction of the stick model. Certainly, in the first iterations the advantage of the stick model will not be evident since there are two additional components in comparison with the 3D model, thus, extra time. However, when the process reaches the multidisciplinary analysis(MDA), the computational efficiency of the stick model should be clear. [25]

In Tab. 4.3 the time cost of each component in both optimizations are presented:

Component	*Full model*$[s]$	*Stick model*$[s]$	*Reduction*$[\%]$
Geometry	7.97	7.81	−2.01
Interpolation	0.371	0.416	−88.79
Total Reduction	–	7.27	–
Aero	7.26	7.25	−0.28
Load Transfer	0.0044	0.00183	−58.41
Structure	3.52	2.19	−37.78
Displacements Transfer	0.00166	0.000605	−63.55
Total Pre − MDA Process	8.341	15.1216	81.29
Total MDA Process	10.78606	9.432435	−12.55
Total Time	116.2016	107.7537	−7.27

Table 4.3: Time reduction of the stick model

Chapter 5

Test Cases

To validate the optimization script, several test cases have been done. Many aspects of the problem had to be tested, in order to obtain information on the effective functioning of the software, as well as to bring out errors and complications not taken into account in the programming phase. Tests were therefore carried out on different wing models, in particular the Goland wing and the NASA Common Research Wing CRM , characterized by a different level of detail of the structural meshes, as well as by different aerodynamic properties. Moreover, in the various tests both drivers chosen for the implementation, COBYLA and SLSQP, were used, with the corresponding comparisons on the results; and different combinations between objective function and constraints were carried out.
As first let's see in details the wing model used for the validation of the software.

5.1 Goland Wing

The Goland wing is a wing model developed by M. Goland, and described in [12]. This model of wing is really simple, and several numerical and experimental studies have been carried out on this wing, which is usually used as reference to validate aeroelastic codes. So for the preliminary approach it's the best choice, because we can obtain fast results and find in literature many works to compare the results.
The Goland wing model that we use is based on the model described in the work of Beran P.S. [21]

The wing is schematized as a cantilevered wing. The wing span is 6.10 m, the chord 1.22 m and the thickness 0.51 m.

The finite element model is built up from shear panels, modeling the spars and ribs, and membrane elements, modeling the wing skins. The spar and rib caps are modeled by rod elements and posts connect the wing skins at every spar/rib intersection.[1] In Fig. 5.1 is showed how the initial FEM model is structured.

The result is a very flexible wing, that it's ideal to show several aeroelastic behavior.

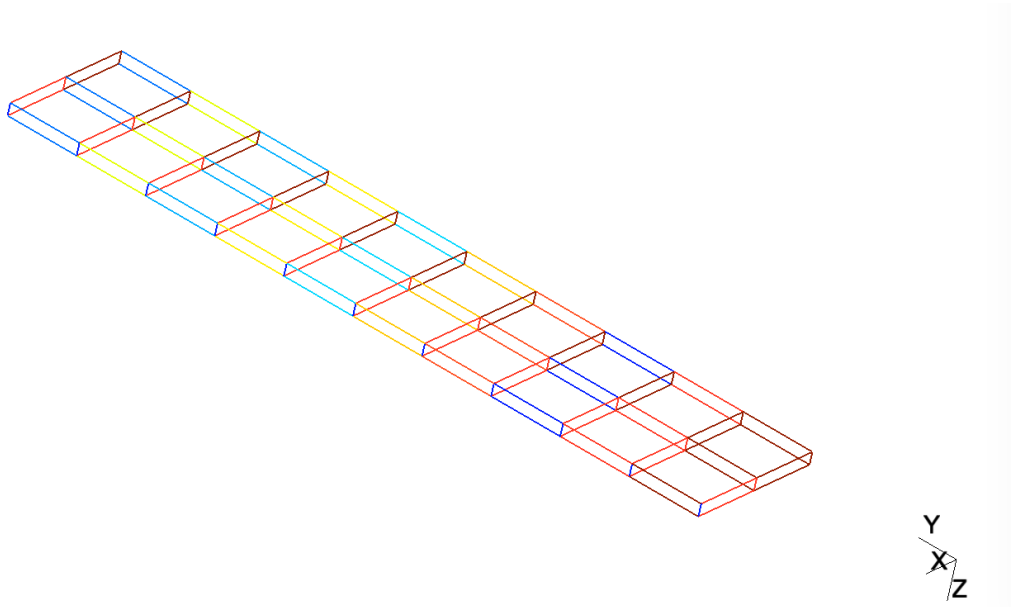


Figure 5.1: Initial FEM model for the Goland wing

For the aerodynamic mesh an airfoil are obtained using a 4% thick parabolic arc, then the mesh is generated using 4 section until the wing span, for each section several point of the airfoil are considered, with a condensation of nodes on the trailing and leading edge, then the panel are obtained joining the nodes and the sections. In Fig. 5.2 is showed how the initial aerodynamic mesh is structured.

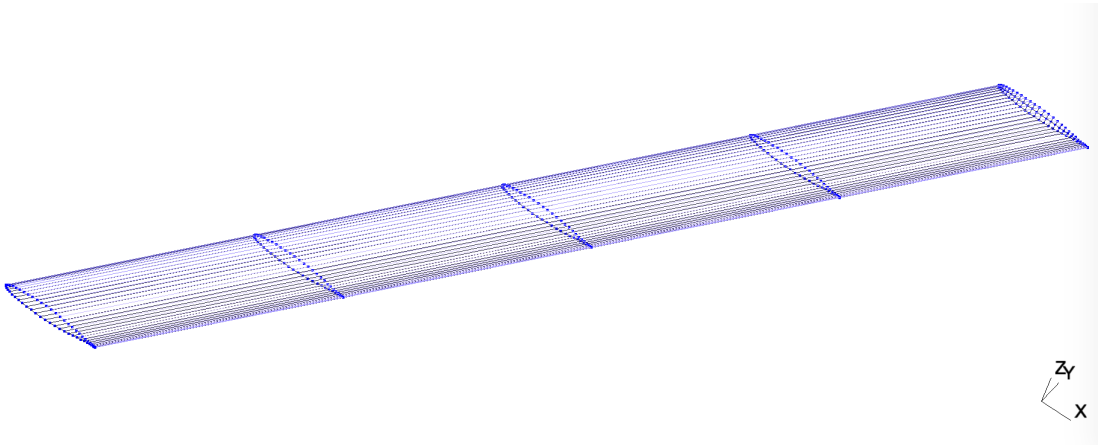


Figure 5.2: Initial aerodynamic mesh for the Goland wing

5.2 CRM wing

The second wing model used for the test cases is the NASA Common Research Model Wing **CRM**. This choice has been adopted in order to have a more complex wing's model, to access to more problematic and design's aspect, compared to the Goland wing, and the choice was the CRM, because several project is based on this model, so we can easily find in literature results to compare, and evaluate the quality of them.

The Common Research Model (CRM) consists of a contemporary supercritical transonic wing and a fuselage that is representative of a widebody commercial transport aircraft. The CRM is designed for a cruise Mach number of $M_\infty = 0.85$ and a corresponding design lift coefficient of $C_L = 0.5$. [2]

The wing consist in a fullscale cantilevered wing. The CRM was generated as open geometry for the research, imagined for transport class aircraft with single-aisle configuration. The geometry of the wing is more complex of the Goland wing, in fact there are sweep angle, taper ratio, etc... .It's caraterized of a wing span of 58.76 m, an aspect ratio of 9, a root cord of 7 m, a taper ratio of 0.275, a leading edge sweep angle of 35 deg, and a break along the trailing edge at 37% of the semi-span, the wing reference area is 383.68 m^2. For the structure the wing box is defined to lie between 10% and 70% of the wing cord.

In Fig. 5.3 the plan view of the CRM wing is showed:

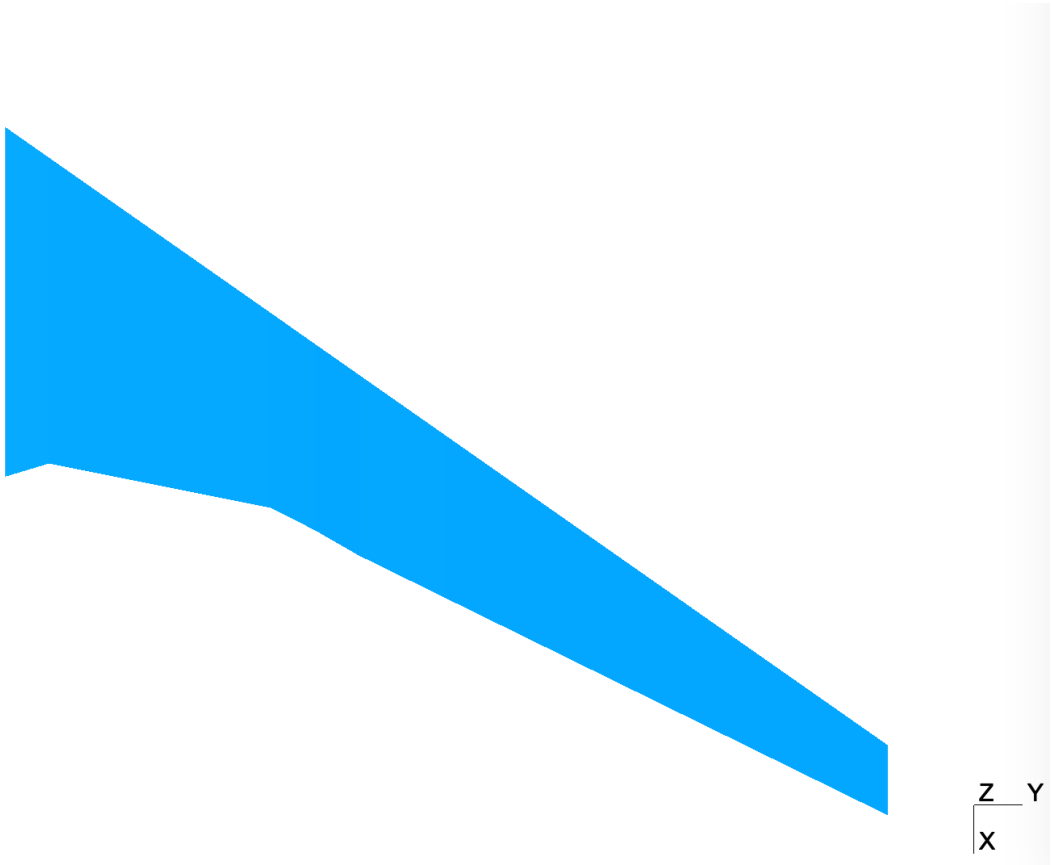


Figure 5.3: Plan view of the CRM wing geometry

For the structural model an high fidelity model have been created. All the component of the wing (ribs, spar, stringer, skin) are modeled using shell elements. Over 25'000 finite elements have been used. In Fig. 5.4 the complete FEM model is showed:

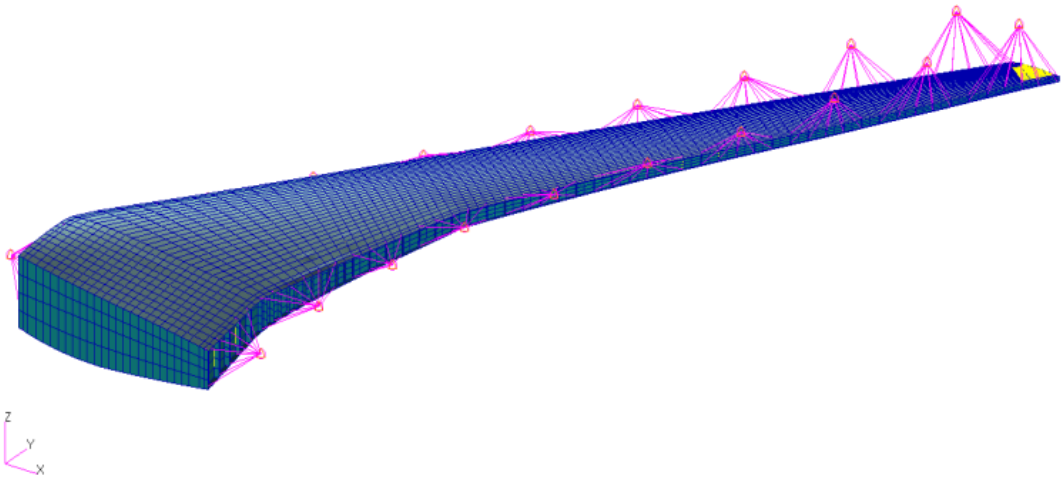


Figure 5.4: FEM model of CRM wing

The airfoil used to obtain the aerodynamic mesh is the CRM-65 in Fig. 5.5, provided by NASA [3]. The aerodynamic mesh is obtained using 50 different section, in order to take account of the taper ratio and the thickness. In Fig. 5.6 the aerodynamic mesh is showed:

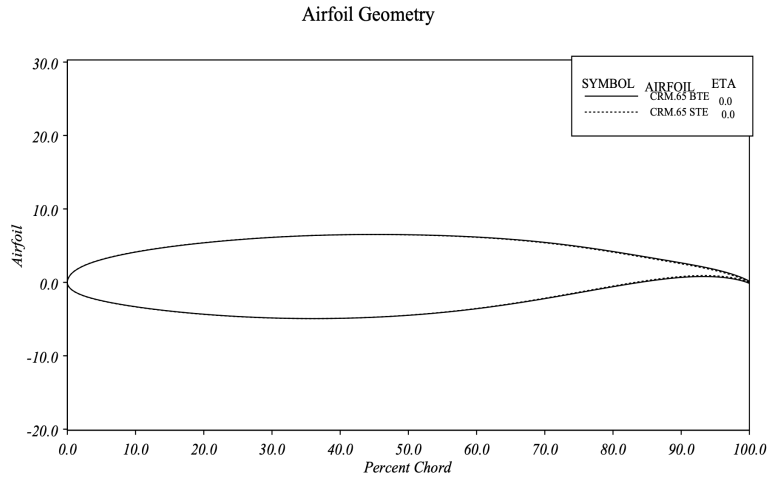


Figure 5.5: CRM-65 airfoil

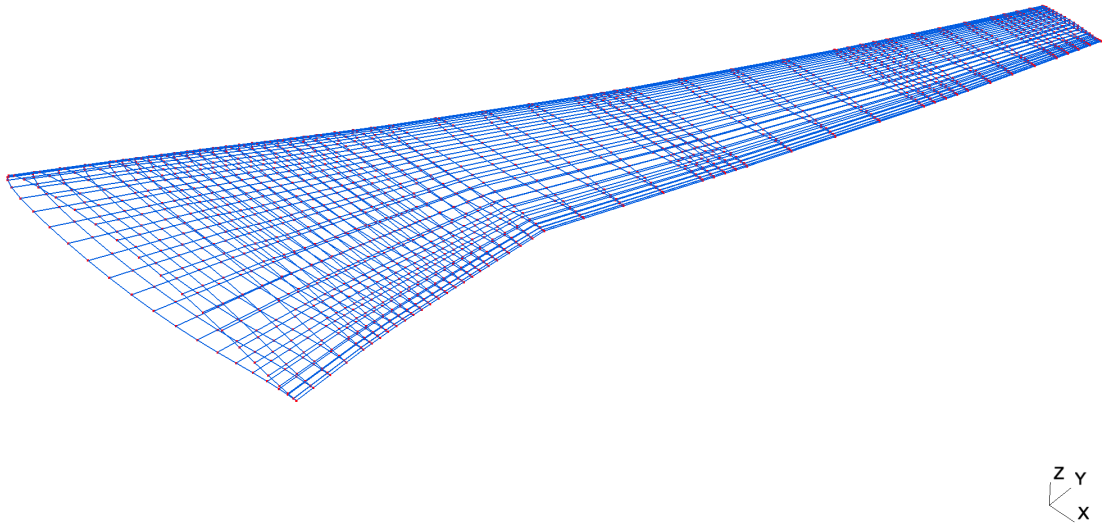


Figure 5.6: Aerodynamic mesh of the CRM wing

5.3 CRM Wing Simple Model

The CRM wing model allow to consider more parameters in the optimization process, but the cost of the structural and aerodynamic analysis for the model that we create are really high. In the test phase of the component, where coding or conceptual errors emerge, the time required to launch the code can be prohibitive. So to solve this problem a simple model of the CRM wing have been used, in order to have all the proprieties of the CRM wing but with a much less computation cost. The quality of the result is, of course, worse, but in that phases we wasn't interested to obtain result, but just to validate the code.

The most important changes are on the structural model. As first much less elements have been used, from 25'000 to 1'000, then instead to use shell elements for all the components, beam elements have been used for stringer and spars.

In Fig. 5.7 the structural mesh of the simple model of the CRM wing is showed, the beam elements are in red while the shell elements are in blue:

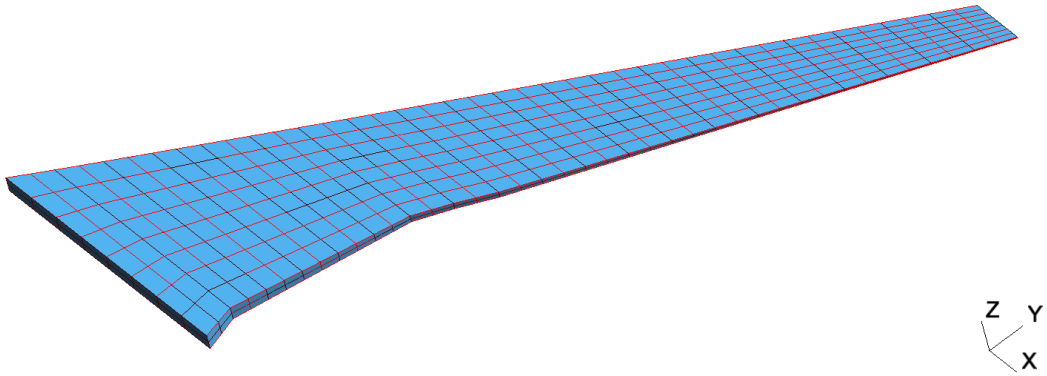


Figure 5.7: Aerodynamic mesh of the simple model CRM wing

Also the aerodynamic mesh is simple. In this case we just reduced the number of section, from 50 to 20. In Fig.5.8 is showed the aerodynamic mesh of the simple CRM wing model:

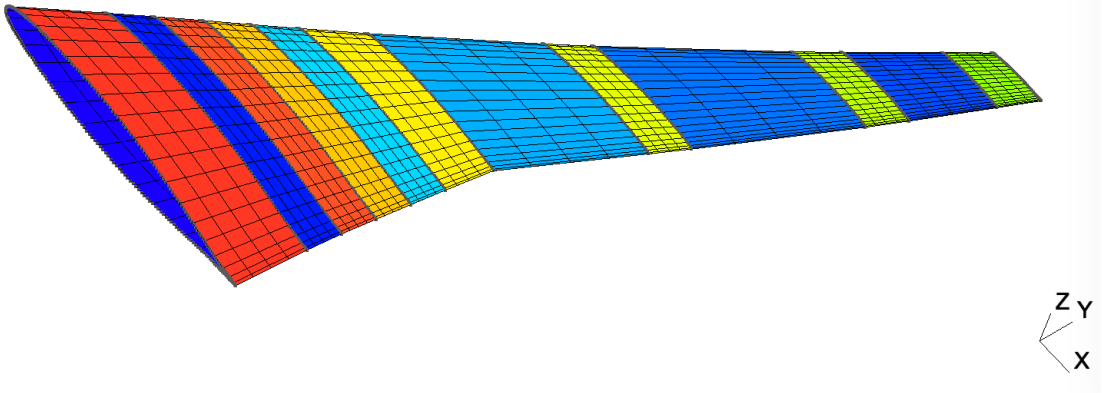


Figure 5.8: Aerodynamic mesh of the simple model CRM wing

5.4 Different Options for the Optimization

Several test cases have been performed in order to validate the different components, to test the different drivers, the different wing model. So each test case represent a

combination of different option and has different goal. In Tab. 5.1 there are collect all the possible option used in the various test case:

Options	*Possible Choice*
Wing Model	Goland Wing
	CRM Wing
	CRM Wing Simplified
Driver	COBYLA
	SLSQP
Design Variables	Angle of Attack α
	Thickness of Shell Elements
	Sweep Angle Λ
	Wingspan b
Objective Function	Mass m
	Induced Drag Coefficient C_{D_i}
	Generic Function f
Generic Options	Constraint Aggregation
	Design Varibles Limit as Constraint
	Reduced Structural model

Table 5.1: Different option selectable for the optimization

5.5 Problems

During the test cases several problems emerge. In this section we will explain the most relevant problems, and the relative solution that we find.

5.5.1 Cobyla Design Variables Limits

One of the first problem that have emerged when we pass to the CRM wing cases was relative to the use of the optimization driver COBYLA. The problem was that when the thickness of the wing tip shell elements, the zone of the wing characterized to the biggest value of displacements , start to be to little, the displacements start to being really impressive. Then this displacements are moving into the aerodynamic mesh using the interpolation. Now if the wing tip section nodes are moving too much, the

aerodynamic analysis will fail, because the mesh assume a weird shape. That's cause the crash of the program and interrupts the optimization. The direct consequence is to set limits for the design variables, in this case limits on the minimum value of the shell elements thickness, in order to avoid that the displacements being huge. Then we set limits on the thickness as 10^{-3} m.

Using the COBYLA optimization we see that, despite the design variables limits was exactly set, the optimization still crash. Then opening the database of the iterations we saw that the limits of the design variables are not respected, as we can see in Fig. 5.9:

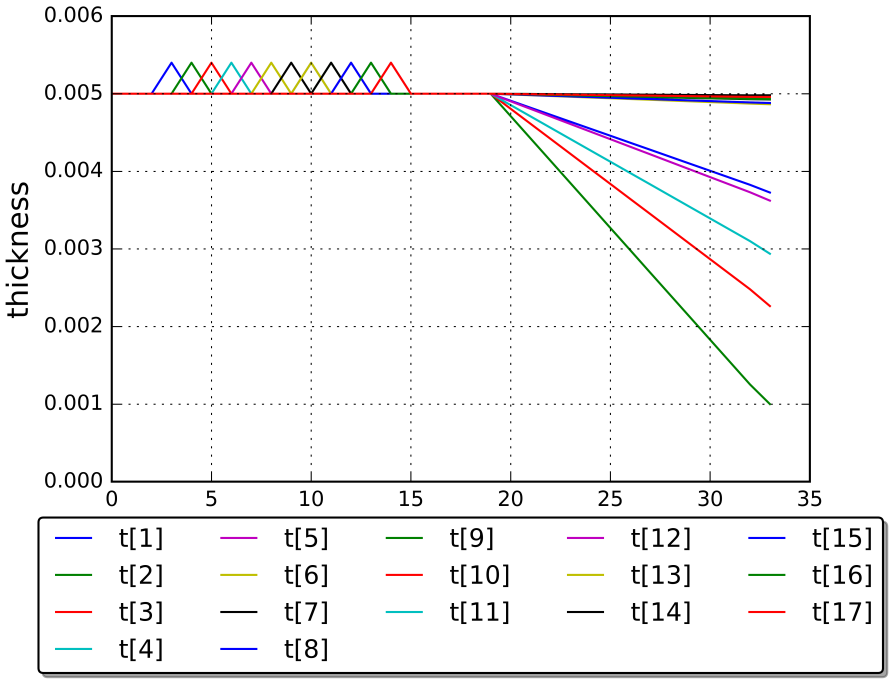


Figure 5.9: Thickness variation in a COBYLA optimization

As we can see the optimization crashes at the iteration 34, because the thickness of the section 16 is lower than 10^{-3} m. The problem is that COBYLA doens't respect the design variables limits. That's is result of the is a consequence of how it was programmed, COBYLA is a driver programmed to get results, so when it decide the direction of the optimization, in this case the direction is to reduce the thickness of section in order to obtain a reduction of the mass of the wing, it still work in order to get the convergence of the objective function respecting the constraints.

To solve this problem, and avoid the crashes, our solution was to set also the thickness limits of the section as constraint, in order to be respected by COBYLA. So a new set of constraint have been created, the constraints equation check for each iteration if the minimum value of the thickness is bigger than the limit which we set:

$$con(t_i) = t_i - t_{i_{min}} > 0 \qquad i = 1,2,...,n_t$$

where $con(t_i)$ is the constraint function, t_i the value of the thickness of the section i, $t_{i_{min}}$ the limit of the thickness and n_t the number of the section with different thickness.

In order to check if the solution works, we start a particular optimization focused on the reduction of the thickness for all the section, and check if the limits on the thickness if finally respected, as is showed in Fig. 5.10:

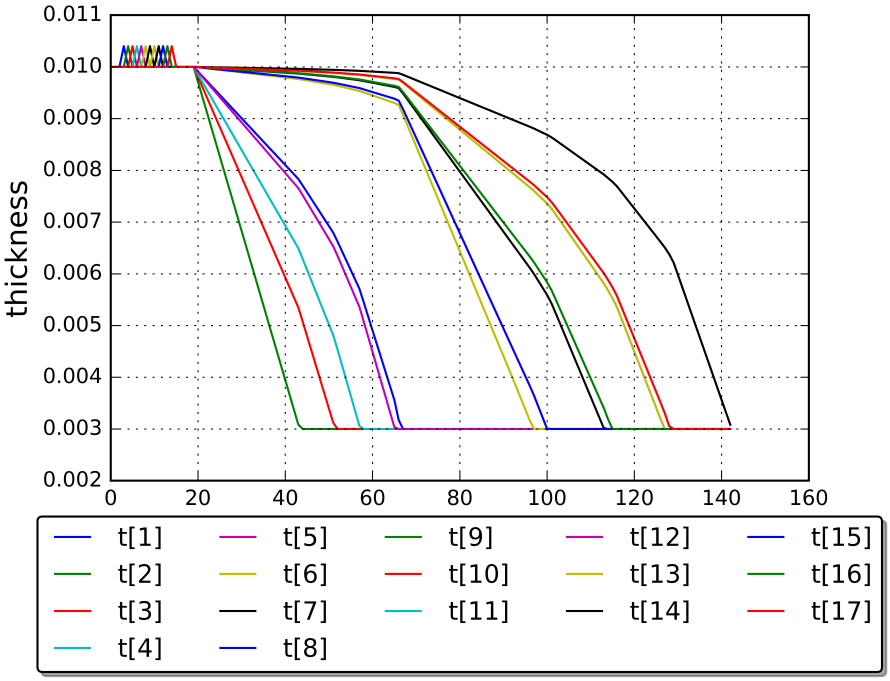


Figure 5.10: Thickness variation after design limits correction

As we can see the correction works well, in fact when the thickness reach the design limit, in this case $3 * 10^{-}3$, that value will not more changed, and the stability of the optimization is guaranteed.

5.5.2 Finite Difference Gradient Evaluation Error

Another error has emerged using the SLSQP driver. As we said in the chapter 2, to evaluate the gradient, for the gradient based optimization using SLSQP, we use the finite difference method. In our specific case one of the gradient that we need for the optimization, being the thickness of the section a design variables, and the Von Mises stress a constraint, like the lift, and the mass an objective, the gradient of these function respect to the thickness.
As we explain the structural analysis and the aerodynamic analysis is performed by the external codes, respectively NASTRA95 and Panair. So to compute the gradient the flow-chart is the following, for example in the case of the gradient of the mass respect the thickness:

1. Set the the starting value of the thickness
2. Perform an static structural analysis
3. Extract from the output file the value of the mass
4. Change the thickness of the finite difference step
5. Perform an static structural analysis
6. Extract the new value of the mass from the output file
7. Compute the gradient using the finite difference equation

The problem that we found was that using the default settings, the solver can't evaluate the gradient, as the mass didn't change after a change of the thickness. That happens because NASTRAN95 use a 8 bit floating point numeration; so the finite difference step set as default for SLSQP is $10^{-}4$, so the effect that the variation of this step induce on the structural mass is really little, and it changes just the 9th or 10th significant digits, then NASTRAN95 cut the information after the 8th significant digit, so he will lose the information on the variation of the mass, and the mass seems unchanged.
To solve this problem is just necessary to specify in the setting of the driver the

new step used for the finite difference, in order to induce a bigger variation on the structural mass, then the information on the variotion of the mass will not lose, in our case a step of $10^{-}2$ it's enough to compute correctly the gradient.

5.5.3 Nastran Output File Reader

Another problem emerged in the test cases is relative to the structural component, specifically for the output file reading method. In order to extract the information of the Von Mises stress for each element we set NASTRAN95 to save this information in a *.pnh* file, characterized by a special structure of the file. Then the structural component after the analysis access to this output file, and an algorithm have been written to associate all the stress to the elements and save these information in a python dictionary. The algorithm have been written in relation at the output file of an analysis performed on the CRM wing, where only shell elements have been used. In the moment that we introduce the simple CRM wing model that contains also beam elements, the structure of the output file changes, then the algorithm can't read successfully it. In the first moment a new algorithm have been written to read correctly also the new output file, but we are working in order to make it universally. The idea is to use an *.exe* file that convert the *.pnh* file. The difference between the output files is that the number of the stresses depends on the number of the degree of freedom of the elements, then the data is collected on more lines. The *.exe* file is structured to convert the file in a file where all the stress of one element are collected on just one line, then is easy to write an algorithm to collect the data independently of the wing structural model chosen.

Chapter 6

Results

In this chapter are collected the results related to the most significant test cases. For each case all the combination of the setting used have given, together with the graph relative to the trend of the design variables, constraints and objective function until the iteration. The data relative to this trend are collected in a database, created using the recording function implemented in the openMDAO package. So to access to te result is required to use the openMDAO database function, for this reason an external component have been created in order to access to the database and plot the graphics.

6.1 Reader code

In order to collect all the information through the optimization process, the recorder function of openMDAO have been used. This function provide to create a database, which contain all the value of the variables. For each iteration a new dictionary will be created, this dictionary contain one dictionary for each type of variables, in the last there are all the value of the variables for that iteration, as showed in Fig. 6.1:

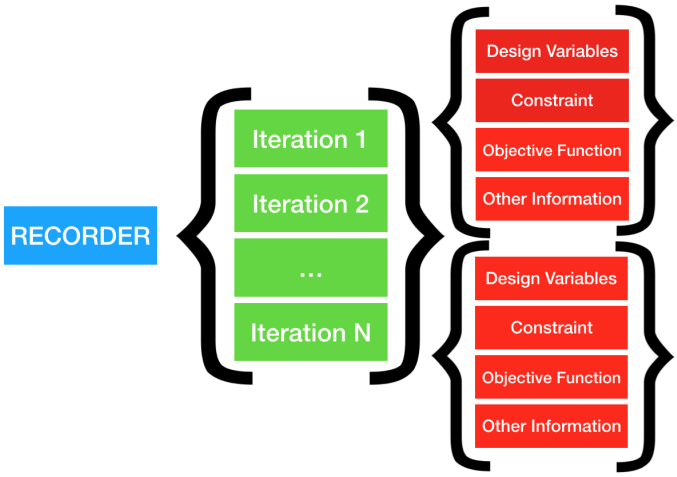


Figure 6.1: Structure of the recorder output file

Since the database file name is set, is possible to extract the keys of the dictionary, which are representative of the iterations. Depends on the optimization drives used, SLSQP or COBYLA, the extraction algorithm is different, so it's necessary to specify the used driver. Then for each iteration the value of the variables will be stored in a bunch of vector, defined by the user. At the end of the process a plotting section have been written in order to plot the trend of the variables. For the constraints plots also the limits of the constraint have been plotted, in order to see when a constraint is violated.
In Appendix A.2 is reported the python code written for extract the result, with indication of the function of each section.

6.2 Test Cases Results

In this section the result relative to the most significant test cases are given. For each case a table indicate all the setting parameters, while the results are stored in the output graph.

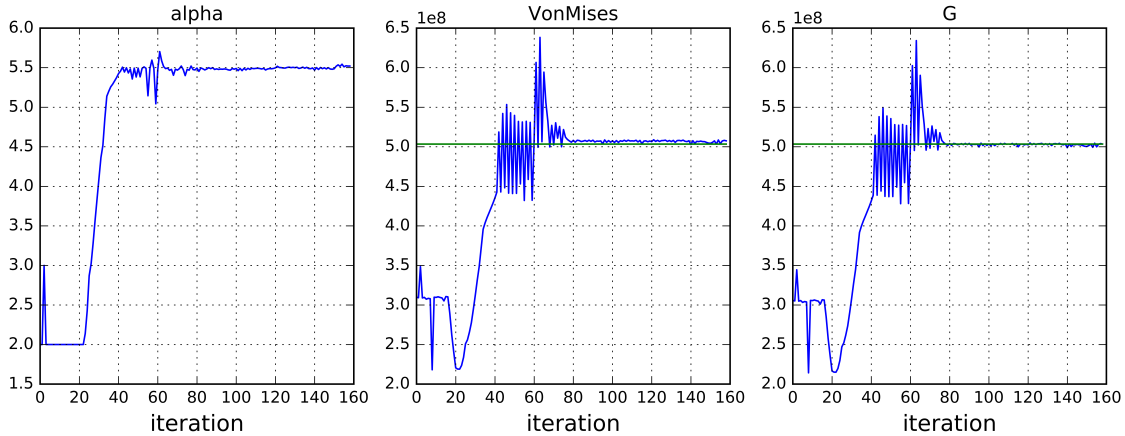
6.2.1 Case 1

In this test case an optimization of the CRM wing, full model, respect to the mass of the wing have performed. The driver chosen in this case is COBYLA, then the design limits have been implemented as constraint. The objective function is the mass of the wing, while the design variables are the angle of attack α and the thicknesses of the 12 shell element section of the structural model. The constraints are the lift constraint and the stress constraint, for the second the aggregation have been performed, using the G^L_{KS} function.

In Fig. 6.2 are shown the trend of the variables until the optimization, for the stress are shown the maximum of Von Mises stress vector obtained using the maximum value function and the relative aggregation function value; for the aerodynamic variables the trend of the induced drag coefficient C_{D_i} and the lift coefficient C_L are shown. In Tab. 6.1 are summarised the information of the optimization settings.

Options	*Choice*	
Wing Model	CRM Wing	
Driver	COBYLA	
Design Variable	Angle of Attack	Thicknesses
Constraints	Lift	Stresses
Objective Function	Mass	
Generic Options	Constraints Aggregation	☑
	Design Limits as Constraints	☑

Table 6.1: Optimization Settings Case 1



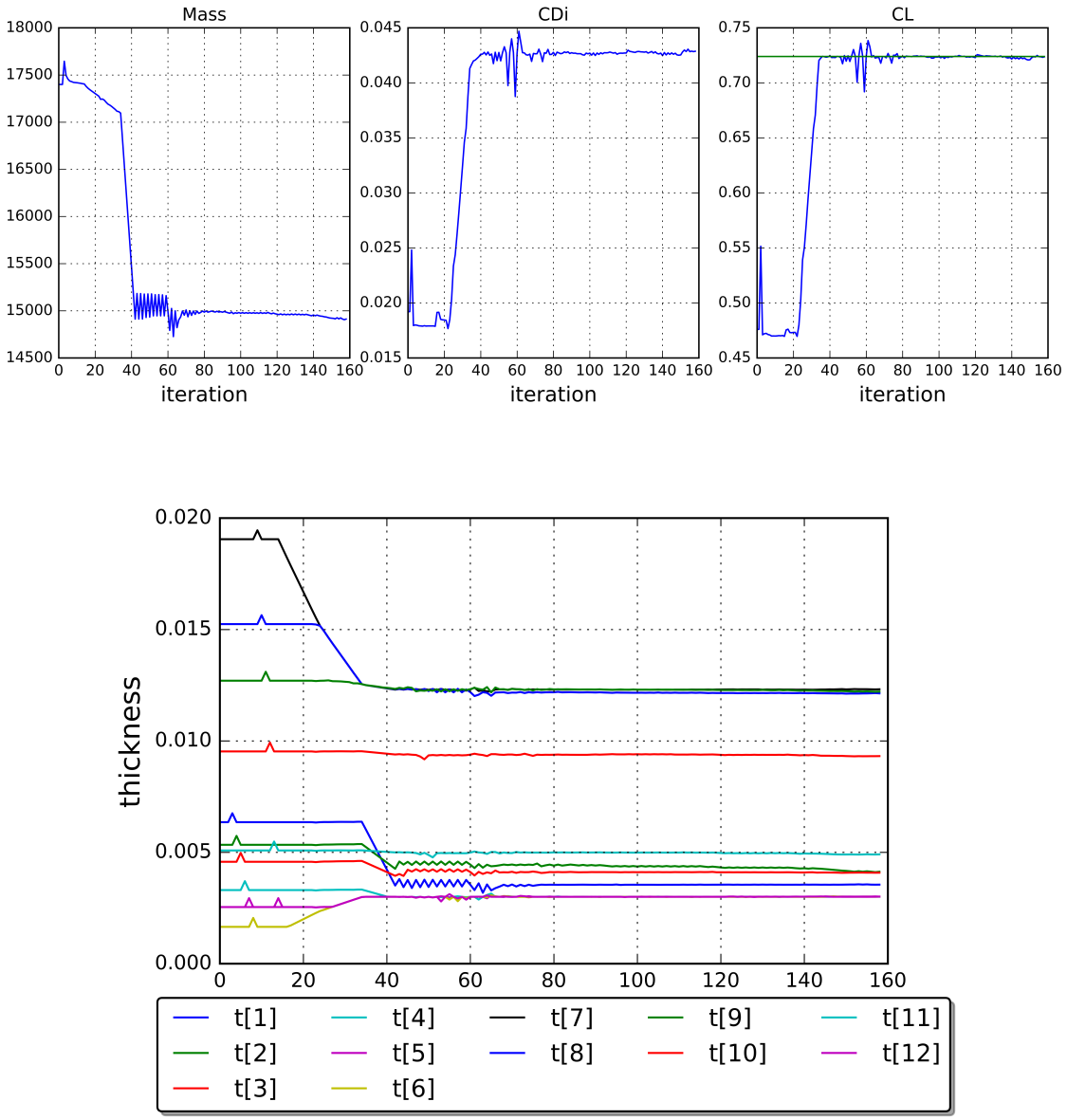


Figure 6.2: Result of the optimization Case 1

As you can see from the graph the optimization reach the convergence of the objective function in around 160 iteration. All the constraint are respected, and the optimization reach a gain in terms of structural mass of around 15%, the initial mass of the wing was 17'200 Kg, at the end of the optimization the mass is around 15'000 Kg. As we can see using a lower bounded aggregation function the maximum of Von Mises stress exceed the limit.

For this optimization we use COBYLA, the gradien free optimization driver implemented. As we can see the driver choose the design direction for the optimization in an evaluation performed in the initial iteration, that we can see on the graph as

a little step on the design variables. Once the direction it's decided he continue in that direction until the convergence. We can see how the direction is to reduce the thickness of the panel until the failure stress criteria allow it. The angle of attack is related to the C_L, so its initial value is chosen in order to respect the lift constraint. At iteration 40 the driver found the best design point, so from that point he start to evaluate the functions changing the design variable from that point, checking step by step the constraints.

6.2.2 Case 2

In this test case we have done an optimization on the CRM wing, in order to minimize the induced drag coefficient C_{D_i} using, this time, the gradient based optimizer SLSQP, where, as we said, the gradient is computed using the finite difference method. The constraint are the lift constraint and the stress constraints, aggregate this time with the upper bounded Kreisselmeier-Steinhsauser function. This time, since COBYLA is not used, isn't necessary to set the design limits as constraint.
In Fig. 6.3 are showed the results, in the same format of the first case, while in Tab. 6.2 there are given the problem settings.

Options	*Choice*	
Wing Model	CRM Wing	
Driver	SLSQP	
Design Variable	Angle of Attack	Thicknesses
Constraints	Lift	Stresses
Objective Function	Cdi	
Generic Options	Constraints Aggregation	☑
	Design Limits as Constraints	☒

Table 6.2: Optimization Settings Case 2

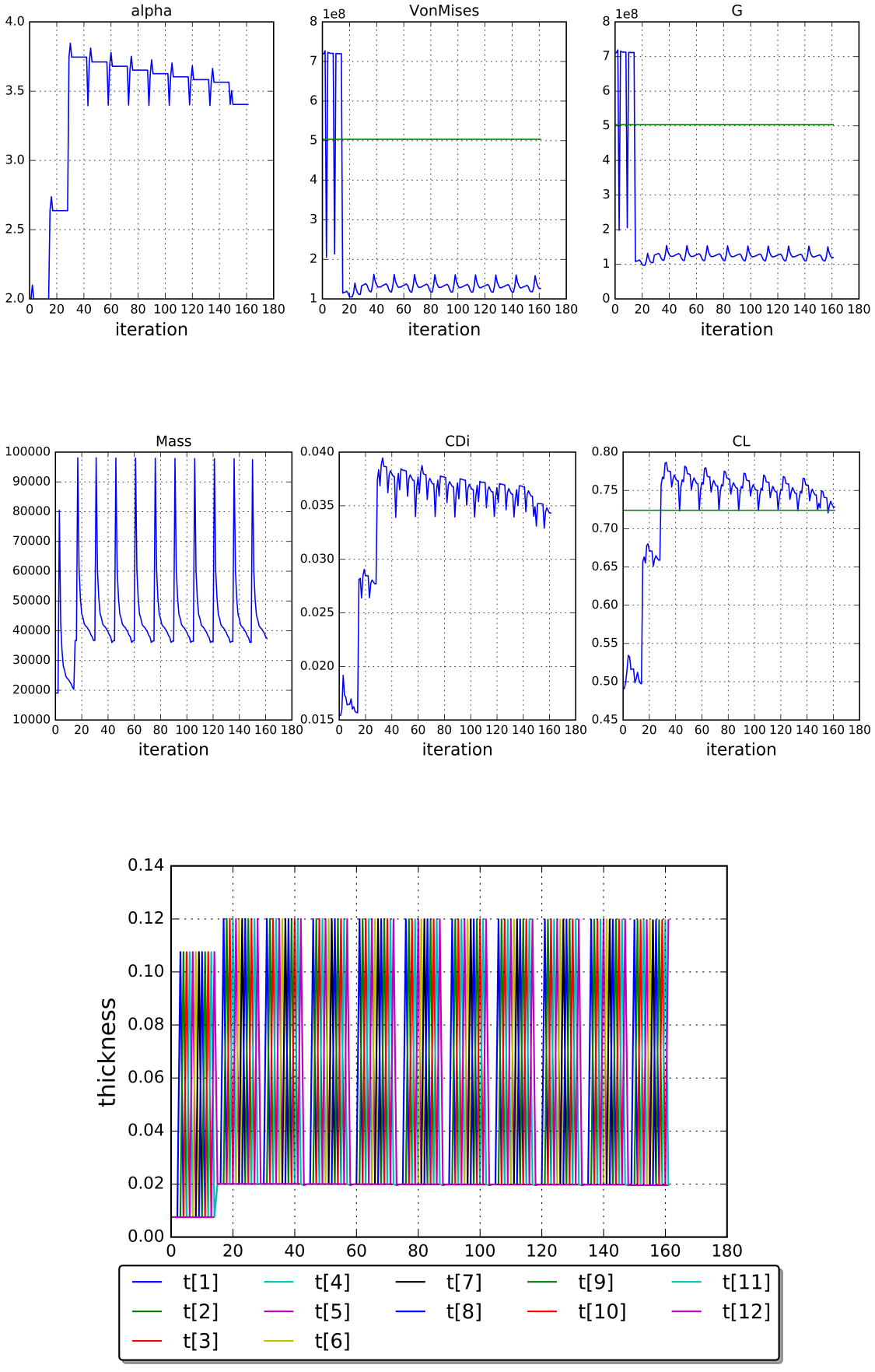


Figure 6.3: Result of the optimization Case 2

In this case more than 220 iteration need to reach the convergence. The optimization target is the reduction of the induced drag coefficient, that see a reduction of the 25%. No limits are imposed on the structural mass and on the minimum stress, so as we can see to reach this gain on the C_{D_i} there is an increase of the mass of the 45%, due to the increase of the thicknesses of the shell elements. This involves that the material is not fully exploited, the stresses are much lower than the yield stress. In this case we used an upper bounded aggregation function, so, as you can see, the value of the aggregation function is bigger than the value of the maximum of the Von Mises stress vector. Both the constraint are respected.
In this case we have used the gradient based optimizer, we can observe how the optimizer work, it start the initial condition, evaluate the gradient using the finite difference centered in that design point, than decide the direction of optimization and continue in that direction until the best condition are reached, then start a new gradient evaluation centered this time in the new design point, and it repeat this until the best is reached. So the graph are typed of strong excursions each time the gradient are computed.

6.2.3 Case 3 and 4

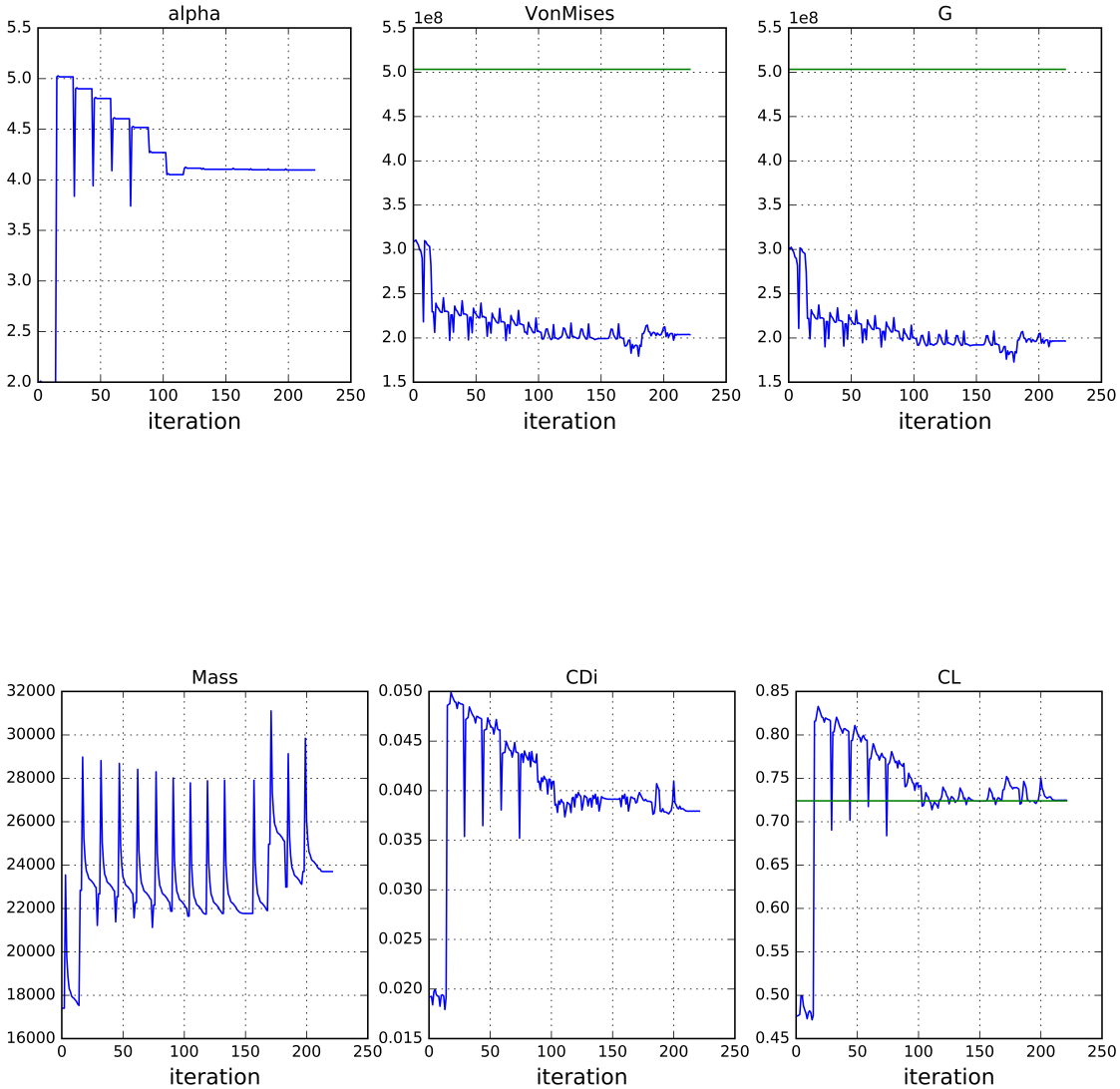
The case 3 and 4 have the same settings, expect for the driver, in fact this test case have been done to got a comparison between the two different driver. In this cases the objective function is given by:

$$f = \alpha C_{D_i} + \beta m$$

where $\alpha = \beta = 0.5$, in order to obtain an optimization where the objective is find the best compromise between the mass and the induced drag, to avoid an optimization as the case 2 where to minimize the drag coefficient the mass of the wing grows disproportionately.
In Tab. 6.3 and Tab. 6.6 there are given respectively the setting option of the two case, while in Fig. 6.4 and Fig. 6.5 are showed the results graphs.

Options	*Choice*	
Wing Model	CRM Wing	
Driver	SLSQP	
Design Variable	Angle of Attack	Thicknesses
Constraints	Lift	Stresses
Objective Function	$f = \alpha C_{D_i} + \beta m$	
Generic Options	Constraints Aggregation	☑
	Design Limits as Constraints	☒

Table 6.3: Optimization Settings Case 3



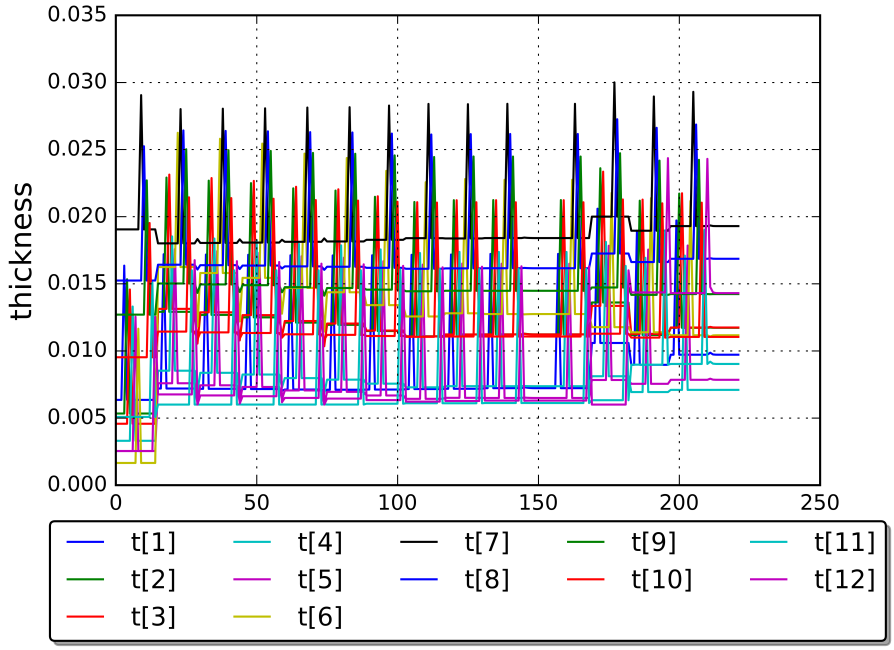
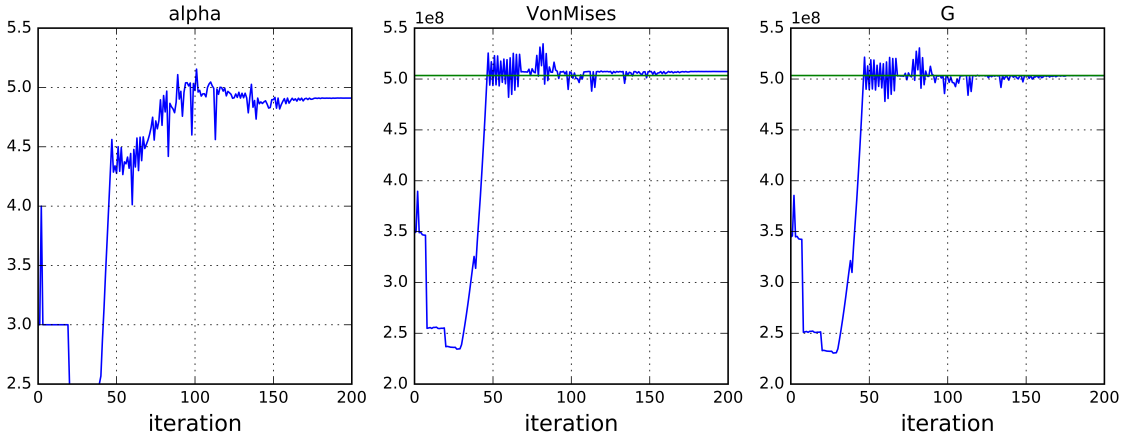


Figure 6.4: Result of the optimization Case 3

Options	*Choice*	
Wing Model	CRM Wing	
Driver	COBYLA	
Design Variable	Angle of Attack	Thicknesses
Constraints	Lift	Stresses
Objective Function	$f = \alpha C_{D_i} + \beta m$	
Generic Options	Constraints Aggregation	☑
	Design Limits as Constraints	☑

Table 6.4: Optimization Settings Case 4



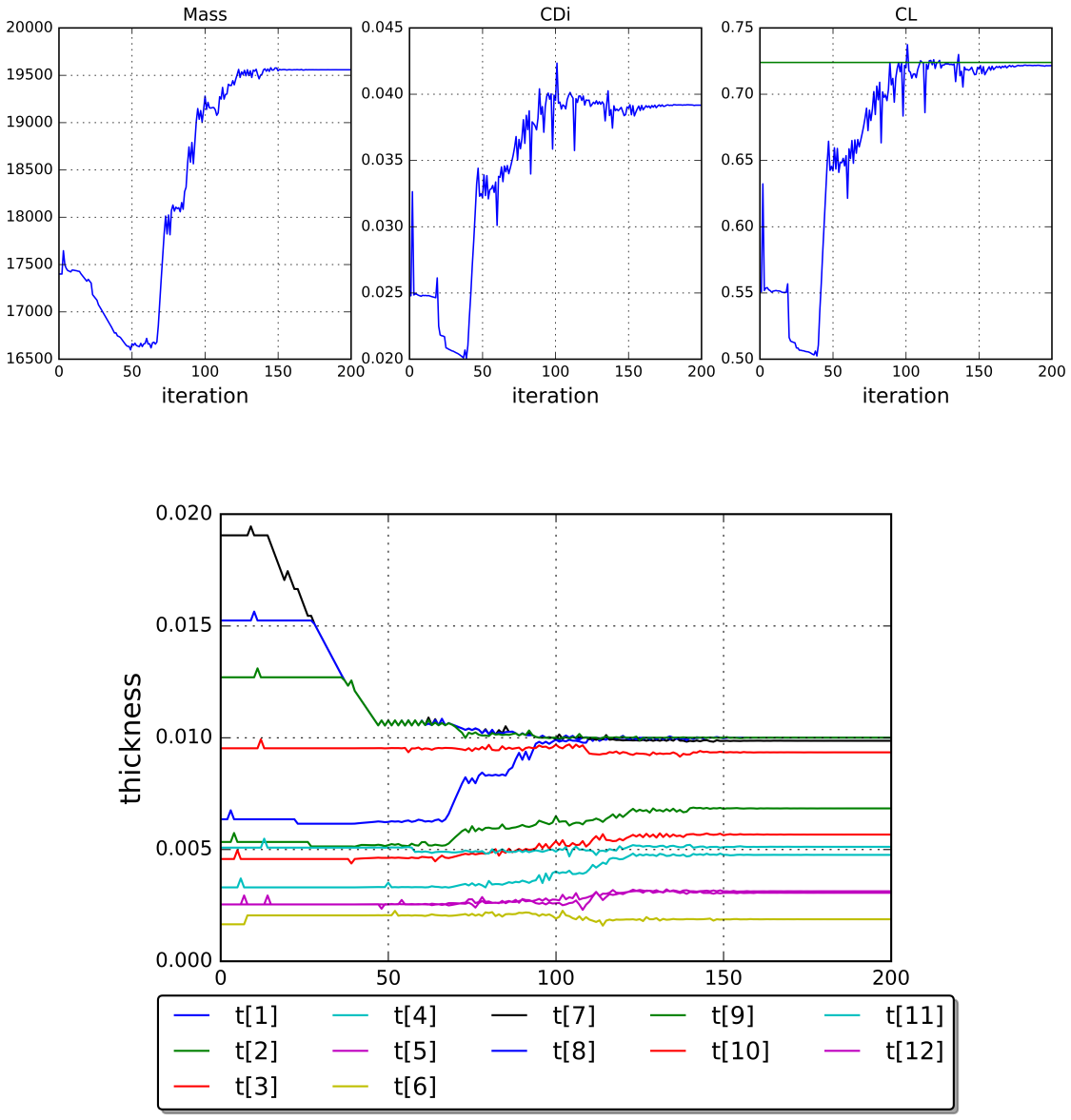


Figure 6.5: Result of the optimization Case 4

The result of the objective functions, as expected, are in between of the result of the case 1 and case 2, which was polarized just on one objective function. In both cases in order to have a reduction of the induced drag coefficient it's necessary to increase the structural mass, but the increase is not bigger as the case 2.Though there are difference between the two optimization:

COBYLA

- ≈200 iterations
- maximize the stresses

SLSQP

- ≈ 220 iterations
- minimize α

- final mass = + 11%
- final $C_{D_i} = 0.037$

- final mass = + 27%
- final $C_{D_i} = 0.039$

In Tab. 6.5 a comparison of the important value of the optimization between the first 4 test case is given:

	Case 1	Case 2	Case 3	Case 4
α [deg]	5.5	3.4	4.1	4.8
C_L	0.70	0.71	0.70	0.69
C_{D_i}	0.044	0.034	0.037	0.039
m $[Kg]$	14900	38000	23700	19600

Table 6.5: Comparison of Results

6.2.4 Case 5

This test case is relative to an optimization performed using the Goland wing, in order to obtain fast result in testing phase. Being the wing model different, the number of the section is different, but the structure of the script is the same. This optimization has as objective the function f, described in the test case 3 and 4, with $\alpha = \beta = 1$.

Options	*Choice*	
Wing Model	Goland wing	
Driver	COBYLA	
Design Variable	Angle of Attack	Thicknesses
Constraints	Lift	Stresses
Objective Function	$f = \alpha C_{D_i} + \beta m$	
Generic Options	Constraints Aggregation	☒
	Design Limits as Constraints	☑

Table 6.6: Optimization Settings Case 2

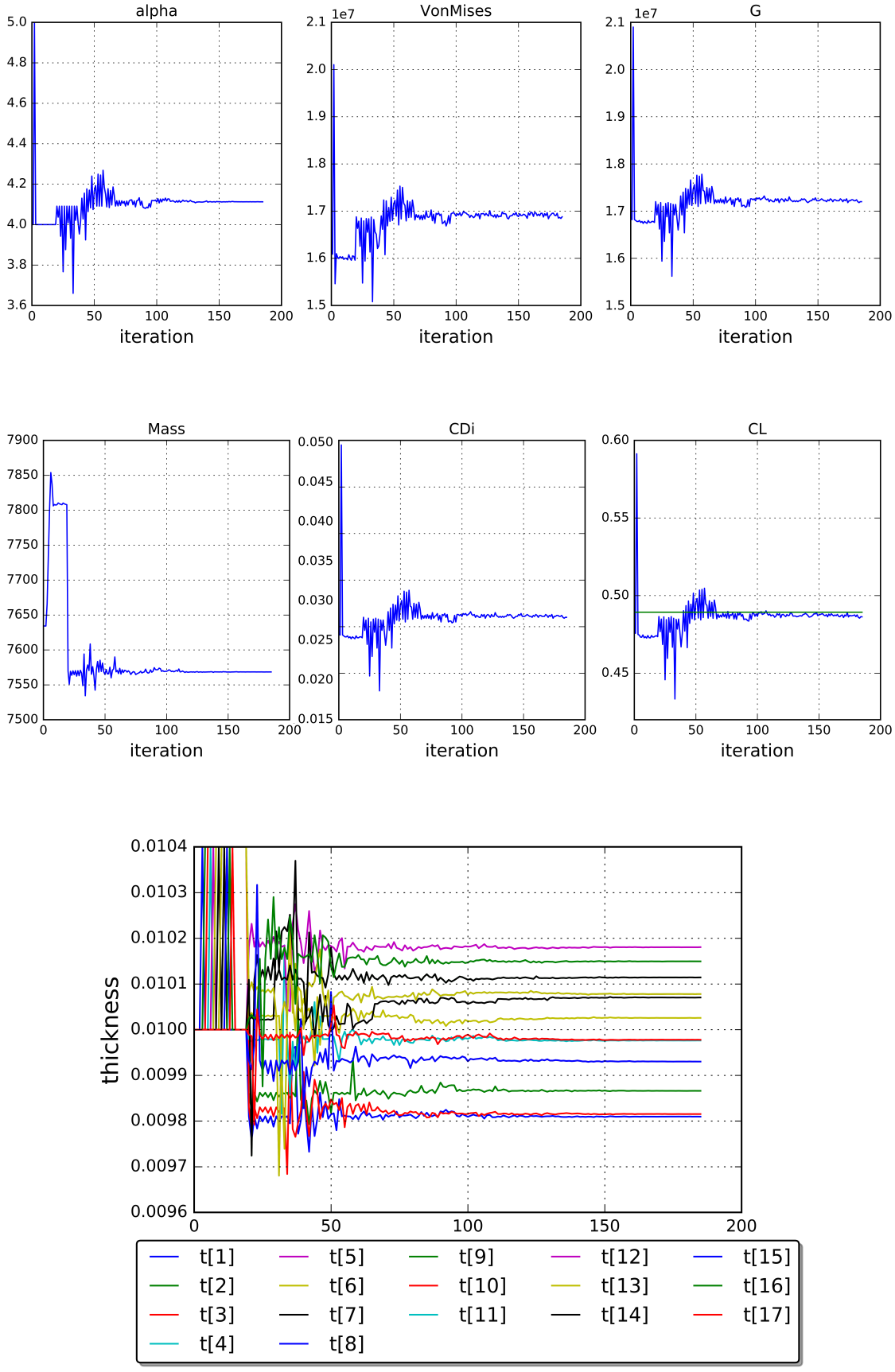


Figure 6.6: Result of the optimization Case 5

Appendix A

Python Codes

A.1 CRM Wing Optimization Test Case 1

```
#MAIN SCRIPT
#Modules Importing
# -*- coding: utf-8 -*-
"""
Created on Tue Mar 29 10:50:10 2016

@author:  a.iacono
"""
from __future__ import print_function

from openmdao.api import Problem, Group, IndepVarComp,
    ScipyGMRES, SqliteRecorder, ExecComp, ScipyOptimizer, view_tree

from aerostructures import NastranStatic, DisplacementTransfer,
    Panair, LoadTransfer,Aggregation, Interpolation,\
StaticStructureProblemDimensions, StaticStructureProblemParams,
    AeroProblemDimensions, AeroProblemParams, NLGaussSeidel

import numpy as np
```

```
#Definition of Specific Problem Parameters
#Note: To choose correctly the aggregation function parameters
    checking the guide.

if __name__ == "__main__":

    #Interpolation function type and setup
    function_type = 'thin_plate'
    bias = (1,50,1)

    #Symmetry plane index
    sym_plane_index = 1

    #Problem parameters
    Sw = 383.689555         # Wing Surface
    V = 250.75              # Velocity
    rho_a = 0.337           # Air Density
    Mach = 0.85             # Flight MACH
    alpha = 0.58465         # Starting Angle of Attack
    b = 58.7629             # Wing Span
    c = 7.00532             # Cord
    E = 6.89e10             # Young Module
    nu = 0.31               # Poisson Coefficient
    rho_s = 2795.67         # Material Density
    t_i_max=0.0125          # Upper Limit for Panel's Thickness
    t_i_min=0.0018          # Lower Limit for Panel's Thickness
    sigma_y = 5.033172e+08 # Yield Stress
    W = 9.81*300000.        # Airplane Weight
    function = 'Gksl'       # Aggregation Function
    p=100.                  # Draw-Down Function
    s0=40000000.0           # Reference Stress
```

```
47 #Definition of Problem Dimensions:
   #Note: To do this we are using some of the methods defined in the
      Aerostructures Package, in the relative guide is declared the
      meaning of each variables and how we determinate it.
49
      structure_problem_dimensions =
      StaticStructureProblemDimensions()
51    aero_problem_dimensions = AeroProblemDimensions()

53    ns = structure_problem_dimensions.ns
      ns_all = structure_problem_dimensions.ns_all
55    node_id = structure_problem_dimensions.node_id
      node_id_all = structure_problem_dimensions.node_id_all
57    n_stress = structure_problem_dimensions.n_stress
      tn = structure_problem_dimensions.tn
59    mn = structure_problem_dimensions.mn

61    structure_problem_params =
      StaticStructureProblemParams(node_id, node_id_all)
      aero_problem_params = AeroProblemParams()
63
      na = aero_problem_dimensions.na
65    network_info = aero_problem_dimensions.network_info

67    node_coord = structure_problem_params.node_coord
      node_coord_all = structure_problem_params.node_coord_all
69    t = structure_problem_params.t
      m = structure_problem_params.m
71
      apoints_coord = aero_problem_params.apoints_coord
73 #Setting OpenMDAO Problem
```

```
75      top = Problem()
        top.root = root = Group()
77  #If you want to used a gradient based optimization like SLSQP the
        gradient is obtained by finite difference. So you have to set
        the step because the default step make really little changes
        on the variables, and the 8 float architecture of nastran
        can't recognize this variation.

79  #================================================
    #     UNCOMMENT JUST FOR SLSQP OPTIMIZER
81  #================================================
    #     top.root.deriv_options['type'] = 'fd'
83  #     top.root.deriv_options['step_size'] = 1.0e-1
    #================================================
85  #Here we are defining the openMDAO variables as Independent
        Variables Component:

87      root.add('wing_area', IndepVarComp('Sw', Sw), promotes=['*'])
        root.add('airspeed', IndepVarComp('V', V), promotes=['*'])
89      root.add('sigma_y', IndepVarComp('sigma_y', sigma_y),
        promotes=['*'])
        root.add('stress_ref', IndepVarComp('s0', s0), promotes=['*'])
91      root.add('air_density', IndepVarComp('rho_a', rho_a),
        promotes=['*'])
        root.add('Mach_number', IndepVarComp('Mach', Mach),
        promotes=['*'])
93      root.add('young_module', IndepVarComp('E', E), promotes=['*'])

        root.add('tick_max',IndepVarComp('t_i_max',t_i_max),promotes=['*'])
```

```
95      root.add('t_min', IndepVarComp('t_i_min',t_i_min),
    promotes=['*'])
        root.add('weight', IndepVarComp('W', W), promotes=['*'])
97      root.add('mat_density', IndepVarComp('rho_s', rho_s),
    promotes=['*'])
        root.add('poisson', IndepVarComp('nu', nu), promotes=['*'])
99      root.add('angle_of_attack', IndepVarComp('alpha', 0.),
    promotes=['*'])
        root.add('wing_span', IndepVarComp('b', b), promotes=['*'])
101     root.add('wing_chord', IndepVarComp('c', c), promotes=['*'])
        root.add('s_coord', IndepVarComp('node_coord', node_coord),
    promotes=['*'])
103     root.add('s_coord_all', IndepVarComp('node_coord_all',
    node_coord_all), promotes=['*'])
        root.add('thicknesses', IndepVarComp('t', t), promotes=['*'])
105     root.add('masses', IndepVarComp('m', m), promotes=['*'])
        root.add('a_coord', IndepVarComp('apoints_coord',
    apoints_coord), promotes=['*'])
107 #Here we are adding the modules for the interpolation between
    aerodinamic and strucutural mesh, and the aggregation module,
    defined in the Aereostructures package:

109     root.add('inter', Interpolation(na, ns, function =
    function_type, bias = bias), promotes=['*'])
        root.add('agrr', Aggregation(n_stress,p,function),
    promotes=['*'])
111
    #Creating the mda group for the convergence of aerodinamyc forces
    and structural displacement:
113     mda = Group()
```

```
115     #Add disciplines to the group
        mda.add('displacement_transfer', DisplacementTransfer(na,
    ns), promotes=['*'])
117     mda.add('aerodynamics', Panair(na, network_info),
    promotes=['*'])
        mda.add('load_transfer', LoadTransfer(na, ns), promotes=['*'])
119     mda.add('structures', NastranStatic(node_id, node_id_all,
    n_stress, tn, mn), promotes=['*'])
#Setting the mda solver type and settings:
121     mda.nl_solver = NLGaussSeidel()
#     mda.nl_solver.options['rtol'] = 1.e-1
123     mda.nl_solver.options['maxiter'] = 15
        mda.nl_solver.options['rutol'] = 1.e-2
125     mda.nl_solver.options['use_aitken'] = True
        mda.nl_solver.options['aitken_alpha_min'] = 0.1
127     mda.nl_solver.options['aitken_alpha_max'] = 1.5
#Adding the mda cycle to the optimization group:
129
        mda.ln_solver = ScipyGMRES()
131
        root.add('mda_group', mda, promotes=['*'])
133 #=====================================================
#       UNCOMMENT JUST FOR SLSQP OPTIMIZER
135 #=====================================================
#=====================================================
137 #     top.root.mda_group.deriv_options['type'] = 'fd'
#     top.root.mda_group.deriv_options['step_size'] = 1.0e-1
139 #=====================================================
#Setting Recorder
141     recorder = SqliteRecorder('opti_g_55')       # In brackets
    the name of the database
```

```
    recorder.options['record_params'] = False
    recorder.options['record_metadata'] = False
    recorder.options['record_resids'] = False
    recorder.options['record_derivs'] = False
    top.root.nl_solver.add_recorder(recorder)
#Setting the Optimitazion
#Defining solver type and the optimizer
    root.ln_solver = ScipyGMRES()                # Solver type
    top.driver = ScipyOptimizer()                # Adding Optmizer
    top.driver.options['optimizer'] = 'COBYLA'  # Defining the
   optimizer and is settings
    top.driver.options['disp'] = True
    top.driver.options['tol'] = 1.e-4
    top.driver.options['maxiter'] = 500
    top.driver.opt_settings['rhobeg']= 0.1      # USE JUST FOR
   COBYLA
#Defining objective and constraint functions
    # Minimize the CDi or the mass
    root.add('obj_function', ExecComp('obj_f = CDi'),
   promotes=['*'])
    # Define constraint CL - W/q
    root.add('con_lift', ExecComp('con_l = CL -
   W/(0.5*rho_a*V**2*Sw)'), promotes=['*'])
    # Define constraint G - sigma_y
    root.add('con_stress', ExecComp('con_s = G - sigma_y'),
   promotes=['*'])
    t_max=0.01*np.ones(tn)
    t_min=0.006*np.ones(tn)
    # Define one constraint t_max and t_min for each
   differentthickness section in the model
    for i in range(tn):
```

```
167         root.add('max_t_'+str(i+1),ExecComp('max_t_'+str(i+1)+' =
    t['+str(i)+'] - t_i_max',

    t=np.zeros(tn,dtype=float)),promotes=['*'])
169         root.add('min_t_'+str(i+1),ExecComp('min_t_'+str(i+1)+' =
    t['+str(i)+'] - t_i_min',

    t=np.zeros(tn,dtype=float)),promotes=['*'])
171 #Setting objective, design variables and constraint
     top.driver.add_objective('obj_f')
173
     alpha_max=10.
175  alpha_min=0.

177  top.driver.add_desvar('alpha', lower=alpha_min,
    upper=alpha_max, adder=-alpha_min,
    scaler=1/(alpha_max-alpha_min))
     top.driver.add_desvar('t', lower=t_min, upper=t_max,
    adder=-t_min, scaler=1/(t_max-t_min))
179
     top.driver.add_constraint('con_l', lower=0.)
181  top.driver.add_constraint('con_s', upper=0.)
     for i in range(tn):
183
    top.driver.add_constraint('max_t_'+str(i+1),upper=0.,scaler=1/t_i_max)

    top.driver.add_constraint('min_t_'+str(i+1),lower=0.,scaler=1/t_i_min)
185 #As u can see we have created one constraint for each different
    thickness section, but for
#the stress constraint we just created one constraint using the
    aggregation function, that is
```

```
187 #representative of the max value of the stress for each iteration.


189 #Setup Problem and Run It
    top.setup()
191 
    view_tree(top, show_browser=False)
193 
    top.run()
195 top.cleanup()
```

opti_con_nt.py

A.2 Python Script to Access Results

```
#ACCESS TO THE RESULTS
#Importing Section
# -*- coding: utf-8 -*-
"""
Created on Mon Jun 25 15:57:07 2018

@author: a.iacono
"""
import sqlitedict
import numpy as np
import matplotlib.pyplot as plt
#Preprocessing Section
#Setting the name of the file that contain the database
name='opti_g_50'
#Creating a dictonary from the database
#The print line just show us the number of the iterations and the
   respective keys for each iteration.
db = sqlitedict.SqliteDict( name, 'iterations' )
print( list( db.keys() ) )

#Defining the number of the iteration and the x-axis for the graph
n=len(db)
x=np.arange(0,n)
#Creating the empy vector to save all the value for each
   iteration of the variables of interest
cd=np.array([])
g=np.array([])
m=np.array([])
vm=np.array([])
cl=np.array([])
```

```
a=np.array([])
vms=np.array([])
t1=np.array([])
t2=np.array([])
t3=np.array([])
t4=np.array([])
t5=np.array([])
t6=np.array([])
t7=np.array([])
t8=np.array([])
t9=np.array([])
t10=np.array([])
t11=np.array([])
t12=np.array([])
#Extrapoling the dictonary of the first iteration to have access
    to some value like the yield stress
data = db['rank0:COBYLA|0|root|1']
c = data['Unknowns']
#Determinating the value of the lift and stress constraint
con_l=(c['W']/(0.5*c['rho_a']*c['V']**2*c['Sw']))*np.ones(n)
con_s=c['sigma_y']*np.ones(n)
#Accessing to the value for each iteration
#As first we obtain a database for each iteration, the print
    command it's just to have a feedback on the runtime.

#From the Data dictonary we can collect all the data in refernce
    at the variables of interest, and append it to the vector
    created in the settings section.
for j in range(0,n):
        data = db['rank0:COBYLA|'+str(j)+'|root|'+str(j+1)]
        print('rank0:COBYLA|'+str(j)+'|root|'+str(j+1))
```

```
#          u = data['Parameters']
57         c = data['Unknowns']
           g = np.append(g,c['G'])
59         cd =np.append(cd, c['CDi'])
           a = np.append(a,c['alpha'])
61         m = np.append(m,c['mass'])
           vms = max(c['VMStress'])
63         vm = np.append(vm,vms)
           cl = np.append(cl, c['CL'])
65         t = c['t']
           t1 = np.append(t1,t[0])
67         t2 = np.append(t2,t[1])
           t3 = np.append(t3,t[2])
69         t4 = np.append(t4,t[3])
           t5 = np.append(t5,t[4])
71         t6 = np.append(t6,t[5])
           t7 = np.append(t7,t[6])
73         t8 = np.append(t8,t[7])
           t9 = np.append(t9,t[8])
75         t10 = np.append(t10,t[9])
           t11 = np.append(t11,t[10])
77         t12 = np.append(t12,t[11])
#Print Results
79 #Here we are printing all the graph of the variation of the
    variables for each iteration.
#The script is also set to save the plot as jpg files in the code
    folder.
81 fig, axes = plt.subplots(1, 3, figsize=(12, 4))
axes[1].grid (True)
83 axes[1].set_xlabel('iteration', fontsize=14)
axes[1].plot(x,vm)
```

```
85 axes[1].plot(x,con_s)
   axes[1].set_title("VonMises")
87 axes[2].set_xlabel('iteration', fontsize=14)
   axes[2].grid (True)
89 axes[2].plot(x,g)
   axes[2].plot(x,con_s)
91 axes[2].set_title("G")
   axes[0].plot(x,a)
93 axes[0].set_title("alpha")
   axes[0].grid (True)
95 axes[0].set_xlabel('iteration', fontsize=14)
   #axes[2].plot(x,32vms)
97 #axes[2].set_title("VonMises Stresses")
   plt.savefig(name+'_1',dpi=1000, bbox_inches='tight')
99 plt.show()
   fig, axes = plt.subplots(1, 3, figsize=(12, 4))
101
   axes[1].grid (True)
103 axes[1].set_xlabel('iteration', fontsize=14)
   axes[1].plot(x,cd)
105 axes[1].set_title("CDi")
   axes[2].set_xlabel('iteration', fontsize=14)
107 axes[2].grid (True)
   axes[2].plot(x,cl)
109 axes[2].plot(x,con_l)
   axes[2].set_title("CL")
111 axes[0].plot(x,m)
   axes[0].set_title("Mass")
113 axes[0].grid (True)
   axes[0].set_xlabel('iteration', fontsize=14)
115 #axes[2].plot(x,32vms)
```

```
#axes[2].set_title("VonMises Stresses")
plt.savefig(name+'_2',dpi=1000, bbox_inches='tight')
plt.show()
plt.plot(x,t1,label='t[1]')
plt.plot(x,t2,label='t[2]')
plt.plot(x,t3,label='t[3]')
plt.plot(x,t4,label='t[4]')
plt.plot(x,t5,label='t[5]')
plt.plot(x,t6,label='t[6]')
plt.plot(x,t7,label='t[7]')
plt.plot(x,t8,label='t[8]')
plt.plot(x,t9,label='t[9]')
plt.plot(x,t10,label='t[10]')
plt.plot(x,t11,label='t[11]')
plt.plot(x,t12,label='t[12]')
plt.grid(True)
plt.xlabel('iteration', fontsize=14)
plt.ylabel('thickness', fontsize=14)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05),
          fancybox=True, shadow=True, ncol=5)
plt.savefig(name+'_3',dpi=1000, bbox_inches='tight')
```

reader.py

Bibliography

[1] URL: http://www.cfd4aircraft.com/4ecerta_testcases_goland.php.

[2] URL: https://commonresearchmodel.larc.nasa.gov/experimental-approach/model-description/.

[3] URL: https://commonresearchmodel.larc.nasa.gov/crm-65-airfoil-sections/.

[4] Mostafa S. A. Elsayed, Ramin Sedaghati, and Mohammed Abdo. "Accurate Stick Model Development for Static Analysis of Complex Aircraft Wing-Box Structures". In: *AIAA Journal* 47.9 (2009).

[5] Martins Joaquim R. R. A. "A COUPLED-ADJOINT METHOD FOR HIGH-FIDELITY AERO-STRUCTURAL OPTIMIZATION". In: (2002).

[6] Martins Joaquim R. R. A. and Lambe Andrew B. "Multidisciplinary Design Optimization: A Survey of Architectures". In: *AIAA JOURNAL* (2013).

[7] Saltelli A. *Sensitivity Analysis for Importance Assessment, Risk Analysis,* 2002.

[8] R.J. Allemang and D.L. Brown. "A correlation coefficient for modal vector analysis". In: *Proceedings of The 1st International Modal Analysis Conference* (1972).

[9] Daniele Brigante, Carlo Rainieri, and Giovanni Fabbrocino. "The role of the Modal Assurance Criterion in the interpretation and validation of models for seismic analysis of architectural complexes". In: *Procedia Engineering* 199 (2017), pp. 3404–3409.

[10] George Done. "Introduction to Aircraft Aeroelasticity and Loads J. R. Wright and J. E. Cooper John Wiley and Sons, The Atrium, Southern Gate, Chichester, West Sussex". In: *The Aeronautical Journal (1968)* (2008).

[11] M.I. et al. Friswell. "The convergence of the iterated IRS methods". In: *Journal of Sound and Vibration* (1998).

[12] M. Goland. "The Flutter of a Uniform Cantilever Wing". In: *Journal of Applied Mechanics* 12 (1945), A197–A208.

[13] B. M. Irons and R. C. Tuck. "A Version of the Aitken Accelerator for Computer Iteration". In: *International Journal for Numerical Methods in Engineering* 1 (1969), pp. 275–277.

[14] Mas Colomer J. et al. "Similarity Maximization of a Scaled Aeroelastic Flight Demonstrator via Multidisciplinary Optimization". In: (2017).

[15] S. Jakobsson and O. Amoignon. "Mesh deformation using radial basis functions for gradient-based aerodynamic shape optimization". In: *Computers & Fluids* 36.6 (2007), pp. 1119–1136.

[16] G. Kreisselmeier and R. Steinhauser. *SYSTEMATIC CONTROL DESIGN BY OPTIMIZING A VECTOR PERFORMANCE INDEX*. Ed. by M.A. CUENOD. Pergamon, 1980, pp. 113–117.

[17] Andrew B. Lambe, Graeme J. Kennedy, and Joaquim R. R. A. Martins. "An evaluation of constraint aggregation strategies for wing box mass minimization". In: *Structural and Multidisciplinary Optimization* 55.1 (2017), pp. 257–277.

[18] M Lombardi, Nicola Parolini, and Alfio Quarteroni. "Radial basis functions for inter-grid interpolation and mesh motion in FSI problems". In: *Computer Methods in Applied Mechanics and Engineering* 256 (2013), pp. 117–131.

[19] Joaquim R. R. A. Martins and Nicholas M. K. Poon. *On Structural Optimization Using Constraint Aggregation*. 2005.

[20] Joaquim R. R. A. Martins, Peter Sturdza, and Juan J. Alonso. "The Complex-step Derivative Approximation". In: *ACM Trans. Math. Softw.* 29.3 (2003), pp. 245–262.

[21] Beran P. et al. “Numerical analysis of store-induced limit-cycle oscillation”. In: *Journal of Aircraft* 41 (2004), pp. 1315–1326.

[22] Pat Piperni, Mohammed Abdo, and Fassi Fafyeke. “The Application of Multi-Disciplinary Optimization Technologies to the Design of a Business Jet”. In: (2004).

[23] Nicholas M. K. Poon and Joaquim R. R. A. Martins. “An adaptive approach to constraint aggregation using adjoint sensitivity analysis”. In: *Structural and Multidisciplinary Optimization* 34.1 (2007), pp. 61–73.

[24] T.C.S Rendall and C.B Allen. “Unified fluid-structure interpolation and mesh motion using radial basis functions”. In: *International Journal for Numerical Methods in Engineering* 74.10 (2008), pp. 1519–1559.

[25] J.C. Serralta. and D. Glenis. “BLENDED WING BODY MODEL REDUCTION FOR AEROELASTICITY”. In: (2017).

[26] MSC Software. *MSC Nastran 2012 Quick Reference Guide*. MacNeal-Schwendler Corporation, 2011.

[27] Alexander Verbart, Matthijs Langelaar, and Fred van Keulen. “A unified aggregation and relaxation approach for stress-constrained topology optimization”. In: *Structural and Multidisciplinary Optimization* 55.2 (2017), pp. 663–679.

[28] G. A. Wrenn. “An indirect method for numerical optimization using the Kreisselmeier– Steinhauser function”. In: *Technical Report CR-4220* (1989).

[29] Qu Z. *Model Order Reduction Techniques with Applications in Finite Element Analysis*. Springer-Verlag London, 2004.

[30] Ke-Shi Zhang et al. “Constraint aggregation for large number of constraints in wing surrogate-based optimization”. In: *Structural and Multidisciplinary Optimization* (2018).