UNIVERSIDAD CARLOS III DE MADRID

DEUTSCHES ZENTRUMFÜR LUFT UND RAUMFAHRT

Validation and extension of an MDO framework including dynamic aeroelastic analysis

---

Author: Lucas Bernácer Soriano

Supervisor: Rauno Cavallaro
Co-supervisor: Francesco Torrigiani

A thesis submitted for the degree of
MSc Aeronautical engineering

September 7, 2020

Acknowledgements

This thesis was performed during the first semester of 2020 at the DLR - Deutsches Zentrum für Luft- und Raumfahrt, at the Institute of System Architectures in Aeronautics, as a collaboration between the mentioned institution and the University Carlos III of Madrid. It is my pleasure to acknowledge several individuals who were significant for the completion of my master's thesis.

I am deeply grateful to my university supervisor professor Rauno Cavallaro for bringing me the opportunity to be a part of this collaboration, and for his guidance along the whole project. I also owe my deepest gratitude to my supervisor Francesco Torrigiani, who made my stay in the DLR so much pleasant. Without his incessant help, the presented thesis would not have been as exhaustive as it is.

Last but not least, I would also like to express my most sincere recognition to all my close relatives and friends who showed me love despite not being physically around them during the development of the thesis. Your encouragement has allowed me to keep going through these raw times that unfortunately we have had to live.

Abstract

The optimization discipline has experienced an increase in popularity over the last years, being published multiple research articles and dissertations in relation to this field of study. Besides, the seek for new improvements in the aircraft design process has made the optimization processes to play a huge role due to the new challenges that nowadays are meant to be faced, such as the need for automation and robustness. The foundation of the thesis lies in the definition of a structural optimization problem subjected to an aeroelastic response, being possible to develop two tools whose aim consists of aiding the task of setting up and providing support at the time of solving this particular type of optimization scenarios.

The NOI tool is in charge of providing a direct connection between the theoretical formulation of the problem and its resolution using *Nastran*, whereas an optimization framework named FAEDO is responsible for computing the aeroelastic response of the optimization problem, accounting for the structural, aerodynamic, and stability analysis. The combination of both tools allows conducting a complete optimization subjected to an aeroelastic response by means of OpenMDAO, as it has been proven through several test cases which verify the proper working of the developed framework.

Contents

List of figures

List of tables

1 Introduction

Over the last few years optimization has been growing attention, eventually turning into one of the most important tools in the design process and a recurrent matter in scientific publications. It can be said that the fundamental idea behind optimization consists of determining the most appropriate set of components that fits a given purpose, following certain criteria. This basic concept has been applied to multiple disciplines such as economics, engineering, and even medicine, in spite of getting the best possible results. From all of them, the thesis targets the one related to the structural discipline, and in particular focuses on its integration in the aircraft design process with the development of an optimization framework.

The aircraft design process is constantly under improvement, seeking new methods and techniques to obtain an enhanced final product. According to the traditional design process breakdown, see Figure 1.1, it is usually divided into the conceptual, preliminary, and detailed phases. Furthermore, and as the product moves along the mentioned phases, the number of modeling details increases, being reduced the number of parameters which can be modified. Otherwise, modern aircraft design approaches try to increase the level of details and fidelity in the early phase of the design process, while retaining the same large number of parameters. The two main reasons for this are reducing the design process time and cost, and therefore the vehicle time-to-market, and also the need for physics-based tools in the early phases due to the fact that, for example, novel configurations have extreme flexibility that classical theory cannot predict correctly.

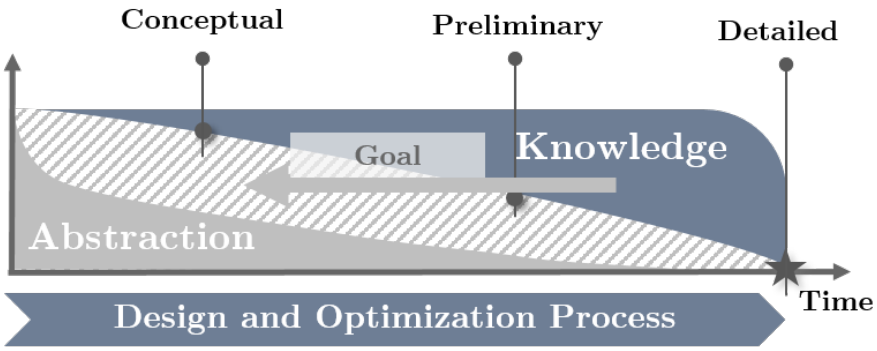


Figure 1.1: Design process breakdown

The nowadays adopted modern design process approach brings mainly two new challenges. One of them corresponds to the seek for automatization, where disciplines like unsteady aeroelastic analysis and methods like FEM and CFD need now to be executed several times and in the shortest possible time. On the other hand, robustness is also a critical aspect to take into account. Not only there are several configurations to be analyzed but these are also very different. These two challenges have been considered at the time of developing the thesis, being directly applied to its main topic which has been the development of a tool able to perform structural optimization problems under flutter constraints on an automatized basis.

In relation to structural optimization, a structure in mechanics can be defined as any assemblage of materials that is intended to endure loads. Hence, structural optimization directly targets making the structure deal with the imposed loads in the best way under certain conditions established as constraints in the problem.

From this point, the reader may have a general idea of which is the aim of optimization, however, it still may be unclear which would be the best way to address this. Many people think that the overall preferred configuration of a structure would be the one that offers light weight and lower cost. Actually, the best output of a structural optimization problem is the one that fulfills all the imposed constraints and still achieves the purpose in an optimum way. In other words, if a structure is being designed in order to avoid buckling, the objective would be to make it as stiff as possible to weave away this phenomenon, but at the same time using the ideal amount of resources. This approach can be summed up in a simple maximization or minimization process involving an objective function subjected to given constraints, which in the structural discipline usually adopt the shape of displacements, stresses, or geometry. At the same time, these constraints could be either defined as objective functions of the problem.

Putting special emphasis now on aerospace structures, the aeroelastic and flutter responses earn much more relevance in the aircraft design process as they can determine the wellness of the overall results, and can be defined as constraints in the process. It is common that these kinds of constraints are included in the optimization problem formulation, in such a way that the output can be profitable due to the fact that, for example, flutter is a safety and certification-critical phenomenon which must be able to be accounted as early as possible in the design process. This condition gives the thesis one of its main purposes, as it has been developed a way of dealing with these types of constraints at the time of setting-up an optimization problem.

Another important aspect that lies down the basis for the thesis has been related to the sensitivity analysis. When performing a gradient-based optimization process, it is extremely useful studying the derivatives, or sensitivity coefficients, of the objective and constraint functions with respect to the design variables. This practice, which goes under the name of sensitivity analysis, allows evaluating how a change in the design variable impacts the problem. Furthermore, its relevance yields in the fact that it aids the optimizer at the time of choosing the proper direction to be followed, and the criteria when updating the design variables.

This brief introduction of elemental optimization related notions offers a glimpse of the topics discussed along the first chapter of the thesis, which mainly concentrates on offering the reader a general overview of the actors involved in a structural optimization problem, the different types that there may be found, the relevance of accounting for aeroelastic constraints, and the remarkable differences with respect the design sensitivity analysis. Moreover, several commercial software provide the possibility to perform structural optimization problems, and sensitivity analysis, focusing on the capabilities offered by *Nastran*.

After all these concepts have been clarified, the goal of the thesis has been developing a tool, denominated NOI, which grants an interface between the chosen structural software and the user-defined optimization analysis, bearing with the inclusion of aeroelastic constraints. As it was previously introduced, *Nastran* is a powerful tool adopted in the industry that allows performing several types of analysis, having a devoted solution sequence to optimization.

The main focus of the tool has been providing the user the means to create an optimization input file to be run by this software, and also a way of dealing with the output results. However, the purpose of NOI does not stop in the optimization solution sequence, it has been included the possibility to provide modal and flutter analysis input files as a way of easing the task at the time of setting-up aeroelastic problems. Chapter 2 presents a complete discussion about the objective, implementation, and limitations of the tool.

Being aware that the thesis has been submitted for an aerospace master's degree, it has been interesting including NOI in a framework capable of solving an optimization problem subjected to an aeroelastic constraint. Therefore, the FAEDO framework was developed in such a way that served as a bridge between the different modules required to define the problem and to compute the aeroelastic constraint. All the involved processes have been discussed along Chapter 3, where a detailed description of the incorporation of NOI into this framework, and the connection with OpenMDAO, has been provided. One of the most important tasks has been granting the precise information flow between the structural and aeroelastic analysis.

Once the NOI tool has been included in the FAEDO framework, it has been required to verify that everything worked in the appropriate way. The fourth chapter gathers all the numerical results regarding several test cases performed to validate the adequate working of the tool, at the time of computing the sensitivity coefficients, and the structural analysis results. Taking as input the Goland wing benchmark model, a DOE and different optimization processes have been evaluated.

1.1. Formulation of the structural optimization problem

A typical optimization problem is usually composed of an objective function, to be minimized or maximized, a set of design variables, and certain constraint functions. The definition of a structural optimization problem can be divided into the same three main topics, which should be established in order to accomplish its complete formulation.

In a general matter, the following variables and functions are the ones required to be specified:

- **Objective function (f) definition**: Denotes the function that points out the goodness of the design. Usually f indicates the weight, stress or displacement in the structure, and a minimization or maximization problem may be formulated, normally prevailing the first ones in the structural discipline.
- **Choice of the design variables (x)**: The design variables can be a function or a vector that represents the design. They are usually related to a geometry feature or even to a choice of material. When talking about geometry design variables, a distinction between shape and size design variable can be performed.
- **State variables (y) designation**: Conforms the responses of the problem for a given design condition. In structural optimization state variables are mainly related to stresses, strains, displacements or forces.

After discussing its fundamental components, the usual structural optimization problem may be presented in the following mathematical way:

$$\begin{cases} \text{Minimize } f(x,y) \text{ with respect to } x \text{ and } y \\ \\ \text{Subject to } \begin{cases} \text{Behavioral constraints on } y \\ \text{Design constraints on } x \\ \text{Equilibrium constraints} \end{cases} \end{cases} \tag{1.1}$$

In certain situations it may arise optimization problems where more than one objective functions may be considered, the so-called multiple criteria optimization problems. For the purpose of solving them, a *Pareto* optimility approach is often followed due to the fact that all the objective functions can not be minimized for the same set of design and state variables. For simplicity's sake, in structural optimization usually there have only been considered single scalar objective function scenarios, see Reference [1] for a deeper discussion.

According to the different types of constraints depicted in Expression 1.1, a more detailed description of each one of them has been provided. The behavioral constraints are the ones acting on the state variable y, in such a way that $g(y) \leq 0$, and often represent a displacement or stress limit. On the other hand, the design constraints would be essentially the same as the behavioral ones, but now involving the design variables x. In relation to the equilibrium constraint, it it is important to write down the structural equilibrium equation to understand its relevance.

$$\boldsymbol{K}(x)\,\boldsymbol{u} = \boldsymbol{F}(x) \tag{1.2}$$

where $\boldsymbol{K}(x)$ corresponds to the stiffness matrix of the structure, $\boldsymbol{u}$ is the displacement vector and $\boldsymbol{F}(x)$ the force acting on the structure. Addressing the mathematical development presented in Reference [1], and treating $\boldsymbol{u}(x)$ as a certain function, the equilibrium constraint can be omitted and replaced by the state variable, reaching the well-known nested formulation expression of the optimization problem:

$$\begin{cases} \min_x & f(x, \boldsymbol{u}(x)) \\ \text{s.t.} & g(x, \boldsymbol{u}(x)) \leq 0 \end{cases} \tag{1.3}$$

At the end, the problem stated through Eq. 1.3 can be solved numerically by the appropriate software, being required the derivatives of f and g, which receive the name of sensitivity coefficients. The process of finding the value of that derivatives conforms the sensitivity analysis, an important step in all the optimization problems, and one of the base topics of the presented thesis.

1.2. Types of optimization problems

Depending on the geometric formulation and the relationships between the design process and the analysis parameters, it is possible to distinguish three different types of optimization problems:

- **Size optimization**: Refers to a design task where the design variables have been defined according to element properties such as thicknesses, cross-sectional areas, Young's modulus, etc.
- **Shape optimization**: For shape optimization, the design variables directly influence the location, the form, or contour of the structural domain. The optimization is mainly based on choosing the integration domain for the differential equations in an optimal way. Boundaries of the problem remain the same, and the outcome corresponds to the best shape that meets the prescribed objective and constraints.
- **Topology optimization**: Mainly based on evaluating if a given element of the structure should be there or not. Hence, the process allows removing mass and modifying the connectivity of the model in view of outputting the optimal response for the proposed objective function.

It is important to note that the shape and topology optimization types refer to different approaches, and therefore conform two different categories although they may seem kind of similar. Furthermore, most of the current optimization software allow setting-up design problems that combine, for example, size and shape design variables, increasing the range of possibilities.

1.3. Incorporation of aeroelastic constraints

Section 1.1 concisely presented the relevance of constraints in the optimization problem formulation and, in particular, structural optimization places special focus on stresses, displacements, and even the geometry, as the main responses that can be defined as constraints. At the time of setting-up optimization problems for aerospace structures, all the relevant physics must be represented so in this way the results can be affordable and constitute a proper response of the model.

As it may be known by the reader, the coupling between the inertial, elastic and aerodynamic forces that act on the body gain relevance in aerospace structures, and play a huge role in optimization, as it has been debated in Reference [2], where wings with very high aspect ratios, and prone to flutter, have been the output of the optimization due to the fact that no aeroelastic constraints were included.

1.3.1. The aeroelastic stability analysis

The evaluation of aeroelastic stability constraints usually involves the solution of eigenvalues of non-linear unsymmetrical and complex matrices. This is the main reason why this section briefly discusses the most common development used to carry out the aeroelastic stability analysis of the structure subjected to optimization, and directly follows the reasoning presented in Reference [3].

The starting point corresponds to the linearized equations of motion, which have been shown in their discretized form through Equation 1.4. There, the vector $\mathbf{u}$ represents the nodal displacements of the FE model, $\mathbf{M}$ the mass matrix, $\mathbf{K}$ the stiffness matrix, and $\mathbf{A}$ the matrix of aerodynamic forces. Furthermore, the dynamic pressure has been defined according to $q = 1/2\,\rho\,v^2$, being ρ the air density and v the flight speed.

$$\mathbf{M\ddot{u}} + \mathbf{Ku} - q\mathbf{Au} = 0 \tag{1.4}$$

From this point, solutions of the form $\mathbf{u} = \bar{\mathbf{u}}e^{i\omega t}$ may be found, and a brand new expression of the motion equations can be formulated:

$$\left[\mathbf{K} - \omega^2\mathbf{M} - q\mathbf{A}(k)\right]\bar{\mathbf{u}} = 0 \tag{1.5}$$

The eigenvalue problem must be solved, being now its main difficulty that the matrix of aerodynamic forces depends on the reduced frequency, which represents the unsteadiness of the problem and has been pictured as $k = \omega b/u$, denoting b the semi-chord of the model. This situation makes the eigenvalue problem shown in Eq. 1.5 to be non-linear since $\mathbf{A}$ is a non-linear function of the reduced frequency.

It can be said that the objective of the aeroelastic analysis is determining the critical flight speed for which the structure becomes unstable. Firstly, the special situation of static instability, where $\omega = 0$, must be analyzed by means of computing the eigenvalues of the real unsymmetric linear eigenvalue problem, see Equation 1.6. This situation allows obtaining the divergence dynamic pressure, which matches the smallest positive real eigenvalue, and then the divergence flight speed can be directly attained following $v_D = \sqrt{2q_D/\rho}$.

$$[\mathbf{K} - q\mathbf{A}]\tilde{\mathbf{u}} = 0 \tag{1.6}$$

On the other hand, the stiffness matrix $\mathbf{K}$ and mass matrix $\mathbf{M}$ are in general sparse, while the matrix of aerodynamic forces $\mathbf{A}$ is complex and dense. Even if the matrix $\mathbf{A}$ is assumed constant, in order to solve Equation 1.5 it is required to employ a subspace in which the eigenvalue problem can be sorted out. The most common approach has been determining this subspace as the space spanned by the eigenvectors, see Reference [3], corresponding to the lowest free vibration frequencies obtained by solving the symmetric generalized linear eigenvalue problem:

$$\left[\mathbf{K} - \omega^2 \mathbf{M}\right] \mathbf{z} = 0 \tag{1.7}$$

After some development, which has been gathered in Reference [3], it is possible to reach Expression 1.8, where $\mathbf{\Omega}$ is diagonal with the free vibration eigenvalues ω_i^2 on the diagonal and $\hat{\boldsymbol{A}} = \boldsymbol{Z}^{\boldsymbol{T}} \mathbf{A} \mathbf{Z}$ is the matrix of generalized aerodynamic forces.

$$\left[\mathbf{\Omega} - \omega^2 \mathbf{I} - q\hat{\mathbf{A}}(k)\right] \hat{\mathbf{u}} = 0 \tag{1.8}$$

Different approaches may be found in order to solve the nonlinear eigenvalue problem. One of them is the famous k-method, where damping is introduced as a computational aid. Then, the eigenvalue problem can be solved for a number of different values of the reduced frequency, and finally the non-dimensional speed and damping factor g, for each eigenvalue and reduced frequency, can be plotted in a graph. The non-dimensional flutter speed has been defined as the lowest speed for which the damping factor becomes positive, being the speed for which positive damping is needed for neutral stability. The resolution followed by other methods such as the p-method or the p-k method have also been presented in Reference [4].

In practice, and once the relevance of the aeroelastic stability analysis has been discussed, the results are provided in the way of V-g and root locus plots. The first one of them represents the speed versus the damping parameter, in this way, it is possible to spot the behaviour of each one of the modes and if coupling happens between any of them. Otherwise, the root locus aids at the time of determining the flutter point of the structure as it illustrates the real and imaginary part of the eigenvalues, being clear that flutter occurs when a crossing of the imaginary axis takes place for the considered branches. All along the thesis, and in particular along Chapter 4, these plots have been used in order to analyze the results of the optimization, and as a way of checking the aeroelastic response of the model.

This section has presented the practical relevance of including an aeroelastic constraint, and how the stability analysis of the model plays a huge role in the overall response of the structure when subjected to aerodynamic loads. Furthermore, and thanks to these types of constraints, it is possible to avoid too conservative designs that may provide heavier and stiffer structures, whereas an unconstrained aeroelastic optimization may also lead to an excessively dangerous flexible response.

1.4. Design sensitivity analysis

As noted in the previous sections, it is important to make a distinction between the design sensitivity and optimization concepts. Both terms have been highly related, and for a given design problem, the sensitivity analysis provides the variation of the structural responses with respect to the design variables. In size optimization, these design variables may correspond to thicknesses and areas, while for shape optimization the grid point locations can also be considered. For the particular scenario of the aerospace sector, it is relatively important to know how a certain change in a design variable may impact a given component, or even the whole structure, so in this way it is possible to assess the best path to be followed.

The design optimization concept goes a step further and involves the process of improving the proposed design. To do so, an optimizer is required and the design sensitivities serve as an input to guide the optimizer in the proper direction.

Coming back to the sensitivity coefficients of the problem, and paying special attention to the sensitivity analysis, they represent the change of a structural response, or objective, with respect to variations in the design variables, serving as the essential gradient information for the optimizer. A design coefficient can also be defined as the rate of change of a given response with respect to a design variable, being these coefficients evaluated at a particular design point:

$$\lambda_{ij} = \left.\frac{\partial g_i}{\partial x_j}\right|_{x^0} \tag{1.9}$$

where j indicates the j-th design variable and i the i-th response, as sensitivity coefficients are required for the whole set of design variables and responses spectrum of the problem. Note that the sensitivity coefficient presented in Equation 1.9 would be just the slope of the response with respect to the design variable. Hereafter, there have been described the two main methods in order to compute the sensitivity coefficients for arbitrary functions and design variables according to Reference [1].

1.4.1. Numerical methods

The starting point relies on the nested optimization formulation, being represented the general optimization problem through Expression 1.10. In a numerical approach, the terms corresponding to $\partial \hat{g}_i / \partial x_j$ are computed by means of finite differences, for example with a forward or central scheme.

$$\begin{cases} \min_x & \hat{g}_0(\boldsymbol{x}) = g_0(\boldsymbol{x}, \boldsymbol{u}(x)) \\ \text{s.t.} & \hat{g}_i(\boldsymbol{x}) = g_i(\boldsymbol{x}, \boldsymbol{u}(x)) \leq 0, \quad i = 1, \ldots, l \\ & \boldsymbol{x} \in \mathcal{X} = \left\{ x \in \mathbb{R}^n : x_j^{\min} \leq x_j \leq x_j^{\max}, j = 1, \ldots, n \right\} \end{cases} \tag{1.10}$$

The following equation illustrates the finite differences forward scheme for the proposed problem:

$$\frac{\partial \hat{g}_i\left(\boldsymbol{x}^k\right)}{\partial x_j} \approx \frac{\hat{g}_i\left(\boldsymbol{x}^k + h\boldsymbol{e}_j\right) - \hat{g}_i\left(\boldsymbol{x}^k\right)}{h} \tag{1.11}$$

where $\boldsymbol{e}_j = [0, \ldots, 0, 1, 0, \ldots, 0]^T$ and the 1 is in row j.

This numerical approach can be easily understood through the set-up of a fictitious example. Let's suppose that the reader wants to compute at given design conditions the sensitivity of the objective response with respect to the thickness of a shell element j in the structure. Equation 1.11 depicts the required terms to calculate the sensitivity of $\hat{g}_0(\boldsymbol{x})$ in relation to the design variable t_j. The contribution coming from $\hat{g}_0(\boldsymbol{x}^k)$ can be directly computed, as it corresponds to the value of the objective at the considered design point. Therefore, a perturbation h has to be added to the design variable seeking to compute the remaining term. After the design variable has been perturbed, the displacements field must be solved again by means of finite element analysis. Then, and once $\boldsymbol{u}(\boldsymbol{x}^k + h\boldsymbol{e}_j)$ is no longer an unknown, the term $\hat{g}_0(\boldsymbol{x}^k + h\boldsymbol{e}_j)$ can be obtained, being able to retrieve the whole sensitivity coefficient.

One of the problems that may appear with this technique would be related to how the value of the perturbation shall be defined. If h is too large, the approximation of the coefficient happens to be less accurate. In contrast, a relatively low value of the perturbation also introduces problems, as the numerical error due to cancellation increases. Using a central finite differences scheme could help to improve the accuracy of the method, being its main advantage the ease of implementation in comparison with the analytical procedure.

1.4.2. Analytical methods

The second approach used to compute the sensitivity coefficients has been based on the analytical formulation. On behalf of retrieving an analytical expression for $\partial \hat{g}_i(\boldsymbol{x}^k)/\partial x_j$, the chain rule must be applied providing the following equation:

$$\frac{d\hat{g}_i\left(\boldsymbol{x}^k\right)}{dx_j} = \frac{\partial g_i\left(\boldsymbol{x}^k, \boldsymbol{u}\left(\boldsymbol{x}^k\right)\right)}{\partial x_j} + \frac{\partial g_i\left(\boldsymbol{x}^k, \boldsymbol{u}\left(\boldsymbol{x}^k\right)\right)}{\partial u} \frac{\partial \boldsymbol{u}\left(x^k\right)\right)}{\partial x_j} \tag{1.12}$$

where the response function $\hat{g}_i(\boldsymbol{x}^k)$ directly depends on the set of design variables x_j and also on the displacement vector $\boldsymbol{u}(x^k)$. This aspect explains the two different terms that appear in Equation 1.12 due to the differentiation process. The following lines try to summarise two analytical resolution procedures, the direct and the adjoint methods, and their working principles in order to attain the sensitivity coefficients.

A deeper discussion of the stated approaches can be found in Reference [1], as the objective of the thesis does not include the theoretical development of these procedures and just seeks to provide the reader with insight about the different methods used to compute the sensitivity coefficients.

Essentially, both methods take as their common starting point the differentiation of the equilibrium equation $\boldsymbol{K}(x)\,\boldsymbol{u} = \boldsymbol{F}(x)$, which can be written as:

$$\frac{\partial \boldsymbol{K}\left(\boldsymbol{x}^k\right)}{\partial x_j}\boldsymbol{u}\left(\boldsymbol{x}^k\right) + \boldsymbol{K}\left(\boldsymbol{x}^k\right)\frac{\partial u\left(\boldsymbol{x}^k\right)}{\partial x_j} = \frac{\partial \boldsymbol{F}\left(\boldsymbol{x}^k\right)}{\partial x_j} \tag{1.13}$$

After rearranging certain terms, the new sorted expression has been shown in Equation 1.14, where the right hand side is usually known as the *pseudo-loads*. From here on, and in relation to the two previously mentioned analytical procedures, their corresponding developments have been presented.

$$\boldsymbol{K}\left(\boldsymbol{x}^k\right)\frac{\partial \boldsymbol{u}\left(\boldsymbol{x}^k\right)}{\partial x_j} = \frac{\partial \boldsymbol{F}\left(\boldsymbol{x}^k\right)}{\partial x_j} - \frac{\partial \boldsymbol{K}\left(\boldsymbol{x}^k\right)}{\partial x_j}\boldsymbol{u}\left(\boldsymbol{x}^k\right) \tag{1.14}$$

1.4.2.1. The direct method

The direct analytical method has been illustrated through the computation of the sensitivity of the compliance $\hat{g}_0(\boldsymbol{x}) = g_0(\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{x})) = \boldsymbol{F}(\boldsymbol{x})^T\boldsymbol{u}(\boldsymbol{x})$. Thus, and after applying differentiation, the terms in the following expression can be found.

$$\frac{\partial g_0\left(\boldsymbol{x}^k, \boldsymbol{u}\left(\boldsymbol{x}^k\right)\right)}{\partial x_j} = \frac{\partial \boldsymbol{F}\left(\boldsymbol{x}^k\right)^T}{\partial x_j}\boldsymbol{u}\left(\boldsymbol{x}^k\right), \quad \frac{\partial g_0\left(\boldsymbol{x}^k, \boldsymbol{u}\left(\boldsymbol{x}^k\right)\right)}{\partial \boldsymbol{u}} = \boldsymbol{F}\left(\boldsymbol{x}^k\right)^T \tag{1.15}$$

Furthermore, from Expression 1.14 it is possible to solve $\partial \boldsymbol{u}(\boldsymbol{x}^k)/\partial x_j$ as presented in Eq. 1.16, and input this expression all together with the differentiation of the compliance to finally obtain the formulation of the analytical sensitivity coefficient through Equation 1.17.

$$\frac{\partial \boldsymbol{u}\left(\boldsymbol{x}^k\right)}{\partial x_j} = \boldsymbol{K}\left(\boldsymbol{x}^k\right)^{-1}\left(\frac{\partial \boldsymbol{F}\left(\boldsymbol{x}^k\right)}{\partial x_j} - \frac{\partial \boldsymbol{K}\left(\boldsymbol{x}^k\right)}{\partial x_j}\boldsymbol{u}\left(\boldsymbol{x}^k\right)\right) \tag{1.16}$$

$$\begin{aligned}\frac{d\hat{g}_0\left(\boldsymbol{x}^k\right)}{dx_j} &= \frac{\partial \boldsymbol{F}\left(\boldsymbol{x}^k\right)^T}{\partial x_j}\boldsymbol{u}\left(\boldsymbol{x}^k\right) + \boldsymbol{F}\left(\boldsymbol{x}^k\right)^T\boldsymbol{K}\left(\boldsymbol{x}^k\right)^{-1}\left(\frac{\partial \boldsymbol{F}\left(\boldsymbol{x}^k\right)}{\partial x_j} - \frac{\partial \boldsymbol{K}\left(\boldsymbol{x}^k\right)}{\partial x_j}\boldsymbol{u}\left(\boldsymbol{x}^k\right)\right) \\ &= 2\boldsymbol{u}\left(\boldsymbol{x}^k\right)^T\frac{\partial \boldsymbol{F}\left(\boldsymbol{x}^k\right)}{\partial x_j} - \boldsymbol{u}\left(\boldsymbol{x}^k\right)^T\frac{\partial \boldsymbol{K}\left(\boldsymbol{x}^k\right)}{\partial x_j}\boldsymbol{u}\left(\boldsymbol{x}^k\right)\end{aligned} \tag{1.17}$$

1.4.2.2. Adjoint method

The adjoint method formulation takes place when Equation 1.14 is combined with Equation 1.12, yielding the following expression for the sensitivity coefficient:

$$\frac{\partial \hat{g}_i\left(\boldsymbol{x}^k\right)}{\partial x_j} = \frac{\partial g_i}{\partial x_j} + \frac{\partial g_i}{\partial \boldsymbol{u}} \boldsymbol{K}\left(\boldsymbol{x}^k\right)^{-1} \left(\frac{\partial \boldsymbol{F}\left(\boldsymbol{x}^k\right)}{\partial x_j} - \frac{\partial \boldsymbol{K}\left(\boldsymbol{x}^k\right)}{\partial x_j} \boldsymbol{u}\left(\boldsymbol{x}^k\right) \right) \tag{1.18}$$

One of the remarkable aspects of the adjoint methodology is the requirement to define Eq. 1.19 in order to account for solving the derivatives with respect the design variables. Hence, and operating with Equation 1.14, it is possible to reach the analytical expression of the sensitivity coefficient as seen in Expression 1.20.

$$\boldsymbol{K}\left(\boldsymbol{x}^k\right) \lambda_i = \left(\frac{\partial g_i}{\partial \boldsymbol{u}} \right)^T \tag{1.19}$$

$$\frac{\partial \hat{g}_i\left(\boldsymbol{x}^k\right)}{\partial x_j} = \frac{\partial g_i}{\partial x_j} + \lambda_i^T \left(\frac{\partial \boldsymbol{F}\left(\boldsymbol{x}^k\right)}{\partial x_j} - \frac{\partial \boldsymbol{K}\left(\boldsymbol{x}^k\right)}{\partial x_j} \boldsymbol{u}\left(\boldsymbol{x}^k\right) \right) \tag{1.20}$$

The main difference between the direct and the adjoint method has been found in the way both procedures compute the coefficients for a given set of $j = 1, ..., n$ responses and $i = 0, ..., l$ design variables. According to the direct formulation, it is compulsory to solve Equation 1.14 for each design variable i, meaning n times in overall. The result is then requested to solve Equation 1.13 a total of $l + 1$ times for each j. On the other hand, the adjoint method solves Equation 1.19 for the objective function and for each constraint function $l + 1$ times, and recalls the results from Eq. 1.20 n times for each design variable. Therefore, it can be concluded that the adjoint method ideally shows better performance when the number of constraints is lower than the number of design variables, otherwise, the direct method is recommended.

1.5. Structural optimization in *Nastran*

These days it can be found several software packages that implement the finite element method in order to solve structural problems. From all of them, it has been decided to pay close attention to *Nastran*, which can be defined as a multidisciplinary structural analysis tool that is capable of carrying out static, dynamic, and thermal analysis. For the reason that *Nastran* has been considered the industry standard for structural analysis simulation, it has been decided to use this FE package for the developed thesis, where the special emphasis has been placed in the structural optimization module that goes under the name of solution sequence 200. Nonetheless, and in case the reader is not used to the *Nastran* format, a brief presentation has been provided along the following lines.

1.5.1. Input and output data format

At the time of performing any kind of analysis using *Nastran* it is required to provide the software a properly formatted input text file. These text files are commonly referred to as decks due to the fact that in the past the information was actually stored into punch cards that were read by the machine, resembling a deck. The input file, which can show a BDF or DAT extension, has to contain in a mandatory fashion the following sections:

- **Executive control section:** Accommodates the lines where the type of solution sequence and various diagnostics are specified. As it has been previously mentioned, *Nastran* has certain predefined solution sequences for each type of study. The most popular ones correspond to solution sequences 101 for linear static analysis, 103 for normal modes and 200 for optimization. Reference [5] includes the complete list of available solution sequences.
- **Case control section:** This section of the input text file is the one in charge of defining the loads, constraints and objective function, determining the contents of the model results output files, setting the coordinate system, and creating the subcase structure for all the performed analysis.
- **Bulk data section:** Considered the main section of the input file as it is where the geometry model, element connectivities, materials, and loads are defined. It is not order dependent, and its entries can be provided in three different formats depending on the kind of delimiter used between tabs, commas and spaces. Furthermore, it is also the place to set-up the particular entries corresponding to the selected solution sequence. For example, in case it has been requested to perform an optimization analysis, the bulk data section must contain the design variables, objective function and response entries.

Once the simulation takes place, the results can be provided by *Nastran* in several output formats. The prevailing one corresponds to the well-known f06 file, which conforms a text file with results from analysis along with diagnostic messages that report how the simulation has performed. Nonetheless, it is also popular to store the results in a binary file, known as OP2, as it is easier to handle the data in order to post-process it. A more detailed description regarding how the input and output files are formatted can be found in *Nastran's* quick reference guide, see Reference [5], but for the purpose of the thesis, just the f06 and OP2 output formats have been used.

1.5.2. The optimization solution sequence

One of the most important aspects, at the time of creating an optimization input file, is the process that relates the finite element model stored in the bulk data section with the design model used to define the optimization problem. This process, summed up through Figure 1.2, has been divided into several steps which essentially gather the ones previously mentioned at the time of formulating the structural optimization problem. Each one of the steps involves including certain *Nastran* cards in the case control or bulk data sections, which would be in charge of defining, for example, the design variables, objective function, or constraints of the problem.

At the time of setting-up the design variables, it is worth to mention that *Nastran* allows defining multiple types of design variables such as element properties, connectivity of elements, or even grid point locations. The DESVAR bulk data entries are the ones responsible for specifying the design variables of the problem, however, they can only affect the finite element analysis if they are linked with designed properties, shapes and/or design responses. This means that the mentioned features of the finite element model must be explicitly related to their corresponding DESVAR entries in the bulk data section through particular *Nastran* bridge cards, see Table 1.1. Furthermore, it is the user's responsibility to define the initial values of these design variables and their upper and lower bounds in the DESVAR card.

Type of design variable	***Nastran* bridge entry**
Element property	DVPREL
Material property	DVMREL
Connectivity of elements	DVCREL
Grid point location	DVGRID

Table 1.1: Bridge entries and types of design variables

Another remarkable matter is the design responses, which are used by *Nastran* in order to specify the objective and constraints functions. Thus, before the objective function and constraints can be included in the bulk data section, the analysis responses must be specified using the DRESP set of bulk data entries. There can be found several types of responses, DRESP1 is the most commonly used card in order to define, for example, the weight, displacements, or stresses of the model.

Once the responses have been included, it is possible to specify the ones conforming the set of constraints, or the objective function. The process recalled to implement the design model constraints involves two new *Nastran* entries. Essentially, each DRESP card must be linked with a DCONSTR entry in order to specify the constraint bounds. These responses must also be gathered in a DCONADD entry depending on the subcase to which they belong. Besides, and now acting on the case control section, it must be specified the particular set of constraints to be used for each one of the subcases through the DESSUB card, or just set the global set of constraints by means of the DESGLB entry.

The objective function must be a scalar quantity that is either minimized or maximized by the optimizer. So, it shall be specified in the case control section using the DESOBJ case control command. This entry points to a design response defined on either a DRESP1, DRESP2 or DRESP3 bulk data card and must be a single scalar response. In structural optimization it usually prevails the weight of the model as the objective function, however, *Nastran* allows specifying a wider range of scalar responses as objectives of the problem. It is important to note that each optimization input file only permits the definition of a unique objective function.

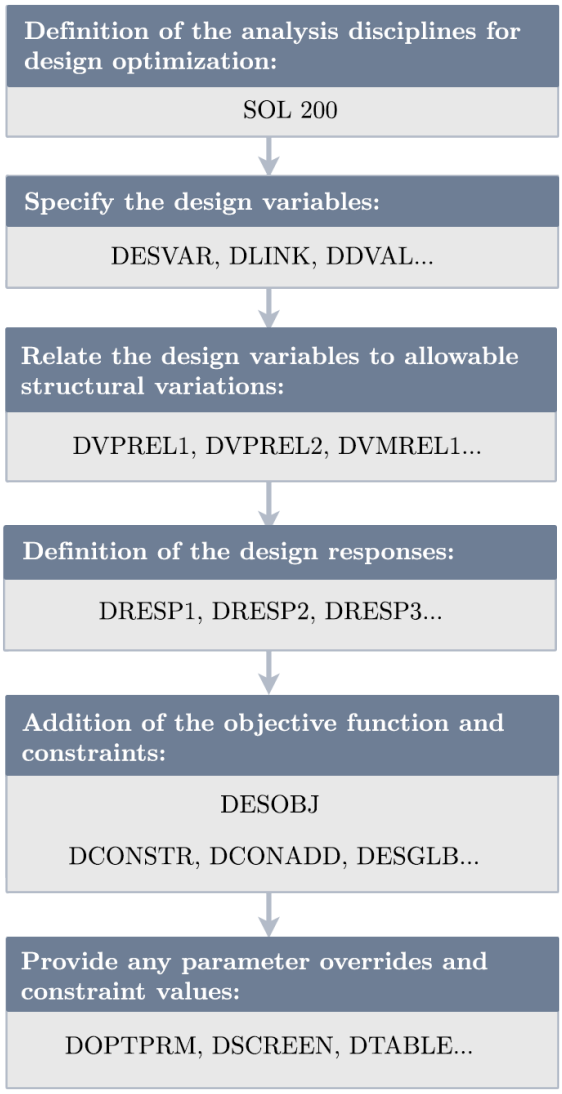


Figure 1.2: *Nastran* optimization input file sequence

The mentioned steps, which have been depicted in Figure 1.2, are the essential ones to be followed in order to accomplish a properly formatted optimization input file. A deeper description has been provided in *Nastran's* design sensitivity and optimization user guide, see Reference [6]. There, each one of the steps has been described in further details and the reader may get a broader understanding of how *Nastran* performs. Nonetheless, the briefly provided explanation in the previous lines happens to be more than enough for the reader to get a general idea.

One of the main advantages of the *Nastran* optimization module is that it allows implementing several analysis scenarios, structuring them in different subcases. Each subcase can be related to a different analysis type, allowing the user to end up creating an MDO interface. At the moment, the solution sequence 200 module only supports static, modal and buckling analysis types, however, and attending to the objective of the thesis, only the first two analysis types have been necessary.

According to the design sensitivity and optimization user guide, Reference [6], and reminding the previous literature discussion about the different procedures to compute sensitivities, it is important to clarify that the software uses finite difference techniques where it is not practical using analytical sensitivities, making a distinction between forward and central finite differences schemes. In relation to the analytical approaches, it supports two different methods: direct sensitivities and adjoint. The direct method uses a semi-analytic formulation, while the adjoint employs a scheme that first solves for the adjoint solutions that were introduced by means of Eq. 1.19.

2 | NOI

The developed optimization related tool, which goes under the name of *Nastran* Optimization Interface, NOI, serves the purpose of granting a connection between a user provided input file, which includes the complete definition of the finite element model, and the solution sequence for design optimization. The main objective that has driven the development of the tool was the extraction of the sensitivity coefficients of the proposed design sensitivity analysis, being the tool able to format all the required *Nastran* definition cards and to handle the output of the simulation.

In order to ease the comprehension of the tool, its implementation has been divided into three different modules. The first one has been devoted to transforming the original BDF input file according to the optimization problem definition, meaning that in this particular stage the model design variables, responses, and the analysis subcases are meant to be established. Nonetheless, inside this module there has also been included the option to write modal and flutter analysis *Nastran* input files. The reason behind this yields in the fact that once the FE model has been provided to the tool, it may be interesting to not limit its capability to the optimization field and provide the user a greater range of possibilities.

The second module has been oriented to the post-processing of results once *Nastran* has provided the output OP2 file, and its purpose would be mainly devoted to the modal, static, and sensitivity results handling. The extraction of the weight, sensitivity matrix, stresses, eigenvalues or eigenvectors has been structured in two different blocks depending on the type of analysis performed. Thus, the user is able to access the information in a much easier way than trying to understand the traditional *Nastran* output formats. This module also aids in the task when the tool has been included in an optimization framework, in such a way that is responsible of providing the optimizer with the required data coming from *Nastran's* analysis.

Last but not least, the third module of NOI has been devoted to creating a platform that takes care of the results stored in a F06 file. It must be clarified that so far there has only been included the possibility to read the flutter results coming from solution sequence 145, however, this module is expected to be extended in order to process all the information that can not be stored in an OP2 file. Regarding the flutter results, the V-g, V-f and root locus representations for the analysis can be directly displayed. This particular module, and although it is not directly related to the optimization area, supposes a great relief when dealing with *Nastran's* flutter output files.

2.1. NOI overview

This section serves the purpose of providing an initial description about how the tool has been developed, how it works, and also to deliver a discussion about its capabilities and limitations. Hence, and as it has been formerly introduced, the tool can be divided into three different modules, being important to remark that each one of them is capable of working in an individual basis. The presented modules have been coded throughout a set of classes in *Python*, which at the same time include a series of methods that allow the program to perform all the required actions.

From the optimization point of view, the fact that one of them prepares the *Nastran* input file and the other one handles the results allows the tool to be incorporated in a workflow environment, where its main task would be the one of extracting the eigenvalues, eigenvectors and sensitivities of the proposed structural model. Even so, and in the scenario of neglecting this collective approach, the tool still can be used to prepare *Nastran* optimization, modal, and flutter files, or as an output reader for modal, static and sensitivity analysis results.

The core of NOI is the *pyNastran* package developed by Steve Doyle, which grants an interface to work with *Nastran's* complicated input/output files. The BDF editor has been used to read and write the optimization, modal and flutter information, supporting more than three hundred different cards, and being these cards in charge of setting-up the output file. Card objects have methods to access data such as design variables, objective functions and constraints definition. Thus, the first module takes as its starting point the mentioned BDF editor and expands its capabilities, conforming the XBDF class that at the same time provides support for the writing of the optimization, modal and flutter input files.

On the other hand, the second module of the tool recalls the OP2 reader in order to access the static/transient information and creates an extension that goes under the name of XOP2. From that point, two branches appear depending on the type of results that should be read. The first one has been directly related to the static analysis, providing the weight, stress and sensitivity information, while the second one is oriented to the modal results. This second branch is capable of outputting the structural matrices of the model, eigenvectors and sensitivities. Furthermore, *pyNastran* also includes an F06 reader that accommodates the basis of the XF06 class that conforms NOI's third module, being able to process the information written in an F06 file and to intercede when the XOP2 capabilities have been exceeded.

As it has been discussed, the *pyNastran* package only assures the purpose of reading and writing information according to the *Nastran* format, NOI is the one responsible of handling, for example, the input of the optimization problem, writing the proper entries, and finally provide a user-friendly format for the results. The breakdown of the tool, which has been depicted in Figure 2.1, eases the comprehension of how it has been structured. From this point, and all along this chapter, a deeper discussion about the implementation and functionalities of each one of its modules has been handed over to the reader.

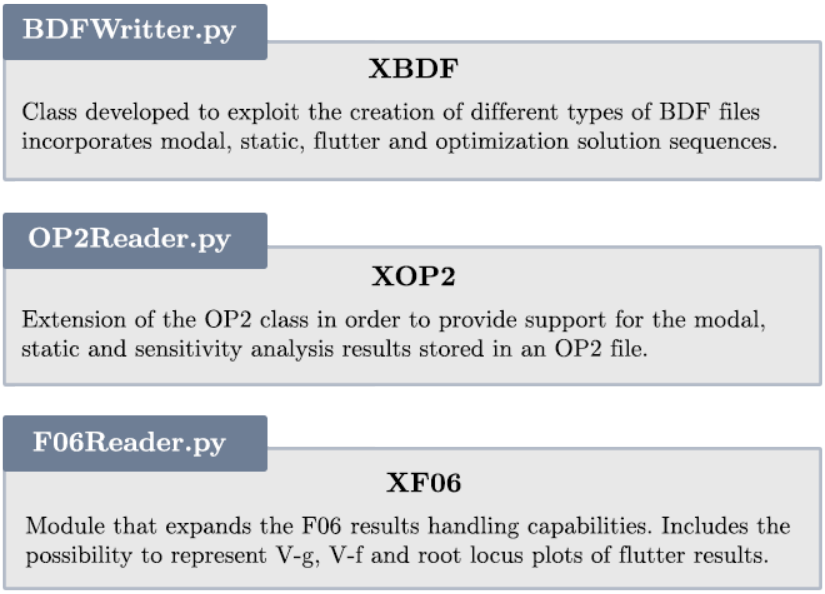


Figure 2.1: NOI breakdown diagram

2.2. BDF writter module

The first module, which contains the parent XBDF class, is responsible of creating input files for optimization, modal, and flutter analysis. As a result of *pyNastran's* restrictions, the input finite element model should be provided in a BDF file with the executive, case control and bulk data sections already defined. From this point, the XBDF class allocates the information regarding the geometry, element connectivity, material, loads, and boundary conditions prior starting the creation of new entities.

Inside the BDF writer module, several classes and methods can be found in order to account for optimization, modal and flutter *Nastran* input files. Each one of them has a devoted class that directly inherits the FE model from XBDF, so in this way it has been avoided redundant coding and the structure of the module happens to be much cleaner. Figure 2.2 depicts all the classes stored in the module, sorted according to the provided output. The first one of them is responsible of including all the required sensitivity and optimization definition cards in the already user prepared *Nastran* model. Otherwise, the second and third classes are in charge of writing the modal and flutter BDF files. It is important to note that due to the way the flutter analysis has been defined, a lot of information coming from the modal class can be used at the time of setting-up the flutter input file.

Each one of the methods inside the classes may require certain input coming from the user, which is usually supplied in the shape of *Python* dictionaries. For example, for the optimization methods there are requested the dictionaries with the design variables, responses and analysis information. However, for the creation of the modal or flutter BDF there is only demanded the number of modes, normalization method and flutter conditions.

Due to the fact that the major topic of the thesis falls towards the structural optimization area, a detailed description about how to properly create an optimization file has been presented in subsection 2.2.1. Moreover, it is worth mentioning that as NOI started to mature, the possibilities of providing modal and flutter BDF files were included and there has also been explained how to take advantage of these capabilities in subsection 2.2.2.

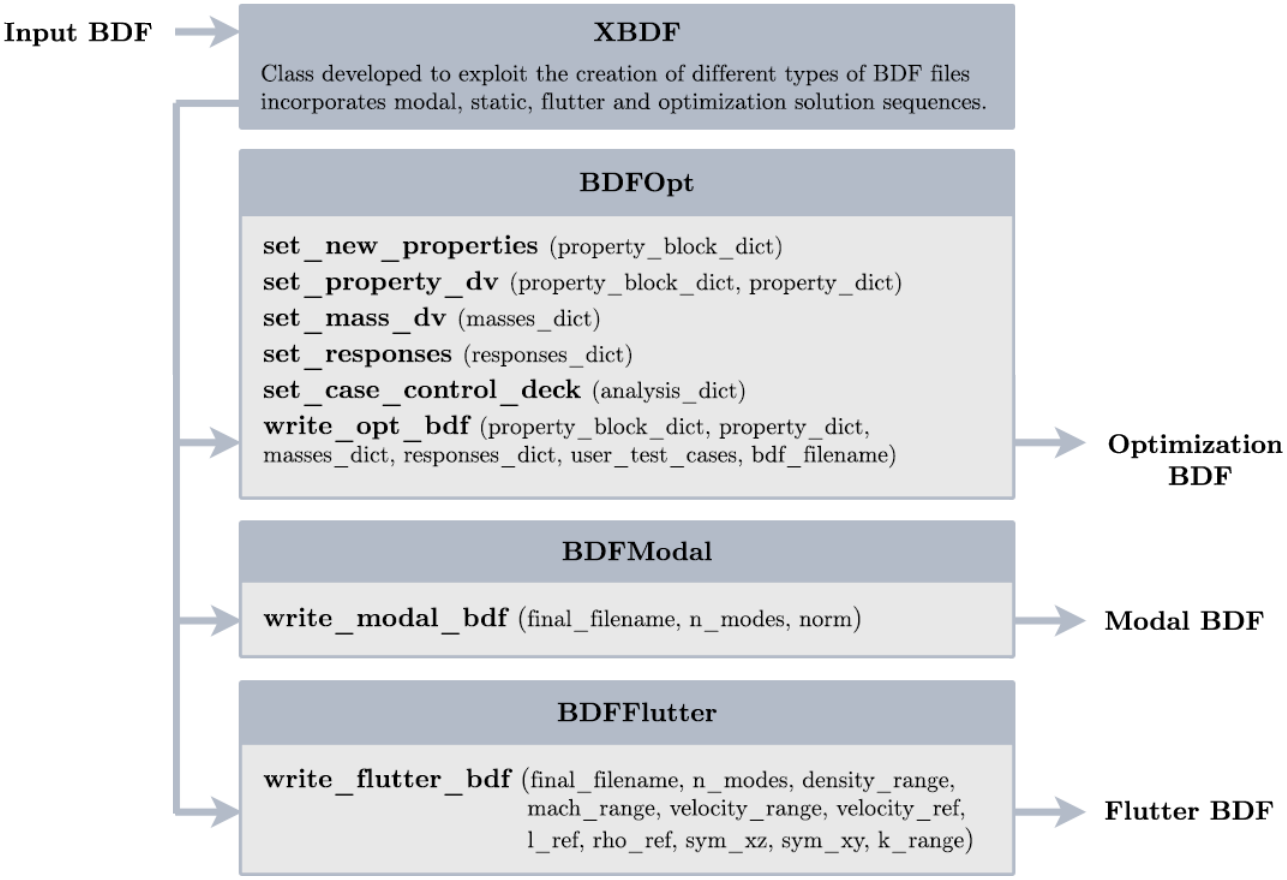


Figure 2.2: BDF writer module breakdown

2.2.1. Optimization input files

In place of writing an optimization file, the BDFOpt class of the first module shall be recalled. This class has been made-up by a set of methods that depend on the optimization problem statement, and being required certain user input. Hence, the class methods are in control of taking the user provided information and translate it into *Nastran* optimization entries by means of *pyNastran's* capabilities. Five main functions, which have been previously introduced in Figure 2.2, have to be discussed in relation to the design model definition.

Regarding the definition of design variables, the tool allows the possibility to set finite element properties or lumped masses as design variables, anyhow, in the near future there has been considered the prospect to introduce shape variables. The property related design variables have been defined according to the "*set_new_properties*" and "*set_property_dv*" methods, while just a single "*set_mass_dv*" function is required when handling punctual masses. The difference between these two types of design variables lays on the optimization problem and in the FE model definition. In certain scenarios it may be interesting to study the influence of the model lumped masses, the element properties, or the combination of both, so the tool accounts for all these conditions.

NOI also takes care of the responses and the case control deck implementation calling the "*set_responses*" and "*set_case_control_deck*" methods. It has been allowed two types of subcases, modal and statics, in such a way that the user is free to combine them. However, and depending on the kind of subcase, the responses of the model have been limited to the weight, stress, eigenvalues and eigenvectors.

From now on, a detailed description of each one of the methods included in this optimization class has been presented. Besides, it is strongly recommended to the reader to recall Figure 2.3 in order to get a general overview of how the BDFOpt class has been structured and to understand the purpose of each one of its methods.

2.2.1.1. Element properties design variables definition

As the tool has been firstly oriented to size optimization, the available element properties design variables depend on the element types of the user specified FE model, meaning that for each element type *Nastran* establishes certain properties that are somehow settled. Table 2.1 gathers the most commonly used element types, their associated *Nastran* property entries, and the properties that could be defined as design variables by NOI. The design sensitivity and optimization user's guide compiles the complete list of properties associated to each element type, but up to now only the most remarkable ones have been included. Nonetheless, it is responsibility of the user to be aware of the FE model elements and their available properties prior trying to set them as design variables in the tool.

Element entry	**Property entry**	**Allowed design variables**
CBAR	PBAR	A, I_1, I_2, I_{12}, J ...
CBEAM	PBEAM	A, I_1, I_2, I_{12}, J ...
CROD	PROD	A, J, C, NSM ...
CQUAD4	PSHELL	T, TST, NSM, Z1 ...
CELAS1	PELAS	K1, GE1, S1 ...

Table 2.1: Elements, properties and design variables in *Nastran*

The first task of the "*set_new_properties*" method is, taking as input the original FE model coming from XBDF, creating the adequate property entries according to the user selected elements that then will be associated with a certain design variable.

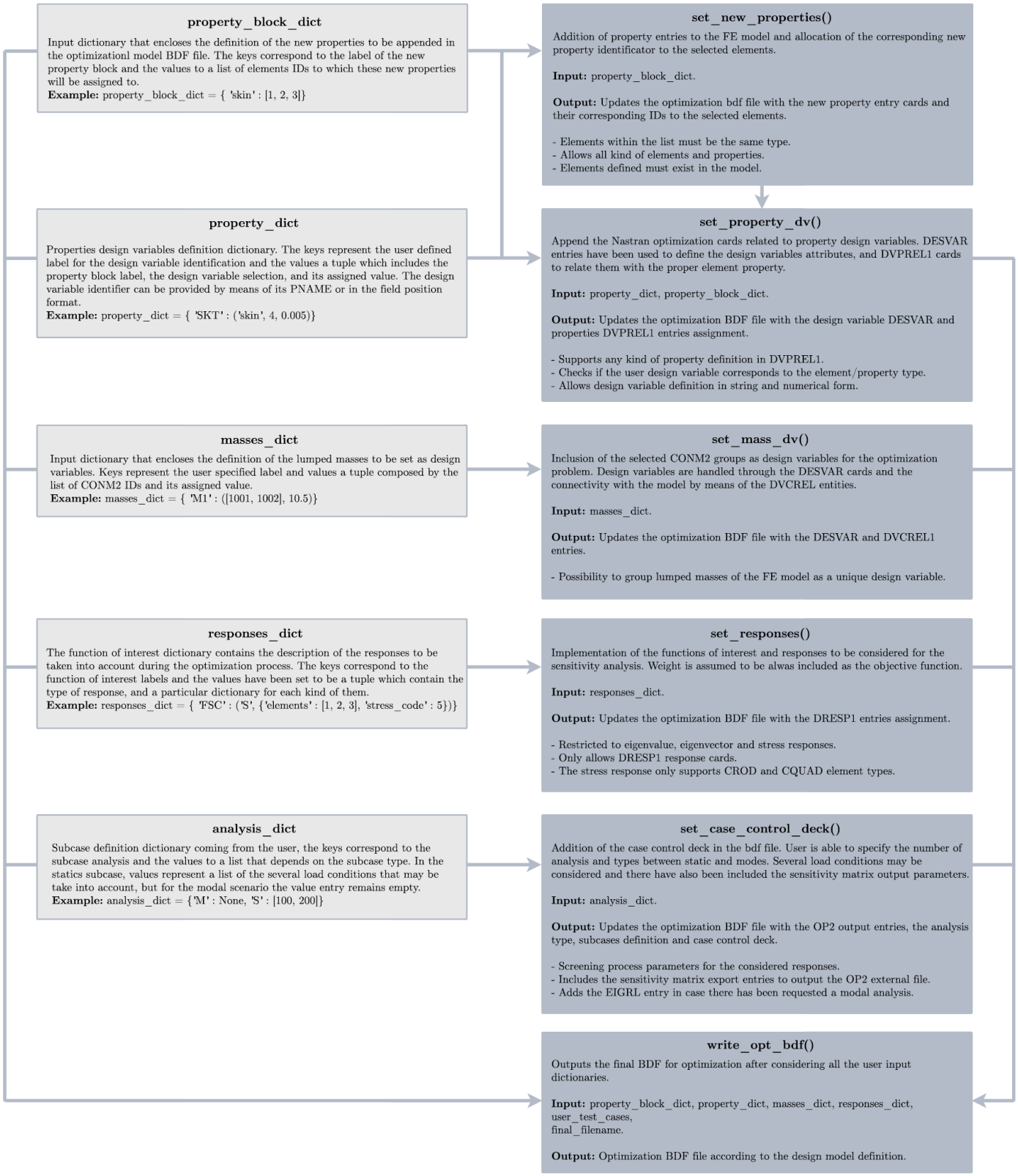


Figure 2.3: BDFOpt class methods diagram

This process has been explained through Figure 2.4, where a new property entry has been included, making possible to split the upper skin of the presented geometry in two different blocks. Despite the fact that this operation seems rather trivial, it is extremely important because it lays down the basis for defining element properties design variables according to new blocks of properties that have been appended to the FE model.

In order to clarify the concept of properties definition, it may be reformulated in the following way: the entries corresponding to element properties design variables in *Nastran* need to be linked with a certain property card, which at the same time is related to an element or block of elements. For example, in the situation the user wants to consider as a design variable the thickness of a given CQUAD4 element, a new PSHELL card has to be created pointing to the chosen element, being the tool responsible of managing this new property creation.

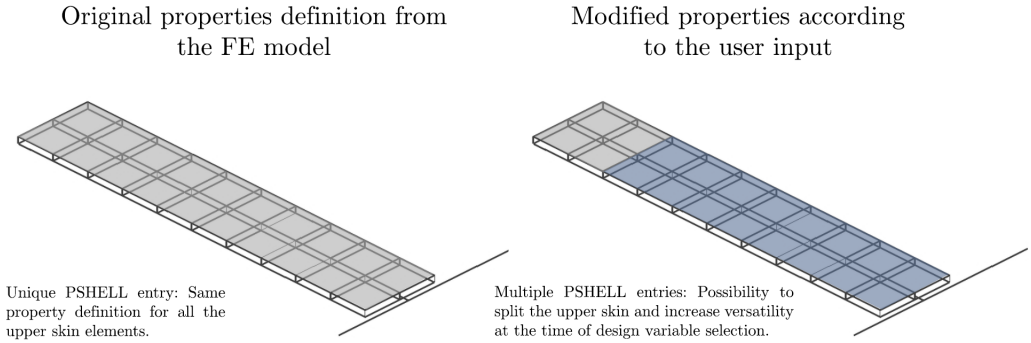


Figure 2.4: Properties definition example in *Nastran*

Then, the user is able to set as many design variables as it wants, not being these design variables limited to just one element. This aspect means that the same design variable could be specified for a block of the same type of elements that shared initially the same property card. The user input format for defining the previously mentioned new blocks of properties has been presented in Figure 2.3, showing a *Python* dictionary entry which holds as keys the label for the components and as values the list of the affected elements for which new properties are going to be created. Furthermore, this method also recognizes the non-used or repeated properties and erases them from the model.

Once the new properties have been added to the design model, the optimization related DESVAR and DVPREL1 cards have to be included by means of the "*set_property_dv*" method. These inputs are required for *Nastran* to identify the desired design variables and to associate them with their corresponding properties, performing the sensitivity analysis in relation to them. In a certain sense, the DVPREL1 card associates the design variable with the property from the FE model, serving as a kind of bridge between the FE definition and the optimization sequence.

According to *Nastran's* optimization manual, the model properties are the quantities that appear on the bulk data property entries such as thicknesses, area moments of inertia, elastic modulii or grid point locations, being the DVPREL1 entry in charge of providing a linear relation between the property and the design variable. It has been assumed that the user is only able to take as design variables the ones that are already set as properties by *Nastran*, the linear relation coefficient would be equal to one and the constant term of relation to zero. No bound limits have been defined for the design variables as they do not impact the sensibility computation.

Attending to the element properties design variables input dictionary, see Figure 2.3 for a complete example, the user is capable of establishing the particular design variables to consider for each one of the element blocks. This action has been implemented through the PNAME/FID entry, allowing the user to specify the property name, such as "T" for thickness, or the field position of the property entry defined in *Nastran's* quick reference guide. Not all the element properties have been defined yet as design variable, see the tool's repository information in order to check the available properties or to request the implementation of newer ones.

There has also been required from the user to input the initial value of the design variable, having this definition a deeper purpose as NOI could also be implemented in a recursive loop with an external optimizer. In that situation, the tool would provide the optimizer with the sensitivities, and the feedback coming from the optimizer would supply the design variables values for the next iteration. Here lies one of the main features of NOI, which allows to updates the DESVAR entries once the model has been initialized. However, it is also possible to call the tool in a non-nested way.

2.2.1.2. Lumped masses design variables definition

As it was previously introduced, NOI also recognizes the definition of punctual masses of the FE model, CONM2 entries, as design variables for the sensitivity analysis. From the implementation point of view, the process of setting-up the masses related design variables happens to be quite similar to the one described for the element properties, with the main difference that in this scenario it is not required to create additional lumped masses entries. This is due to the fact that it has been assumed that the CONM2 cards that have been provided in the FE model are immutable, being only possible to update their value.

The "*set_mass_dv*" method takes as input a *Python* dictionary that includes as keys the labels for the new design variables, and as values a tuple with the list of masses identifiers to consider, and its corresponding assigned value. From that point, the method adds the required DESVAR and DVCREL1 cards to the optimization model. Due to the fact that it is possible to group lumped masses to create new design variables, the method also accounts at the time of adding new design variables for the already existing ones, updates them in case the same masses have been selected, and removes the non-used cards. Lastly, the only remaining task has been redefining the value of the concerned CONM2 entries with the same one set in the design variable entry.

Again, it has been established that the connectivity relation between the lumped masses and the design variables is direct, using a null constant term of relation and being the linear coefficient equal to one. A more detailed example of the input dictionary and description of the method has been presented in Figure 2.3.

2.2.1.3. Design model responses

One of the most important steps in the optimization sequence module is including the responses linked to the design model. Certain degree of flexibility, in terms of user input, has been take into account. Nonetheless, and keeping in mind that the main focus of this work yields in the static and modal analysis for optimization, the possible responses have been restricted and presented in Table 2.2. Even though the mentioned consideration reduces in some sense the applicability range of the tool, solution sequences 101 and 103, for static and modal subcases, are currently ones of the most used in the *Nastran* structural optimization environment.

Subcase type	**Supported responses**
Modal	WEIGHT, EIGN, DISP
Static	WEIGHT, STRESS

Table 2.2: BDFOpt supported analysis types and responses

It has been presumed the weight of the structure to be always enforced as an active response, and it has been defined in the tool as the fixed objective function due to the fact that *Nastran* requires in a mandatory way the definition of an objective function. Thus, it was concluded that the weight would be rather cheap to compute once the FE model has been provided, in comparison with the rest of responses.

In relation to the remaining responses definition, and recalling Table 2.2, the user is able to consider several options depending on the analysis type. The responses are handled by means of the DRESP1 entries in the bulk data section, where it must be specified the label, type of response, and its own characteristics. The *Nastran* design sensitivity and optimization manual compiles all the possible responses accepted by the DRESP1 card. However, in NOI there have only been accepted the stress, eigenvalue, displacement and weight ones up to now.

For the static subcase, there only exists the possibility to define the stress as response. Nonetheless, NOI allows the stress constraint to be applied to multiple sets of elements. The tool is also capable of identifying the kind of stress that the user wants to consider by means of the information stored in the responses input dictionary and to write it into the DRESP1 response attribute field. This process involves the fact that each element has a set of associated stress item codes. For example, the Von Mises stress in a CQUAD4 element has a value of nine for its stress code, however, the only recognized element types for stress responses in BDFOpt are the CROD and CQUAD4, and in view of expanding this capability.

With respect to the modal analysis, the user can select as responses the eigenvalues or eigenvectors for a given mode. The eigenvector implementation had to be specified for each node of the FE model, being that the reason behind using the displacement response type. At the end, this feature allows retrieving the sensibilities for the three displacement components x, y, z per node and stated mode.

After the DRESP1 entries have been included, and in order to account for them in the sensitivity analysis, they must be gathered by a DCONSTR card so *Nastran* sets them as constraints in the optimization process. In reality, and because of the fact that the tool is only interested on computing the sensitivity, they are not strictly considered constraints of the optimization problem but *Nastran* requires this formality to provide the sensitivity. Besides, these responses stored in DCONSTR entries shall be grouped in two different DCONADD cards to make a distinction between the static and modal subcases responses.

All in all, the user is only responsible of properly setting-up the responses input dictionary, which contains as keys the label for the response, and its values correspond to a nested dictionary with the response type identifier as the new key. The response identifiers have been set to be "S" for stress, "EVAL" for eigenvalues and "EVEC" for eigenvectors. Furthermore, the nested dictionary must include, for the stress response the elements and the stress item code, and for the modal responses the mode number information. See Figure 2.3 for a proper example of the responses input dictionary and a detailed description of the "*set_responses*" method.

2.2.1.4. Executive control section and case control deck

The tool is able to properly set the executive control section, the case control deck, and the bulk section parameters in the BDF input file, being these actions performed by means of the "*set_case_control_deck*" method. These particular lines of code are the ones subjected of allowing the user to specify the number of statics and/or modal subcases to be performed and of specifying *Nastran's* solution sequence.

The case control deck, in case the reader in not familiarized with the *Nastran* environment, is the section of the input file devoted to compiling the subcase structure, certain control parameters, boundary conditions, load scenarios, and constraints of the model. On the other hand, and for the executive control, the optimization solution sequence 200 must be specified.

In relation to the subcases incorporation, it has been required from the user to define the ones it wants to perform through the analysis input dictionary. This dictionary compiles the subcase definition, so its keys are meant to be the subcase identifiers, "S" for static and "M" for modal. The static subcase recognizes an extra input list which corresponds to the load identifiers to be appended as additional static subcases in the subcase definition. Another important aspect about the subcases section is the scenario when a modal analysis is being specified by the user. There, a EIGRL entry must be included in the bulk data section, printing the maximum number of modes to evaluate and the eigenvector normalization method.

Once all the subcases have been appended to the model, the DESSUB entry has to be specified for each one of them, pointing to the previously mentioned DCONADD set of responses to be evaluated. Moreover, the DESOBJ card is also involved, being its task to designate the DRESP1 entry to be considered as the objective function, which for NOI always will be the weight of the structure. At the end, the main outputs of the optimization which are witten in the OP2 file are the sensitivity matrix, DSCM2, and the DSCMCOL table that comprises the coefficients identifiers of *Nastran's* sensitivity matrix.

Regarding the output parameters, which are defined at the beginning of *Nastran's* bulk data section with the PARAM card, there have been included the ones that write the sensitivity results in an external OP2 file, see Table 2.3. Besides, and as the main purpose of NOI yields in the sensitivity analysis of the model, *Nastran* allows to just stop there and not keep going with the full optimization process, so this has also been specified with the corresponding POST equal to -1 entry value.

Output parameter	**Value**
POSTEXT	Yes
OPTEXIT	-4
SPARSEDR	No

Table 2.3: BDFOpt output parameters

One of the most important remarks in relation to the sensitivity matrix coefficients is the screening process, which is defined as the operations used to identify those responses that are likely to drive the redesign process. This means that by default *Nastran* removes all the sensitivity coefficients that are not meaningful enough in the optimization. However, NOI is capable of clearing away these actions and to provide the user with the corresponding sensitivity coefficient using the DSCREEN entries.

2.2.2. Modal and flutter input files

The last two blocks that belong to the BDF4OPT class are related with allowing the user to output modal and flutter analysis files to be run by *Nastran*. The implementation of these blocks was included in NOI once the tool started to grow in complexity, and after the optimization part was already finished. In behalf of the fact that in the optimization sequence there is the possibility to perform modal analysis, and that one of the most popular constraints in the aero-structural optimization is the flutter speed, it was concluded that being able to check beforehand the response of the model under either a modal or flutter analysis, without having to set-up the complete optimization problem, was an appealing feature.

In the first place, and in relation to the modal analysis file, there has not been required any input dictionary coming from the user, as it was at the time of dealing with the optimization block. For this scenario, there is only demanded from the user to define the number of modes that *Nastran* has to account for.

To assemble the modal file, it is needed to recall the solution sequence 103 in the executive control section, and to define the proper subcases, which are already established in the FE model input file. The *Nastran* entry responsible to define the modal analysis parameters goes under the name of EIGRL, collecting the number of modes to consider and the normalization method for the eigenvectors, being the user able to select between normalizing respect the unit value of the generalized mass or the value of the largest eigenvector displacement.

On the other hand, the flutter analysis definition directly embeds from the modal block as the EIGRL card and number of modes are requested by the solution sequence 145. Additional parameters related with this particular analysis such as velocity, Mach, and density ranges, stored in FLFACT entries, can also be provided by the user. It is also important to set-up properly the reference magnitudes of the problem like the length, velocity and density. Altogether, the tool is able to output a properly formatted flutter analysis file, being its only limitation that it does not account for setting-up the CAERO and PAERO entries, which must be provided in the FE input file and that are related with the aeroelastic analysis.

2.3. OP2 reader module

The purpose of the output processing module is providing the user a tender way of handling the simulation output. The XOP2 class, by means of the *pyNastran* package, is capable of reading the OP2 files that *Nastran* provides after each run and to extend its potential. These results are usually stored in a raw format, and not sorted in any kind of way, that's why this module had to be developed. Figure 2.5 offers a general overview of the different classes and methods that have been discussed along this section.

As it was previously introduced, two branches can be found that directly inherit from the XOP2 class. The mentioned parent class is capable on its own of providing results related to the weight, its sensitivity, and the center of gravity of the considered model. In this way, the extracted results are some kind of generic and independent of the considered analysis scenarios, being this the main reason why these methods belong to the partner class. Then, two child classes have been sorted and named depending on the type of subcase for which they have been developed.

2.3.1. Static analysis results

The OP2Static class gathers the methods in charge of the static analysis results, being able to extract the stresses recorded for the finite model elements and also to output the sensitivity matrix in a general basis when this type of analysis has been performed. Up to now the main focus of the thesis, in relation to the handling of results, has been oriented towards the modal resolution, being this the reason why only two methods can be found in this class. The "*stress_output*" method provides the stresses in each element when an static analysis type has been performed. This particular method outputs a dictionary that has been sorted according to the element type, subcase, and element identifier.

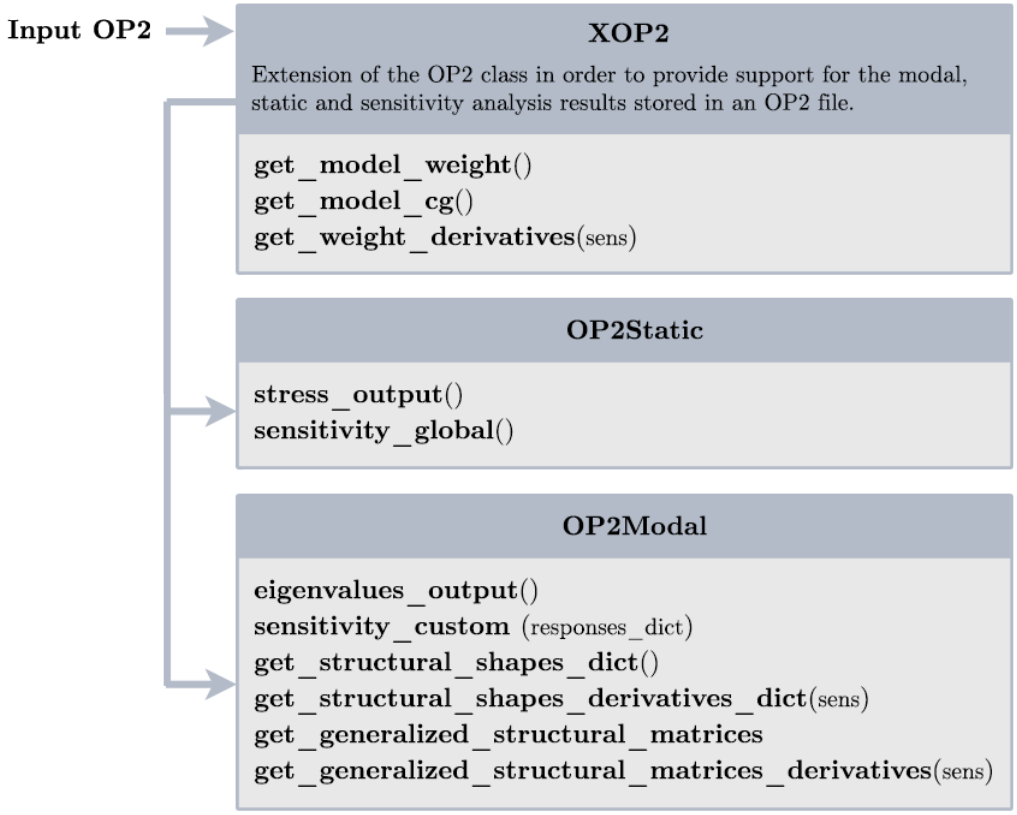


Figure 2.5: OP2 reader module breakdown

On the other hand, the global sensitivity matrix can be accessed throughout the "*sensitivity_global*" method, providing a dictionary that includes all the sensitivity coefficients sorted by their response type and *Nastran's* identifier. This first method for outputting the sensitivity covers the reading of any sensitivity matrix which contains whatever type of response, with the only restriction that it must be supported by the DSCMCOL table. However, its interpretation may be difficult as the output dictionary is sorted according to the response identifiers. That's why an additional method to handle the sensitivity matrix has been developed for the modal branch.

2.3.2. Modal analysis results

The OP2Modal class has been oriented towards building an interface for the sensitivity and modal analysis results computed by *Nastran*. This branch was born after attempting to include NOI in an optimization framework, and serves the purpose of aiding the handling and formatting of modal results. As it was presented along the introduction of the former thesis, the structural model, apart from its geometrical definition, is characterized by a set of matrices known as stiffness, mass and damping, which play a huge role with respect its aerodynamic behaviour.

In this way, the OP2Modal class is responsible of retrieving the modal and sensitivity results and building the corresponding matrices and their derivatives whenever possible. All the methods stored in the class are available for OP2 files that include results coming from solution sequences 103, 145 and 200 when a modal analysis has been executed. Nonetheless, not only the structural matrices are being considered, the eigenvectors can also be written in an output array sorted according to the FE model nodes. All in all, these would be the main considerations of the OP2Modal class, whose overview and I/O structure has been presented in Figure 2.6. A deeper discussion is going to be provided for each one of the methods

In a first instance, it is required to provide as inputs to the OP2Modal class the analysis results stored in an OP2 file, and the number of modes to take into consideration. The number of modes is going to define the modal shapes to be stored and also the dimensions of the structural matrices.

After the class has retrieved the corresponding input values, the first method named "*eigenvalues_output*" stores the eigenvalues up to the user defined number of modes. This method is then recalled at the time of setting-up the structural matrices through "*get_generalized_structural_matrices*", where once the values of the generalized stiffness, mass and damping have been obtained for each one of the evaluated modes, they are stashed in the corresponding position of the generalized matrices. It is important to note that due to the fact that the eigenvectors have been subjected to mass normalization, the resulting mass matrix mirrors the identity matrix. Furthermore, and as no structural damping has been considered, the damping matrix is a null. Lastly, the stiffness matrix is conformed by the eigenvalues of each one of the modes in their corresponding position of the diagonal matrix.

In relation to the eigenvector output, they have been stored in an array which its shape has been determined according to the number of nodes of the finite element model. This output is provided by the "*get_structural_shapes_dict*" method, storing in the numerical array up to the defined number of shapes sorted by the node number and coordinate. Nonetheless, and due to a *Nastran* constraint, it is not possible to output the eigenvectors when a solution sequence 145, flutter analysis, has been performed for the reason that these displacements are not stored in the input OP2.

The previously discussed methods do not involve the treatment of the sensitivity analysis results, meaning that they can be applied directly to solution sequences 103 and 145 in order to retrieve the mentioned matrices, or the modal shapes of the model. One of the most important features of the OP2Modal class has been enabled in the "*get_generalized_structural_matrices_derivatives*" method, which is capable of outputting the derivatives of the structural matrices when the sensitivity analysis results are stored in the input OP2 file. This method allows to account for analytical derivatives when an optimization problem is being set-up and NOI is used to cover the structural section of the optimization framework. Moreover, and in the scenario an aero-structural mapping process has to be performed involving the eigenvector derivatives, their values can be extracted by means of the "*get_structural_shapes_derivatives_dict*" method.

Figure 2.6: OP2Modal methods diagram

It is also worth mentioning, as the reader may notice, that the two methods related to the derivatives output contain as input a certain dictionary. Thus, the the main drawback of these methods is that there has been requested to input a dictionary containing the eigenvector responses definition due to the fact that *Nastran* does not provide information about the evaluated mode number in the DSCMCOL table for each one of the eigenvector entries. In this way, the tool is now able to match the displacement responses with their corresponding mode number.

The extraction of the sensitivity matrix results has been performed throughout the "*sensitivity_custom*" method, whose output gathers the sensitivity coefficients in a nested dictionary according to their response type, subcase, element/mode and design variable. Besides, for responses that include dependencies with element numbers or grid entries, such as the stress and eigenvectors ones, this kind of information has also been included. Keeping in mind that the tool may be incorporated in an optimization loop, this procedure eases the way of dealing with sensitivity coefficients in order to input them to an external optimizer, being this process recalled in several methods of the OP2Modal class at the time of dealing with sensitivity results.

2.4. F06 reader module

The reason behind the implementation of this module lies in the fact that the user may require reading results contained in a F06 file, being the OP2 reader not capable of assessing this matter. Although this particular module is still in development, some features have been included regarding the F06 flutter solutions as *Nastran* has not been able to provide an straight forward way of processing the solution sequence 145 results. Further work includes expanding the competences of this modules, as it may be easier for the user to work with F06 files despite the fact that from the computational point of view its performance is much lower than handling OP2 files.

Right now, and in relation to the flutter results, the user is able to display the V-g, V-f and root locus plots recalling the methods presented in Figure 2.7. Furthermore, this module can be combined with the BDFFlutter class from the BDF writer module, conforming a powerful blend when the user wants to set-up and perform a flutter analysis using *Nastran*. The unique requested input in order to display the results happens to be the raw F06 file that *Nastran* outputs once the flutter analysis has been completed. From that point, and via the *pyNastran* package, it is possible to access the flutter results such as damping, frequency, complex eigenvalues and velocity. Moreover, the user is able to specify the desired modes to be represented, with the only restriction that they must had been stored in the F06 file.

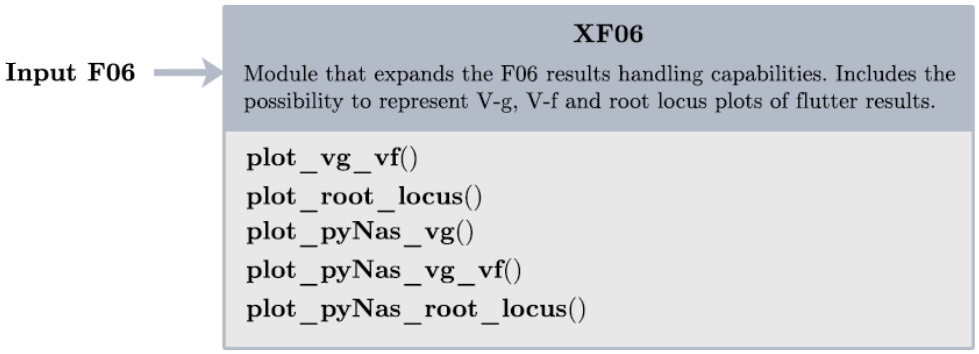


Figure 2.7: F06 reader module breakdown

The first two methods shown in Figure 2.7 have been own implemented and provide the V-g, V-f and root locus plots, being able to identifying the flutter speed and removing the non-valid points where the imaginary component of the eigenvalue goes to zero. Nonetheless, the output value of the flutter speed can show certain variability as it directly depends on the range of speeds considered by *Nastran* when setting-up the flutter analysis.

The coded method to retrieve the flutter point only evaluates the individual velocity points defined by *Nastran*, meaning that for example if the flutter speed falls within the 110 and 120 m/s range and the only considered points by *Nastran* are that two, the method is going to output 120 m/s as the flutter speed because it is the first evaluated point with a positive damping sign. This method is subjected to be improved in the near future in order to provide a more robust determination of the flutter speed, but up to now the obtained results are adequate. On the other hand, the remaining methods have been directly embedded from the *pyNastran* package, showing the same purpose but with the main difference that they are not able to identify the flutter speed or to select the modes the user wants to represent.

To verify the proper working of the module, a test has been conducted taking as input the FE model of the Goland+ wing benchmark problem that has been presented in Subsection 4.1.1, nonetheless, certain changes were performed. The first one has been transforming the units of the FE model from imperial to the I.S, and the second to increase the number of velocity points to evaluate. It should be remarked that all these actions have been made by means of the BDFFlutter class. Finally, and once the flutter BDF file has been produced, and *Nastran* has stored the results in the F06, it is possible to display the V-g and V-f plots for the first 4 modes of the Goland+ wing, see Subsection 4.1.1 for the complete discussion about the results and for the visualization of the plots.

3 FAEDO

This chapter of the thesis draws together the discussion concerning the framework developed to solve a structural optimization problem with the incorporation of an aeroelastic constraint. The framework has been entrenched in such a way that allows a great degree of flexibility at the time of setting-up the design optimization problem, allowing the possibility to define different types of objective functions and responses, being NOI the one in charge of the sensitivity and structural analysis. Once the structural results have been computed by *Nastran* and handled through NOI, a particular set of tools has been recalled in order to account for the aerodynamic and stability analysis of the model. This means that the framework is capable of fully performing the structural and aeroelastic analysis, however, OpenMDAO would be the one responsible for the optimization when nested with FAEDO.

Assessing the described framework, Figure 3.1 illustrates all the required steps to be followed. The process, which starts with the structural analysis and ends up providing the flutter speed and its derivatives, has been divided into different blocks that also encompass the aero-structural mapping and the aeroelastic analysis. It is important to take care of all the dependencies between the different blocks of the framework, being critical to guarantee the proper flow of information between them. Once all the information demanded by the optimizer has been gathered, the recursive loop can take place, updating the starting values of FAEDO for each iteration, always attending to the optimizer requests.

Regarding the optimizer, the framework has been coupled with OpenMDAO, Reference [7], which is an open-source high-performance computing platform written in *Python*. A noticeable aspect that conditioned the selection of the optimizer has been the possibility to account for analytic derivatives, which in the framework have been provided by NOI. OpenMDAO not only endorses this feature, as it is focused on supporting gradient-based optimization, but also provides assistance at the time of checking the values of the analytic derivatives, comparing them with a finite difference method in order to verify that they have been properly computed.

The following pages briefly discuss the implementation of FAEDO, drawing special attention to the structural analysis section of the tool in which NOI has been included, but also granting the reader a general description of all the involved processes. Moreover, it has been presented the way OpenMDAO has been incorporated into the framework, the details regarding the preferred optimization algorithm, and its working parameters.

3.1. FAEDO overview

According to the workflow presented in Figure 3.1, the work performed along the thesis has been focused on the structural analysis section and its embodiment into the FAEDO framework. In a first instance, it is worth mentioning that in order to account for the stability analysis of the model, and thus its aeroelastic response, it has been required to carry out a modal analysis that provides all the necessary information. NOI has been the platform responsible for writing the optimization problem BDF file, and once the modal analysis results have been provided by *Nastran*, to make them available to the rest of the framework. By means of its XOP2 class, NOI is capable of transferring the data regarding the generalized stiffness, mass and damping matrices, and also the eigenvectors and sensitivities for the evaluated number of modes and design variables.

After the structural part has been introduced, the aerodynamic related processes can take place. All the methods for this block of the framework have been supplied by Francesco Torrigiani, one of the supervisors of the thesis, in such a way that it was only needed to rearrange them to build the aerodynamic section of the framework. Although the implementation of FAEDO has not been one of the main objectives of the thesis, due to the contribution to its development, and for the sake of completeness, along this part of the project there has been included a brief discussion corresponding to each one of the involved blocks rather than just providing a description of the structural analysis section.

The main inputs of the FAEDO framework, when solving for the flutter speed and its sensitivities, correspond to the FE model to be considered, the set of *Python* dictionaries used in order to fully define the optimization problem, and the aerodynamic mesh of the evaluated geometry. Furthermore, additional parameters involved in the aerodynamic analysis can be also defined by the user inside the tool, but not directly provided on an external basis. In the end, it has only been required to provide the exact same inputs previously discussed in the NOI section, the aerodynamic mesh, and also the OpenMDAO related parameters.

3.2. Structural analysis

The set-up for the structural analysis block includes the FE model definition, which is directly associated with a certain structural grid and constitutes the main input for this section of the framework. From this point, NOI has been adopted in order to constitute the optimization problem design model, and with the purpose of handling the results coming from *Nastran's* sensitivity and modal analysis.

Although NOI allows a great degree of flexibility at the time of producing an optimization BDF file, and according to the target of the framework, it has been mandatory to carry out a modal analysis that would lie down the basis for the stability study capable of computing the aeroelastic response. This has been the main reason why the responses and analysis dictionaries required by NOI have been already defined inside FAEDO, instead of being provided by the user.

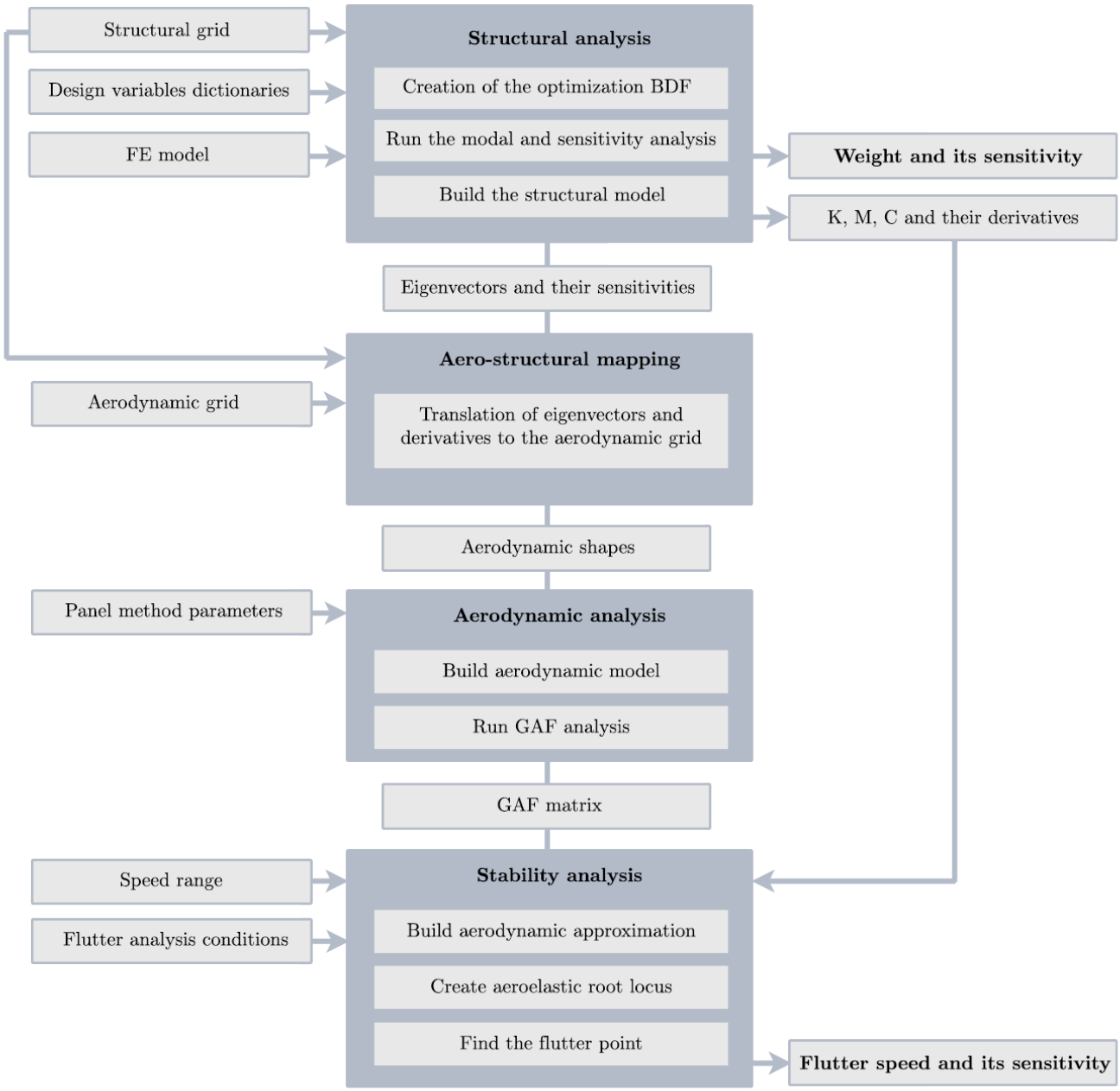


Figure 3.1: FAEDO flutter speed analysis diagram

This circumstance implies that there has only been required to specify the number of modes to evaluate, and then FAEDO takes care of writing the appropriate responses dictionary including the eigenvalues and eigenvectors up to the preferred number of shapes. Besides, and as it was brought in along the NOI discussion, the objective function has been set to be the weight of the structure, however, the design variables directly depend on the problem the user wants to solve, being this the reason why its corresponding dictionary is a mandatory input of the structural block.

Therefore, NOI allows the addition of all this type of information to the input FE model and provides the final optimization BDF file for *Nastran*. This file accounts for a sensitivity and modal analysis, including the user-selected number of modes and defining only as responses the eigenvalues and eigenvectors. Since the tool is going to be subjected to an optimization loop, the only entries that have to be updated when moving on along the iterations would be the values of the design variables dictionary.

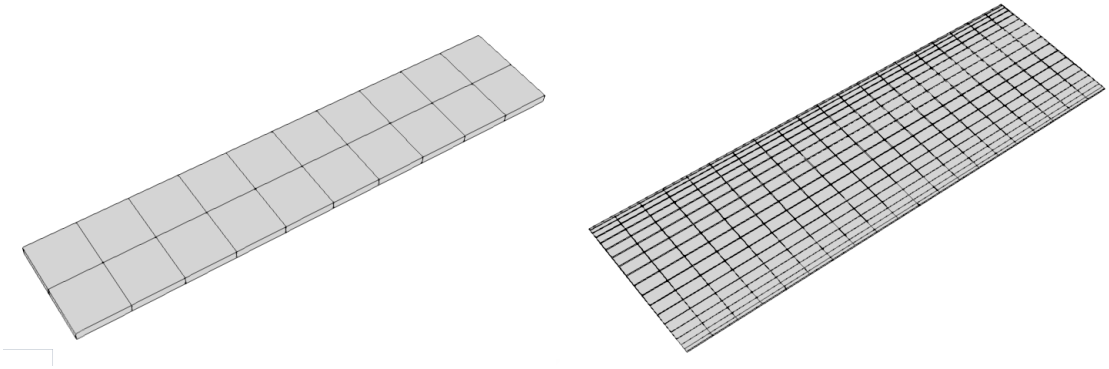
Once *Nastran* has provided the results for the sensitivity and modal analysis, and stored them in an OP2 file, NOI is able to extract the structural matrices, eigenvectors, and sensitivities. At this point in the thesis, it has been already justified the need for the OP2Modal class, which has been discussed along Section 2.3.2, as all the information for the aeroelastic portion of FAEDO can be provided in a straight forward basis. The mentioned class also outputs the generalized mass, stiffness, and damping matrices, which are further requested by the stability analysis. Moreover, there have been stored the eigenvectors, the objective function value and their corresponding derivatives in a set of dictionaries handled inside the framework.

In the end, the activities that take place inside this block can be summed up into the creation of an optimization BDF file according to the user-defined optimization problem, running *Nastran* in such a way that the modal results can be obtained, and finally handling these results which are being required by the aero-structural mapping and stability analysis sections of FAEDO.

3.3. Aero-structural mapping

The second block of the FAEDO workflow comprises the aero-structural mapping of the model. The required information in order to complete this process corresponds to the aerodynamic grid of the geometry, the eigenvectors that have been previously computed by the structural analysis section, and the structural grid that contains the FE model description. After the mentioned data has been retrieved by FAEDO, a set of methods supplied by Francesco Torrigiani perform a mapping of the structural modal displacements to the aerodynamic grid of the geometry. Due to the fact that NOI also provides the sensitivity information, it has been required to include the mapping of the modal shape derivatives due to the fact that they may be further requested in the aerodynamic and stability analysis blocks.

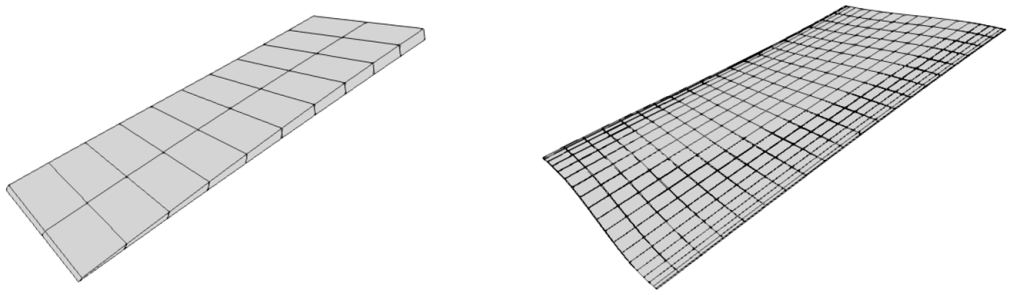
In place of completing the mapping between both grids, the Infinite Plate Spline (IPS) has been deployed. A more detailed discussion on the topic has been offered through References [8] and [9] in case the reader is not familiar with these techniques. Furthermore, and before proceeding with the aerodynamic analysis, the user is able to check the output of the aero-structural mapping through an external generated VTK file. This feature, and although it may seem trivial, holds great importance at the time of checking if the mapping has been successful. The straightforward visualization and comparison of both structural and aerodynamic shapes can determine if the information between grids has been properly transferred and also its accuracy degree.



(a) Structural grid (b) Aerodynamic grid

Figure 3.2: Structural and aerodynamic Goland+ grid comparison

Figure 3.2 shows the reader the differences that may arise between the employed structural and aerodynamic grids, and the relevance of carrying out the aero-structural mapping process in a proper manner. The mentioned figure depicts two different grids that represent the Goland+ wing benchmark model, see Section 4.1, where the displacements computed in the structural nodes must be translated to every single node of the aerodynamic grid by means of a least-square RBF method or a similar procedure. As it can be clearly seen, the number of nodes and locations may vary between the two grids, being this the main reason why an accurate mapping has been required to get accurate results. Moreover, this transformation happens to be more relevant when translating modal shapes between both grids, as seen in Figure 3.3. There, the second modal shape, torsion, of the Goland+ FE model has been represented in both grids.



(a) Structural grid (b) Aerodynamic grid

Figure 3.3: Structural and aerodynamic Goland+ grid torsion mode comparison

3.4. Aerodynamic analysis

After retrieving the modal shapes in the aerodynamic grid, which correspond to the eigenvectors and their derivatives, this section of the framework springs into action aiming at computing the aerodynamic analysis. For this purpose, it has been used an unsteady panel method that formerly goes under the name of GAF analysis in the framework. The Morino's method, see Reference [10], has been implemented and employed to perform steady and unsteady potential aerodynamic analyses, and the methods developed in References [11] and [12] have been directly incorporated into the aerodynamic section of the framework.

Following the workflow presented in Figure 3.1, this section of FAEDO gathers the information coming from the aero-structural mapping of the model. Additional parameters connected to the panel method set-up, such as the number of panels in the wing and in the wake, have also to be defined, as well as, the set of evaluated reduced frequencies or the reference length. Usually, the number of panels in the body and wake are decided according to a compromise between the accuracy of the results and the expense of the computation. All these actions related to the panel method set-up have been defined inside FAEDO with a set of default parameters that can be modified by the user depending on the characteristics of the selected geometry to be solved.

After everything has been properly defined, the aerodynamic analysis can take place providing as its main output the GAF database and its derivative. Both results are required by the finite state analysis, which is in charge of computing the flutter speed of the model. It is important to note that the derivative of the aerodynamic forces' database plays a huge role at the time of computing the sensitivity of the aeroelastic response that may be imposed in the optimization problem formulation.

3.5. Stability analysis

The final block of the framework corresponds to the the stability analysis, which includes the finite state approximation for the aerodynamic results. This first process has been performed following the guidelines provided by Reference [13]. Furthermore, the methods described in Reference [12] have been the ones employed in order to account for the flutter speed computation and its sensitivity.

As stated in the mentioned references, the finite state aerodynamic modelling allows for the use of a simple root locus method for the computation of the flutter point. In this way, the complex iterative procedure typical of p-k methods is avoided, and the accuracy of aeroelastic eigenvalues computation far from the flutter point is increased. Moreover, and according to Reference [12], by using the finite state finite state approach it is possible to define analytically the flutter speed derivative starting from the derivative of the structural and the aerodynamic terms. The derivatives of the structural mass, stiffness and damping matrices can be obtained analytically, whereas for the GAF matrix, a discrete adjoint procedure is set up.

That being said, the stability analysis can take place, taking as inputs the results provided by the finite state and from the structural analysis. Besides, it has been needed to account for the generalized stiffness, mass and damping matrices, the flight conditions and the speed range. FAEDO also grants the user the possibility to define the flight conditions inside the framework, being their default values the ones matching standard ISA sea level. In the end, the stability analysis is capable of providing the flutter speed and frequency by means of computing the root locus and its intersection points. Besides, it has also been implemented the computation of the flutter speed and frequency derivatives with respect to the design variables, in such a way that these sensitivities directly guide the optimizer at the time of selecting the new design variable values.

3.6. Connection with OpenMDAO

The preferred optimization platform with which the FAEDO framework has been coupled happens to be OpenMDAO. Several options may be found regarding optimizers, nonetheless, and after considering all the available alternatives, it has been concluded that OpenMDAO was the most attractive option, being one of its most remarkable aspects that it has been focused on supporting gradient-based optimization with analytic derivatives.

Furthermore, its ease of use and research background, regarding being used for several multidisciplinary optimization publications, have been also positive aspects to consider. It is also worth mentioning that although OpenMDAO has been the selected option, the framework could be perfectly coupled with any other external optimizer capable of supporting gradient-based optimization. From this point, the attention falls towards OpenMDAO, and how it is capable of solving coupled systems by means of Newton-type algorithms that ease the the assembly of the global derivative [7].

According to Reference [14], OpenMDAO was developed at NASA Glenn and uses the modular analysis and unified derivatives theory to allow for modular construction and execution of complicated models. Besides, several models can be arranged in a certain manner with any user-defined hierarchy of solver, and it has been employed to optimize a variety of problems, including composite fan blades [15] or even a boundary layer ingestion aircraft [16].

The interface required in order to bond OpenMDAO with FAEDO allows the user to take control of many optimization parameters such as the optimization algorithm, tolerance, maximum number of iterations, etc. For the purpose of FAEDO, it has been required to define the desired design variables, the objective function, and also the aeroelastic constraint. Then, it is possible to point out the location where the sensitivities are stored in such a way that OpenMDAO is capable of retrieving their values and account for the new updated ones. For a deeper explanation regarding how it internally works, and how to set it up, the reader may visit Reference [17].

4 | Cases of study

After the fundamental topics of the thesis have been discussed, now the focus lies in the cases of study performed over a certain wing structure that has been subjected to a flutter constraint. The objective of the presented chapter is verifying the proper working of NOI in an autonomous basis, and what is more important, its integration in an optimization framework serving the purpose of providing the required information regarding the structural analysis. In the evaluated test cases, FAEDO and OpenMDAO have also been set to assessment in order to check the correct flow of information between its components and the accurate computation of the aeroelastic response. Altogether, a design of experiments and several optimization scenarios have been conducted based on two different FE models of the well-known Goland wing configuration, see Reference [18].

The test cases have been grouped according to the employed geometry, where a detailed discussion of the models themselves has been presented in Subsection 4.1 of this chapter. Figure 4.1 offers a quick summary of all the solved cases of study. In the first instance, it is important to note that two FE models have been used due to the fact that it has been required to test different types of design variables. Attending to the cases of study that deal with the Goland+ model, their main feature would be that the design variables correspond to the set of lumped masses that compose the front and rear spars. Thus, a DOE in charge of verifying the accuracy of the sensitivity coefficients and an optimization process have been carried out.

Once the DOE grants the proper working of the framework, the first optimization on the Goland+ focuses on accomplishing a lighter structure while keeping its flutter speed within a given range. The design variables have been defined as the set of lumped masses that conform the front and rear spars of the model, so a more realistic approach has been followed instead of acting on each one of the single lumped masses that compose the wing.

Regarding the classic Goland wing scenarios, which take as input a FE model based on a beam approximation, just a single optimization study has been performed. The main difference between the Goland+ cases and this one can be found in the design variable definition, as their objective function and aeroelastic constraint are exactly the same. The cross-sectional area of the beam has been set as the design variable, exploring the capabilities of NOI when dealing with finite element properties design variables.

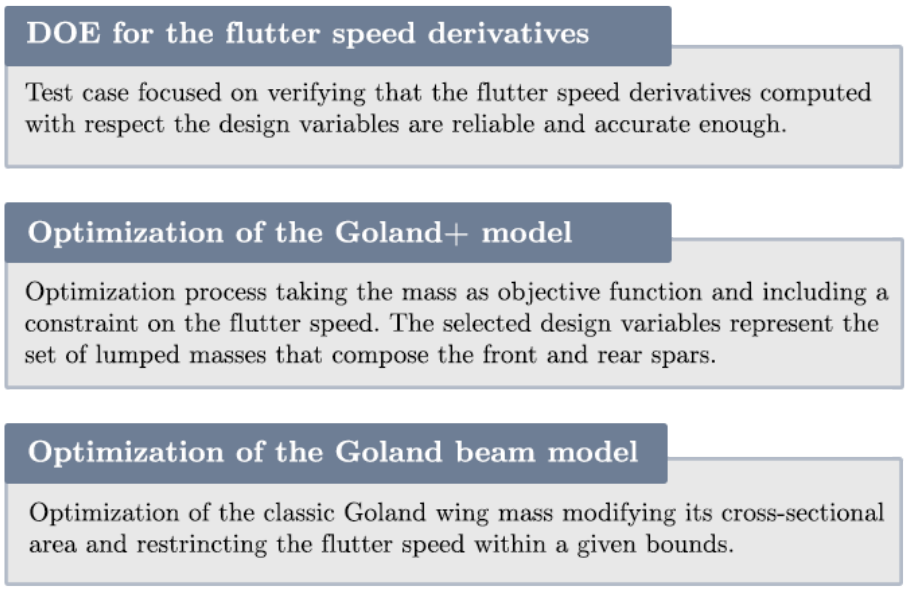


Figure 4.1: Summary of the conducted cases of study

4.1. Finite element models

The Goland wing has been considered a reference model mainly used to perform analysis and validations of aeroelastic methods in the aeronautical community. In the presented cases of study, and in order to validate the proper working of the developed tool, two finite element representations of the Goland wing have been recalled to perform the required computations. Taking as starting point the geometry described in Reference [18], the FE models have been evaluated by means of *Nastran*. Before assessing the employed finite element models, a brief introduction about the Goland wing shall be provided to the reader in order to understand the relevance of this model and its evolution over the years.

The original Goland wing was firstly introduced in 1945 by Martin Goland [18], who aimed to study the flutter speed of a uniform cantilever wing by the integration of the differential equations for the wing motion. Since its first appearance in 1945, many research articles have been conducted using the classic Goland wing as a way of verifying aeroelastic results. Nonetheless, in 1980 the first modification of the model appeared, going under the name of "Heavy" Goland wing.

Due to the analytical difficulties to predict the flutter stability when in transonic range, F.E. Eastep and J.J Olsen [19] recalled the finite-difference relaxation method to determine the oscillatory transonic aerodynamic forces on a uniform stiff cantilever rectangular wing. The model resembled the classic Goland wing, keeping almost the same elastic properties and introducing changes regarding a heavier mass distribution in order to fulfill a certain flutter performance, see Table 4.1 for the main differences between both wing models.

Parameter	Goland	"Heavy" Goland
c	1.828 m	1.828 m
l	6.096 m	6.096 m
m	35.7 kg/m	534.7 kg/m
I	8.642 kg/m	129.5 kg/m
S	6.514 $kg\,m/m$	97.71 $kg\,m/m$
EI_b	$9.786{\cdot}10^6$ N m^2	$9.786{\cdot}10^6$ N m^2
GJ	$0.989{\cdot}10^6$ N m^2	$0.989{\cdot}10^6$ N m^2

Table 4.1: Comparison between the classic and "Heavy" Goland wing

Taking as a starting point the "Heavy" Goland wing, a FE model was developed in Reference [20], modeling the wing according to a box structure in order to enable the option of attaching a given store. The authors intended to study the store-induced limit-cycle oscillation of a rectangular wing with tip store in transonic flow, and this FE model of the "Heavy" Goland wing is nowadays known as the Goland+, being the only difference the wing box structure. A full description of the wing characteristics and of the finite element model was presented in [20], being this the basis from which the FE model for the lumped masses optimization has been developed. However, and at the time of evaluating element properties design variables, a beam model of the traditional Goland wing has been recalled, see Reference [21].

4.1.1. Goland+ FE model

One of the main characteristics of the Goland+ model is that the mass distribution of the wing has been modeled by means of lumped masses instead of considering the material density, so this situation offers the possibility to test how NOI behaves under this particular type of design variables. As the objective of the test case would be the structural optimization of the wing under a flutter constraint, it seems convenient to study how the placement of lumped masses in the structure may impact its aeroelastic behaviour.

The main reason why the Goland+ model decided to use lumped masses was in order to obtain a certain aeroelastic response. Avoiding the weight contribution of the elements allowed the authors to use specified thicknesses that provided a given elastic response of the model that would not be possible to accomplish without placing lumped masses. In the end, properties such as the elastic axis and the center of gravity locations where two of the main contributors to the flutter response that were governed independently by means of lumped masses.

According to the FE model, see Figure 4.2 for its schematic representation, there has been depicted a cantilevered wing with a span corresponding to 6.096 m and a 1.219 m chord. As it was previously mentioned, the FE model represents a simplified wing box through a combination of around 230 elements. The skin has been modelled by means of membrane elements, and the ribs and spars with shear panels.

It is also worth to mention that the geometry includes rod elements posts which are in charge of connecting the skin with the matching spar or rib at each intersection. Table 4.2 summarizes the *Nastran* element type and their dimensions, thicknesses for the CQUAD4 and cross-sectional areas for the CROD elements, for each one of the mentioned components.

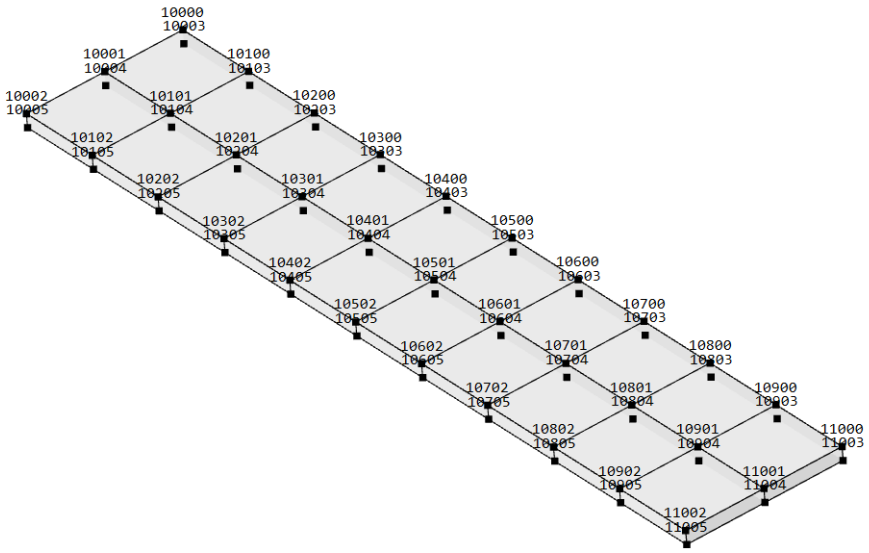


Figure 4.2: Schematics of the Goland+ FE model

Component	Element type	Number of elements	Dimensions
Wing skin	CQUAD4	40	0.004724 m
Front spar web	CQUAD4	10	0.000183 m
Front spar cap	CROD	20	0.0038600 m^2
Rear spar web	CQUAD4	10	0.000183 m
Rear spar cap	CROD	20	0.003860 m^2
Center spar web	CQUAD4	10	0.027096 m
Center spar cap	CROD	20	0.01386 m^2
Rib spar web	CQUAD4	22	0.010576 m
Rib spar cap	CROD	44	0.003920 m^2
Posts	CROD	33	0.000074 m^2

Table 4.2: Goland+ FE model description

The most remarkable feature of the Goland+ FE model is the lumped masses, whose locations have been presented in Figure 4.3. In reality, happens many times that the theoretical response of the wing does not match with the experimental one, being the weight distribution and placement of punctual masses one of the easiest things to modify rather than redesigning the wing. Nonetheless, the value of these punctual masses was determined following two criteria according to Reference [20].

Firstly, the value of the mass entry depends directly on its chord-wise position, meaning that different values can be found depending on if the mass entry belongs to the front, central or rear spar. Moreover, the value also finds influence by the span-wise coordinate, meaning that entries in the root and tip of the wing locations share the same value, which differs from the others in the rest of the spar. These circumstances have been clarified with Table 4.3, where each lumped mass entry has been sorted by its FE model identifier and value.

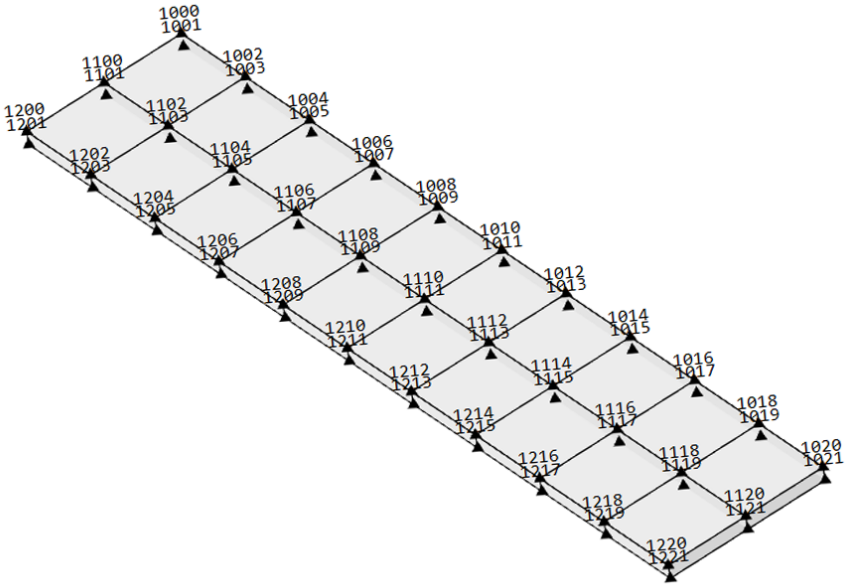


Figure 4.3: Goland+ lumped masses

FE model identifier	**Value**
1000, 1001, 1020, 1021	14.338 kg
1002 ... 1019	28.677 kg
1100, 1101, 1120, 1121	28.780 kg
1102 ... 1119	57.561 kg
1200, 1201, 1220, 1221	38.964 kg
1202 ... 1219	77.928 kg

Table 4.3: Goland+ FE lumped masses locations

With respect to the material properties, all the elements of the model have been defined with a Young's modulus of $71.7 \cdot 10^9$ MPa, a shear modulus of $26.9 \cdot 10^9$ MPa and a structural density of 10^{-5} kg/m^3. It is important to note that due to the fact that the Goland+ benchmark model was oriented to attain a certain aeroelastic behaviour, the material density was set to almost zero due to the lumped masses approach. According to the Goland+ Young's and shear modulus, it has been presumed that the selected material resembles the characteristics of an aluminum alloy, whose characteristics have been compiled in Table 4.4.

Property	**Goland+**	**A7075-T6**
Young Modulus	71.7 GPa	71.7 GPa
Shear Modulus	26.9 GPa	26.9 GPa
Poisson ratio	0.33	0.33
Density	10^{-5} kg/m^3	2810 kg/m^3

Table 4.4: Goland+ model material comparison

Once the model has been set-up, a modal and flutter *Nastran* analysis have been conducted in order to validate the response of the wing. According to the modal solutions, the first eigenvalue shows a frequency of 1.98 Hz while the second one yields a frequency of 4.05 Hz. It is also worth mentioning that the modal results have been compared against the ones provided in Reference [20], showing a good agreement in terms of eigenvalues and eigenvectors.

With respect the flutter solutions, which have been evaluated by means of the corresponding NOI module presented in Section 2.4, in Figure 4.4 there can be spotted a frequency coalescence of the two first modes, being the second one responsible for producing the flutter of the structure due to the positive damping shown around 120 m/s. Attending to the root locus representation displayed in Figure 4.5, it is even easier to spot that the second mode is the one causing flutter as the root locus branch that represents this particular mode transitions from the positive to the negative plane. Besides, an exact value of 119.09 m/s for the flutter speed has been obtained.

4.1.2. Classic Goland beam FE model

The classic Goland beam finite element model, which was firstly presented in Reference [21], keeps the same geometric features of the classic wing shown in Table 4.1, being the main difference with the Goland+ model that now the structure has been represented by means of CBEAM elements and RBE2 rigid body entities. This type of elements allows defining the distance between the elastic and the inertial axial, which responsible of the coupling between bending and torsional modes. The aeroelastic response of the described FE model happens to be the same as the classic Goland, being this the reason why this beam approximation has been used for one of the studies performed along the thesis. In relation to the mentioned study, because of this FE model it is possible to consider element properties design variables, such as the cross-sectional areas, neutral axis location, or even the moments of inertia.

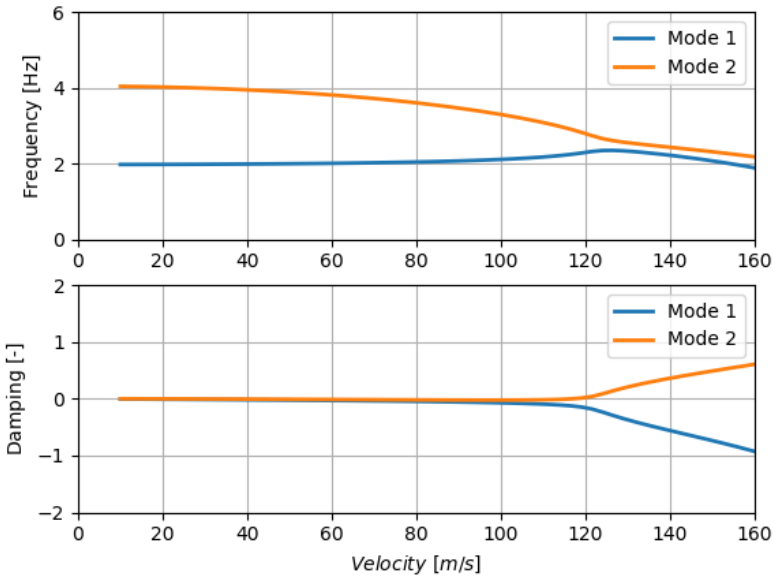


Figure 4.4: V-g and V-f plots for the Goland+ FE model

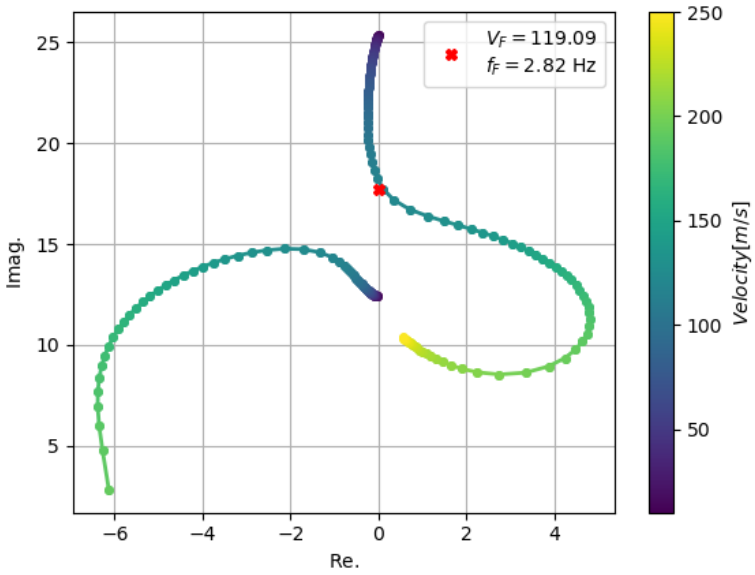


Figure 4.5: Root locus plot for the Goland+ FE model

Figure 4.6 represents a first overview of the beam, where its CBEAM elements have been highlighted. The aerodynamic lifting surface has been aligned with the freestream and modelled as a flat plate divided into an appropriate number of CAERO panels placed along the wing span and chord. The RB2 elements have been designed to match the leading and trailing edge of the aerodynamic surface, providing a natural support for splining. On the other hand, the employed material happens to be very similar to the one used in the Goland+ wing, clearly being an aluminium alloy whose characteristics have been laid out in Table 4.6. A more detailed description of the model can be found in Reference [21].

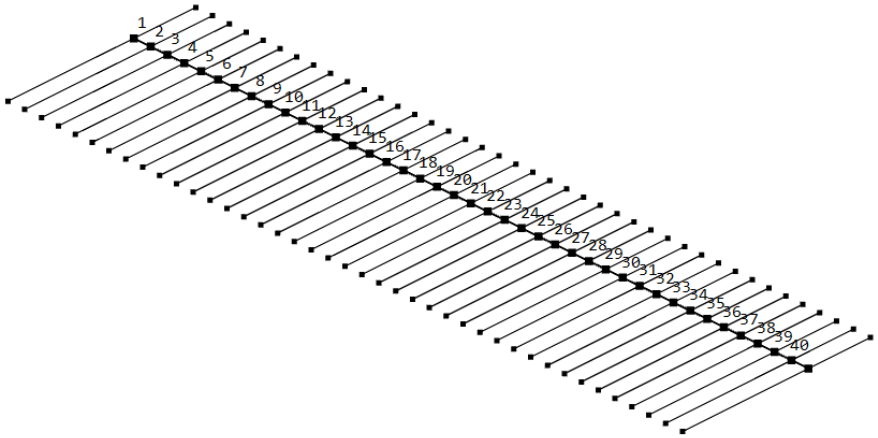


Figure 4.6: Schematics of the classic Goland beam model

Property	Value
A	0.01323 m^2
I_1	0.00262 m^4
I_2	0.0001396 m^4
J	$3.809 \cdot 10^{-5}$ m^4

Table 4.5: Model properties

Property	Value
Young Modulus	71.02 GPa
Shear Modulus	25.9 GPa
Poisson ratio	0.35
Density	2700 kg/m^3

Table 4.6: Material properties

From this point, it has been presented the aeroelastic response of the Goland beam model by means of a *Nastran* modal and flutter analysis. Once the modal response of the structure has been computed, the first eigenvalue shows a frequency of 7.66 Hz, while the second one falls towards 15.22 Hz. These values, all together with the obtained eigenvectors, show a relatively low deviation with respect the expected response of the classic Goland wing discussed in [18], validating the modal response of the beam model. Then, and in relation to the flutter solutions, the V-g and root locus representations have been computed recalling the corresponding NOI module and presented in Figure 4.7 and Figure 4.8 respectively. Again, it is easy to notice the coupling between the bending and torsion mode at a speed of 154.8 m/s with a frequency of around 11.02 Hz, which are similar to the results shown in [12].

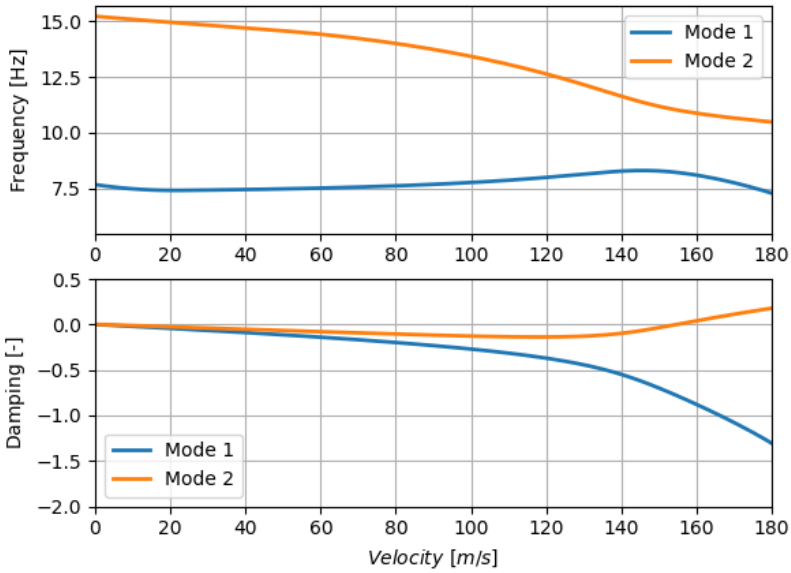


Figure 4.7: V-g and V-f plots for the Goland beam model

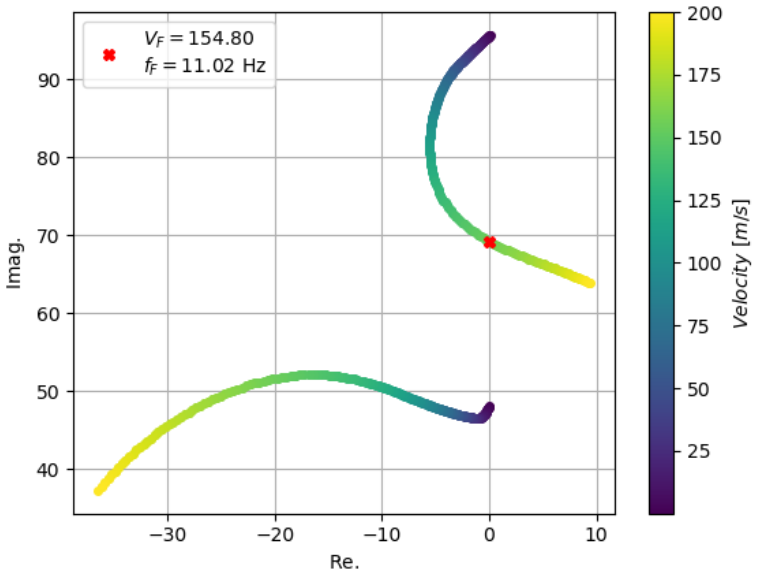


Figure 4.8: Root locus plot for the Goland beam model

4.2. DOE for the flutter speed derivatives

This section brings together the design of experiments performed with the objective of verifying that the flutter speed derivatives computed by the optimization framework are reliable and accurate enough. In order to do so, the analytical derivatives of a proposed optimization scenario have been subjected to comparison with the ones obtained by means of the finite differences method. Being aware that the sensitivity of the flutter speed ends up coming from a combination of several procedures, which all of them have discussed along Section 3.1, this situation justifies the need of proving that the sensitivities show the proper values due to their huge influence in the optimization process, as they drive the direction to be followed by the optimizer.

4.2.1. Methodology

The evaluated DOE takes as its starting point the Goland+ wing geometry and defines a baseline optimization problem with two design variables, which respond to the groups of front and rear spar masses. Table 4.7 clarifies the design variables definition, however, it should be noted that in this scenario all the lumped masses that share the same chord-wise component have been subjected to an identical value.

Design variable	**Set of masses identifiers**
FSM	1000 ... 1021
RSM	1200 ... 1221

Table 4.7: DOE design variables definition

The remaining aspects of the optimization problem happen to be exactly the same, being the mass of the structure the objective function and the flutter speed its aeroelastic constraint. Nonetheless, it is important to clarify that the information provided by FAEDO during the DOE is not going to be handed over by OpenMDAO. Instead of an optimization loop, there has been implemented a recursive process that intends to map a user-defined domain composed of multiple design variable combinations. These combinations correspond to all the possible pairs of design variables when their respective masses have been allowed to range between 1 and 100 kilograms, being evaluated twenty values for each design variable. In this way, the process ends up conforming a bi-dimensional design of experiments where the main output of the framework corresponds to a map of the flutter speed, and its derivatives, for around 400 pairs of values of the design variables.

The actual analytical derivatives computed at each point of the map have been compared with the ones calculated with the FE method. The followed process to obtain the finite differences derivatives has been relatively easy to implement. For each combination of design variables, this means for each point in the flutter speed map, two finite differences scenarios arise. Each one of them is solved setting fixed one of the design variables and introducing a small perturbation of value 10^{-3} in the other one, meaning forward differences.

Then, FAEDO recomputes the flutter speed and it is possible to figure out the sensitivity with respect to the perturbed design variable according to Expression 4.1. After repeating the described process for all the design variable combinations, there is also possible to represent a map of the finite differences flutter speed derivatives for each design variable, allowing a direct comparison of the sensitivity coefficients through the two mentioned methods.

$$\frac{\partial f(x_1, x_2)}{\partial x_2} = \lim_{\Delta x_2 \to 0} \frac{f(x_1, x_2 + \Delta x_2) - f(x_1, x_2)}{\Delta x_2} \tag{4.1}$$

4.2.2. Results

The conclusions coming from the design of experiments have been presented in the following lines. In order to assess them, it has been determined that using contour plots that represent the mapping of the studied magnitudes was the most understandable way. Firstly, the results for the flutter speed mapping have been displayed in Figure 4.9. There, the reader is able to spot how the flutter speed of the Goland+ model changes according to the different combinations of design variables. The x axis of the plot represents the value of each one of the masses located in the front spar, while the y axis is in charge of the masses that compose the rear spar.

One of the first phenomenons that the reader may notice are the void regions in the contour plot, meaning that the FAEDO framework has not been able to provide a valid flutter speed for that combination of design variables in the considered range of evaluated flutter speeds. These situations, analyzed from the aeroelastic point of view, can be explained due to two main reasons. The first one would be that, and due to the fact that acting in the front and rear spar masses directly impacts the center of gravity position, the elastic axis location is now rearwards with respect to the center of gravity of the wing. This can be checked plotting the center of gravity position, as shown in Figure 4.10. There, and for the scenarios where there is a greater increase of the front spar mass compared with the rear spar one, the center of gravity position moves forward and overtakes the 33% chord position of the elastic axis, meaning that no flutter can be found in the predefined velocity range.

So, this situation directly answers the lower right boundary of null flutter speed values shown in Figure 4.9. On the other hand, the upper right boundary of null results may hold similar reasoning, but now the center of gravity positions moves towards the rear spar, making the wing flutter very early and providing wrong reading in the FAEDO flutter speed computation.

It is also interesting to discuss how the flutter speed directly reacts when modifying the design variables, and this can be done in a much easier way attending to its derivatives. Figure 4.11 shows the analytical values of the flutter speed respect both design variables. In the left image, which illustrates the sensitivity with respect to the front spar masses, two regions can be spotted. The fact that there have been found two regions means that depending on the combination of design variables, the behavior of the flutter speed respect the front spar masses would be one or another.

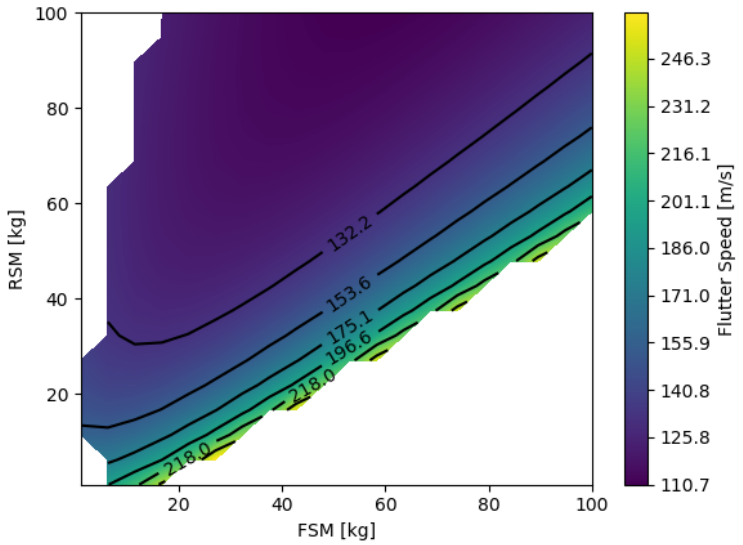


Figure 4.9: Flutter speed mapping of the DOE

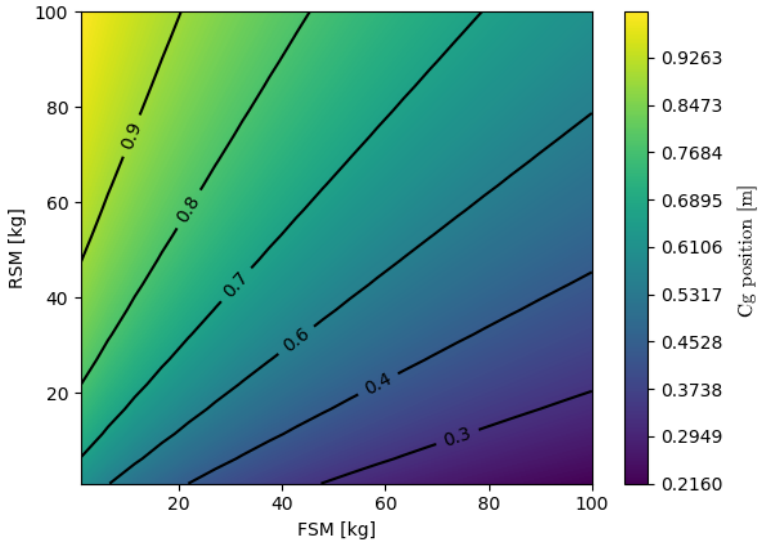
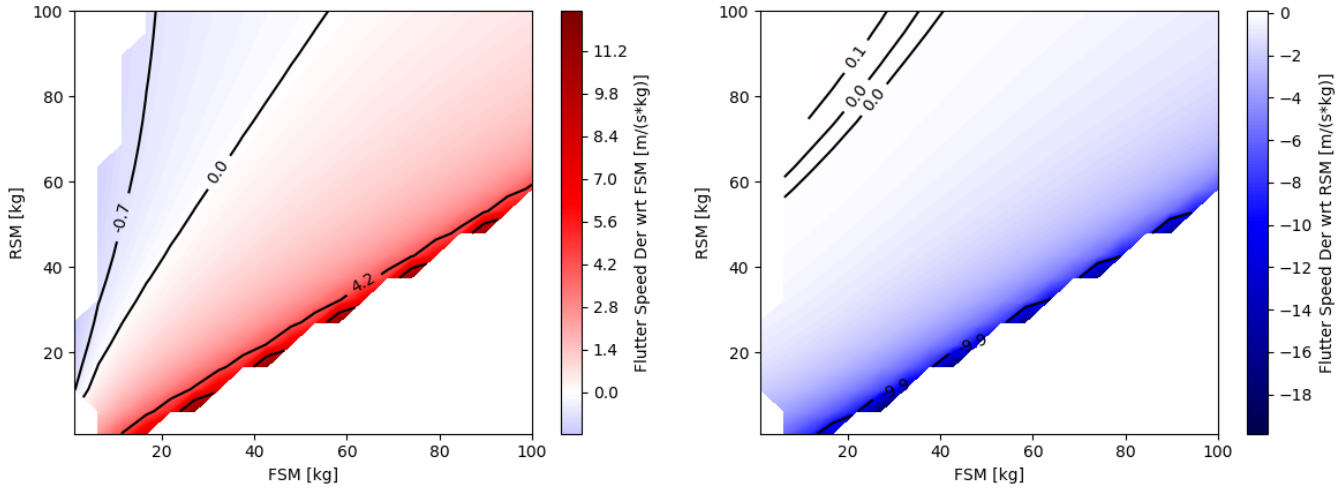


Figure 4.10: Centre of gravity mapping of the DOE

The red regions hold a positive sensitivity coefficient, meaning that increasing the design variable directly rises the flutter speed, while in the blue regions the effect is just the opposite. An increase in the front spar masses supposes that the flutter speed decreases. However, and now paying attention to the right side image, the trend followed by the sensitivity of the rear spar masses only shows one possible region in which increasing the rear spar masses ends up reducing the flutter speed.



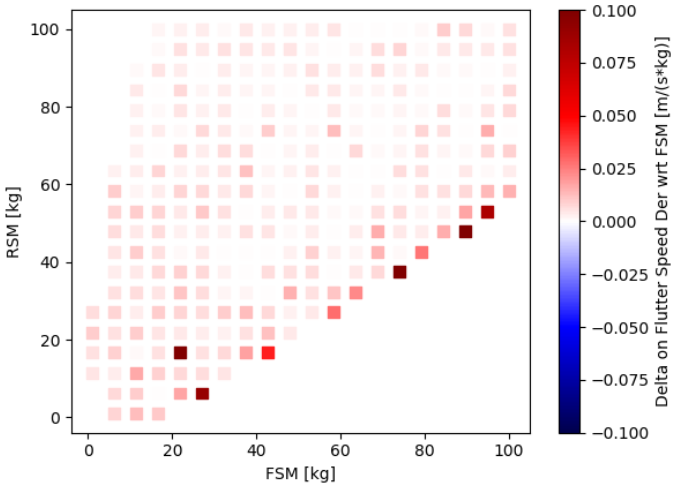
(a) FSM (b) RSM

Figure 4.11: Analytical derivatives computed in the DOE

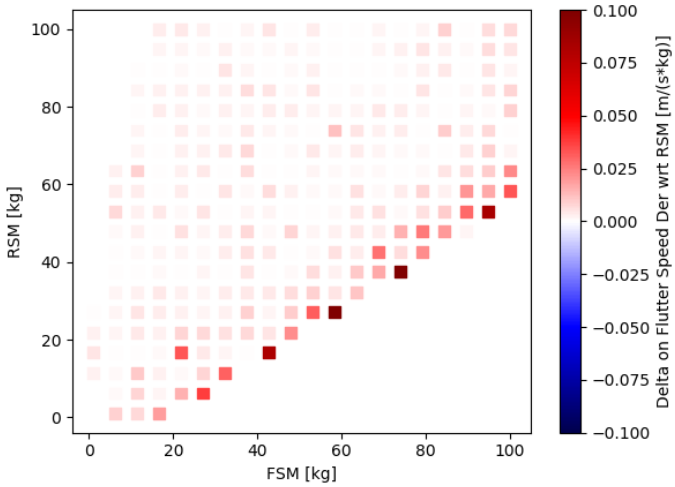
Being aware that the overall objective of the design of experiments has been verifying the values of the analytical derivatives, now there has been presented a comparison between the values of the analytical and finite differences sensitivities according to the previously introduced methodology. This study has been performed in a two stages fashion. The first one, see Figure 4.12, portraits the difference in the value of the sensitivities provided by each one of the methods. For both design variables, there can be seen how the difference is fairly low, showing a greater error between sensitivities located around the regions close to the boundaries, where discrepancies happen to be greater due to the flutter speed computation method limitations.

On the other hand, Figure 4.13, depicts the same comparison but now displaying the error in percentage between the analytical and finite differences sensitivities. The percentage error in both maps happens to be quite low, except in the areas where the value of the sensitivities tends to zero. In those regions, and due to numerical errors, the percent error shows a sudden increase. Overall, and after analyzing both figures for the sensitivities of the front and rear spar masses design variables, it can be confirmed that the FAEDO framework has been accurately computing the derivatives of the flutter speed.

The conclusions achieved along this test case hold a significant meaning because enable the use of NOI and FAEDO for solving optimization problems that include a flutter constraint. From this point, and further in the thesis, several optimization scenarios have been solved with the confidence that the computed sensitivities guide the optimizer in the proper direction at the time of computing the minimum of the considered objective function and fulfilling the imposed constraints.

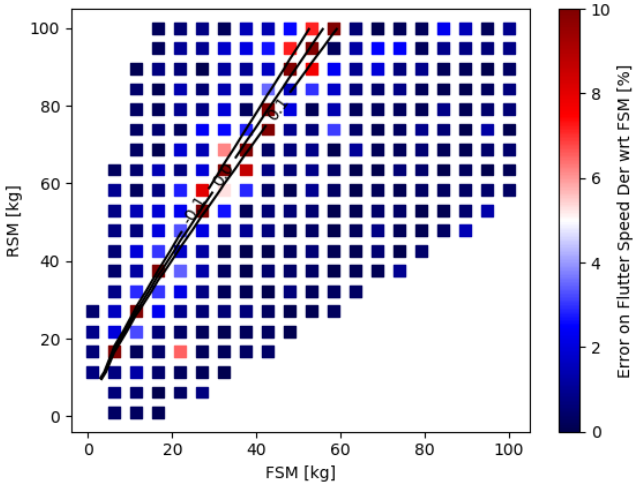


(a) FSM

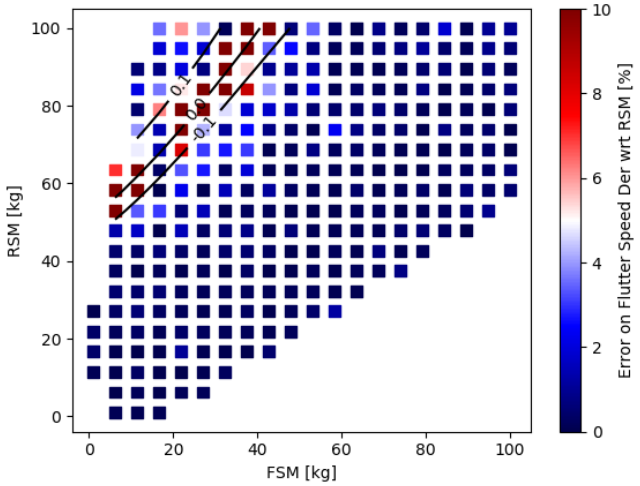


(b) RSM

Figure 4.12: Difference between analytical and FD derivatives



(a) FSM



(b) RSM

Figure 4.13: % error between analytical and FD derivatives

4.3. Optimization of the Goland+ model

As a matter of verifying that NOI is capable of properly working with lumped masses, it has been included in this section of the thesis the study of an optimization process with this kind of design variables. From an aircraft design perspective, and in order to obtain meaningful results, the static aeroelastic constraints should be added, but this is not the purpose of this study. For this particular scenario, the selected design variables correspond to the set of lumped masses that compose the front and rear spar. In this way, it has been appealing to determine how the mass distribution, and the lumped masses placement, impact the flutter speed of the model at the time of minimizing the objective function.

4.3.1. Problem definition and set-up

In a first instance, and attending to the theoretical formulation, the design optimization problem has to be defined. Expression 4.2 represents the problem constitution, where for the case of study the objective function corresponds to the mass of the structure and the aeroelastic constraint has been presented in such a way that the flutter speed has to fall inside a certain range. As it has been previously introduced, the Goland+ FE model makes a distinction on the lumped masses of each one of the spars depending on their spanwise coordinate. Taking this into account, it has been decided that only two design variables are going to be subjected to study, being each one of them composed by a set of lumped masses. For the two groups of design variables that portrait each one of the spars, only the masses that do not belong to the tip and root positions have been selected, see Table 4.8 for further details on the design variable definition.

$$\begin{aligned} \text{Minimize} \quad & m(x) \\ \text{w.r.t} \quad & x \in \mathbb{R}^{x_n} \\ \text{Subject to} \quad & V_{f\,min} \leq V_f(x) \leq V_{f\,max} \end{aligned} \tag{4.2}$$

Keeping in mind that the main goal of the optimization has been the mass minimization of the structure, the objective function defined within NOI and FAEDO happens to be the mass of the FE model. The aeroelastic constraint to be considered acts directly on the Golands's wing flutter speed, which must fall within the range of 100 and 140 m/s. The bound values of the flutter speed range have been selected in such a way that it is not desirable to get early flutter speeds as they compromise the integrity of the structure, however, relatively higher flutter speeds can also penalize the response and characteristics of the final wing, for example, in terms of weight or materials. In order to account for the flutter speed computation, the FAEDO framework has been employed and then OpenMDAO carries out the optimization.

Furthermore, and due to the nature of the considered FE model, there have been only evaluated the first two structural modes, as they have been found to be enough in order to study the aeroelastic behaviour of the Goland+ wing. After setting-up the design optimization model, now it is mandatory to properly arrange the aerodynamic section of the FAEDO framework.

Design variable	FE identifier	Initial value	Lower bound	Upper bound
FSM	1002 ... 1019	28.677 kg	1 kg	100 kg
RSM	1202 ... 1219	77.928 kg	1 kg	100 kg

Table 4.8: Goland+ model optimization design variables

Attending to the FAEDO framework diagram, see Figure 3.1, and in place of accounting for the aero-structural mapping and the aerodynamic analysis, it has been required to provide an aerodynamic mesh of the geometry to be solved. The preferred one has been a Goland wing grid that employs a NACA 0001 airfoil, which has been arranged by a total of 840 nodes and 740 cells. Another remarkable aspect to set-up prior solving the aerodynamic analysis corresponds to the flight conditions, being selected ISA and sea level for this specific optimization scenario.

In addition, it is also important to properly establish the optimizer parameters that OpenMDAO allows acting upon, which have been presented in Table 4.9. The adopted method has been the SLSQP, being a sequential least squares programming algorithm that uses the Han–Powell quasi–Newton method. The maximum number of iterations and the tolerance for termination have also been defined according to the optimization requirements and in order to avoid excessive waste of resources or wrongful results. Scaling of 0.1 has been introduced in the objective function as a matter of easing the method and to avert the nervous behaviour of the optimizer.

Opt. method	Max. iterations	Tolerance	Obj. function scaling
SLSQP	50	10^{-5}	0.1

Table 4.9: Lumped masses optimization parameters

4.3.2. Results

The final results of the optimization have been shown in Table 4.10, however, it is interesting to represent the variation of the objective, design variables, and constraint values along the optimization so it is possible to understand the reasoning behind the results. An important remark is that the process ends with a total number of 50 iterations and 409 function evaluations, and although the optimizer reached the maximum number of iterations, it can be considered that the process converged as the difference between the proposed design variables and the values of objective, and constraint, are within adequate margins.

For the former analysis of results, it has been assumed that the function evaluations play the same role as the optimization iterations although it is well-known their difference. Phenomenons such as gradient computations increase the number of function evaluations, being this one of the reasons behind the difference between these two magnitudes. In the further depicted plots, the values of the functions of interest have been presented versus the function evaluations but labeled as iteration number for the previously discussed reasons.

In a first instance, and paying attention to the objective function evolution of the Goland+ wing, see Figure 4.14, it is noticeable the substantial drop in its overall value at the early stages of the optimization process. After that response, the change in the mass of the structure stabilizes all along with the remaining iterations until further convergence to the minimum. Besides, it is clearly noticeable how a constraint violation has been encountered, in which the flutter speed upper limit has not been fulfilled and the optimization direction must be reconsidered. The small peaks in the objective function, which can also be spotted in the design variables and constraint plots, correspond to the previously mentioned spurious behaviour of the optimization algorithm but do not compromise the validity of the results. The Goland+ wing mass final value, which keeps almost constant after 40 function evaluations, corresponds to a reduction of around the 44 % of the starting mass of the model.

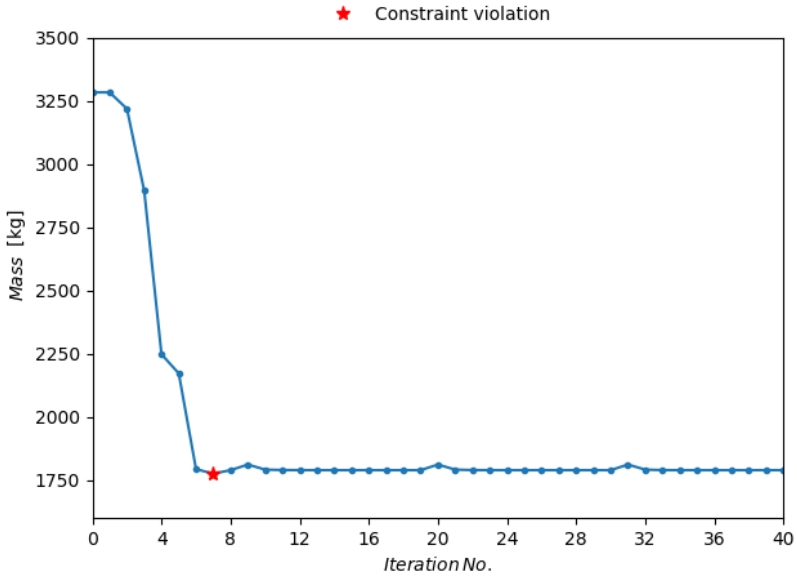
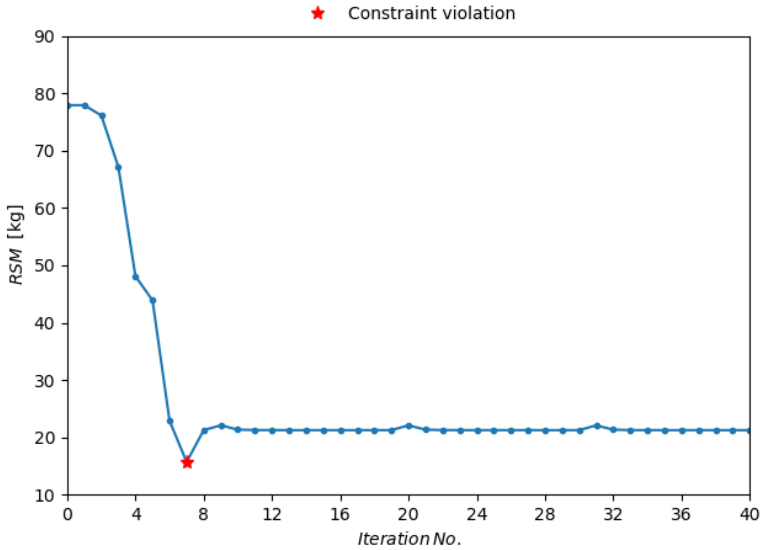


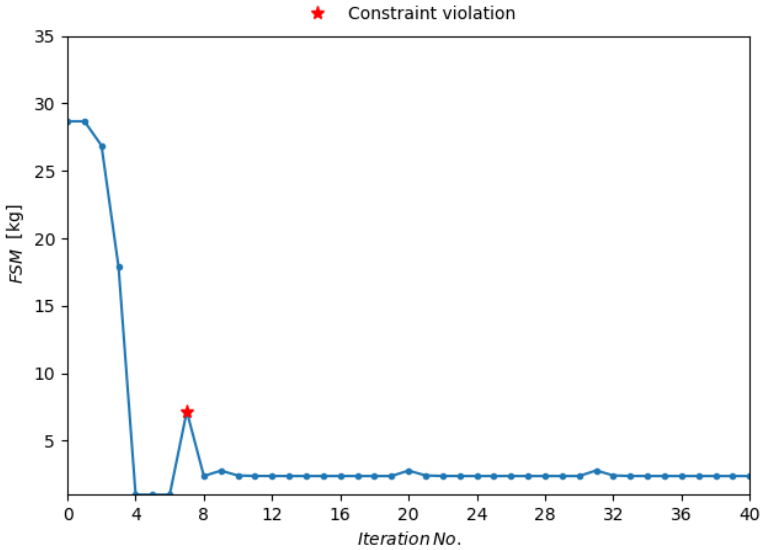
Figure 4.14: Goland+ objective function evolution

This drop on the wing's mass of the FE model has been caused mainly due to the decrease in the lumped masses that compose the front and rear spear, and that have been defined as the design variables. Figure 4.15 shows how there can be found a reduction in both design variables, meaning lighter front and rear spars, however, the most remarkable drop is the one presented in the rear spar. This change on the mass distribution directly swifts the center of gravity location of the model and impacts its aeroelastic response, being this the reason why the constraint in the flutter speed was required. The appreciable trend supposes that depending on the combination of rear and front spar masses, it impacts the flutter speed in such a way that increases with heavier front spars and lighter rear ones.

Again, the noticeable spikes in both design variable histograms find reasoning in the way the optimization algorithm works, introducing greater variations in order to account for situations where a local minimum could be reached instead of the global one, or also due to its nervous behaviour along the process.



(a) FSM



(b) RSM

Figure 4.15: Goland+ design variable evolution

The flutter speed variation versus the iteration number, see Figure 4.16, shows also an interesting behaviour. There, it can be directly spotted how the upper bound has been violated at the start of the optimization, nonetheless, the optimizer modifies the design variables values accordingly to the sensitivity analysis information provided by NOI and redirects the flutter speed inside the allowable bounds. Through the mentioned picture it is also worth mentioning how the flutter speed increases when decreasing the front and rear spar masses, as it has been mentioned. To get a clearer overview of this phenomenon, in Figure 4.10 there has been presented the tendency of the flutter speed when modifying the considered design variables.

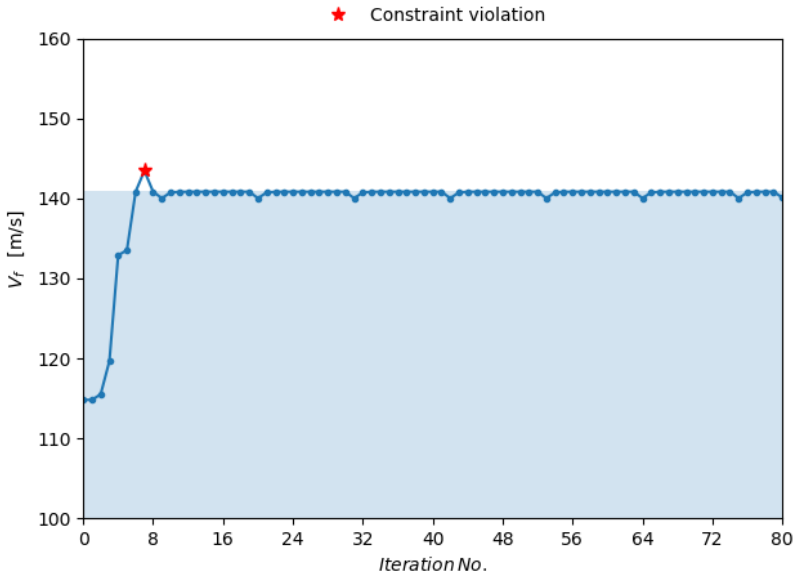


Figure 4.16: Goland+ constraint function evolution

Paying attention now to the optimization results shown in Table 4.10, some further conclusions may be found. First, it is clearly seen how a significant reduction in the structure mass has been achieved by acting on the front and rear spar groups of lumped masses. This circumstance supposes a 44% decrease that has been achieved following a 90% and 77% reduction on FSM and RSM respectively. On the other hand, and as it has been previously mentioned, this mass optimization comes along with an increase in the flutter speed of the Goland+ model of around a 22% over its initial baseline. All in all, it has been possible to confirm the success of this optimization case of study at the time of verifying the capabilities of NOI and FAEDO when working with lumped masses design variables. Furthermore, it has been optimized the mass of the Goland+ FE model keeping its flutter speed inside an acceptable range.

Parameter	**Initial value**	**Final value**	**% change**
m	3283.33 kg	1812.359 kg	-44%
FSM	28.677 kg	2.787 kg	-90%
RSM	77.928 kg	22.097 kg	-77%
V_f	114.808 m/s	140 m/s	+22%

Table 4.10: Goland+ model optimization results

4.4. Optimization of the Goland beam model

The presented optimization scenario takes into account the classic Goland wing with the objective of verifying the appropriate working of NOI and FAEDO at the time of setting-up finite element properties as design variables. The effects of the beam's cross-sectional area on the aeroelastic response of the wing have been analyzed, and its total structural mass has been minimized. For this purpose, the FE beam model discussed in Subsection 4.1.2 has been considered, paying close attention to the fact that the aeroelastic constraint applied on the flutter speed must be satisfied at the end of the optimization process.

4.4.1. Problem definition and set-up

Recalling the optimization case of study presented in Section 4.3, the theoretical formulation of the problem to be solved matches exactly Expression 4.2. For this particular scenario, the objective function happens to be the total mass of the FE beam model, which directly depends on the geometric characteristics and material properties of the beam. Due to the fact that the material properties remain constant along the optimization process, the considered parameter to act on the objective function of this case of study happens to be the cross-sectional area, defined as the unique design variable according to Table 4.11. Otherwise, the aeroelastic constraint has been imposed in such a way that the flutter speed has to fall inside a given range of values, between 100 and 140 m/s, being the reasoning and values of the flutter speed limits the same as the ones shown in Section 4.3.

Design variable	**Initial value**	**Lower bound**	**Upper bound**
A	0.0132 m^2	0.005 m^2	0.05 m^2

Table 4.11: Goland beam model optimization design variables

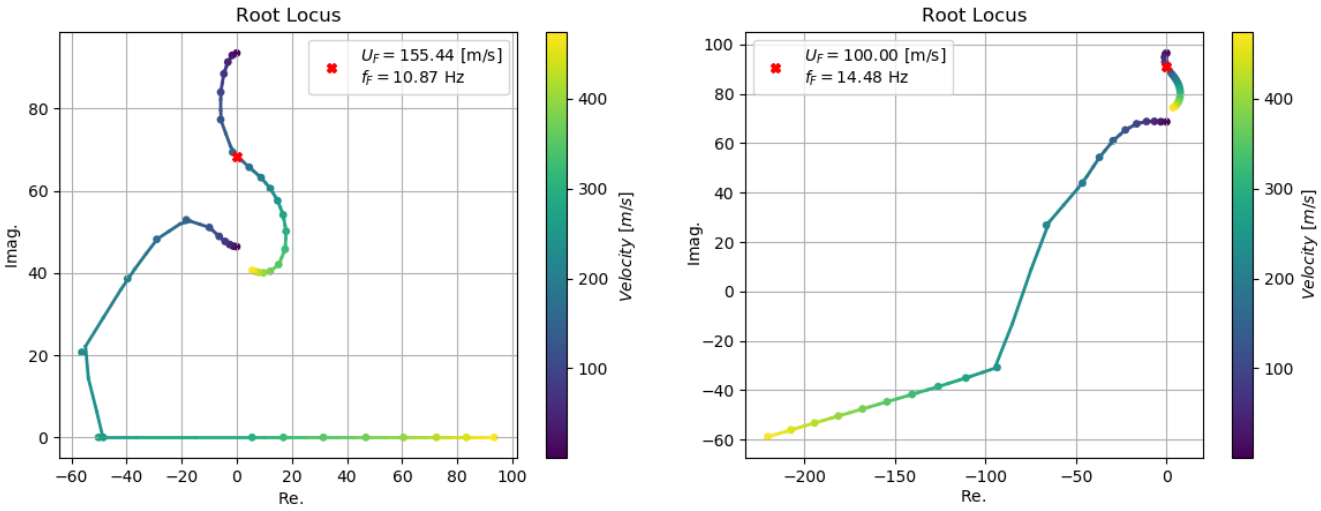
Once the problem has been properly defined, it is possible to set-up the FAEDO framework parameters. The first two eigenvalues of the Goland beam model have been assessed, as the flutter of the structure has been found to be caused due to the coupling of the first bending and torsion modes. Furthermore, the employed aerodynamic mesh has been identical to the one used in the previously carried out optimization process, keeping the same number of panels in the body and wake when solving for the aerodynamic analysis of the model. The atmospheric conditions at which all the computations have been performed directly match ISA sea level, being possible to verify the initial results with Reference [18].

The optimizer's running parameters have been gathered in Table 4.9, where it is important to note that scaling on the objective function has been applied for the shake of smoothing the spurious behaviour of the algorithm that may show up along the optimization. Besides, a maximum number of iterations has been set in case the method is not able to reach the specified tolerances and to avoid excessive waste of computational resources.

4.4.2. Results

The solved optimization problem reaches an optimum minimum of the objective function after 17 iterations and 104 function evaluations. The reader may not be familiar with the difference between iterations and function evaluations, being the second one greater because, for example, gradient estimations increase the function count over the iteration one. For the shake of ease, and along the presented discussion of results, the function evaluations have been treated as iterations, being fully aware of their difference and with the objective of getting a global overview of how the functions of interest vary along with the optimization. In this way, it has been decided to represent the values of the objective, design variable, and constraint function at every single step of the optimization process.

Prior assessing the final results, it may be possible to predict what may happen beforehand. As the final intention is minimizing the mass of the classic Goland wing acting on its cross-sectional area, it can be assumed that lower areas directly mean a mass reduction of the structure. However, the value of the cross-sectional area also impacts the flutter speed of the model. According to the flutter response of a generic wing, several factors play a huge role, being the relative location of its center of gravity and elastic axis two of the most important ones. When acting on the value of the beam's area, its inertia properties are directly modified, influencing its aeroelastic response. For the considered scenario, lower areas are translated into lower flutter speed due to the fact that an earlier coupling of the torsion and bending modes happens. This can be directly seen when comparing the root locus plots of the initial and end results of the optimization, see Figure 4.17.



(a) Iteration No. 1

(b) Iteration No. 17

Figure 4.17: Root locus of the Goland beam optimization

Once it has been clarified what it can be expected from this optimization scenario, it is possible to analyze the conclusions provided by the coupling of FAEDO with the optimizer. A summary of the actual results has been gathered in Table 4.12.

Focusing on the objective function history, which corresponds to the mass evolution of the Goland beam model, Figure 4.18 represents the previously mentioned trend when acting on the cross-sectional area, and how its value has been reduced until reaching a certain minimum. The initial plateau happens due to the initialization of the method, where two evaluations of the function have been performed without introducing any perturbation on the design variable. There, several constraint violations can be found due to the fact that in the early stages the optimizer is still trying to guess the proper minimization direction following the gradient information.

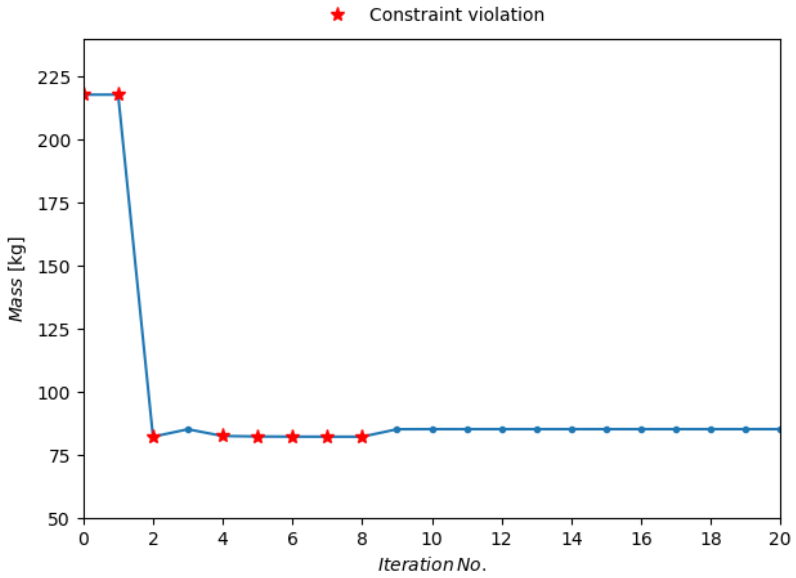


Figure 4.18: Goland beam objective function evolution

Moreover, it is important to compare this behaviour with the optimization history of the design variable and the flutter speed. Figure 4.19 depicts the optimizer guess values for the cross-sectional area of the beam, and how each time that it changes the flutter speed and mass of the model are also modified. When representing the constraint evolution, see Figure 4.20, the limits on the flutter speed have also been portrait in the shape of a blue-colored area. There, it is even easier to spot the constraint violation, as the lower limit on the flutter speed has not been met for the earlier function evaluations. From that point, the method is able to correct this deviation and keep on with the optimization until reaching a minimum.

Parameter	**Initial value**	**Final value**	**% change**
m	217.75 kg	85.29 kg	-61%
A	0.0132 m^2	0.0052 m^2	-61%
V_f	155.44 m/s	100 m/s	-36%

Table 4.12: Goland beam model optimization results

In the end, the results have been presented in Table 4.12 for the objective, design variable, and flutter speed. It is worth to mention that the framework has been able to achieve a reduction of around a 61% in the FE model mass while keeping a flutter speed of 100 m/s and acting only on the cross-sectional area. The obtained results allow concluding that the FAEDO framework, all together with NOI, is capable of solving optimization processes with finite element properties design variables, and aeroelastic constraints.

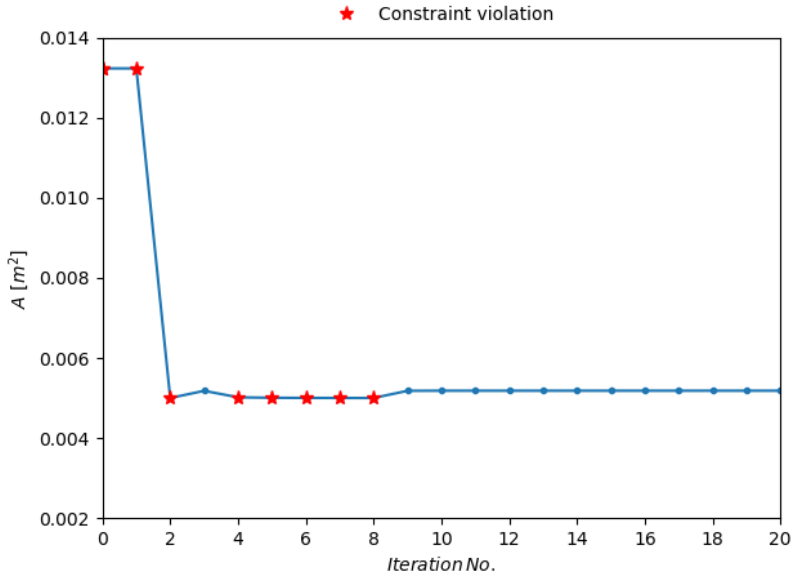


Figure 4.19: Goland beam design variable evolution

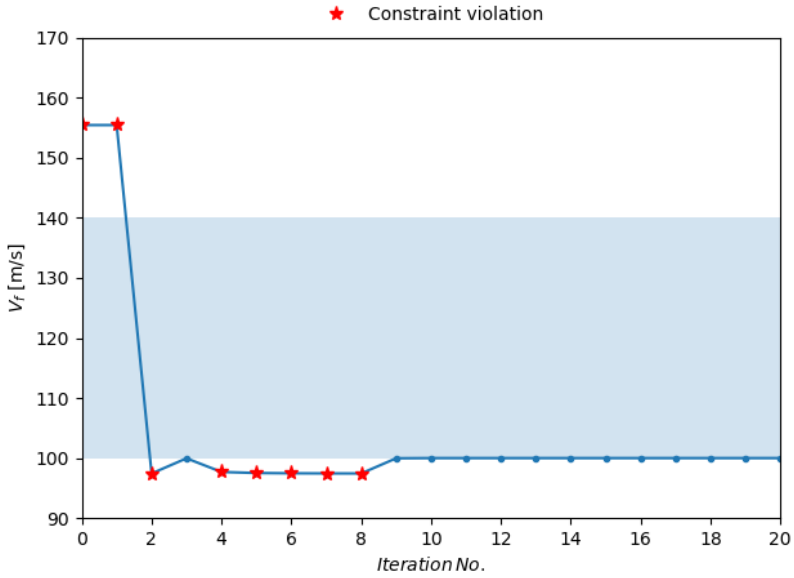


Figure 4.20: Goland beam constraint function evolution

5 Conclusions

The modern aircraft design process requires the inclusion of new methods capable of dealing with automatization, and which provide a high degree of robustness, being this the main reason under the development of the NOI tool and the FAEDO framework, which is capable of dealing with aeroelastic constraints. In that particular sense, the presented set of tools fulfil these requirements and add new features that could be directly applied, and aid the improvement of the aircraft design process.

In the first instance, the NOI tool has been proven to be a reliable asset at the time of solving structural and sensitivity analysis by means of *Nastran*. Additional features have been included, which allow the creation of optimization, modal and flutter input files, and also the post-processing of OP2 and f06 flutter output result files. Its integration in the FAEDO framework has granted the possibility to account for the structural analysis, while additional modules have been developed in order to solve for the aerodynamic and stability analysis at the time of computing the aeroelastic response of the input FE model. Once all the required methods have been properly arranged, the FAEDO framework brings the possibility to solve for the optimization of a given geometry, directly dealing with a flutter constraint, and being possible to couple it with an optimization platform such as OpenMDAO.

The characteristics described at the time of creating NOI and FAEDO happen to be appealing aspects to be potentially included in the aircraft design process. Furthermore, the FAEDO framework and NOI have been validated through several different test cases and models such as beam elements, shell elements, concentrated masses, and different design variables. Overall, the obtained results give an idea of the huge degree of flexibility and robustness of the developed tools.

Different improvements and test cases have been left for the future, being possible to expand the capabilities of both NOI and FAEDO. An attractive feature to be included in NOI would be accounting for shape design variables, as right now it is only able to work with size design variable, and also the implementation of a wider range of element properties to be set as design variables. On the other hand, the FAEDO framework could be also an object of improvement in terms of program comprehension, including the possibility to work with a global input file.

A | NOI user's guide

A.1. What is NOI?

Nastran Optimization Interface, also known as NOI, is a self-developed *Python* tool that provides the capability of writing *Nastran* optimization, modal and flutter input files given the finite element or design model description. NOI is also able of reading the modal, static, and sensitivity analysis results stored in a OP2 file, and to represent the flutter response when a proper F06 file has been supplied.

A.2. How to install NOI

For DLR users:

```
pip install git+https://gitlab.dlr.de/sl-fsi-mdo-collaboration/noi
```

For developers:

```
git clone https://gitlab.dlr.de/sl-fsi-mdo-collaboration/noi
cd noi
python setup.py develop
```

A.3. What is NOI capable of?

- Writing optimization, solution sequence 200, *Nastran* input files.
- Writing normal modes, solution sequence 103, *Nastran* input files.
- Writing flutter analysis, solution sequence 145, *Nastran* input files.
- Reading the optimization OP2 file and providing the sensitivity analysis results.
- Reading the optimization OP2 file and providing the structural analysis results.
- Reading the normal modes OP2 file and providing the structural results.
- Reading the flutter analysis F06 file and represent the V-g, V-f, and root locus.

A.4. Writing an optimization input file

This section of the user's guide gathers a detailed discussion concerning how to operate the tool at the time of writing an optimization BDF file. Some of the information provided here has already been introduced in Section 2.2.1 of the thesis, which is highly recommended to be carefully reviewed prior trying to work with the tool. Furthermore, the needed input dictionaries, which should be provided by the user, have been presented with practical examples so in this way the reader may find it even easier to understand how the tool has been structured.

A.4.1. NOI's input for optimization

The main blocks of information that must be supplied by the tool's user correspond to a properly formatted *Nastran* BDF file, which must contain the finite element model, and a set of *Python* dictionaries in charge of setting-up the optimization problem. Attending to the characteristics of the input BDF file, it has been required to include in a mandatory fashion the FE model and a set of entries that NOI may demand depending on the analysis type to be performed. In a general basis, the tool requires from the user supplied BDF the following information:

- FE model (nodes, element and properties entries)
- Materials
- Loads
- Boundary conditions

On the other hand, the following lines concerning the writing of an optimization input file have been devoted to show the reader how the input *Python* dictionaries have been in charge of setting-up the design optimization model, the proper way to define them, and also to discuss their limitations and the tool's range of capabilities.

A.4.2. Optimization input dictionaries

The input dictionaries have been the preferred way to define the characteristics of the optimization problem. In such manner, they have been sorted in three different categories depending on the information that they provide:

- Design variables input dictionaries
- Responses input dictionary
- Analysis input dictionary

A.4.2.1. Design variables input dictionaries

According to *Nastran*, and for the shake of defining the design variables of the model, it has been needed to include a set of DESVAR entries which have to be linked to a given element property, connectivity or grid point location through the DVPREL, DVCREL and DVGRID cards.

At the moment, NOI is only capable of including design variables related to element properties and lumped masses, meaning this last group the CONM2 entries stored in the FE model. The element properties design variables have been handled by means of the *set_property_dv* method and the *property_dict* and *property_block_dict* dictionaries. Otherwise, the lumped masses design variables have been included attending to the *set_mass_dv* method and the *masses_dict* input dictionary.

- **Element properties design variables:** Due to the fact that in order to deal with element properties it has been needed to write new property cards depending on the specified design variables, two user-input dictionaries are directly requested by NOI. The first one, *property_block_dict*, is responsible of writing the new property blocks, while the second one, *property_dict*, is accountable of appending the corresponding DESVAR and DVPREL cards following the user supplied information.
- **Lumped masses design variables:** In this particular scenario, just the definition of the *masses_dict* has been required, granting the set of masses to be defined as design variables, their corresponding values, and appending the DESVAR and DVCREL entries to the optimization BDF file.

The following blocks of code depict the structure that must be followed at the time of setting-up the design variables input dictionaries and also two practical examples. In a general matter, the keys of the dictionaries correspond to a representative label, and the values to a tuple that contains the design variable information.

property_block_dict = {'label of the block of properties': [list of affected elements]}
property_dict = {'label of the design variable': ('label of the block of properties', type of design variable, value of the design variable)}
masses_dict = {'label of the design variables': ([list of affected masses], 'value of the design variable')}

Definition of skin thicknesses of elements 1 and 2 as a design variable:
property_block_dict = 'skin': [1,2]
property_dict = 'SkT': ('skin', 4, .0155)

Definition of a set of CONM2 entries as a design variable
masses_dict = 'M1': ([1001, 1002, 1003], 15)

A.4.2.2. Responses input dictionary

The model responses have been included in the optimization BDF file through the DRESP1 *Nastran* entry and recalling the *set_responses* method which takes as input the *set_responses_dictionary*. Depending on the type of responses considered, different analysis may be included in the BDF file, see Section 2.2.1.3.

The responses dictionary compiles the definition of the responses to be taken into account along the optimization. The keys of the dictionary correspond to the user defined labels for the responses, and values have been set to be a tuple that contains the type of response and a dictionary dependent on each of them. The user is free to specify whatever label for the response, but it is recommended to use a representative one depending on the type of response considered. For eigenvalues and eigenvector responses it has been required a normal modes analysis subcase, as well as, for a stress response it has been mandatory including a statics one.

In case of including a stress response, it has been allowed to specify the elements of the FE model to which this response shall be applied and the type of stress to be evaluated through *Nastran's* stress code formulation. Otherwise, for eigenvalue and eigenvector responses it is needed to supply the mode number. Bounds have not been required for either the stress, eigenvalue, or eigenvector responses.

The presented blocks of information represent the formatting to be followed by the responses dictionary depending on the type of included responses, and also several practical examples for an easier understanding. The first of them compiles the differences between the dictionaries that must be supplied after the responses identifier depending on each type.

Stress (S): {'elements': [eids], 'stress_code': stress_code, 'bounds': [lw, ub]}
Eigenvalue (EVAL) : {'mode': mode_number, 'bounds': [lw, ub]}
Eigenvector (EVEC): {'mode': mode_number, 'bounds': [lw, ub]}

Von Misses stress response in elements 1, 2, 3:
responses_dict = {'FsS':('S', {'elements': [1,2,3],'stress_code':5})}

First two eigenvalues with a stress response in element 1:
responses_dict = {'FsS':('S',{'elements':[1],'stress_code':5}), 'EV1':('EVAL', {mode: 1}), 'EV2':('EVAL', {mode: 2})}

First eigenvalue and eigenvector:
responses_dict = 'EV1':('EVAL', {mode: 1}), 'EVC1':('EVEC', {mode: 1})

A.4.3. Analysis input dictionary

The case control deck of the optimization BDF file compiles the analysis that shall be performed, and it has been required to input the *analysis_dict*. Then, by means of the *set_case_control_deck* method NOI includes all the information regarding the analysis types, subcase structure, output parameters, etc. The structure of the *analysis_dict* is relatively easy to understand, the keys of the dictionary correspond to the identifier of the subcase type that the user wants to perform, 'M' for modal and 'S' for static, and the values provide NOI additional information to be appended in the subcase definition section, for example the load identifier in the static subcase.

analysis_dict = {'identifier of the subcase type': 'information on the subcase'}

Modal analysis:
analysis_dict = {'M': None}

Modal and static analysis under the action of a load:
analysis_dict = {'S': [100] , 'M': None}

A.4.4. Example of writing an optimization BDF file

This section illustrates how NOI allows writing an optimization *Nastran* file requesting as input the FE model described in Section 4.1.1. The obtained output corresponds to a properly formatted solution sequence 200 file that defines as design variables the thickness of the FE model skin elements and a set of three lumped masses. Besides, a modal and static analysis has been included due to the fact that the preferred responses correspond to the Von Misses stress in elements 1 and 2, the first eigenvector and the first eigenvalue, see Table A.1.

The presented block of code compiles all the necessary steps to be covered in the *Python* script. Firstly, it has been required by the user to properly install the NOI tool, see Section A.2, and to call the BDFOpt class in order to access the methods to write the optimization BDF file. Then, the user must include the input dictionaries that set-up the design optimization model. In this particular situation, a combination of element properties and lumped masses design variables has been established, being this the reason why both the *property_dict* and *masses_dict* dictionaries must be provided. Finally, the user must recall the *write_opt_bdf* method included in the BDFOpt class to output the final optimization file.

Initialization:
from noi.BDFWriter import BDFOpt

Input dictionaries definition:
property_block_dict = {'skin': list(range(1, 20))}
property_dict = {'SKT': ('skin', 'T', .1)}
masses_dict = {'M1': ([1210, 1211, 1212], 6.)}
responses_dict = {'FsS': ('S', {'elements': [1,2], 'stress_code': 5}),'EV1': ('EVec', {'mode': 1}), 'EV1': ('EVal', {'mode': 1})}
analysis_dict = {'M':None, 'S':[]}

Requesting the output file:
bdf_filename = 'goland_clean_103.bdf'
mdl = BDFOpt(bdf_filename)
mdl.write_opt_bdf('goland_optimization_output.bdf', responses_dict, analysis_dict, property_block_dict=property_block_dict, property_dict=property_dict, masses_dict= masses_dict)

Design variables	Thickness of elements [1 to 20], set of lumped masses [1210, 1211, 1212]
Responses	Eigenvalue and eigenvector no. 1, Von Misses stress in element 1
Analysis types	Modal and static

Table A.1: Summary of the example optimization problem

A.5. Writing a modal analysis file

The process required to output a modal analysis file happens to be much easier in comparison with the optimization one. The user has only been requested to input the finite element model. Then, NOI is capable of properly formatting a solution sequence 103 output file, recalling its BDFModal class. Through the implemented methods the tool is able to account for the desired number of modes to evaluate, and also recognizes multiple spc entries to conform different analysis subcases. Besides, the tool allows the possibility to define the normalization mode for the eigenvectors, and the default number of shapes to be considered has been set to two.

A.5.1. Example of writing a modal BDF file

Attending to the input finite element model described in Section 4.1.1, the presented block of code details the process to accurately output its corresponding modal analysis file. For the evaluated scenario, the normalization mode matches the mass one, and there have been included up to three modal shapes.

Initialization:
from noi.BDFWriter import BDFModal

Calling the tool:
bdf_filename = 'goland_clean_103.bdf'
mdl = BDFModal(bdf_filename)
mdl.write_modal_bdf('goland_modal_analysis.bdf', norm='MASS', n_modes=3)

A.6. Writing a flutter analysis file

At the time of writing a flutter analysis file, and according to the way *Nastran* works, the BDFFluter class directly relies upon the BDFModal one and contains the *write_flutter_bdf* method. It is also worth to mention that the input BDF must include the CAERO and PAERO entries in order to generate the appropriate solution sequence 145 file. Furthermore, the user is able to specify the most remarkable flutter analysis parameters, see Table A.2, such as velocity, reduced frequency, and density range, symmetry, or even the number of modes.

Flutter parameters	Number of modes, density range, Mach number range, velocity range, k range, reference length, reference density, symmetry and reference velocity

Table A.2: Writing a flutter file parameters

A.6.1. Example of writing a flutter BDF file

The flutter analysis *Nastran* file has been obtained according to the BDFFlutter class, and attending to the geometry discussed in Section 4.1.1. The depicted block of code features the required actions to output the flutter analysis file. Along this particular example, there has been included the definition of the reference length and the velocity range parameters.

Initialization:
from noi.BDFWriter import BDFFlutter

Calling the tool:
bdf_filename = 'goland_clean_103.bdf'
mdl = BDFFlutter(bdf_filename)
mdl.write_flutter_bdf('goland_flutter.bdf', l_ref=1.83, velocity_range=[10... 200])

A.7. Reading of modal and static analysis results

As it has been discussed along the thesis, NOI also allows the post-processing and verification of the obtained results when modal, static and sensitivity analysis results have been stored in an external OP2 file. For this purpose, it is possible to rely upon two different classes which go under the name of OP2Static and OP2Modal, depending on the type of results the user wants to analyze.

A.7.1. Example of reading static analysis results

Initialization:
from noi.OP2Reader import OP2Static

Calling the tool:
op2_filename = 'goland_static_results'
op2 = OP2Static(op2_filename)
op2.stress_output()

A.7.2. Example of reading modal analysis results

Initialization:
from noi.OP2Reader import OP2Modal

Calling the tool:
op2_filename = 'goland_modal_results'
op2 = OP2Modal(op2_filename)
op2.eigenvalues_output()

A.7.3. Example of reading sensitivity analysis results

A particular scenario arises at the time of trying to read the sensitivity analysis results, as two ways may be followed. In the situation the user wants to output the general, and unsorted, sensitivity matrix, it has been required to use the method stored inside the OP2Static class. However, a more precise method for the modal scenarios has been developed, as discussed in Section 2.3.2, where the responses dictionary must be provided, and the sensitivity matrix has been provided in a more structured fashion according to the OP2Modal class.

Initialization:
from noi.OP2Reader import OP2Static

Calling the tool:
op2_filename = 'goland_sensitivity_results'
op2 = OP2Static(op2_filename)
op2.sensitivity_global()

Initialization:
from noi.OP2Reader import OP2Modal

Input dictionaries definition:
responses_dict = {'EV1': ('EVec', {'mode': 1}), 'EV1': ('EVal', {'mode': 1})}

Calling the tool:
op2_filename = 'goland_sensitivity_results'
op2 = OP2Modal(op2_filename)
op2.sensitivity_custom(responses_dict=responses_dict)

A.8. Reading a flutter F06 file

NOI also supports reading and post-processing the results of a conducted flutter analysis that have been stored in a F06 file. Once the input file has been evaluated by the tool, the user is able to request the V-g, V-f and root locus plots by means of the methods coded inside the F06Reader class.

A.8.1. Example of reading a flutter F06 file

A concise example in relation to the analysis of a solution sequence 145 F06 file has been presented in the following block of code. There, the first two modes have been the ones selected to be depicted in the V-g, V-f and root locus representations.

Initialization:

```
from noi.F06Reader import XF06
```

Calling the tool:

```
f06_filename = 'goland_model_145.f06'
flutter = XF06(f06_filename, modes=[1, 2])
flutter.plot_root_locus()
flutter.plot_vg_vf()
```

Bibliography

[1] Peter W Christensen and Anders Klarbring. *An introduction to structural optimization*. Vol. 153. Springer Science & Business Media, 2008.

[2] Gaetan Kenway, Graeme Kennedy, and Joaquim Martins. "A scalable parallel approach for high-fidelity aerostructural analysis and optimization". In: *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference 20th AIAA/ASME/AHS Adaptive Structures Conference 14th AIAA*. 2012, p. 1922.

[3] UT Ringertz. "On structural optimization with aeroelasticity constraints". In: *Structural optimization* 8.1 (1994), pp. 16–23.

[4] Dewey H Hodges and G Alvin Pierce. *Introduction to structural dynamics and aeroelasticity*. Vol. 15. cambridge university press, 2011.

[5] MSC Software. *Quick Reference Guide*. MSC Software, 2018.

[6] MSC Software. *Design Sensitivity and Optimization User's Guide*. MSC Software, 2017.

[7] Justin S. Gray et al. "OpenMDAO: An open-source framework for multidisciplinary design, analysis, and optimization". In: *Structural and Multidisciplinary Optimization* 59.4 (Apr. 2019), pp. 1075–1104. DOI: 10.1007/s00158-019-02211-z.

[8] Armin Beckert and Holger Wendland. "Multivariate interpolation for fluid-structure-interaction problems using radial basis functions". In: *Aerospace Science and Technology* 5.2 (2001), pp. 125–134.

[9] Thomas CS Rendall and Christian 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.

[10] Luigi Morino. "A general theory of unsteady compressible potential aerodynamics". In: (1974).

[11] Francesco Torrigiani and Pier Davide Ciampa. "Development of an unsteady aeroelastic module for a collaborative aircraft MDO". In: *2018 Multidisciplinary Analysis and Optimization Conference*. 2018, p. 3879.

[12] Francesco Torrigiani et al. "FLUTTER SENSITIVITY ANALYSIS FOR WING PLANFORM OPTIMIZATION". In: (2019).

[13] L Morino et al. "Matrix fraction approach for finite-state aerodynamic modeling". In: *AIAA journal* 33.4 (1995), pp. 703–711.

[14] John P Jasa et al. "How Certain Physical Considerations Impact Aerostructural Wing Optimization". In: *AIAA Aviation 2019 Forum*. 2019, p. 3242.

[15] Rula M. Coroneos. "Structural Analysis and Optimization of a Composite Fan Blade for Future Aircraft Engine". In: *NASA/TM 217632*. Aug. 2012.

[16] Justin S. Gray et al. "Coupled Aeropropulsive Optimization of a Three-Dimensional Boundary-Layer Ingestion Propulsor Considering Inlet Distortion". In: *Journal of Aircraft* (2020). DOI: 10.2514/1.C035845.

[17] Christopher Heath and Justin Gray. "OpenMDAO: framework for flexible multidisciplinary design, analysis and optimization methods". In: *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference 20th AIAA/ASME/AHS Adaptive Structures Conference 14th AIAA*. 2012, p. 1673.

[18] Martin Goland. "The flutter of a uniform cantilever wing". In: *Journal of Applied Mechanics-Transactions of the Asme* 12.4 (1945), A197–A208.

[19] FE Eastep and JJ Olsen. "Transonic flutter analysis of a rectangular wing with conventional airfoil sections". In: *AIAA Journal* 18.10 (1980), pp. 1159–1164.

[20] Philip S Beran et al. "Numerical analysis of store-induced limit-cycle oscillation". In: *Journal of Aircraft* 41.6 (2004), pp. 1315–1326.

[21] Marco Berci and Rauno Cavallaro. "A Hybrid Reduced-Order Model for the Aeroelastic Analysis of Flexible Subsonic Wings—A Parametric Assessment". In: *Aerospace* 5.3 (2018), p. 76.