Frequency-Domain Method for Flutter on a Highly Deformed Wing

Rodrigo Gaspar Pinto Ramos Correia

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisors:	Prof. Fernando José Parracho Lau
	Dr. Frederico José Prata Rente Reis Afonso

November 2020

Acknowledgments

As the author of this work, I would like to thank:

My coordinator, Prof. Fernando Lau, for the availability and helpful advice.

My co-coordinator, Frederico Afonso, for the extensive and tireless feedback and for always being available whenever I needed assistance.

The IST Aerospace Group, for the assistance provided in several occasions.

My mom and my sister, for support and help along the way.

Abstract

The objective of this work is to build a frequency domain tool through the assembly of already existing frameworks, which performs a flutter analysis on high aspect-ratio wings where the deformed wing configurations and its influence over the flutter speed are taken into account. From a Nonlinear Aeroelastic Framework, the tool gets as input the wing nonlinear aeroelastic static equilibrium positions and through a MATLAB code builds the structural and aerodynamic models (based on the Euler-Bernoulli Beam Theory and on the Doublet-Lattice Panel Method, respectively). Then, the solver NASTRAN performs the aeroelastic analyses by applying the p-k method at those equilibrium positions. These analyses are performed in an iterative cycle, so it is possible to take into account the wing deformation due to the airspeed while calculating the flutter speed. The obtained updated flutter speed is then compared with one calculated by a time domain approach. This comparison showed that despite some identical trends in both approaches, it was not possible to predict exactly the same flutter speed in both techniques. Still, it could be concluded that the developed frequency domain tool managed to produce consistent results at a low computational cost.

Keywords: Aeroelasticity; flutter; frequency-domain; geometric nonlinearities; high aspect-ratio wings.

Resumo

Este trabalho tem por objetivo construir uma ferramenta no domínio da frequência a partir de ferramentas já existentes, que efetue uma análise de flutter em asas de elevado alongamento, onde a configuração deformada da asa e sua influência na velocidade de flutter sejam consideradas. A partir de uma Ferramenta de Aeroelasticidade Não-linear, a ferramenta no domínio do tempo obtém como input a posição de equilíbrio estático não-linear aeroelástico da asa e, por meio de um código MATLAB, constrói os modelos estruturais e aerodinâmicos (baseados na Teoria de Viga de Euler-Bernoulli e no *Doublet-Lattice Panel Method*, respetivamente). De seguida, o solver NASTRAN realiza as análises aeroelásticas aplicando o método p-k nessa posição de equilíbrio. Estas análises são realizadas num ciclo iterativo, permitindo assim considerar a deformação da asa devido à velocidade do ar durante o cálculo da velocidade de flutter. Para avaliar a precisão da ferramenta, é feita a comparação da velocidade de flutter atualizada com uma calculada por um método no domínio do tempo. Esta comparação mostrou que, apesar de terem sido identificadas algumas tendências idênticas em ambas as abordagens, não foi possível prever exatamente a mesma velocidade de flutter. Ainda assim, pôde-se concluir que a ferramenta desenvolvida no domínio da frequência conseguiu produzir resultados consistentes a um baixo custo computacional.

Palavras-chave: Aeroelasticidade; flutter; domínio da frequência; não-linearidades geométricas; asas de elevado alongamento.

Contents

List of Tables

List of Figures

Nomenclature

A_C	aerodynamic damping matrix
A_K	aerodynamic stiffness matrix
$A(p)$	matrices of aerodynamic stiffness and damping
AR	wing aspect-ratio
b	wing span
c	wing chord
C	damping matrix
E	Young modulus
EI_1	flap bending stiffness
EI_2	chord bending stiffness
F	applied load vector
F_k	aerodynamic forces
F_g	structural forces
g	gravity acceleration vector
G	shear modulus
GJ	torsional stiffness
G_{kg}	interpolation matrix
I_2/I_1	inertia ratio
k	reduced frequency
K	stiffness matrix
L	reference length
M	mass matrix
M_∞	Mach number
p	nondimensional Laplace number
q	system degrees of freedom
q_0	static aeroelastic equilibrium state
$\hat{q}$	vector of the amplitudes of the disturbed degrees of freedom
Re	Reynolds number
s	complex frequency
S	wing area
t	time
U	airspeed
u_g	structural grid point deflections
u_k	aerodynamic grid point deflections
u_x	horizontal tip displacement
u_z	vertical tip displacement
$u_z/(b/2)$	dimensionless vertical tip displacement
V_f	flutter speed

α	angle of attack
δ	real part of the nondimensional Laplace parameter
θ_y	angle of twist
ρ	air density

1. Introduction

1.1. Background and Motivation

Over the years, the aviation industry has constantly pursued more efficient aircraft solutions. With this goal in mind, high aspect-ratio wings have been one of the most studied aircraft configurations. The idea behind it is that an increase in the wing aspect-ratio reduces the aerodynamic induced drag [1], which then results in a decrease in the aircraft fuel consumption. However, despite the aerodynamic benefit that grants a higher lift-to-drag ratio and a longer range to the aircraft, there are a few structural disadvantages related to this configuration, like higher wing flexibility and (if there is an increase in the wing span) larger root bending moment.

The increase in wing flexibility derived from a higher aspect-ratio while keeping a light structure leads to large deformations at normal operating conditions, which in turn results in the occurrence of aeroelastic effects on such a wing, at a lower speed than in a stiffer wing [2].

Despite causing strain, some of these effects are benign, like buffeting. However, other effects such as flutter can be quite catastrophic [3]. And since flutter is the most nefarious aeroelastic phenomenon, it is generally the key focus of analysis in the aeroelasticity field.

Although it is not under the scope of this work, some innovation in the field of augmenting aircraft aerodynamic performance goes a lot further than increasing the wing aspect-ratio, and has resulted in quite original aircraft configurations, such as the Blended Wing Body [4] or the Joined Wing [5]. Other wing configurations, less extravagant than the ones just mentioned, like the Truss-Braced Wing [6], aim to have higher wing aspect-ratio and span without having the aforementioned aeroelastic problem.

In the last few decades, highly flexible high aspect-ratio wings were the subject of analysis through different numerical studies which used low-medium fidelity computational tools [7], such as the University of Michigan Nonlinear Aeroelastic Simulation Toolbox [8], the Nonlinear Aeroelastic Trim And Stability of High Altitude Long Endurance (HALE) Aircrafts [9], the Nonlinear-Aerodynamic/Nonlinear-Structure Interaction Methodology for a HALE Wing [10], or the Unsteady Vortex Lattice Method [11]. Most of these aeroelastic models are rooted in the coupling of nonlinear beam equations with 2D aerodynamic models, and the flutter speed can be estimated either by employing a time marching scheme or by using a frequency domain approach.

In cases where structural nonlinearities are likely to occur, such as in highly flexible wings, the accuracy of time domain methods is usually superior to that of frequency domain methods, regarding the determination of the flutter speed [12]. However, time domain methods carry several disadvantages, like a higher computational cost, the need of various runs at different airspeeds and a more complex interpretation of the results. These are obstacles that make the application of such type of methods to aircraft design and optimization problems quite impracticable. Unlike time domain methods, frequency domain methods can be run with a lower computational cost and be easier to apply to an optimization or an aircraft design problem. Although in highly flexible wings, because of the nonlinearities that may cause flutter at lower speeds than what linear analyses predict, the frequency domain methods need to be accordingly updated in order to accurately predict the flutter phenomenon.

There are not many studies in the literature which compare these two approaches for predicting flutter speeds. In fact, the author did not find any which featured a comparison for highly flexible wing structures, where the geometrical nonlinearities would be relevant. In 2007, Salvatori and Borri [13] compared a time domain method with a frequency domain method for an aeroelastic analysis of a bridge model. Despite verifying that indeed their frequency domain approach arrived at the same results of their time domain approach at a lower computational cost, it is thought that such a frequency domain approach (which is not updated) would not apply to structures that present nonlinearities [14]. Therefore, in this context, there would be value in developing an updated frequency domain tool with the goal of accurately estimating the flutter speed of high aspect-ratio wings, (which are structures that easily present relevant geometrical nonlinearities) and comparing its results to those produced by a time domain method.

1.2. Objectives

The objective of this work is to build a frequency domain tool which, through the assembly of already existing frameworks, performs a flutter analysis on high aspect ratio wings where the deformed wing configurations and its influence over the flutter speed are taken into account. This tool will be used to study wings of different aspect ratios in order to assess the impact of the large wing deformation (a characteristic of high aspect ratio wings) on the flutter speed. In this work, such an updated frequency domain method will be conceived and its accuracy in determining the flutter speed of high aspect-ratio wings will be evaluated through a comparison of its results with the ones drawn from a time domain method developed by the IST Aerospace Group [15].

1.3. Thesis Layout

This thesis is structured in the following way:

1. In the first chapter, the context and motivation behind the work are introduced and its objectives are stated.
2. The second chapter focuses on the theoretical background which supports the work.
3. In the third chapter, the computational tool which was developed to reach the objectives is explained in detail.
4. In the fourth chapter, the application case of study is presented, and the results obtained with the frequency domain tool are shown and compared to the ones obtained through a time domain method.
5. Lastly, in the fifth chapter, concluding remarks are laid out and notes for the future are suggested.

2. Theoretical Background

In this work, the source of all the results obtained by both the time domain approach and the frequency domain approach is the Nonlinear Aeroelastic Framework developed by the IST Aerospace Group [15] [16]. In this framework, a Fluid-Structure Interaction (FSI) algorithm was implemented to couple the structural component of the model (a condensed 3D beam model) with the aerodynamic component of the model (a 3D panel code method) for nonlinear unsteady and steady analyses.

The nonlinear unsteady analyses, which estimate the wing's flutter speed based on the aeroelastic response to a prescribed gust at a given time interval, constitute the time domain approach. And for this work, these unsteady simulations are only used for comparison purposes with the updated frequency domain methodology.

On the other hand, the steady simulations are carried out to provide a position of stationary equilibrium, where the structural and the aerodynamic meshes converge, to the updated frequency domain method. The convergence process during the steady simulations will be now succinctly described. In these simulations, the aerodynamic loading is applied on the structural mesh. Then, this deformation on the structural mesh changes the aerodynamic mesh, which on its turn originates a new loading that will deform the structure again. This iterative process goes on until the point where both the current structural deformation and the current aerodynamic loading are equal to the previous ones (under a certain tolerance factor). Therefore, when both the aerodynamic loading and the structural deformation converge, we say that a position of stationary equilibrium has been reached.

These converged positions of stationary equilibrium are what feeds the developed frequency domain analysis tool, which will be described in depth in the next chapter. This frequency domain tool resorts to a MATLAB code which generates the whole frequency domain analysis. It takes as input the wing converged positions and from there it builds the structural and aerodynamic models. The structural model is based on a one-dimensional cantilever beam, while the aerodynamic model is based on a 2D aerodynamic panel method. Then, on these models, NASTRAN performs the aeroelastic analyses, calculating the flutter speed by applying the p-k method at the nonlinear aeroelastic static equilibrium position. These analyses are performed in an iterative cycle, so it is possible to take into account the wing deformation due to the airspeed while calculating the flutter speed. We so obtain the updated flutter speed, to be then compared with the one calculated by the time domain approach.

This chapter will be focused on describing the theoretical background behind the steady and unsteady aeroelastic analyses, as well as behind the structural and aerodynamic components of the aeroelastic modelling. Also, the solver used in the frequency domain tool (NASTRAN) will be introduced and the way how it connects the two components of the model through spline interpolation and employs the p-k method to perform the aeroelastic analysis will be looked upon.

2.1. Aeroelasticity

Before proceeding with the theoretical background behind aeroelastic analysis, it is useful to first introduce precise terminology to describe some aeroelastic problems. The ones that will be looked upon in this work are two particular cases of what can be described as Instability Boundary Problems [17]. These problems are concerned with determining the boundary which separates a stable fluid interaction from an unstable one and can be either static or dynamic. While in static problems the inertia effect is not accounted for, dynamic problems are the more general type where inertia effects must be included. The diagram shown below presents two aeroelastic instability problems which will be mentioned in this work.

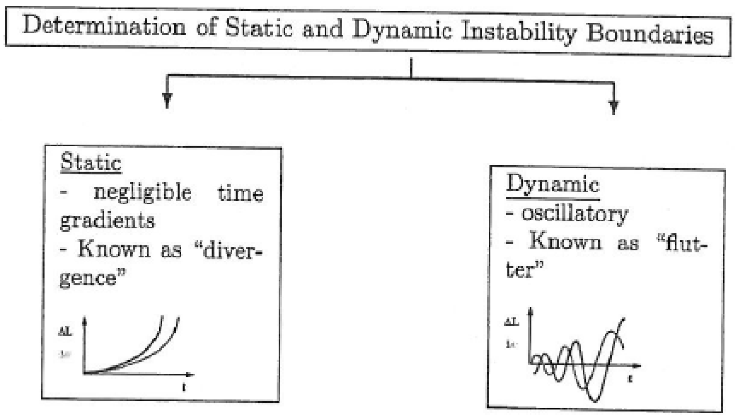

Determination of Static and Dynamic Instability Boundaries
Static
- negligible time gradients
- Known as "divergence"
Dynamic
- oscillatory
- Known as "flutter"
ΔL
t
ΔL
t

Figure 1 - Static and Dynamic aeroelastic instability problems [17]

A more general way of visualizing aeroelastic problems is the Collar's Triangle, shown in the picture below. This shows the positioning of some aeroelastic problem types in relation to the forces considered within the model. Beside the aeroelastic problems, it is possible to locate several other classic aerospace problems on the diagram.

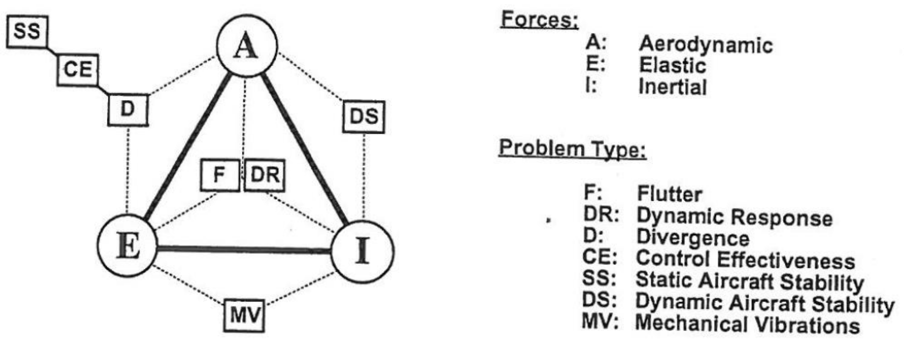


Figure 2 - Collar's Triangle (1946) [17]

As it can be seen in the diagram above, the aeroelastic phenomenon results from the interaction of aerodynamic, inertial and elastic forces. A typical way of presenting such interaction is to describe the aeroelastic problem as a system of second-order differential equations in terms of its degrees of freedom $\{q\}$:

$$[M]\{\ddot{q}\} + [C]\{\dot{q}\} + [K]\{q\} = \{F\} \tag{1}$$

where $[M]$, $[C]$ and $[K]$ respectively represent the mass, damping and stiffness matrices of the system's structure; and $\{F\}$ represents the total load vector. The aeroelastic problem shown in the equation above can be considered as either steady or unsteady.

2.1.1. Steady Aeroelastic Analysis

In a steady aeroelastic analysis, the solution is assumed stationary and thus all the time dependent terms are eliminated. By doing so, only the interaction between the elastic and the aerodynamic forces is considered, which results in the following reduced system:

$$[K]\{q\} = \{F\} \tag{2}$$

The load vector in the equation above contains both aerodynamic and gravity loads:

$$\{F\} = [A_K]\{q\} + [M]\{g\} \tag{3}$$

where $[A_K]$ and $\{g\}$ are the aerodynamic stiffness matrix and the gravity acceleration vector, respectively. As it is possible to see from the first term of the right-hand side member of the above equation, the aerodynamic loads depend on the system's degrees of freedom, which makes this analysis a coupled problem. A way to solve this problem is through the implementation of a Fluid-Structure Interaction algorithm, which purpose is to transfer the loads from the aerodynamic mesh to the structural mesh and the structural displacements to the aerodynamic mesh, while ensuring convergence at generalized coordinates. There are two main FSI approaches to establish an equilibrium between aerodynamic and structural solvers: strongly-coupled; and loosely-coupled. The former consists in solving the fluid and structure equations simultaneously, while in the latter the equations of fluid and structure are solved separately and coupled using a scheme until convergence is reached. In the framework developed by the IST Aerospace Group, a loosely-coupled method is employed to reach convergence at a position arbitrarily close to the equilibrium state:

$$[K]\{q\} - [A_K]\{q\} - [M]\{g\} \approx 0 \tag{4}$$

It is this converged position resulting from the static analysis that will feed the frequency domain tool described ahead.

2.1.2. Unsteady Aeroelastic Analysis

In unsteady aeroelastic problems Eq. 1 is used, involving all three interaction forces: aerodynamic, elastic and inertial. This means that the load vector when compared with the steady case will now include the aerodynamic damping contribution:

$$\{F\} = [A_C]\{\dot{q}\} + [A_K]\{q\} + [M]\{g\} \tag{5}$$

where [A$_C$] is the aerodynamic damping matrix. Aerodynamic mass contribution is neglected in this work. The key aeroelastic problem focus of study in this work is flutter, which can be studied either in the time domain or in the frequency domain.

2.1.2.1. Time Domain Approach

A time-marching scheme is required for the task of solving Eq. 1 that represents the aeroelastic problem. Like for the steady analyses, a proper FSI algorithm is required to ensure convergence between the two solvers (aerodynamics and structures). As for the steady FSI algorithms both strongly-coupled and loosely-coupled approaches can be followed. In the aeroelastic framework from the IST Aerospace Group, the loosely-coupled FSI approach is used to ensure a convergence between aerodynamic and structural meshes by means of a predictor-corrector scheme at each time step used [15]. The results obtained through this method are the ones to which we will compare our results obtained through the frequency domain tool.

2.1.2.2. Frequency Domain Approach

To allow for solving the same problem (Eq. 1) in the frequency domain, it is necessary to perform adequate modifications to the degrees of freedom, which consists in assuming disturbances at the aeroelastic static equilibrium state $\{q_0\}$. The disturbances are represented as harmonic oscillations $\{\hat{q}\}e^{st}$ at a given damped complex frequencies s and so the total deformation for the aeroelastic dynamic problem becomes:

$$\{q\} = \{q_0\} + \{\hat{q}\}e^{st} \tag{6}$$

If we consider this assumption, then the equilibrium equation of the dynamic aeroelastic problem can be written as:

$$s^2[M]\{\hat{q}\}e^{st} - s[A_C]\{\hat{q}\}e^{st} + ([K] - [A_K])\{\hat{q}\}e^{st} + ([K] - [A_K])\{q_0\} - [M]\{g\} = \{0\} \tag{7}$$

where the matrices of aerodynamic stiffness $[A_K]$ and damping $[A_C]$ depend nonlinearly on the complex frequency s. As mentioned before, the vector of amplitudes of the disturbed system degrees of freedom and the aeroelastic static deformed state are represented by $\{\hat{q}\}$ and $\{q_0\}$, respectively. To estimate the flutter speed, the mass $[M]$, the damping $[C]$ and the stiffness $[K]$ matrices are set constant and equal to the ones obtained for the steady problem (Eq. 4). Also, no structural damping $[C]$ was considered in this study and therefore with all these assumptions, the problem is considerably reduced to:

$$\left[\, s^2[M] - s[A_C] + [K] - [A_K]\right]\{\hat{q}\} = \{0\} \tag{8}$$

In this work, the equation above is solved iteratively using the p-k method in NASTRAN.

2.1.2.2.1. The p-k method

The p-k method emerged in the mid 1960's as an alternative to previous frequency domain flutter predicting methods, such as the k-method, but with better properties for aerospace problems [17, p. 120] . By avoiding the computational costs that characterize a time domain solution, it aims at generating an approximate flutter diagram as a solution to our problem. While explaining this method, we will employ the following non-dimensional expression for the nondimensional Laplace number *p*:

$$p = \frac{L}{U}(\sigma + i\omega) = \delta + ik \tag{9}$$

where $p = \left(\frac{L}{U}\right)s$, δ is the real part of the nondimensional Laplace parameter and the reduced frequency is $k = \omega\frac{L}{U}$. The dynamic equation of motion can then be written as:

$$\left[\frac{U^2}{L^2}[M]p^2 + [K]\right]\{\hat{q}\} = \frac{1}{2}\rho U^2[A(p)]\{\hat{q}\} \tag{10}$$

where $A(p)$ represents the matrices of aerodynamic stiffness and damping as a function of the Mach number M_∞ and the Reynolds number Re.

The p-k method is based on the assumption that we may approximate the aerodynamics of sinusoidal motions with slowly increasing or decreasing amplitudes using purely harmonic aerodynamic results. *i.e.*:

$$[A(p)] = [A(\delta + ik, M_\infty, Re, \dots)] \approx [A(k, M_\infty, Re, \dots)] \tag{11}$$

The full equations of motion are then used to determine the correct value of *p* for each vibration mode at a given flight condition.

With our assumed form of the aerodynamic forces, the equations of motion can be written as:

$$\left[\frac{U^2}{L^2}[M]p^2 + [K]\right]\{\hat{q}\} = \frac{1}{2}\rho U^2[A(k, M_\infty, Re, \dots)]\{\hat{q}\} \tag{12}$$

or

$$[F(p,k)]\{\hat{q}\} = \{0\} \tag{13}$$

This equation can be used to determine values of *p*. Normally, an iterative solution method is used, with guesses for *p* determined from the previously analyzed flight condition. The most common technique is known as "determinant iteration" [18].

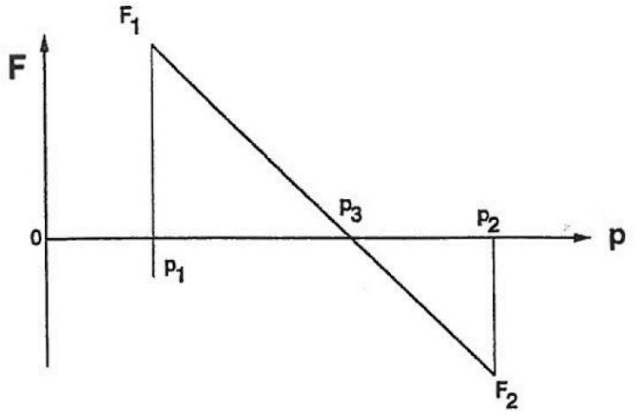


Figure 3 - The determinant iteration procedure *[17]*

The basic procedure is the following. First, the vibration mode which will be tracked is chosen, along with a starting value for U near $U = 0$. The Mach and Reynolds number are then computed for that U. A first guess, p_1, can be made using the natural frequency of the mode in the absence of aerodynamic forces. Another initial guess, p_2, can be made by adding a small value of damping to p_1. Then, the following iterative process is performed as described by Hulshof [17]:

1. Compute or interpolate for $[A(k_1, M_\infty, Re, \dots)]$ and $[A(k_2, M_\infty, Re, \dots)]$.
2. Compute the determinants $F_1 = |[F(p_1, k_1)]|$ and $F_2 = |[F(p_2, k_2)]|$.
3. Update *p* using: $p_3 = \frac{p_2 F_1 - p_1 F_2}{F_1 - F_2}$
4. Set $p_2 = p_3$ and $p_1 = p_2$, then repeat from step 1 until it converges.
5. Choose the next flight condition $(\rho, U, M_\infty, Re, \dots)$.
6. Choose two new values of *p* based on extrapolation from the previous flight conditions.
7. Return to step 1 and repeat to find the new *p* at the next flight condition.

(It is helpful to view steps 1-4 as an inner loop and steps 1-7 as an outer loop.)

Once the vibration mode is tracked through its desired range, a different mode is selected and the entire procedure is repeated. Note that the formula for p_3 in step 3 can be derived using the diagram shown in Figure 3. Also, only one determinant evaluation is required per iteration, since F_1 can be taken from the previous iteration. The following graphs help to understand what the results of the p-k method look like, and in the next chapters, they will be referred to as V-f and V-g diagrams.

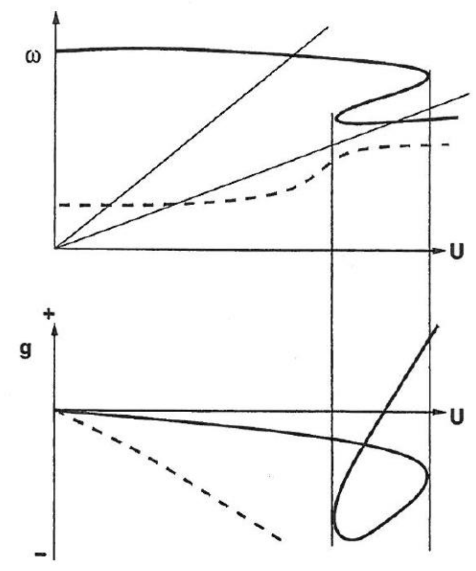


Figure 4 - Non-physical representation of p-k method diagrams [17]

For each tracked vibration mode, for each U value, we get the corresponding frequency and damping values that allow us to build these graphs. These are essential in determining for which conditions it exists flutter, *i.e.*. when there is a change in damping from negative to positive (or null) while the frequency is a positive value. In those conditions, a flutter point has been found and it is known its frequency, its speed and in which vibration mode it occurred. In addition, through such graphs, it is also possible to find the phenomenon of divergence, if there is a point where the damping changes from negative to positive (or null) but the frequency remains null.

In general, it is possible to say that the diagrams generated from the p-k method are more easily interpreted and can be closer to those of the actual system than those obtained via previously existing flutter predicting methods (like the k-method) [18]. They are still only exact, however, at the flutter point.

2.2. Aeroelastic Modeling

2.2.1. Structural Model

The wing model present in the IST Aerospace group aeroelastic framework is quite different than the one used in the frequency domain analysis tool described in this work. Their structural components, however, are both based on beam models constructed through the finite element method. Despite having different structural formulations (the aeroelastic framework employs a tridimensional beam model, while the frequency domain tool employs a bidimensional beam model), the results drawn from them will tendentially point in the same direction, since one of the degrees of freedom of the beam model employed by the aeroelastic framework is constrained. This wing/beam approximation is possible because we make the assumption that since we are dealing with high aspect-ratio wings, their length is many times longer than their thickness and width. Therefore, it is feasible to approximate such wings as beams, using classical beam theory.

The wings are assumed to be cantilevered at their root in both models, thus in the frequency domain analysis tool only half wing is considered. However, in the aeroelastic framework both halves are portrayed in its beam model, since the aerodynamic model implemented requires so.

There are two major beam theories: Euler-Bernoulli and Timoshenko. In the Euler-Bernoulli theory, it is assumed that plane cross sections perpendicular to the axis of the beam remain plain and perpendicular to the axis after deformation [19, p. 233]. On the other hand, in the Timoshenko theory, this normality assumption is not used, *i.e*., plane sections remain plane but not necessarily normal to the longitudinal axis after deformation [19, p. 261]. Therefore, shear deformation is taken into account. By analyzing the governing equations of both beam theories [19, p. 262], it is worth to notice that when the shear strain factor is zero (*i.e*. for long slender beams) on the Timoshenko theory governing equations, these become the Euler-Bernoulli theory ones.

Both the wing/beam models of the aeroelastic framework and of the frequency domain tool follow the Euler-Bernoulli beam theory and use the same model, which is built in the aeroelastic framework. This model is generated from a pre-defined wing-box where caps, webs and skin thickness are known as well as the material. Based on these structural parameters it is possible to compute the inertial properties (such as area and inertia moments) which will be transmitted to the frequency domain tool and used to generate a finite element model based on one-dimensional elements. Thus, the same structural model is considered in both computational approaches. The main difference is in how the analyses are conducted: in the aeroelastic framework a non-linear solver (Newton-Raphson) is used; while in the frequency domain tool a linear solver is employed. In the next section, the basics behind how the solver NASTRAN builds the structural model on the frequency domain model will be looked into.

2.2.1.1. Beam modeling in NASTRAN

NASTRAN is a Finite Element Analysis program, which means that it uses the finite element method (FEM) to model and solve problems. This finite element method is a numerical method that is quite powerful in its application to real-world problems that involve complicated physics and boundary conditions [19, p. 13]. The key concept of such method is that a given complex domain should be viewed as a collection of simpler subdomains, called finite elements, which are independent from each other within their respective domains [19, p. 13]. By doing so, it is possible to systematically construct the approximation functions needed to get the approximated solution of a problem over each of those finite elements. The idea behind this is that it is easier to represent a complex function by a collection of simpler ones, normally polynomials. These simpler functions, which are called interpolation functions, will govern each finite element in a way that the overall functions for the dependent variables of the problem are approximated by them and its unknown coefficients.

More specifically, regarding two-node beam elements such as the CBEAM elements used in this work, S. Raghu [20] is a great source to understand how NASTRAN represents the displacement of an element by means of the linear combination of shape functions. Once a finite element mesh is defined based on the geometry provided, these shape functions are developed for each element in regard to the overall governing equations. Finally, after reaching mesh convergence by increasing the number of finite elements to a desired point, the solution for the whole structure is then obtained from the assembly of the element solutions.

2.2.2. Aerodynamic Model

The purpose of the aerodynamic component of the models is to simulate the airflow and the aerodynamic surfaces as accurately as possible. In the aeroelastic framework, the aerodynamics were modeled through a 3D panel method with compressibility and viscosity corrections [15] [16]. Based on the established wing geometry (airfoil and wing planform dimensions) a mesh using rectangles (with several chordwise and spanwise divisions) is built and a convergence study is conducted before running the aeroelastic analysis. In the frequency domain tool, the best option available was to employ a simpler 2D panel method. This method, known as the Doublet-Lattice Method (DLM) [21] or the Doublet-Lattice Subsonic Lifting Surface Theory [21], was selected from the subsonic aerodynamic theories available in NASTRAN, since it was the one most adequate to the conditions of our problem [21].

2.2.2.1. Doublet-Lattice Method

The Doublet-Lattice Method is used for interfering lifting surfaces in subsonic flow. While its theory is presented in detail in [21] and [22], the following explanation will summarize the essential features of this method.

Like in other aerodynamic theories, the DLM requires three matrix equations that lead to the development of a matrix of aerodynamic influence coefficients (AIC) [23], which was previously referred to in the p-k method section as the matrix A. These must establish the relations between:

- Air downwash (normalwash) velocity at the surface of an element and lifting pressure.
- Deflection of the lifting element and downwash (substantial derivation matrix).
- Aerodynamic loads and lifting pressure (integration matrix).

The AIC matrix is then formed through a “combination” of these three matrices, making it possible to relate the aerodynamic forces acting on an element to the element deflection at the control points. We will be back to this topic after the following introduction to how a lifting surface is modeled by the DLM.

The theoretical basis of the DLM is a linearized aerodynamic potential fluid theory [21]. In the DLM, the undisturbed flow is uniform and its speed can be either steady or vary harmonically. All lifting surfaces are assumed to be parallel to the flow, and each of these surfaces (or panels) is divided chord-wise and span-wise into small trapezoidal lifting elements (or “boxes”), such that the boxes are arranged in strips parallel to the flow direction. This configuration is exemplified in Figure 5.

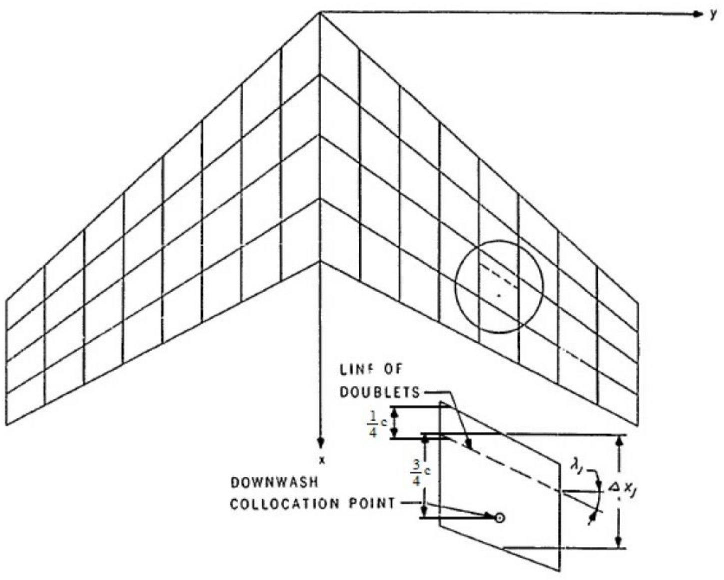


Figure 5 - Representation of a lifting surface model by the DLM [22]

Each box is considered to have an array of acceleration potential doublets of equal yet unknown strength along the ¼-chord line. The unknown lifting pressures are assumed to be concentrated uniformly across the ¼-chord line of each box. Through a matrix integral equation, the downwash resulting from the doublet lines of all boxes is related to the lifting pressure on each panel element. The substantial derivative in the stream-wise direction of the normal deflection gives the downwash boundary conditions [21]. To represent the deflection distribution, a series of mode shapes and generalized coordinates is used [22]. There is one control point per box, centered spanwise on the ¾-chord line of the box, and the downwash boundary conditions must be satisfied at each of these points. The last relationship is established by an integration matrix, which integrates the pressure distribution over the lifting surface to obtain the aerodynamic loads acting on each element. Having the three relationships established, the only thing left before calculating the AIC matrix is to obtain the loads acting on the element control points located at ¾-chord of each element, by chord-wise interpolation of the loads at ¼-chord along the strip.

According to NASTRAN's *Aeroelastic Analysis User's Guide* [23], there are some guidelines that should be followed when building an aerodynamic element mesh through the DLM method, in order to obtain accurate results. The ones considered relevant to our aerodynamic model are the following:

- no less than four boxes should be used per strip;
- the chord lengths of adjacent boxes should change gradually in the stream-wise direction;
- the chord length of the boxes should be less than 0.08 times the velocity divided by the greatest frequency (in Hz) of interest, *i.e.*, $\Delta x < 0.08 V_\infty / f$;
- the aspect ratio of the boxes should approximate unity;
- the mesh should be finer next to areas where large pressure gradients and downwash discontinuities occur (e.g. the wing edge).

The way how NASTRAN implements the DLM will be described further ahead.

2.3. NASTRAN

As introduced before, all aeroelastic analyses conducted in the frequency domain tool are performed by the solver NASTRAN. The purpose of this section is to provide insight on how the solver NASTRAN works, how the structural and aerodynamic models are built and connected through spline interpolation and how it employs the p-k method to perform the aeroelastic analysis. Also, it aims to provide context to the next chapter, where the frequency domain tool is thoroughly described.

The NASTRAN software is able to perform various specific tasks, such as processing model geometry, assembling matrices, applying constraints, solving matrix problems, and calculating output quantities. This software, also called a finite element "solver", takes an input file written by the user (or generated by another program), runs it and presents an output file (or several output files).

Each type of analysis available in NASTRAN is called a "solution sequence". Each solution sequence is a pre-defined collection of hundreds or thousands of DMAP commands. DMAP (Direct Matrix Abstraction Programming) is a high-level programming language with its own compiler and grammatical rules [24]. Once a solution sequence is selected, its particular set of DMAP commands sends instructions to the modules that are needed to perform the requested solution. One particularity of the NASTRAN software is that in each run it can perform various types of analyses on the same model. Each of these independent analyses is called a subcase.

2.3.1. NASTRAN's Files

All the information regarding the analyses requests, as well as the model data, is written on an input file (typically a .bdf file). NASTRAN then processes such input file and writes the analysis results in an output file (typically a .f06 file), to be later post-processed by the user.

As mentioned above, NASTRAN's input files are typically .bdf files written in ASCII. Each file is 80 characters (or columns) in length on each line [25] and it is divided into fields according to the data format. There are two basic categories of input data formats in NASTRAN:

- "Free" format data, where the data fields are simply separated by commas.
- "Fixed" format data, where the data must be aligned in columns of specific width. There are two subcategories of this format that differ based on the size of the fixed column width:
  - Small field format, where a single line of data is divided into 10 fields of 8 characters each.
  - Large field format, where a single line of input is expanded into two lines for greater numerical accuracy. The first and last fields on each line are 8 characters wide, while the intermediate fields are 16 columns wide.

In this work, the .bdf input files generated by the frequency domain tool will be written according to the free field format.

Each NASTRAN input file has three mandatory sections: Executive Control, Case Control and Bulk Data. These sections will be followingly described, along with commands and entries used in the present work.

2.3.1.1. Executive Control Section

The Executive Control section is where the solution sequences to be run by NASTRAN are specified, along with other solution parameters. The solution sequence for aerodynamic flutter analysis is known to NASTRAN as “SOL 145” and it will be the only one that is entered in the Executive Control section in this work. The last entry of this section is the CEND statement, which is a required statement that designates the end of the Executive Control section and the beginning of the Case Control section.

2.3.1.2. Case Control Section

The Case Control Section contains commands that are used to specify and control the analyses and their output. This is where a title or a label can be assigned to the routine, as well as where constraints and loads that are used in the Bulk Data section must be declared first, with their corresponding entry numbers. In addition, it is where analysis subcases and sets are defined. Also, the output requests for printing and plotting are selected here. In this work, this section is where the set identification numbers of the flutter analysis method (FMETHOD entry) and of the eigenvalue extraction method (METHOD entry) are declared. The V-g and V-f plot output is defined in this section as well. Lastly, the Case Control section must end with the BEGIN BULK delimiter.

2.3.1.3. Bulk Data Section

The Bulk Data section contains the entries that define the model geometry, its material properties, element connectivity and properties, loads, boundary conditions and some analysis parameters. As mentioned above, it begins with the BEGIN BULK entry.

2.3.1.3.1. Structure

In the part of the Bulk Data section that corresponds to the structural model, the material properties, the geometry of the structure and the boundary conditions are all defined. Through the MAT1 command, the properties of an isotropic material can be defined for the structure. The base on which the geometry is built are the so called grid points. The command GRID defines the location of a geometric grid point, the directions of its displacement, and its single-point constraints. They can be set in the general coordinate system or in user-defined coordinate systems and they are used to indicate the position of element nodes. As mentioned before, the structural elements used in this work are one dimensional beam elements. Each beam element is defined through the CBEAM entry and its properties through the PBEAM entry. With the CBEAM command, the grid points that correspond to the two nodes of each beam element are identified, as well as the orientation vector that displays how the beam element planes are placed (see Figure 6).

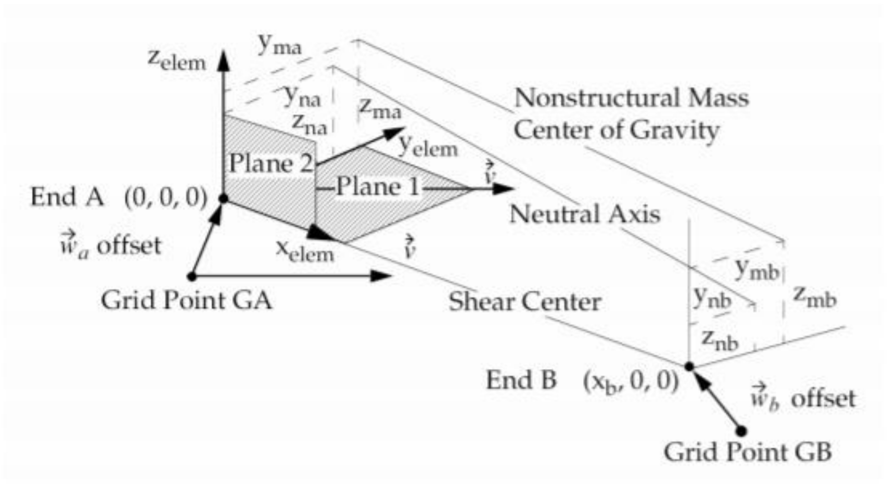


Figure 6 - CBEAM element coordinate systems [26]

The components of this vector, specified on fields X1, X2 and X3 of the CBEAM entry, must be chosen while having the desired orientation of the element coordinate system in mind. It is important to keep in mind the orientation of the element coordinate system, as many of the beam properties specified in the PBEAM entry are defined in respect to this system. As just mentioned, the PBEAM command defines the properties of a beam element, such as its material, cross section areas, area moments of inertia, cross products of inertia and torsional stiffnesses. Also, in order to be working with the Euler-Bernoulli beam theory, it is possible to neglect shear deformation by setting the shear stiffness factors to zero.

2.3.1.3.2. Aerodynamics

As introduced before, in this work, the aerodynamic surfaces are modeled through discretization in aerodynamic finite elements based on the Doublet-Lattice Method. Each surface has an aerodynamic grid point, which does not need to coincide with the grid points of the structural model. In addition, the configuration of the aerodynamic element mesh is independent from that of the structural mesh, which is quite convenient, since one model may require a finer mesh on a certain area where such degree of refinement is not required on the other one. Unlike the beam elements, which have to be declared individually (*i.e.* each CBEAM command represents a single beam element), the lifting panel elements are declared in a collective way, by defining macro elements. Resorting to the CAERO1 command, it is possible to define an aerodynamic macro element (panel) in terms of two leading edge locations and side chords.

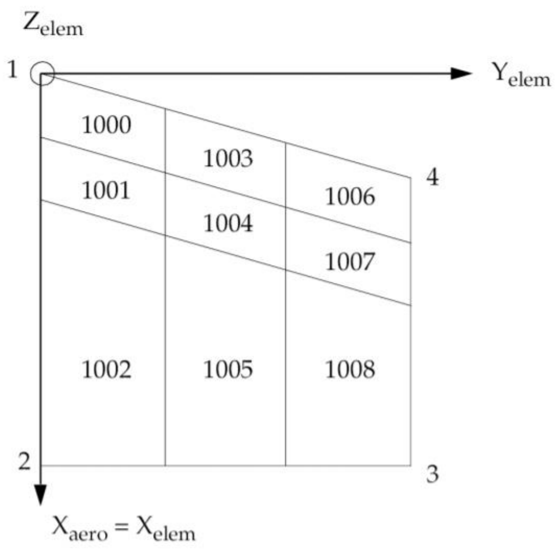


Figure 7 - Coordinate system of a CAERO1 panel [26]

As it will be addressed further ahead in this work, in order so that the structural/aerodynamic interaction meets certain criteria, each aerodynamic panel is “covered” by a different spline and is defined between two nodes of the structural model. Likewise, the number of spanwise and chordwise boxes per panel are also defined in order to meet such criteria. The command PAERO1 defines an aerodynamic property for the present panels.

2.3.1.3.3. Splines

The interaction between the structural model and the aerodynamics model results from the definition of the so-called splines. A spline is an interpolation method by which it is possible to reflect the aerodynamic forces on the structure and the structural motion on the aerodynamic mesh. As the aerodynamic mesh moves along with the structural mesh, the structural deflections are transferred to the aerodynamic deflections and the aerodynamic forces are considered in the structural equivalent forces acting on the structural grid points. This is fundamental to the analysis of a dynamic aeroelastic phenomenon such as flutter, since it ensures the coupling of the inertial, elastic and aerodynamic forces of the model. The splining methods lead to an interpolation matrix $[G_{kg}]$ that relates the components of structural grid point deflections $\{u_g\}$ to the deflections of the aerodynamic grid points $\{u_k\}$:

$$\{u_k\} = [G_{kg}]\{u_g\} \tag{14}$$

To interpolate the forces transformation, the two force systems must be “structurally equivalent” rather than statistically equivalent. This means that the two force systems deflect the structure equally, rather than having the same resultant loads. Therefore, the aerodynamic forces $\{F_k\}$ and their structural equivalent values $\{F_g\}$ do the same virtual work in their respective deflection modes:

$$\{\delta u_k\}^T\{F_k\} = \{\delta u_g\}^T\{F_g\} \tag{15}$$

where $\{\delta u_k\}$ and $\{\delta u_g\}$ are virtual deflections. By substituting Eq. 14 into the left-hand side of Eq. 15 and rearranging, the required force transformation is obtained because of the arbitrariness of the virtual deflections:

$$\{F_g\} = [G_{kg}]^T\{F_k\} \tag{16}$$

Eq. 14 and Eq. 16 are both required to complete the formulation of the aeroelastic problems in which the aerodynamic and structural grid points do not coincide. The splining theories used to calculate the above-mentioned interpolation matrix are different for each type of spline and are explained in detail in the *NASTRAN Aeroelastic Analysis User's Guide* [23]. For high aspect ratio wings or beam-like structures like the ones analyzed in this work, linear splines are the most appropriate. A linear spline is a "beam" function *w(x)* which is derived using the known deflections and twist angles at the structural grid points as boundary conditions. This function is then used to obtain the deflections and twist angles at the aerodynamic grid points. Any aerodynamic panel or body can be subdivided into subregions for interpolation, using a separate function for each. In this work, every spline is created through the command SPLINE2, which defines a beam spline for interpolating motion and forces for aeroelastic problems. When a spline is defined, the user must declare the structural and the aerodynamic grid points involved in the interpolation, along with specifying the degrees of freedom that are to be interpolated. The command CORD2R defines a rectangular coordinate system for each spline, where the y-axis of each system defines the axis of each spline. The command SET1 defines a set of grid points for each spline that indicate which structural grid points are to be interpolated for each spline.

2.3.1.3.4. Flutter solution

As introduced before, the p-k method treats the aerodynamic matrices as real frequency dependent springs and dampers. A frequency is estimated, and the eigenvalues are found. From an eigenvalue, a new frequency is found and the convergence to a consistent root is rapid.

One issue with the aerodynamic matrices computed by interaction theories such as the Doublet-Lattice Method is that they are quite expensive to generate. An effective method to evaluate these matrices for a large number of parameter values is to compute the matrices for a few selected values and to interpolate to the remaining values. This parametric interpolation is an automatic feature of the solution modules for aeroelastic analysis. It is also important to notice that these complex influence coefficient matrices can be defined as depending upon two parameters of the flow: reduced frequency (dimensionless ratio of frequency to velocity) and Mach number (ratio of velocity to speed of sound). Therefore, instead of building the aerodynamics matrices for every reduced frequency used in the p-k method iterations, NASTRAN only does so for a user-defined list of Mach numbers and reduced frequencies. The MKAERO1 entry specifies the Mach number and reduced frequencies for which the generalized aerodynamic forces are computed. Therefore, this entry can be a key factor for the accuracy of the flutter results. It is important to provide enough reduced frequency values, which cover a broad enough range. The reason why is that if during the flutter analysis any of the reduced frequency values is outside this range, the aerodynamic matrices will be obtained from extrapolation, which will likely produce inaccurate results. Sources in the MSC NASTRAN Users Community Forum have advised in using values not lower than 0.001 and not higher than 8.0. According to these, such a low value is necessary in order to accurately follow the development of torsional divergence by any mode. The highest value should be estimated by multiplying the highest expected frequency by the reference length and dividing it by the lowest speed considered in the analysis. Through a study that was conducted, a minimum value of 0.001 and a maximum value of 7.0 were proven to be sufficient for all the analyzed cases. In this work, the analyses of all models were performed using the same reduced frequency discretization in the MKAERO entry, ranging from 0.001 to 7.0.

The AERO entry specifies a reference air density and a reference length to be used with the reduced frequency. The EIGRL command defines the parameters required to perform the necessary real eigenvalue analysis. With the modal analysis performed and the splining and aerodynamic matrices computed, the flutter analysis can be performed. Once the generalized modal matrices are generated and the splining and aerodynamics data is processed, all the computed matrices are assembled to form the flutter problem equations. For the p-k method, these equations are solved for each user-defined combination of air density, Mach number and air speed. As mentioned before, during the p-k method iterative loops, the aerodynamic matrices for each reduced frequency are obtained through an interpolation of the previously computed matrices. Through the FLUTTER entry, it is both chosen the p-k method as the flutter analysis method, and also the aerodynamic parameters for which the flutter problem will be solved, namely: air density, Mach number and air velocity range. The concrete values of such aerodynamic parameters are defined in the FLFACT entry. The PARAM entry is used to specify values for parameters used in solution sequences. For example, the parameter LMODES indicates the number of vibration modes to be requested in modal formulation.

Lastly, the Bulk Data section ends with the ENDDATA command.

3. The Frequency Domain Analysis Tool

In this chapter, we will describe the tool that was developed in MATLAB for the frequency domain analysis.

All the input wing configuration data we have for the frequency domain analysis comes from the Non-linear Aeroelastic Framework. This computational tool can run both steady and unsteady simulations resorting to steady and unsteady FSI solvers. For this work, the unsteady simulations are only used for comparison purposes with the updated frequency domain methodology; while the steady simulations are carried out to provide a position of stationary equilibrium, where the structural and the aerodynamic meshes converge, to the updated frequency domain method. These converged positions, for different wind speeds, different angles of attack at different altitudes, for the different aspect-ratio wings, will provide the several deformed wing configurations that will constitute the file database to be used as input in the frequency domain analysis tool.

So, the MATLAB code would have to be able to receive as inputs the deformed wing configurations generated by the FSI Solver and its inertial properties, and then build the structural and aerodynamic components of the wing models on a .bdf input file to be analyzed with NASTRAN.

This NASTRAN analysis calculates a flutter speed for each air speed and its correspondent deformed structure configuration. Finally, the program proceeds to find the actual flutter speed for the aspect ratio, angle of attack and altitude combination in focus, by running this analysis in an iterative process to try to converge the wind and flutter speeds. This way, it is possible to use a frequency domain approach to find the minimum flutter speed for a certain wing configuration, while taking into account its deformation due to the airspeed.

More specifically, the developed MATLAB tool is constituted by 5 functions (Figure 8). The main function, flutter.m, (Figure 9) runs the main iterative calculation routine, which in each cycle resorts to the flutter_SOL145.m function to find the flutter speed for a specific input airspeed and matching wing configuration. The main function also contains the flutter_param.m function, which has some analysis parameters, as well as the interpolate_deflections.m function (Figure 10), which in each cycle interpolates a new deformed wing configuration for the required input airspeed. Then, in each cycle run, the flutter_SOL145.m (Figure 11) function runs the NASTRAN flutter analysis for a specific input airspeed and corresponding deformed wing configuration. It writes the Nastran input file, runs NASTRAN, reads the result file, plots V-G and V-F graphs, analyzes the results and returns the estimated flutter speeds, as well as the corresponding frequencies and vibration modes. It also resorts to the beamwriter.m function (Figure 12) to build both the structural and the aerodynamic components of the beam model that will simulate the required wing configuration.

While the following subchapters will explain in more detail the above mentioned functions, the diagram and flowcharts below aim to help understanding the structure of the frequency domain analysis tool. An overview of this updated frequency domain tool is depicted in Figure 8, while in Figures 9 to 12 each main function of this tool is detailed.

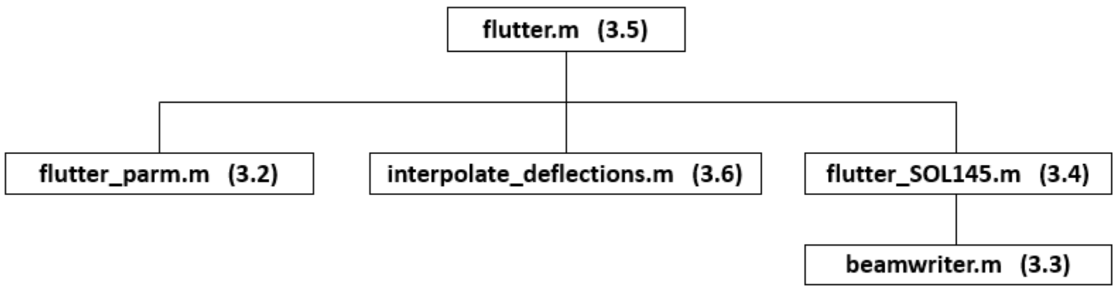


Figure 8 - Function structure of the Tool

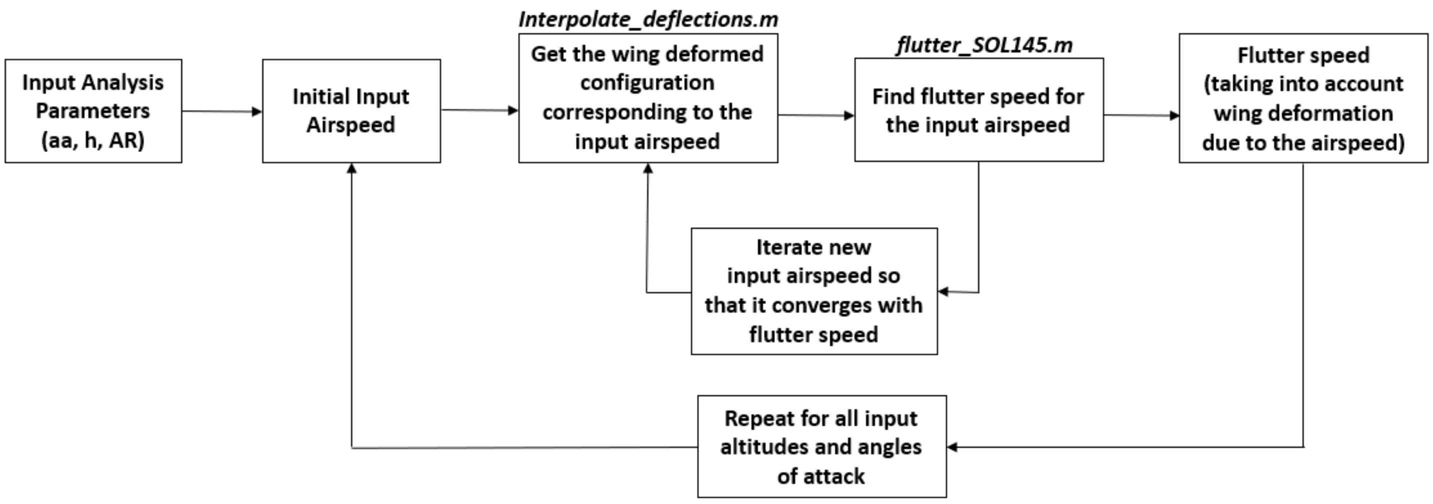

Input Analysis Parameters (aa, h, AR)
Initial Input Airspeed
Interpolate_deflections.m
Get the wing deformed configuration corresponding to the input airspeed
flutter_SOL145.m
Find flutter speed for the input airspeed
Flutter speed (taking into account wing deformation due to the airspeed)
Iterate new input airspeed so that it converges with flutter speed
Repeat for all input altitudes and angles of attack

Figure 9 - flutter.m function flowchart

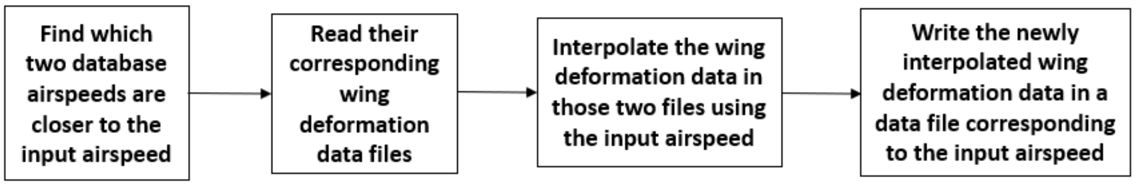

Find which two database airspeeds are closer to the input airspeed
Read their corresponding wing deformation data files
Interpolate the wing deformation data in those two files using the input airspeed
Write the newly interpolated wing deformation data in a data file corresponding to the input airspeed

Figure 10 - interpolate_deflections.m function flowchart

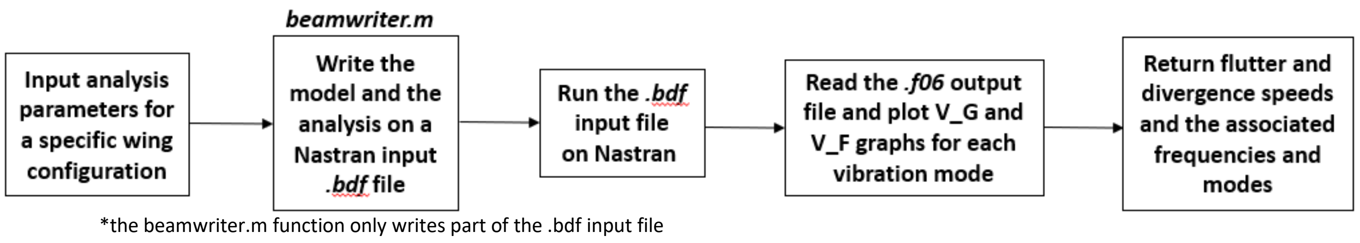

beamwriter.m
Input analysis parameters for a specific wing configuration
Write the model and the analysis on a Nastran input .bdf file
Run the .bdf input file on Nastran
Read the .f06 output file and plot V_G and V_F graphs for each vibration mode
Return flutter and divergence speeds and the associated frequencies and modes
*the beamwriter.m function only writes part of the .bdf input file

Figure 11 - flutter_SOL145.m function flowchart

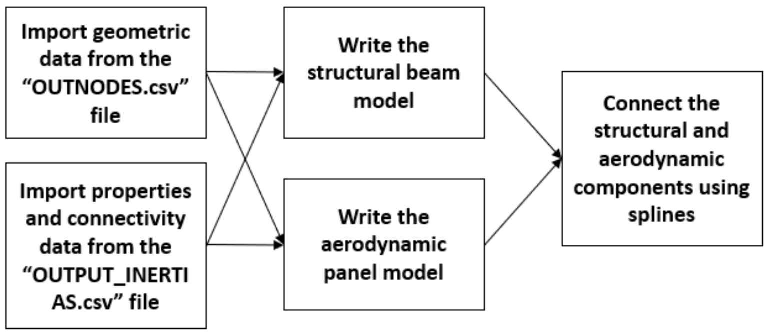

Import geometric data from the "OUTNODES.csv" file
Import properties and connectivity data from the "OUTPUT_INERTIAS.csv" file
Write the structural beam model
Write the aerodynamic panel model
Connect the structural and aerodynamic components using splines

Figure 12 - beamwriter.m function flowchart

3.1. Input Data

The input data we will base our model on comes in the form of two files generated by the Non-linear Aeroelastic Framework, which contain geometrical and inertial data of different converged wing configurations subject to different analysis parameters.

These inputs from the Non-linear Aeroelastic Framework are the OUTNODES.csv file and the OUTPUT_INERTIAS.csv file.

The OUTNODES.csv file contains the wing geometrical data. Its rows represent the different nodes and its columns present the following relevant data:

- Column 1: Node ID
- Column 2: Chordwise direction node coordinate component – *x*
- Column 3: Spanwise direction node coordinate component– *y*
- Column 4: Vertical direction node coordinate component – *z*
- Column 5: Leading edge coordinate component at the node location – *x_LE*
- Column 11: Chord length at the node location – *c*
- Column 12: Chordwise direction nodal displacement – *ux*
- Column 13: Spanwise direction nodal displacement – *uy*

- Column 14: Vertical direction nodal displacement – *uz*
- Column 15: Chordwise direction nodal rotation – *theta_x*
- Column 16: Spanwise direction nodal rotation – *theta_y*
- Column 17: Vertical direction nodal rotation – *theta_z*

The OUTPUT_INERTIAS.csv file contains wing properties and inertial data. Its rows represent the different elements and its columns present the following relevant data:

- Columns 1 and 2: Connectivity Matrix (pair of nodes that correspond to the finite element) – *N1* and *N2*
- Column 3: Area of the beam element cross section – *A*
- Column 4: Area moment of inertia for bending about the chordwise axis - *Ixx*
- Column 5: Area moment of inertia for bending about the vertical axis - *Izz*
- Column 6:Torsional stiffness - *Jt*
- Column 8: Angle used to rotate each element vector from the global referential to the element principal referential of inertia - *theta_e*

It is important to notice that in the steady FSI analysis, while parameters like the AR, the altitude and the angle of attack obviously don't change as a function of the air speed, the deformed configuration changes, *i.e.* the higher the airspeed, the greater deformation is observed. Therefore, as inputs for the developed tool, for the same AR, altitude and angle of attack, but different airspeeds, there will be a single "OUTPUT_INERTIAS.csv" input file, but different "OUTNODES.csv" input files. Each one representing the deformations of the same wing configuration due to a different airspeed. And unlike the "OUTNODES.csv" data, the "OUTPUT_INERTIAS.csv" data is independent from the wing deflection.

3.2. The Flutter Parameters Function

This is a simple function where we define some analysis parameters to be used in the following functions.

The header of the *flutter_parm* function is:

```
function [title_prop, label_files, MAT, ref_chord, y_mandat_breakpt, ...
        panels, altitude_data, aa_data, V_data,...
        branch_nodes] = flutter_parm ()
```

and its inputs are the following:

- `title_prop` will return the name of the input file with the wing inertial data;
- `label_files` will return the input wing Aspect Ratio;
- `MAT` will return the properties of an isotropic material (Young's modulus, shear modulus and density) to be used in the wing beam model;
- `ref_chord` will return the reference chord;

- `y_mandat_breakpt` will return the number of mandatory break points for the splines;
- `panels` will return a vector that will influence the number of spanwise and chordwise panel boxes;
- `altitude_data` will return a vector with the altitudes for which we have wing configuration data in our database;
- `aa_data` will return a vector with the angles of attack for which we have wing configuration data in our database;
- `V_data` will return a matrix with vectors that represent the airspeeds for which we have wing configuration data in our database, for the respective angle of attack and altitude combinations. As it is possible to see in the example below, the rows of the matrix represent the different angles of attack and the columns represent the altitudes for which we have available airspeed vectors, *i.e.*. available deformed wing configurations in our database.
- `branch_nodes` will return the option of excluding some nodes from the breakpoint writing process.

An illustrative example of these inputs is presented below.

```
% Connectivity data with element properties
title_prop = 'OUTPUT_INERTIAS.csv';

label_files = 'AR12';

% [E, G, rho]
MAT = [70E9,27E9,2700];

%reference chord
ref_chord = 1.2910;

% breakpoint
y_mandat_breakpt = 2;

% aerodynamic mesh
panels = [80,8];

% branch nodes
branch_nodes = NaN;

% Available flight conditions

%                1     2    3    4    5    6    7       8
altitude_data = 0;

%          1  2  3 4 5 6 7 8
aa_data = [-4,-2,0,2,4,6,8,10];

V_data(:,1) = {[10,40,60,80,100,120,140,160,180],...  %-4
    [10,40,60,80,100,120,140,160,180],...         %-2
    [10,40,60,80,100,120,140,160,180],...
    [10,40,60,80,100,120,140,160,180],...         %2
    [10,40,60,80,100,120,140,160,180],...
    [10,40,60,80,100,120,140,160,180],...         %6
    [10,40,60,80,100,120,140,160,180],...
    [10,40,60,80,100,120,140,160,180]};           %10
```

3.3. The Beamwriter Function

The first goal was to build a model in NASTRAN that would represent the wing´s structure and the aerodynamics surrounding it. As mentioned before, our structural model is based on the Euler-Bernoulli Beam Theory and the aerodynamic model is grounded on the Doublet-Lattice Panel Method.

The *Beamwriter* function will import geometric and inertial data from the files "OUTNODES.csv" and "OUTPUT_INERTIAS.csv" generated by the non-linear aeroelastic framework, and will write the beam model in a .bdf input file to be read by the NASTRAN solver.

Here is a general overview of this function by steps:

1. Import wing deflection data and change references.
2. Import wing connectivity and property data and rearrange it.
3. Write *GRID* points that represent the beam nodes.
4. BEAM CYCLE:
    - Write *CBEAM* beam elements
    - Write *PBEAM* beam properties
5. Write break points for the splines
6. AERODYNAMIC/SPLINE CYCLE:
    - Write points for the splines' coordinate systems
    - Write *CAERO1* aerodynamic panel elements
    - Write *CORD2R* coordinate systems for the splines
    - Write *SPLINE2* beam splines
    - Write splines' *SET1* node sets
7. Write *PAERO1* aerodynamic panel property

These steps are presented next in more detail, starting with the function header and its inputs, which come from the *flutter_SOL145.m* function.

Some stress and mass calculation options available will be ignored in the following description, since they ended up not being used in our analysis.

The *Beamwriter* function header is:

```
function
flutter_Beam_Writer4(title_prop,fout,title_deform,aa,panels,mandat_breakpt,n
_g,branch_nodes,PM_data,undeformed)
% Writes deformed beam data in title_out, including GRID, CBEAM, PBEAM,
% CAERO1, SPLINE2, CORD2R, PAERO1 and SET1 entries.
```

The inputs of this function are the following:

- `title prop` represents the OUTPUT_INERTIAS.csv input file;

- `fout` represents the title_out.bdf output file;

- `title_deform` represents the OUTNODES.csv input file;
- `aa` represents an input angle of attack;
- `panels` represents a vector that will influence the number of spanwise and chordwise panel boxes;
- `mandat_breakpt` represents the option of having mandatory break points for the splines.
- `n_g` represents a scale factor (this option is to set to 1 in this work, since no scaling methodology is employed in this work);
- `branch_nodes` represents the option of excluding some nodes from the breakpoint writing process;
- `PM_data` represents the option of adding mass points;
- `Undeformed` indicates whether wing deformations are being analyzed;

So, first, we define some parameters/variables, like the structure in which the *.bdf* output file will be written, and some inputs related with the writing of the aerodynamic panels and the beam splines.

Then, we import the deflection data contained in the "OUTNODES.csv" file (title_deform variable). More concretely, for each beam node that will constitute our model, we have its position and deflection information, as well as the leading edge and chord data for the wing section corresponding to the nodes. We also change the wing reference. The code implementation is the following:

```
% Import deflection data and change reference
data_raw = csvread(title_deform);

Nnodes = length(data_raw(:,1));

for i=1:Nnodes

    if abs(data_raw(i,3))<1E-10

        root_index = i;
    end
end

origin = data_raw(root_index,2:4);

data_def = data_raw;

for i=1:Nnodes

    data_def(i,2) = data_raw(i,2) - origin(1);
    data_def(i,4) = -data_raw(i,4) + origin(3);
    data_def(i,5) = data_raw(i,5) - origin(1);
end

x = data_def(:,2)*n_g;
y = data_def(:,3)*n_g;
z = data_def(:,4)*n_g;
c = data_def(:,11)*n_g;

if undeformed

    ux = zeros(Nnodes,1);
    uy = zeros(Nnodes,1);
    uz = zeros(Nnodes,1);
    theta_x = zeros(Nnodes,1);
    theta_y = zeros(Nnodes,1);
    theta_z = zeros(Nnodes,1);
else

    ux = data_def(:,12)*n_g;
    uy = data_def(:,13)*n_g;
    uz = -data_def(:,14)*n_g;  %-
    theta_x = -data_def(:,15); %-
    theta_y = -data_def(:,16); %-
    theta_z = data_def(:,17);
end
```

Next, we import the beam properties and the connectivity data from the OUTPUT_INERTIAS.csv file (variable title_prop) and store them in the corresponding variables, as follows:

```
% Import connectivity and property data and rearrange it

data_prop= csvread(title_prop);

N1 = data_prop(:,1);
N2 = data_prop(:,2);
A = data_prop(:,3)*n_g^2;
Ixx = data_prop(:,4)*n_g^4;
Izz = data_prop(:,5)*n_g^4;
Jt = data_prop(:,6)*n_g^4;
theta_e = -data_prop(:,8);
```

This way, we know which nodes constitute which beam elements as well as the properties of each element.

After that, using the *GRID* command we write the grid points and its coordinates corresponding to the beam nodes locations on the *.bdf* output file. It is possible to notice that while the input file contains data from both wings, since the structural model we are building is a cantilever beam that represents a wing attached at the root, we will only work with the “positive y” half of the nodes, that represent half the wing span. Therefore, we must not forget to constrain all degrees of freedom of the grid point that represents the wing root.

```
for i = 1:Nnodes

    if ~(y(i) < 0) || abs(y(i)) < 1E-10

        if large_field_struct

            fprintf(fout,'GRID*,%i,,%3.9E,%3.13f,*\n*,%3.13f',...
                        i,x(i)+ux(i),y(i)+uy(i),z(i)+uz(i));
        else

            fprintf(fout,'GRID,%i,,%3.1E,%3.5f,%3.5f',...
                        i,x(i)+ux(i),y(i)+uy(i),z(i)+uz(i));
        end

        if i == root_index;

            fprintf(fout,',,123456');
        end


        fprintf(fout,'\n');
    end
end

fprintf(fout,'\n');
```

After laying out the grid points where the elements would be, we can then write the *CBEAM* beam elements. In addition to the nodes of the beam elements, we have to define the components of its orientation vectors. This is done by using the angle *theta_e* to rotate each element vector from the global referential to the element principal referential of inertia.

```
for i=1:Nnodes-1

    if ~((y(N1(i)) < 0 && abs(y(N1(i))) > 1E-10) || ...
            (y(N2(i)) < 0 && abs(y(N2(i))) > 1E-10))

        % components of vector N1->N2 (basic coordinates)
        u = x(N2(i))+ux(N2(i))-x(N1(i))-ux(N1(i));
        v = y(N2(i))+uy(N2(i))-y(N1(i))-uy(N1(i));
        w = z(N2(i))+uz(N2(i))-z(N1(i))-uz(N1(i));

        mod = sqrt(u^2+v^2+w^2);
        u = u/mod;
        v = v/mod;
        w = w/mod;

        % components of orientation vector, after sweep
        v_x1 = -v;
        v_y1 = u;
        v_z1 = 0;

        % rotate orientation vector about element axis using theta_e
        v_x2 = u*(u*v_x1+v*v_y1+w*v_z1)*(1-cos(theta_e(i)))+...
            v_x1*cos(theta_e(i))+(-w*v_y1+v*v_z1)*sin(theta_e(i));

        v_y2 = v*(u*v_x1+v*v_y1+w*v_z1)*(1-cos(theta_e(i)))+...
            v_y1*cos(theta_e(i))+(w*v_x1-u*v_z1)*sin(theta_e(i));

        v_z2 = w*(u*v_x1+v*v_y1+w*v_z1)*(1-cos(theta_e(i)))+...
            v_z1*cos(theta_e(i))+(-v*v_x1+u*v_y1)*sin(theta_e(i));

        fprintf(fout,'CBEAM,%i,%i,%i,%i,%4.5f,%4.5f,%4.5f,GGG\n',...
            100+i-1,2000+i-1,N1(i),N2(i),v_x2,v_y2,v_z2);
```

These CBEAM elements need then to have properties assigned. Basing ourselves on the input data, using the PBEAM command, we define those properties for all beam elements: material, cross section areas at both beam ends, area moments of inertia for bending on both planes at both beam ends, cross products of inertia at both beam ends, and torsional stiffnesses. We also choose to neglect shear deformation (*i.e.* to obtain the Euler-Bernoulli beam theory) by setting the shear stiffness factors to 0.0.

```
        % Calculation of the torsion component of the rotational
        % deformations, that is, the component of the resultant rotational
        % deformation along the beam element direction:

        % cos(beta) = V_AB.theta/(/V_AB./theta)
        % theta_T = /theta*cos(beta) = V_AB.theta/ /V_AB

        % Where beta is the angle between the resultant (theta) and the
        % element, and theta_T is the torsion angle.

        XA = x(N1(i)) + ux(N1(i));
        XB = x(N2(i)) + ux(N2(i));

        YA = y(N1(i)) + uy(N1(i));
        YB = y(N2(i)) + uy(N2(i));

        ZA = z(N1(i)) + uz(N1(i));
        ZB = z(N2(i)) + uz(N2(i));

        V_AB = [XB-XA, YB-YA, ZB-ZA];

        theta_A = [theta_x(N1(i)), theta_y(N1(i)), theta_z(N1(i))];
        theta_B = [theta_x(N2(i)), theta_y(N2(i)), theta_z(N2(i))];
        theta_T_A = dot(V_AB,theta_A)/norm(V_AB);
        theta_T_B = dot(V_AB,theta_B)/norm(V_AB);

        if large_field_struct

            fprintf(fout,'PBEAM*,%i,1000,%6.10E,%6.10E,*\n',...
                2000+i-1,A(i),...
                Izz(i)*cos(theta_T_A)^2+Ixx(i)*sin(theta_T_A)^2);
            fprintf(fout,'*,%6.10E,%7.9E,%6.10E,,*\n',...
                Izz(i)*sin(theta_T_A)^2+Ixx(i)*cos(theta_T_A)^2,...
                -0.5*sin(2*theta_T_A)*(Izz(i)-Ixx(i)),Jt(i));

            if stress

                SO = 'YES';
                fprintf(fout,'*,%7.9E,%7.9E,,,*\n',stress_y(N1(i)),...
                    stress_z(N1(i)));
                fprintf(fout,'*,,,,,*\n');
            else

                SO = 'NO';
            end

            fprintf(fout,'*,%s,1.,%7.9E,%6.10E,*\n',SO,A(i),...
                Izz(i)*cos(theta_T_B)^2+Ixx(i)*sin(theta_T_B)^2);
            fprintf(fout,'*,%6.10E,%7.9E,%6.10E,,+\n',...
                Izz(i)*sin(theta_T_B)^2+Ixx(i)*cos(theta_T_B)^2,...
                -0.5*sin(2*theta_T_B)*(Izz(i)-Ixx(i)),Jt(i));

            if stress

                fprintf(fout,'*,%7.9E,%7.9E,,,*\n',stress_y(N2(i)),...
                    stress_z(N2(i)));
                fprintf(fout,'*,,,,,*\n');
            end

        else

            fprintf(fout,'PBEAM,%i,1000,%6.2E,%6.2E,',...
                2000+i-1,A(i),...
                Izz(i)*cos(theta_T_A)^2+Ixx(i)*sin(theta_T_A)^2);
            fprintf(fout,'%6.2E,%7.1E,%6.2E,,+\n',...
                Izz(i)*sin(theta_T_A)^2+Ixx(i)*cos(theta_T_A)^2,...
                -0.5*sin(2*theta_T_A)*(Izz(i)-Ixx(i)),Jt(i));

            if stress

                SO = 'YES';
                fprintf(fout,'+,%7.1E,%7.1E,,,',stress_y(N1(i)),...
                    stress_z(N1(i)));
                fprintf(fout,',,,,+\n');
            else

                SO = 'NO';
            end

            fprintf(fout,'+,%s,1.,%7.1E,%6.2E,',SO,A(i),...
                Izz(i)*cos(theta_T_B)^2+Ixx(i)*sin(theta_T_B)^2);
            fprintf(fout,'%6.2E,%7.1E,%6.2E,,+\n',...
                Izz(i)*sin(theta_T_B)^2+Ixx(i)*cos(theta_T_B)^2,...
                -0.5*sin(2*theta_T_B)*(Izz(i)-Ixx(i)),Jt(i));

            if stress

                fprintf(fout,'+,%7.1E,%7.1E,,,',stress_y(N2(i)),...
                    stress_z(N2(i)));
                fprintf(fout,',,,,+\n');
            end
        end

        fprintf(fout,'+,0.0,0.0\n');
    end
end

fprintf(fout,'\n');
```

The following part of the *beamwriter* program is the writing of the aerodynamic panels and the splines. As introduced in the previous chapter, these splines are the tool that will allow us to connect the structural part to the aerodynamic component of our model.

First, we will define the breakpoints corresponding to the nodes between which the splines will be defined. The purpose of this definition is to make sure that the aerodynamic surface covered by each spline meets the following criteria:

- The difference between *theta x* of the beginning and ending node of each spline doesn't exceed a certain predefined value, so that the aerodynamic macro element covered by the spline isn't too "curved".
- The *AR* of the surface covered between two breakpoints isn't greater than 1.
- The first and last break points are the root and the tip, respectively.
- The last *CAERO1* element shouldn't be so narrow that it would include only a single column of panels. In such case, the corresponding surface area should be assigned to the previous *CAERO1* element instead.

```
% Write coordinate systems needed for splines.

% First, as before, exclude any nodes that don't belong to the elastic axis or to the
% "positive y" wing.

if isnan(branch_nodes)

    data_def_sorted = sortrows(data_def,3);
else

    data_def_clean = data_def;

    branch_nodes = sortrows(branch_nodes',1);

    for i=1:length(branch_nodes)

        data_def_clean(branch_nodes(i)-(i-1),:) = [];
    end

    data_def_sorted = sortrows(data_def_clean,3);
end

root_index = 1;

while(abs(data_def_sorted(root_index,3)) > 1E-10)

    root_index = root_index + 1;
end

data_def_sorted_positive_y = data_def_sorted(root_index:end,:);

node_num = data_def_sorted_positive_y(:,1);
xs = data_def_sorted_positive_y(:,2)*n_g;
ys = data_def_sorted_positive_y(:,3)*n_g;
zs = data_def_sorted_positive_y(:,4)*n_g;
x_LEs = data_def_sorted_positive_y(:,5)*n_g;
cs = data_def_sorted_positive_y(:,11)*n_g;
uxs = data_def_sorted_positive_y(:,12)*n_g;
uys = data_def_sorted_positive_y(:,13)*n_g;
uzs = -data_def_sorted_positive_y(:,14)*n_g; %-

% Then, find span points breakpt(i) such that
% thetax_breakpt(i+1)-thetax_breakpt(i) doesn't exceed a certain value,
% delta_tan.

Nbpts = 1;
```

```
breakpt(1) = 1;
last_bpt = 1;

Npositive_nodes = length(data_def_sorted_positive_y(:,2));

b = 1;
for i = 2:Npositive_nodes-1

    if node_num(i) == mandat_breakpt(b) || (ys(i) == 0)

        Nbpts = Nbpts + 1;
        breakpt(Nbpts) = i;
        last_bpt = i;

        if length(mandat_breakpt) > b

            b = b + 1;
        end

    elseif ~isnan(max_delta_tan)

        tan_theta_x = ...
            ((zs(i+1)+uzs(i+1))-(zs(i)+uzs(i)))/...
            ((ys(i+1)+uys(i+1))-(ys(i)+uys(i)));

        tan_theta_x_last = ...
            ((zs(last_bpt+1)+uzs(last_bpt+1))-(zs(last_bpt)+uzs(last_bpt)))/...
            ((ys(last_bpt+1)+uys(last_bpt+1))-(ys(last_bpt)+uys(last_bpt)));

        if abs(tan_theta_x - tan_theta_x_last) > max_delta_tan &&...
            node_num(i+1) ~= mandat_breakpt(b)

            Nbpts = Nbpts + 1;
            breakpt(Nbpts) = i+1;
            last_bpt = i+1;
        end

    % Else if AR of surface covered since last breakpoint reaches 1.
    % Condition also ensures that new breakpoint is not set if the next
    % surface would reach a mandatory breakpoint before AR=1.
    elseif (cs(last_bpt)+cs(i))/2 < (ys(i)+uys(i))-(ys(last_bpt)+uys(last_bpt))

        if isnan(mandat_breakpt)

            Nbpts = Nbpts + 1;
            breakpt(Nbpts) = i;
            last_bpt = i;

        elseif((cs(i)+c(mandat_breakpt(b)))/2 < ...
                y(mandat_breakpt(b))+uy(mandat_breakpt(b))-(ys(i)+uys(i)) ||...
                ys(mandat_breakpt(b))-ys(i) < 0)

            Nbpts = Nbpts + 1;
            breakpt(Nbpts) = i;
            last_bpt = i;
        end
    end
end

% The first and last break points are the root and the tip, respectively.
if last_bpt ~= Npositive_nodes

    Nbpts = Nbpts + 1;
    breakpt(Nbpts) = Npositive_nodes;
    last_bpt = Npositive_nodes;
end

% The last CAERO1 element shouldn't be so narrow that it would include only
% a single column of panels. In such case, the corresponding surface area
% should be assigned to the previous CAERO1 element instead:
if panels(2)*2*(ys(last_bpt)-ys(breakpt(Nbpts-1)))/...
        (cs(last_bpt)+cs(breakpt(Nbpts-1)))<1.5

    breakpt(Nbpts-1) = [];
    Nbpts = Nbpts - 1;
end
```

Next, we calculate the points that will define the spline coordinate systems.

```
Nsplines = Nbpts - 1;
Nset = 501;
for s=1:Nsplines
    XA = xs(breakpt(s)) + uxs(breakpt(s));
    XB = xs(breakpt(s+1)) + uxs(breakpt(s+1));
    YA = ys(breakpt(s)) + uys(breakpt(s));
    YB = ys(breakpt(s+1)) + uys(breakpt(s+1));
    ZA = zs(breakpt(s)) + uzs(breakpt(s));
    ZB = zs(breakpt(s+1)) + uzs(breakpt(s+1));

    S1 = [XA,YA,ZA];
    %S2 = [XA,YA+ZB-ZA,ZA+YA-YB];
    %S3 = [XA-(YB-YA)^2-(ZB-ZA)^2, YA+(XB-XA)*(YB-YA), ZA+(XB-XA)*(ZB-ZA)];

    % With inclusion of the angle of attack:

    %Upwards spline z
    %S2 = [XA - sin(aa)*(YA-YB), YA + sin(aa)*(XA-XB) + cos(aa)*(ZA-ZB),...
    %      ZA - cos(aa)*(YA-YB)];

    %S3 = [XA + (XA-XB)*(ZA-ZB)*sin(aa) + ((YA-YB)^2+(ZA-ZB)^2)*cos(aa),...
    %      YA + (YA-YB)*((ZA-ZB)*sin(aa) - (XA-XB)*cos(aa)),...
    %      ZA - ((XA-XB)^2 + (YA-YB)^2)*sin(aa) - (XA-XB)*(ZA-ZB)*cos(aa)];

    %Downwards spline z
    S2 = [XA + sin(aa)*(YA-YB), YA - sin(aa)*(XA-XB) - cos(aa)*(ZA-ZB),...
        ZA + cos(aa)*(YA-YB)];

    S3 = [XA - (XA-XB)*(ZA-ZB)*sin(aa) - ((YA-YB)^2+(ZA-ZB)^2)*cos(aa),...
        YA - (YA-YB)*((ZA-ZB)*sin(aa) - (XA-XB)*cos(aa)),...
        ZA + ((XA-XB)^2 + (YA-YB)^2)*sin(aa) + (XA-XB)*(ZA-ZB)*cos(aa)];
```

We write the *CAERO1* aerodynamic macro elements. Each aerodynamic panel is placed between the two breakpoint nodes of each spline and the number of spanwise and chordwise aerodynamic boxes is defined so that the above-mentioned criteria are met. Also, the angle of attack is taken into consideration when writing the two leading edge locations and side chords that define each panel.

```
    if force_NswPanels
        swPanels = round(panels(1)*(YB-YA)/...
            (ys(Npositive_nodes)+uys(Npositive_nodes)));
    else

        swPanels = round(panels(2)*2*(YB-YA)/...
            (cs(breakpt(s))+cs(breakpt(s+1))));
    end
    fprintf(fout,'CAERO1*,%i,4000,0,%i,*\n*,%i,,,1,*\n',10000*s+1,...
            swPanels,panels(2));
    fprintf(fout,'*,%7.9E,%7.9E,%7.9E,%2.13f,*\n',...
        XA + (x_LEs(breakpt(s)) + uxs(breakpt(s)) - XA)*cos(aa),...
        YA,...
        ZA - (x_LEs(breakpt(s)) + uxs(breakpt(s)) - XA)*sin(aa),...
        cs(breakpt(s)));
    fprintf(fout,'*,%7.9E,%7.9E,%7.9E,%2.13f\n',...
        XB + (x_LEs(breakpt(s+1)) + uxs(breakpt(s+1)) - XB)*cos(aa),...
        YB,...
        ZB - (x_LEs(breakpt(s+1)) + uxs(breakpt(s+1)) - XB)*sin(aa),...
        cs(breakpt(s+1)));
```

Then we write the *CORD2R* rectangular coordinate systems for the splines, using the coordinates of the three previously defined points.

```
    fprintf(fout,'CORD2R*,%i,0,%7.9E,%7.9E,*\n',300+s,S1(1),S1(2));
    fprintf(fout,'*,%7.9E,%7.9E,%7.9E,%7.9E,*\n',S1(3),S2(1),S2(2),S2(3));

    fprintf(fout,'*,%7.9E,%7.9E,%7.9E\n', S3(1),S3(2),S3(3));
```

After that, we write the *SPLINE2* beam spline elements that will interpolate the forces in this aeroelastic problem, connecting the structural to the aerodynamic components of our model.

```
    if s==1

        fprintf(fout,'SPLINE2,%i,%i,%i,%i,%i,1.0,1.0,%i,+\n+,1.0,0.0\n',...
             400+s,10000*s+1,10000*s+1 + panels(2),...
             10000*s + swPanels*panels(2),Nset,300+s);
    else

        fprintf(fout,'SPLINE2,%i,%i,%i,%i,%i,1.0,1.0,%i,+\n+,1.0,0.0\n',...
             400+s,10000*s+1,10000*s+1,...
             10000*s + swPanels*panels(2),Nset,300+s);
    End
```

Also, we write the *SET1* grid point list corresponding to the structural nodes of each spline.

```
    if spline_local

        fprintf(fout,'SET1,%i',Nset);
        field = 3;
        for i=1:Npositive_nodes

            if (ys(i) == ys(breakpt(s)) || ys(i) > ys(breakpt(s))) &&...
                 (ys(i) == ys(breakpt(s+1)) || ys(i) < ys(breakpt(s+1)))
                if field == 10;
                    fprintf(fout,',+\n+');
                     field = 2;
                end
                fprintf(fout,',%i',node_num(i));
                field = field + 1;
            end
         end
         fprintf(fout,'\n\n');
         Nset = Nset + 1;
    end

    if ~spline_local
        fprintf(fout,'SET1,501,1,THRU,%i\n',Npositive_nodes);     end
```

And finally, we write the *PAERO1* aerodynamic panel property for the panels we created.

```
fprintf(fout,'PAERO1,4000\n\n');
```

That concludes the *beamwriter.m* file. This file was responsible for writing the model part of the code on the *.bdf* file.

We will now describe the *flutter_SOL145.m* file (which includes the *beamwriter.m* file), that is responsible for writing the whole flutter analysis code in the *.bdf* file.

3.4. The Flutter_SOL145 Function

This function will perform the flutter analysis (MSC Nastran SOL145) for a given wing configuration and input parameters (like airspeed, altitude and wing angle of attack). It will write the whole Nastran input *.bdf* file, run NASTRAN, read the result file, analyze the results and return the estimated divergence and flutter speeds, as well as the corresponding frequencies and vibration modes.

Here is a general overview of this function:

- Define some analysis parameters
- Open the Nastran input .bdf file and start its writing
  - Write the Executive Control section
  - Write the Case Control section
  - Write the Bulk Data section
    - Write material properties of the beam model
    - **Call the *beamwriter* function**
    - Write AERO aerodynamic parameters and its CORD2R coordinate system
    - Calculate air density and Mach speed
    - Define the list of Mach numbers and reduced frequencies for the aerodynamic matrix using the MKAERO1 entry
    - Define the flutter analysis using the FLUTTER command and its aerodynamic factors through FLFACT
- Close the Nastran input .bdf file
- Call Nastran to run the input file
- Read the output .f06 result file
- For each vibration mode, store the velocities and corresponding damping factors and frequencies in the respective variables
- For each mode, plot the V-g and V-f graphs
- Determine existing divergence and flutter speeds, and its frequencies and modes associated
- Close the output .f06 result file and show the plots
- Return the summary of the analysis results

The above steps are presented next in more detail, starting with the function header and its inputs, which come from the *flutter.m* function.

The *flutter_SOL145* function header is:

```
function [ mode_num_F, V_F, f_F, mode_num_D, V_D, flutter_sum] = ...
    flutter_SOL145_3(title_prop,title_deform,title_out,Vinf,h,aa,MAT,...
    ref_chord,panels,mandat_breakpt,n_g,branch_nodes,PM_data,...
    single_speed,undeformed)
```

Its entries are the following:

- `mode_num_F` will return the mode associated with the minimum flutter speed.
- `V_F` will return the minimum flutter speed.
- `f_F` will return the frequency associated with the minimum flutter speed.
- `mode_num_D` will return the mode associated with the minimum divergence speed.
- `V_D` will return the minimum divergence speed.
- `flutter_sum` will return all the flutter speeds and its associated frequencies and modes.
- `title prop` represents the OUTPUT_INERTIAS.cvs input file.
- `title_deform` represents the OUTNODES.csv input file.
- `title_out` represents the .bdf output file.
- `Vinf` represents an airspeed.
- `h` represents an altitude.
- `aa` represents an input angle of attack.
- `MAT` represents the material properties.
- `ref_chord` represents a reference chord.
- `panels` represents a vector input of spanwise and chordwise number of panel boxes.
- `mandat_breakpt` represents the option of having mandatory break points for the splines.
- `n_g` represents a scalarization factor.
- `branch_nodes` represents the option of excluding some nodes from the breakpoint writing process.
- `PM_data` represents the option of adding mass points.
- `Undeformed` indicates whether wing deformations are being analyzed.

First, we will define some parameters for the analysis, like how many normal modes and which velocity range we would like to evaluate.

```
Nmodes = 18;

V_range = [1,300];
```

Then there are some options that can be defined, like if we want to use a mode tracking function, in which format we want the output file, or what are the ranges of the graphic plots. Also, we initialize the vector variables that will hold the results of the analysis (flutter frequency, flutter speed and divergence speed).

```
PKS = false;
if PKS

    Hfreq = 720.;  %maximum frequency
end

track_modes = false;

%plot options
max_freq = 60;
max_damp = 0.05;
min_damp = -0.1;

post = 0;

aa = aa*pi/180;

V_Fmodes = zeros(Nmodes,1);
V_Fmodes(:) = Inf;

V_Dmodes = zeros(Nmodes,1);
V_Dmodes(:) = Inf;

f_Fmodes = zeros(Nmodes,1);
f_Fmodes(:) = Inf;
```

Next, we open the .bdf file and start its writing. In the executive control section of the file we indicate that we will run Nastran's SOL145, which is the flutter analysis. In the case control section, we define several parameters regarding the previously chosen analysis, like its title, its eigenvalue extraction parameters, its flutter analysis method parameters and mostly graphic plot parameters. The bulk data section is the last section of the Nastran input file, and it is the one where the whole analysis' content will be defined.

```
fout = fopen(title_out,'w');

fprintf(fout,'NASTRAN SPLINE_METRICS, IFPSTAR = NO\n');
fprintf(fout,'SOL,145\nCEND\n\n');
fprintf(fout,'TITLE = Test2 SOL145 with %i m/s NL deflection\n',Vinf);
fprintf(fout,'ECHO = NONE\n');
fprintf(fout,'METHOD = 10\n');
fprintf(fout,'SVEC = ALL\n');
fprintf(fout,'FMETHOD = 40\n');
fprintf(fout,'DISP = ALL\n');
fprintf(fout,'RESVEC = NO\n');

if ~single_speed

    fprintf(fout,'OUTPUT(XYOUT)\n');
    fprintf(fout,'  CSCALE = 2.0\n');
    fprintf(fout,'  PLOTTER, NASTRAN\n');
    fprintf(fout,'  CURVELINESYMBOL = -6\n');
    fprintf(fout,'  YTTITLE = DAMPING G\n');
    fprintf(fout,'  YBTITLE = FREQUENCY F HZ\n');
    fprintf(fout,'  XTITLE = VELOCITY V (M/SEC)\n');
    fprintf(fout,'   XMIN = 0.0\n');
    fprintf(fout,'   XMAX = %2.1f\n',V_range(2));
    fprintf(fout,'   YTMIN = %2.1f\n',min_damp);
    fprintf(fout,'   YTMAX = %2.1f\n',max_damp);
    fprintf(fout,'   YBMIN = 0.0\n');
    fprintf(fout,'   YBMAX = %2.1f\n',max_freq);
    fprintf(fout,'  XTGRID LINES = YES\n');
    fprintf(fout,'  XBGRID LINES = YES\n');
    fprintf(fout,'  YTGRID LINES = YES\n');
    fprintf(fout,'  YBGRID LINES = YES\n');
    fprintf(fout,'  UPPER TICS = -1\n');
    fprintf(fout,'  TRIGHT TICS = -1\n');
    fprintf(fout,'  BRIGHT TICS = -1\n');
    fprintf(fout,'  XYPLOT VG / 1(G,F), 2(G,F), 3(G,F), 4(G,F), 5(G,F), 6(G,F)\n');
%    fprintf(fout,'              7(G,F),8(G,F),9(G,F),10(G,F),11(G,F),12(G,F)\n');
End
```

After the *BEGIN BULK* command, the first thing we write is the material properties (*E*, *G* and *ρ*) of our beam model.

```
fprintf(fout,'BEGIN BULK\n\n');
fprintf(fout,'MAT1*,1000,%6.10E,%6.10E,,*\n%2.3f\n\n',MAT(1),MAT(2));
fprintf(fout,'*,%2.11f\n\n',MAT(3));
```

Then we call the *beamwriter* function, that will do everything described in the previous chapter.

```
flutter_Beam_Writer4(title_prop,fout,title_deform,aa,panels,...
    mandat_breakpt,n_g,branch_nodes,PM_data,undeformed);
```

Next, we write the *CORD2R* coordinate system for the general aerodynamic parameters, based on the input angle of attack. And after that, we write the *AERO* entry to define those aerodynamic parameters for the analysis, like the reference density and the reference chord for the reduced frequencies.

```
% Aerodynamic coordinate system, followed by AERO entry
fprintf(fout,'CORD2R*,300,0,0.0,0.0,*\n');
fprintf(fout,'*,0.0,%3.13f,0.0,%3.13f,*\n',-sin(aa),-cos(aa));
fprintf(fout,'*,%3.13f,0.0,%3.13f\n\n',-cos(aa),sin(aa));

fprintf(fout,'AERO,300,,%2.5f,1.0,+1\n\n',ref_chord);
```

Following that, using the input altitude and considering the standard atmosphere model, we calculate the air density and the Mach speed that will feature in the analysis.

```
T0 = 288.15;

if h < 11000

    T = T0 - 0.0065*h;
    rho = 1.225*(1 - 0.0065*h/T0)^4.25588;
else

    T = 216.65;
    rho = 0.36392*exp(-0.000157688*(h-11000));
end

a = 340.3*sqrt(T/T0);
M = Vinf/a;
```

Then, resorting to the *MKAERO1* entry, we define the list of Mach numbers and reduced frequencies for the aerodynamic matrix calculation, that will constitute part of the flutter analysis.

We also define some data needed to perform real eigenvalue analyses in Nastran with the Lanczos method, using the *EIGRL* entry.

```
fprintf(fout,'MKAERO1,%3.6f,,,,,,,,+\n',M);
fprintf(fout,'+,0.001,0.01,0.1,0.5,1.,3.,5.,7.\n');
fprintf(fout,'EIGRL,10,10.E-10,,50,,,,MAX\n\n');
```

Next, using the *FLUTTER* command, we choose the p-k method as the method of flutter analysis and the triplet of aerodynamic data for which the flutter problem will be solved, namely: air density, Mach number, and the velocity range of the analysis. These aerodynamic factors are defined using the *FLFACT* command.

```
if PKS

    fprintf(fout,'FLUTTER,40,PKS,1,2,3,TCUB,%4.3f,0.001,\n',Hfreq);
else

    fprintf(fout,'FLUTTER,40,PK,1,2,3,TCUB,,,\n');
end

fprintf(fout,'FLFACT,1,%2.6f\n',rho);
fprintf(fout,'FLFACT,2,%2.6f\n',M);

if single_speed

    fprintf(fout,'FLFACT,3,%2.4f\n\n',-Vinf);
else

    fprintf(fout,'FLFACT,3,%2.4f,THRU,%2.4f,%i\n\n',...
    V_range(1),V_range(2),500);
End
```

After defining some more file parameters, we finish the Bulk Data section with the *ENDDATA* command, finally concluding the writing of the .bdf Nastran input file.

```
fprintf(fout,'PARAM,LMODES,%i\n',Nmodes);
fprintf(fout,'PARAM,GRDPNT,1\n');
fprintf(fout,'PARAM,POST,%i\n',post);
fprintf(fout,'ENDDATA\n');

fclose(fout);
```

Since the Nastran input file is written, we now call the MSC Nastran Software to run the input file.

```
command = ['C:\MSC.Software\MSC_Nastran\20131\bin\nastran ' title_out];

system(command);
```

Logically, we will then read and analyze the results. We begin by reading the .f06 result file, which for every required normal mode will present, for the whole velocity range (V), corresponding frequencies (F) and damping factors (G).

```
i=1;
while title_out(i)~='.'

    i=i+1;
end

title_f06 = title_out;
title_f06(i+1:end)='f06';
```

```
ff06 = fopen(title_f06,'r');

text_f06 = fileread(title_f06);

index = strfind(text_f06,'FLUTTER  SUMMARY');

fseek(ff06,index(1),'bof');

for i=1:6

    fgets(ff06);
end

mode = 1;

for m=1:Nmodes

    i1 = 1;

    while mode == m

        line = textscan(ff06, '%s %f %f %f %f %f %f', 1);
        line = [line{1,2:7}];

        if m == 1

            V(i1) = line(2);
        end

        G(i1,m) = line(3);
        F(i1,m) = line(4);

        i1 = i1 + 1;

        R = textscan(ff06, '%s', 1, 'delimiter', ' ');

        if R{1,1}{1,1} == '1'

            for i=1:5

                fgets(ff06);
            end

            mode = textscan(ff06, '%s %s %d', 1);
            mode = mode{1,3};

            for i=1:4

                fgets(ff06);
            end
        end
    end
end

if track_modes;

    [V,G,F] = mode_tracking(V,G,F);
End
```

After storing the velocities and the corresponding damping factors and frequencies in the respective variables, we then draw two graphs that will plot the damping as a function of the velocity range (V-g) and the frequency as a function of the velocity range (V-f), for every vibration mode analyzed.

```
if single_speed
```

```
    xls_gf_title =
strcat('AA_h',num2str(h),'_v',num2str(Vinf),'_aa',num2str(aa*180/pi),'.xls');

    xlswrite(xls_gf_title,G');
    xlswrite(xls_gf_title,F',1,'B1');
else

    figure;
    s(1) = subplot(2,1,1);
    s(2) = subplot(2,1,2);

    plot(s(1),V,G);
    grid(s(1),'on');
    xlabel(s(1),'Velocity');
    ylabel(s(1),'Damping [-]');
    xlim(s(1),[V_range(1),V_range(2)]);
    ylim(s(1),[min_damp,max_damp]);

    plot(s(2),V,F);
    xlabel(s(2),'Velocity');
    ylabel(s(2),'Frequency [Hz]');
    xlim(s(2),[V_range(1),V_range(2)]);
    ylim(s(2),[0,max_freq]);
end
```

Then, for each vibration mode, we determine any existing cases of flutter or divergence and the velocities at which they occur. Also, in case of flutter, we determine the frequency at which it occurs. This is done by checking, for every mode, if there is a point (an increment in the velocity range variable) where the damping changes from negative to positive or null. At that point, if the frequency is positive, we have found a flutter point, and we register the velocity and frequency at which it occurs. On the other hand, if there is a point where the damping changes from negative to positive or null, but the frequency is zero, it means we have found a divergence point, and so we register the velocity at which it occurs.

```
NV = length(V);
NM = size(F,2);

for i=1:NV-1

    for j=1:NM

        if G(i,j) < 0 && (G(i+1,j) == 0 || G(i+1,j) > 0)

            if F(i+1,j) > 1.0E-10 && V_Fmodes(j) == Inf
                % flutter is found
                % interpolates flutter speed and frequency

                V_Fmodes(j) = (((V(i+1)-V(i))/(G(i+1,j)-G(i,j)))*(0-
G(i,j))+V(i));

                f_Fmodes(j) = (((F(i+1,j)-F(i,j))/(G(i+1,j)-G(i,j)))*(0-
G(i,j))+F(i,j));

            elseif F(i+1,j) < 1.0E-10 && V_Dmodes(j) == Inf
                % divergence is found
                %interpolates divergence speed


                V_Dmodes(j) = (((V(i+1)-V(i))/(G(i+1,j)-G(i,j)))*(0-
G(i,j))+V(i));
            end
        end
    end
end
```

After this analysis, we close the .f06 result file and show the plots described before.

Lastly, we store a summary of the minimum divergence speed found and the mode for which it occurs, as well as a summary of the minimum flutter speed found, and the frequency and mode for which it occurs. We also store all the remaining flutter speeds found, their corresponding frequencies, and which modes they represent. These are the return variables of the *flutter_SOL145* function.

```
fclose(ff06);

%Plot results
command = ['plotps ' title_f06(1:length(title_f06)-4)];
[status,cmdout] = system(command);

% The lowest V_Fmodes value is the flutter speed V_F
[V_F, mode_num_F] = min(V_Fmodes);
f_F = f_Fmodes(mode_num_F);

% The lowest V_Dmodes is the divergence speed V_D
[V_D, mode_num_D] = min(V_Dmodes);

i=0;
for m=1:Nmodes

    if V_Fmodes(m) ~= Inf

        i = i + 1;

        flutter_rand(i,1) = V_Fmodes(m);
        flutter_rand(i,2) = f_Fmodes(m);
        flutter_rand(i,3) = m;
    end
end

if single_speed

    flutter_sum = NaN;
elseif i>0

    flutter_sum = sortrows(flutter_rand,1);
else

    flutter_sum = [Inf,NaN,NaN];
end
end
```

3.5. The Flutter Main Function

We will now describe the main function of the frequency domain analysis tool. This *flutter.m* function is an iterative calculation routine and here is a brief explanation of the idea behind this iterative process. As explained before in the Input Data section, a certain wing configuration that we choose to analyze will not change its AR, angle of attack and altitude with the variation of the airspeed. It will, however, have a different deformed configuration for each airspeed. In addition, for every airspeed/deformed wing configuration combination that we introduce as input, the calculated minimum flutter speed is expected to be different. For example, at a certain altitude, it is possible to calculate that a wing with an aspect-ratio of 20 with 2 degrees of angle of attack (AR20 aa2) that is deformed at the airspeed of 40 m/s will have a predicted flutter speed of 236 m/s. While if the airspeed is increased to 100 m/s, the deformation will be much higher for the same wing, and the calculated minimum flutter speed will be significantly lower, 113 m/s. Therefore, the goal of this iterative calculation routine is to converge the input airspeed (and its corresponding deformed wing configuration) with the flutter speed. This way, it is possible to find the flutter speed for a certain wing configuration, while taking into account its deformation due to the airspeed.

Here is a general overview of this function:

- Define analysis parameters;
- Import data (including available database range) from the ***flutter_parameters* function**;
- Initiate the iterative cycle for all input altitudes and input angles of attack;
- Define the condition at which the airspeed iterations will stop;
- Define how the airspeeds are iterated;
- Write the name of the deformed wing configuration input file corresponding to the airspeed for which we want to calculate the flutter speed;
- **Call the *interpolate_deflections* function** to obtain such wing configuration input file;
- Write the name of the Nastan input *.bdf* file matching the analysis' parameters;
- **Call the *flutter_SOL145* function** to calculate the minimum flutter speed for the airspeed/deformed wing configuration in analysis;
- Display the input parameters and the return variables of function *flutter_SOL145*;
- Iterate until the airspeed and flutter speed converge;
- Repeat the cycle for all input angles of attack and input altitudes.

First, we define some key parameters, such as the angles of attack and the altitudes of the wing configurations we will analyze. We also define the *.bdf* file label corresponding to how many vibration modes we will look at, as well as some iterative parameters like *Vfact*, which will be addressed in more detail below.

In the *flutter_parameters* file, which was introduced before, there are some important variables that we will now import. In addition to a label that represents the aspect ratio of the wing we will analyze, there are also velocity lists for different angles of attack and altitudes that represent the wing configurations for which we already have input data sheets. The remaining variables that we import from this file have already been described in previous chapters.

After that, there are some mass points and scaling factors options that will not be used in the analyses presented here. The initialization of the variables is as follows:

```
% Iterative flutter calculation routine
% NOTES:
% - "Nmodes", and "V_range" are defined in flutter_SOL145_3.m
% - "PKS" is determined to be true or false in flutter_SOL145_3.m
% - "spline_local" is defined in flutter_Beam_Writer4.m
% - "max_delta_tan" is defined in flutter_Beam_Writer4.m
% - force_NswPanels is defined in flutter_Beam_Writer4.m
% - "n_g" is defined in flutter_Beam_Writer4.m
% - "polynomials" is defined in interpolate_deflections2.m
% - in "undeformed" analyses, it is recommended to say so in "end_label"

clear all

% First airspeed value
Vinf0 = 40;

% Iterative parameters
iterative_flutter = true;
Vfact = 3;
undeformed = false;
single_speed = false;

end_label = '_18modes_8x';

altitude = 0;

aa = 2;%[-4,-2,0,2,4,6,8,10];

% Name of function must correspond to desired case!
[title_prop, label_files, MAT, ref_chord, mandat_breakpt, ...
        panels, altitude_data, aa_data, V_data,...
        branch_nodes] = flutter_parm_M3357();

% Point masses. If necessary, masses must be "true", and correct function
% with PM data to pass on to PM_data must be indicated in the "if"
% statement.
% PM_data comprises of mass and principal inertias of each point mass, as
% well as node numbers of the nodes the masses are to be attached to and
% offsets of PM center of mass to said nodes.
masses = false;

if masses

    PM_data = flutter_PM_M3357();
else

    PM_data = NaN;
end

% Scaling factors
n_g = 1;
n_w = 1;
n_rho = 1;
n_v = n_w*n_g;

ref_chord = ref_chord*n_g;
MAT(1:2) = MAT(1:2)*n_rho*n_v^2;

mode_num_F = zeros(length(altitude));
V_F = zeros(length(altitude));
```

```
f_F = zeros(length(altitude));
mode_num_D = zeros(length(altitude));
V_D = zeros(length(altitude));

beep on;
```

Then we begin the iterative cycle. The cycle will run for all input altitudes, initial input airspeeds and input angles of attack. At each time, it will compare the input altitudes and angles of attack with the rows and columns of the *V_data* matrix that contains all the available velocity lists that correspond to all already available "OUTNODES.csv" wing deformation input files. This way, we know which list (*Vdata*) of airspeeds/wing deformation input files we have available for the combination of altitude and angle of attack that we wish to analyze.

We also convert the altitude and angle of attack in focus to strings, so we can use them for part of the input files' names (both for the "OUTNODES.csv" file with the required wing deformation data to analyze, and for the *.bdf* Nastran input file that we will write).

```
for h=1:length(altitude)
        saltitude = num2str(altitude(h));
            for i=1:length(saltitude)
        if saltitude(i) == '.';
            saltitude(i) = '-';
        end
    end
     j = 1;
    while altitude(h) ~= altitude_data(j)
            j = j + 1;
    end
     for v=1:length(Vinf0)
         for a=1:length(aa)
              saa = num2str(aa(a));
                 for i=1:length(saa)
                    if saa(i) == '.';
                              saa(i) = '-';
                end
            end
            i = 1;
            while aa_data(i) ~= aa(a)
                      i = i + 1;
            end
            Vdata = V_data{i,j};
            iter = 2;
            Vinf = Vinf0(v);
            last_V = Inf;
```

Next, we define the tolerance value between the airspeed and the flutter speed at which the cycle will stop. Meaning that if the flutter speed is already really close (within the tolerance value) to the airspeed, there is no need to calculate its corresponding flutter speed, since we have already determined it.

```
% Condition defines tolerance value for flight speed vs flutter
            while (abs(V_F(h,iter-1)-Vinf) > 0.1 || Vinf == 0);

                if V_F(h,iter-1) == Inf,

                    break
                end
```

We will then define how the airspeed that is used for the flutter calculation of each new cycle is interpolated. This new airspeed will be the difference between the previous airspeed and its corresponding flutter speed (calculated in the previous iteration), divided by a factor *Vfact.*

```
if abs(V_F(h,iter-1)-Vinf) > 0.1 && iter > 2

    Vinf = Vinf + (V_F(h,iter-1) - Vinf)/Vfact;
End
```

The next condition states that if a new airspeed value was iterated in the previous operation, we convert it to a string to become part of the name of the input files, and use it to calculate the new flutter speed. On the other hand, if there isn't a new airspeed value, it means that it had already converged (under the defined condition value) in the previous cycle, and so there is no need to calculate a new flutter speed.

```
if abs(Vinf-last_V) > 0.0001

    last_V = Vinf;

    sVinf = num2str(Vinf,7);

    for i=1:length(sVinf)

        if sVinf(i) == '.';

            sVinf(i) = '-';
        end
    end
```

Once we know all the information about the wing configuration for which we want to determine its flutter speed, we write the title of the "OUTNODES.csv" input file that will match such configuration. That information being the altitude, the angle of attack and the airspeed of the current cycle iteration.

```
    title_deform = ...
    strcat(label_files,'_aa',saa,'_deflections_',sVinf,...
    '_h',saltitude,'.csv');
```

After that, we call the *interpolate_deflections* function that based on the *Vdata* vector (which contains the airspeed list corresponding to the appropriate *V_data* matrix entry) will check if the wing configuration for the required airspeed is present in the input database. If that isn't the case, this function will interpolate (based on the already existing deflection data) a new deformed wing configuration corresponding to the airspeed for which we want to calculate the flutter speed, and create a new input file with such data.

```
    interpolate_deflections2(title_deform,label_files,...
    saltitude,saa,Vinf/n_v,Vdata);
```

Also, we write the title of the Nastan input *.bdf* file matching the analysis' parameters. Since we now have the input file with the required wing configuration for the analysis, as well as the remaining required input variables, we can then call the *flutter_SOL145* function, that will return the desired flutter speed.

```
                    title_out = ...
                        strcat(label_files,'_aa',saa,'_flutter_V',sVinf,...
                        '_h',saltitude, end_label,'.bdf');

                    % Obtain flutter boundary with SOL145
                    [mode_num_F(h,iter),V_F(h,iter),f_F(h,iter),...
                        mode_num_D(h,iter),V_D(h,iter),flutter_sum] = ...
                    flutter_SOL145_3(title_prop,title_deform,title_out,Vinf,...
                    altitude(h),aa(a),MAT,ref_chord,panels,mandat_breakpt,...
                    n_g,branch_nodes,PM_data,single_speed,undeformed);
```

Next, we will display the current input altitude and angle of attack for which we are iterating the flutter speed. This will be followed by the airspeed that is the current input of the *flutter_SOL145* function and by its return variables, which are: the minimum flutter speed, its frequency and the vibration mode for which it occurs; the additional flutter speeds found, their corresponding frequencies and which modes they represent; the minimum divergence speed found and the mode for which it occurs.

```
                    if ~single_speed

                        disp(strcat('altitude=',num2str(altitude(h)),...
                            ' aa=',num2str(aa(a))));

                        disp(strcat(int2str(iter-1),':',' Vinf=',num2str(Vinf),...
                            ', V_F=',num2str(flutter_sum(1,1)),...
                            ', f_F=',num2str(flutter_sum(1,2)),...
                            ', mode #=',num2str(flutter_sum(1,3))));

                        if length(flutter_sum(:,1)) > 1

                            for i = 2:length(flutter_sum(:,1))

                                disp(strcat('Additional flutter speed:',...
                                ', V_F=',num2str(flutter_sum(i,1)),...
                                ', f_F=',num2str(flutter_sum(i,2)),...
                                ', mode #=',num2str(flutter_sum(i,3))));
                            end
                        end

                        disp(strcat(', V_D=',num2str(V_D(h,iter)),...
                            ', mode #=',num2str(mode_num_D(h,iter))));
                    end

                    iter = iter + 1;

                    beep
                end

                if ~iterative_flutter

                    break
                end
```

Finally, the cycle will be repeated until the airspeed and the flutter speed converge. This will then happen for all input angles of attack, input initial airspeeds, and input altitudes. For each cycle that is completed a beep signal is produced.

```
            end
            beep
```

```
        end
        beep
    end
    beep
end
```

3.6. The Interpolate_deflections Function

As mentioned before in the *flutter* function, the *interpolate_deflections* function's objective is to provide the appropriate deformed wing configuration corresponding to the airspeed for which we want to calculate the flutter speed.

It will check if the wing configuration for the required airspeed is already present in the input database and, if that isn't the case, will generate a new deformed wing configuration by interpolation of the already existing deflection data and create an input file with such data.

Here is a general overview of this function:

- If the input airspeed doesn't already exist in the available airspeed database, find which two database airspeeds are closer to the input airspeed, so to be used in the interpolation.
- Once those two airspeeds are determined, read their corresponding wing deformation data files.
- Interpolate the wing deformation data in those two files using the input airspeed.
- Write the newly interpolated wing deformation data in a data file corresponding to the input airspeed.

This function header is shown below.

```
function interpolate_deflections2(title_deflections,label_files,...
                      saltitude,saa,Vinf,Vdata)
% interpolate_deflections2
% Writes new deflection file by interpolation of deflection values of
% existing deflection data.

% Specify whether interpolation is to be done with polynomials. If not, the
% program will resort to linear interpolation.
```

The inputs of this function are the following:

- title_deflections represents the name of the wing deformation data file corresponding to the input airspeed.
- label_files represents a string with the input Aspect Ratio.
- saltitude represents a string with the input altitude.
- Saa represents a string with the input angle of attack.
- Vinf represents the input airspeed.
- Vdata is a vector that represents the list of airspeeds that correspond to the wing deformation input files we have available (from the non-linear aeroelastic framework) for the input altitude and input angle of attack.

When the input airspeed doesn't already match one of the available airspeeds in our database, we perform a linear interpolation which always resorts to two airspeeds, VA and VB, for which we already have deformation data in our database.

Firstly, if the input airspeed (*Vinf*) (for which we want the corresponding deformed wing configuration) is lower than the lowest airspeed for which we already have available deformation data, the linear interpolation will use the lowest and the second lowest airspeeds (of the *Vdata* vector) as VA and VB. On the other hand, if the input airspeed is higher than the highest airspeed for which we already have available deformation data, the linear interpolation will use the highest and the second highest airspeeds as VA and VB.

```
if Vinf < Vdata(1)

        VA = Vdata(1);
        VB = Vdata(2);

    elseif Vinf > Vdata(length(Vdata));

            VA = Vdata(length(Vdata)-1);
            VB = Vdata(length(Vdata));
```

The two remaining possibilities are the case where the input airspeed matches one of the available airspeeds on the *Vdata* vector, and the case where the input airspeed is a value between two available airspeeds on the *Vdata* vector. On the first case, the function simply ends because there already is a deformed wing configuration in our database for such input airspeed. On the second case, however, we will use those two "adjacent" available airspeeds as VA and VB, to interpolate the new wing configuration.

```
else

            i = 1;
            while Vdata(i) < Vinf

                i = i + 1;
            end

            if Vdata(i) == Vinf

                return
            end

            VA = Vdata(i-1);
            VB = Vdata(i);
    end
```

Once we know the VA and VB that will be used for the interpolation, we convert those values to strings, so we can assemble the name of the corresponding wing deformation data files.

```
    sVA = num2str(VA,7);

    for i=1:length(sVA)

        if sVA(i) == '.';
```

```
            sVA(i) = '-';
        end
    end

    sVB = num2str(VB,7);

    for i=1:length(sVB)

        if sVB(i) == '.';

            sVB(i) = '-';
        end
    end

    title_deflections_A = ...
                    strcat(label_files,'_aa',saa,'_deflections_',sVA,'_h',...
                    saltitude,'.csv');

    title_deflections_B = ...
                    strcat(label_files,'_aa',saa,'_deflections_',sVB,'_h',...
                    saltitude,'.csv');
```

We then read those two files and use Vinf's relative value to VA and VB to linearly interpolate all the wing deformation data present in the input data files. This newly interpolated data will be written in a new wing deformation file corresponding to the input airspeed.

```
    dataA = csvread(title_deflections_A);
    dataB = csvread(title_deflections_B);

    dataC = dataA;

    for i=1:length(dataA(:,1));

        for j = 12:17

            %dataC(i,j) = interpolate(Vinf,dataA(i,j),dataB(i,j),VA,VB);
            dataC(i,j) = (((dataB(i,j)-dataA(i,j))/(VB-VA))*(Vinf-
VA)+dataA(i,j));
        end
    end
end

csvwrite(title_deflections,dataC);
end
```

4. Results

In this chapter, we will first describe the application case that is the object of all analyses. Then, the results obtained through the frequency domain tool will be laid out. Next, the results obtained through the time domain method will be presented. Lastly, a comparison between both methods will be made.

4.1. Application Case

The application case of this work is a simple rectangular NACA 0012 wing model with 20 meters of span *b* and variable chord *c* and, as it can be seen in Figure 13, it has an internal structure with a wing-box that starts and ends, respectively, at 25% and 75% of the chord.

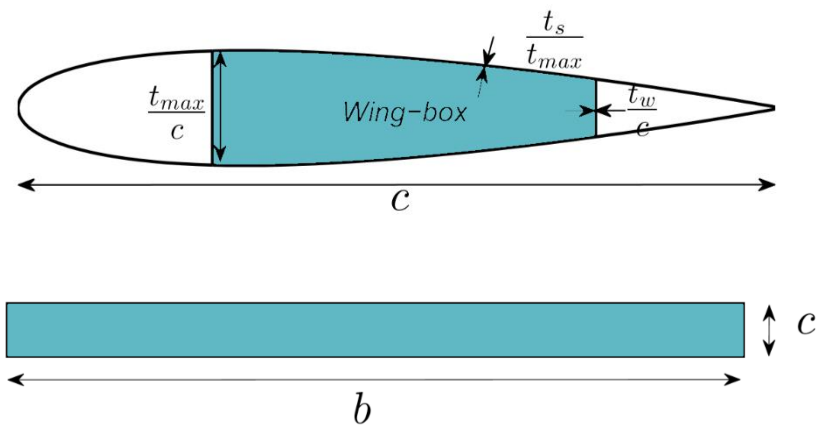


Figure 13 - External and Internal geometries of the wing model

This model is made of aluminum with a Young modulus of 70 GPa, a density of 2700 kg/m^3 and a shear modulus of 27 GPa. In order to evaluate the effect of aspect-ratio, wing span was set fixed to 20 m, while the chord was changed to generate three different values of aspect-ratio *AR*: 12 (*c* = 1.67 m), 20 (*c* = 1 m) and 28 (*c* = 0.71 m). Such geometric modifications change the wing area. For instance, by reducing the chord a smaller wing area *S* and a larger aspect-ratio *AR* were reached. Moreover, since the wing-box parameters such as the thicknesses of the webs and caps are normalized to the chord and airfoil maximum thickness, as the wing chord decreases, the mass and inertia moments also decrease. However, despite the differences in wing area and mass, they still have the same inertia ratio I_2/I_1 (ratio between the two bending stiffnesses, chord EI_2 and flap EI_1) and undergo different deformed states for the same flight conditions. This is what allows a comparison between different flutter prediction methods while considering wing models that present different aeroelastic behavior. A summary of the main characteristics of the considered wings is presented in Table 1.

These are the different wings that will be studied through different methods in order to try to understand the influence that large wing deformations (characteristic of high aspect-ratio wings) have on the wing aeroelastic behavior.

AR	***c* [m]**	***b* [m]**	***S* [m^2]**	***EI$_1$* [N m^2]**	***EI$_2$* [N m^2]**	***GJ* [N m^2]**	***I$_2$/I$_1$* [-]**	***Mass* [kg]**
12	1.67	20.00	33.33	3.76E+06	1.02E+08	3.57E+06	27.1	679.34
20	1.00	20.00	20.00	4.88E+05	1.32E+07	4.63E+05	27.1	244.56
28	0.71	20.00	14.29	1.27E+05	3.44E+06	1.20E+05	27.1	124.78

Table 1 - Main characteristics of three different wing configurations

4.2. The Frequency Domain Method Results

In this section, the results obtained from the frequency domain tool will be presented. First, we will analyze three different conditions in which the tool can be operated and we will draw conclusions from it. Then, the “end product” results of the frequency domain method to which we will draw a comparison with the time domain ones will be laid out. Lastly, we will observe the flutter mechanisms of different wings through the analysis of V-G (Velocity-Damping) and V-F (Velocity-Frequency) plots.

4.2.1. Frequency Domain Tool under different conditions

First, we will show a contrast between three different conditions through which is possible to use the frequency domain tool:

- Condition 1 – Undeformed wing at normal operating conditions.
- Condition 2 – Deformed wing at normal operating conditions.
- Condition 3 – Deformed wing at predicted flutter speed.

In the first condition we estimate the flutter speed of a wing at a certain (low) airspeed without deformation. In the second condition we estimate the flutter speed of a wing at a certain (low) airspeed while taking into account the deformation due to the airspeed. The third condition represents the final product of the frequency domain tool, after a convergence cycle, where the wing is subject to an airspeed that matches the predicted flutter speed, while considering the wing deformation due to the airspeed.

Two intermediate wings (AR = 16 and AR = 24) were considered for comparison purposes. These wings also have the same inertia ratio and wing span of the remaining wings, with intermediate wing area, chord and mass values.

The predicted flutter speed and dimensionless vertical displacement at the wing tip ($u_z/(b/2)$) are shown in Table 2, 3 and 4 for the above stated conditions.

Condition 1 – Undeformed wing at normal operating conditions

AR [-]	U [m/s]	$u_z/(b/2)$ [-]	Flutter Speed [m/s]
12	40	0	286.00
16	40	0	233.40
20	40	0	197.05
24	40	0	172.63
28	40	0	160.64

Table 2 - Flutter speed and dimensionless wing tip displacement for an undeformed wing at normal operating conditions of 40 m/s at an angle of attack of 2 degrees

Condition 2 – Deformed wing at normal operating conditions

AR [-]	U [m/s]	$u_z/(b/2)$ [-]	Flutter Speed [m/s]
12	40	-1,87E-05	285.55
16	40	0,0001	232.33
20	40	0,0016	194.75
24	40	0,0073	168.27
28	40	0,0245	152.63

Table 3 - Flutter speed and dimensionless wing tip displacement for a deformed wing at normal operating conditions of 40 m/s at an angle of attack of 2 degrees

Condition 3 – Deformed wing at predicted flutter speed

AR [-]	U [m/s]	$u_z/(b/2)$ [-]	Flutter Speed [m/s]
12	218,23	-0,0019	218,23
16	160,76	0,0045	160,76
20	127,40	0,0212	127,40
24	106,10	0,0539	106,10
28	92,24	0,1095	92,24

Table 4 - Flutter speed and dimensionless wing tip displacement for a deformed wing at the predicted flutter speed at an angle of attack of 2 degrees

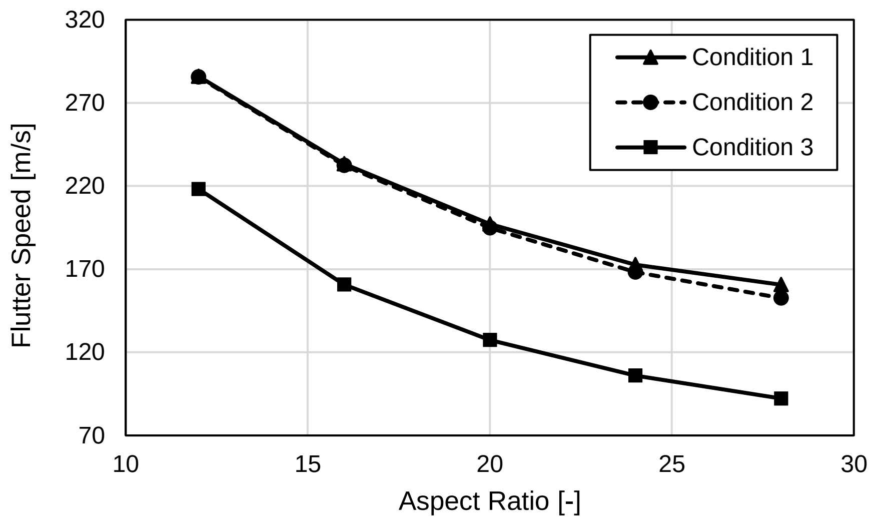


Figure 14 - Flutter speed and dimensionless wing tip displacement for a deformed wing at the predicted flutter speed at an angle of attack of 2 degrees

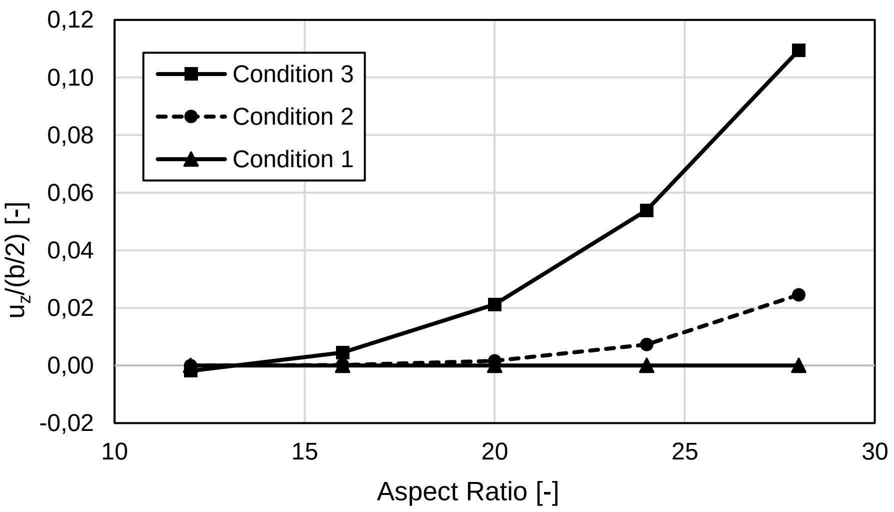


Figure 15 - Dimensionless wing tip displacement *vs* AR

As expected for all conditions, the predicted flutter speed decreases with an increase in the wing aspect-ratio. While comparing conditions 1 and 2, in Figure 14 it is possible to see that even a slightly deformed wing (at an airspeed of 40 m/s) has an influence in the predicted flutter speed. As expected, the prediction is that a deformed wing would reach the flutter point sooner than an undeformed one. The higher the aspect-ratio, the higher the deformation (as shown in Figure 15) and consequently the larger the difference between predicted flutter speeds.

Regarding condition 3, in Figure 14, we can see that it presents a similar trend to that of condition 2, although slightly steeper because of the higher deformation at a higher airspeed (which is again shown in Figure 15). Also, for the different aspect-ratios, the flutter and airspeed converge at values that are lower than the one predicted in condition 2.

4.2.2. Frequency Domain Results

Now, the "end product" results obtained through the frequency domain tool (which were addressed above as "condition 3") will be looked upon. These are the predicted flutter speeds and corresponding dimensionless wing tip displacements at the aeroelastic static equilibrium states for wings of different aspect-ratio at different flight conditions (where the angle of attack was swept from -4 degrees to 10 degrees).

α [º]	***AR* = 12**		***AR* = 20**		***AR* = 28**	
	u_z/(b/2) [-]	*V_f* [m/s]	*u_z/(b/2) [-]*	*V_f* [m/s]	*u_z/(b/2) [-]*	*V_f* [m/s]
-4	-0.0296	217.84	-0.1321	132.14	-0.3376	101.51
-2	-0.0204	218.20	-0.0826	129.47	-0.2191	91.58
2	-0.0019	218.23	0.0212	127.40	0.1095	92.24
4	0.0073	217.88	0.0723	128.53	0.2488	98.32
6	0.0164	217.31	0.1209	130.71	0.3551	101.39
8	0.0254	216.55	0.1664	133.42	0.4315	101.52
10	0.0341	215.66	0.2077	136.14	0.4892	100.30

Table 5 - Predicted flutter speeds and corresponding dimensionless wing tip displacements for wings of different aspect-ratios, where the angle of attack was swept from -4 degrees to 10 degrees.

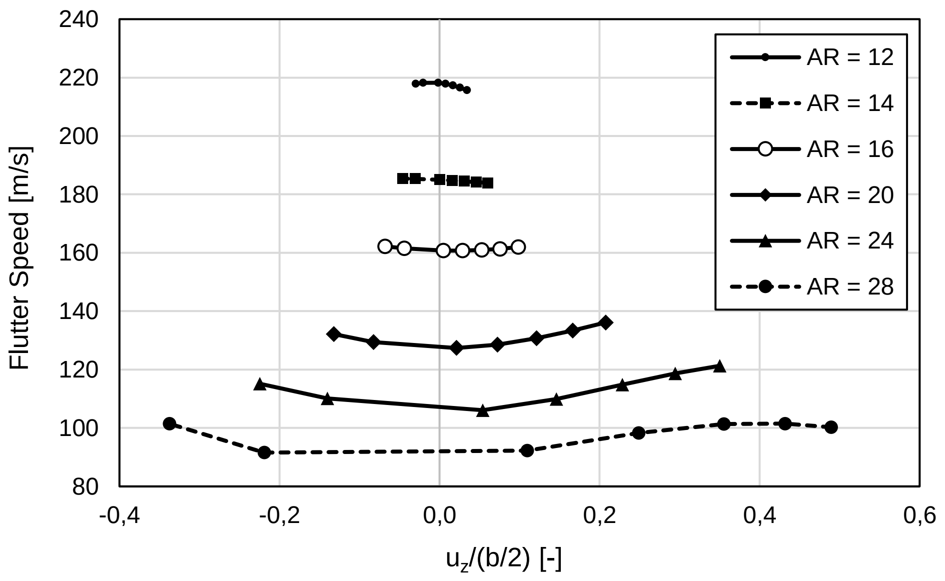


Figure 16 - Predicted flutter speeds and corresponding dimensionless wing tip displacements for wings of different aspect-ratios, where the angle of attack was swept from -4 degrees to 10 degrees.

As one can observe from Table 5 and Figure 16, the changes made on the wing to produce different aspect-ratios highlighted the non-linear of behavior of these flexible wings as the flight conditions change. For low aspect-ratios, such as *AR* = 12, flutter speeds decrease as the dimensionless vertical tip displacement increases caused by the different angles of attack. Then, for an aspect-ratio of 14, the flutter speed is nearly constant with the increase of the vertical tip displacement. Finally, for aspect-ratios higher than 14, the flutter speed increases as the vertical tip displacement increases, inverting the previously observed trend, up to a given point as noted for dimensionless vertical tip displacements above 40% for the wing aspect-ratio of 28. Another relevant factor to notice is that flutter is predicted to occur at increasing dimensionless tip displacements as the aspect-ratio increases. An explanation to this could be assigned to the fact that the higher flexibility effect exceeds the reduction of the loads due to the lower airspeed.

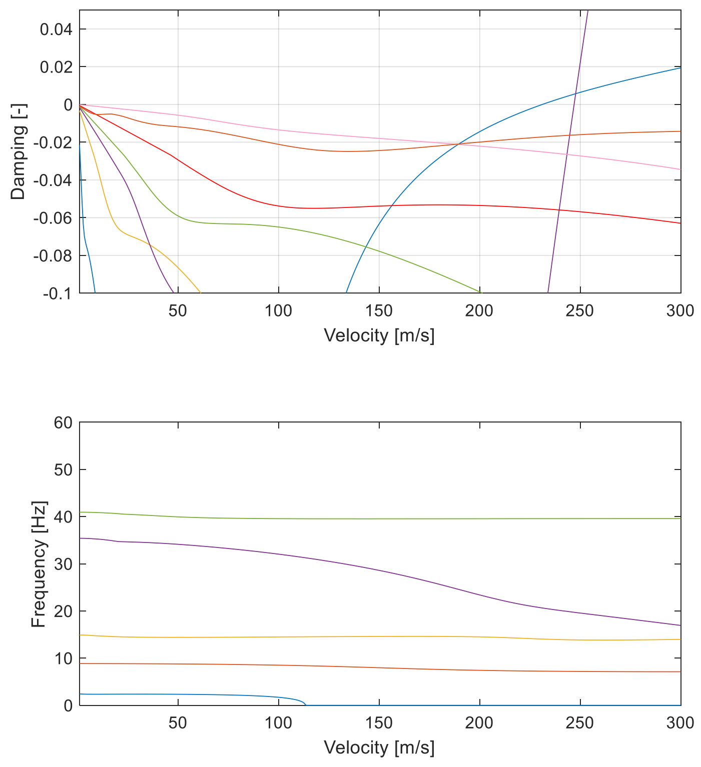
4.2.3. Flutter Mechanism

We will now address the observed flutter mechanism of the different wings at different flight conditions. For this, we will analyze some V-G and V-F graphs that were obtained at each flutter speed boundary (the point after convergence where the frequency domain tool computes the predicted flutter speed). In each case, we will look at the first few vibration modes.

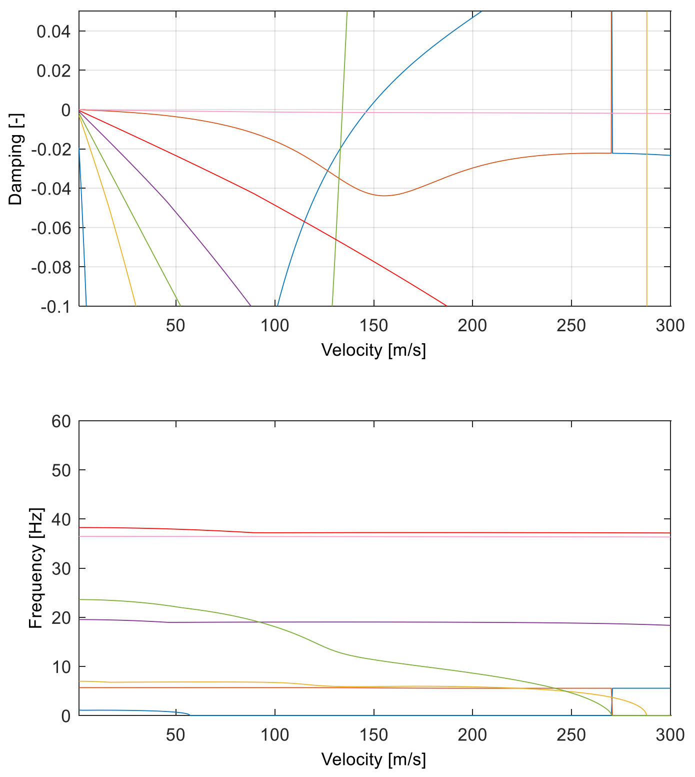
Figure 17 shows a comparison between 3 pairs of V-G and V-F plots corresponding to the wings of aspect-ratios 12, 20 and 28 at their respective flutter boundaries for an angle of attack of 6 degrees. Table 6 represents the identified vibration mode frequencies and their identified mechanisms for those different wings.

By analyzing the V-G and V-F plots (Figure 17), it is possible to estimate the flutter points. For the wing of AR = 12, flutter seems to occur when the fourth mode (first torsion) damping becomes positive and its frequency starts to converge with the one of the third mode (second flap), which suggests some mode coupling between these two modes. For the wing of AR = 20, flutter appears to occur when the fifth mode (first torsion) damping becomes positive and its frequency converges with the one of the fourth mode (third flap), suggesting some mode coupling between these two modes. The wing of AR = 28 seems to have a similar behavior as the wing of AR = 20, where flutter appears to occur when the fifth mode (first torsion) damping becomes positive and its frequency converges with the one of the fourth mode (third flap), suggesting some mode coupling between these two modes. Also, as expected, it can be observed that the predicted flutter speed decreases as the wing aspect-ratio increases. These graphs highlight the non-linear behavior of the mechanism leading to flutter.

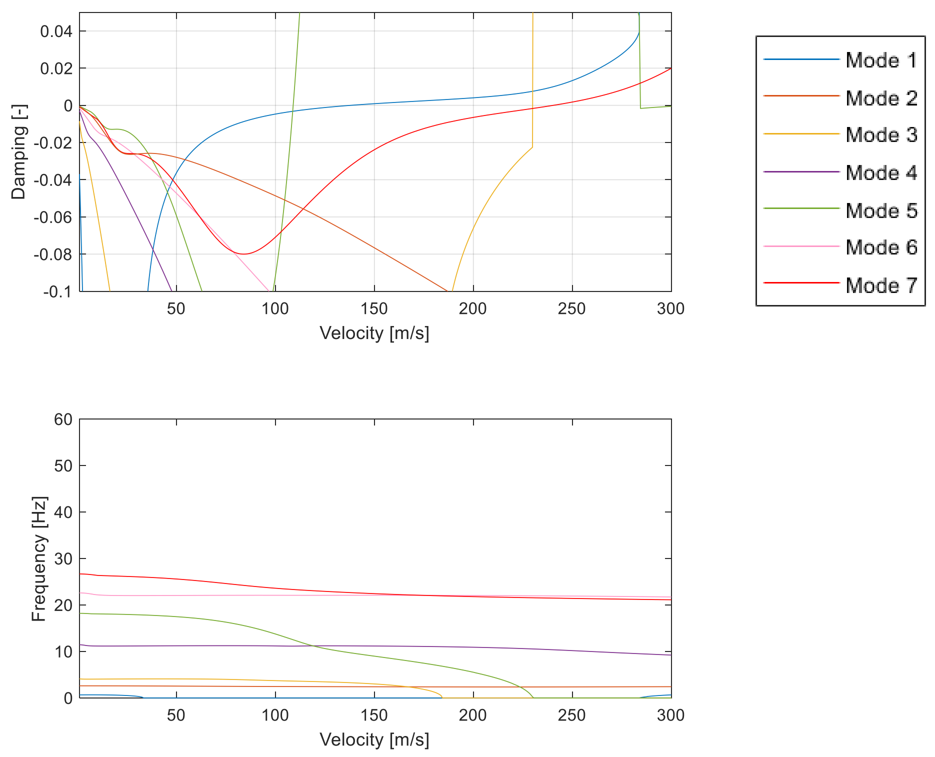
Regarding divergence, for all different wings, it is the first vibration mode that seems to present divergence, when its damping becomes positive while its frequency is null.



(a) AR = 12



(b) AR = 20



(c) AR = 28

Figure 17 - V-G and V-F plots at the flutter speed boundaries for the wings of aspect-ratios of 12, 20 and 28, at an angle of attack of 6 degrees

Mode	***AR* = 12**		***AR* = 20**		***AR* = 28**	
	Frequency [Hz]	*Vibration Mode*	*Frequency [Hz]*	*Vibration Mode*	*Frequency [Hz]*	*Vibration Mode*
1	2.39	1st Flap	1.11	1st Flap	0.67	1st Flap
2	9.46	1st Chord	5.78	1st Chord	3.46	1st Chord
3	14.91	2nd Flap	6.95	2nd Flap	4.16	2nd Flap
4	35.17	1st Torsion	19.44	3rd Flap	11.63	3rd Flap
5	41.65	3rd Flap	23.37	1st Torsion	19.76	1st Torsion

Table 6 - Natural frequencies and vibration mechanisms of the first few vibration modes of three wings with different aspect-ratios

Looking at Table 6, it can be noticed that in wings of aspect-ratio lower than 20, the first torsion mode is the fourth vibration mode and third flap mode is the fifth one. On wings of aspect-ratio of 20 or above, this tendency switches, and the third flap mode appears before the first torsion mode. The reason why we think that this happens is the fact that as the aspect-ratio increases, the inertia moments change along the wing, leading the wing to become weaker when resisting to flap bending. As expected, the wing becomes weaker as the aspect-ratio increases, and all the vibration frequencies decrease. The flap bending modes, however, are particularly weakened. And this is what we think that originates the observed switch between the first torsion and the third flap modes. In addition, it is relevant to note that in all the considered wing models the critical flutter vibration mode seems to be the first torsion mode.

Also from this study, some issues were found with the first chord mode for lower angles of attack, where it seemed to present flutter and some coupling with other modes. These issues with the first chord mode were damped as the angle of attack and the wing aspect-ratio were increased, indicating a possible numerical error, which requires an experimental study.

4.3. The Time Domain Method Results

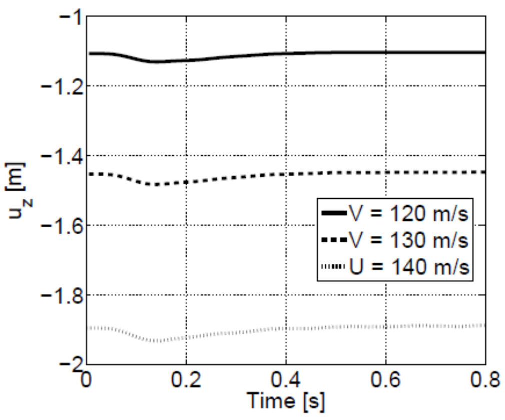
Regarding the time domain approach, the analyses that were carried out by the IST Aerospace Group considered a converged structural mesh of 50 nonlinear beam elements, as well as a converged aerodynamic mesh formed by 210 spanwise divisions and 40 chordwise divisions spaced using a cosine distribution, plus one fixed wake panel per each spanwise division of 100 chord lengths aligned with the wing chord direction. As previously mentioned, the nonlinear aeroelastic framework that feeds the frequency domain tool with the static equilibrium configurations, was now used to obtain the time domain solutions, recurring to a time-marching scheme. In order to disturb the wing structure and then estimate the flutter speed, a frontal triangular wind gust profile was applied at the nonlinear aeroelastic static equilibrium computed for different airspeeds. All wing models are subjected to the same triangular gust profile: starting at t = 0 s and ending at t = 0.05 s, reaching a maximum velocity of 5 m/s at its peak. The time step chosen was 0.01 s. The structural dynamic analyses were conducted using the Newmark method and a predictor-corrector scheme was used to assure aeroelastic convergence [15]. Even though the nonlinear aeroelastic response to above described gust profile was obtained for all wings and for several airspeeds, only three for each considered wing will be presented here. Figures 18 to 20 illustrate the dynamic response obtained at the wing tip in plunge u_z, pitch θ_y and chord u_x degrees of freedom. These figures show a common pattern: as the airspeed increases, the response to the applied gust becomes more oscillatory in plunge, pitch and chord degrees of freedom until a divergent motion is observed for the highest airspeed shown. It is also evident that there is a second oscillatory motion caused by the gust profile, which disperses for the lower airspeeds considered. Based on Figures 18 to 20, it is possible to infer that the critical flutter mode is the first torsion for the three considered wing models, which is in line with what was predicted by the frequency domain method. In addition, flutter was observed to occur as a coupling between the first torsion mode and a bending mode (although it was not clearly noticeable which bending mode it was). This is once again in agreement with what was predicted by the frequency domain method.

The damping ratio is then determined for plunge, pitch and chord degrees of freedom of the dynamic response before and after the divergent motion appears. By interpolating the airspeed *vs.* the damping ratio, it is possible to estimate the flutter speed as the speed for which the damping ratio is zero. In Table 7, there is a summary of the speed and dimensionless vertical tip displacement values at which flutter was predicted to occur, using the time domain method, for the three wings of different aspect-ratios.

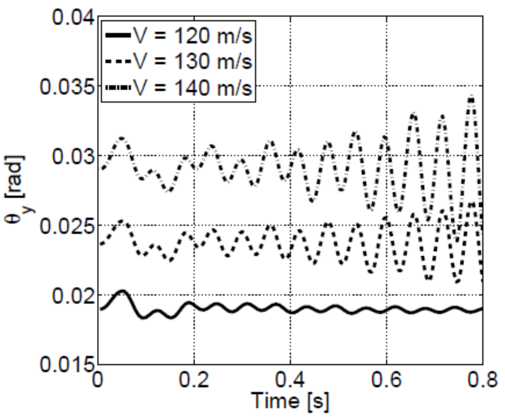
AR	***u_z/(b/2) [-]***	**Flutter Speed [m/s]**
12	0.1201	121.27
20	0.6274	113.17
28	0.6778	82.80

Table 7 - Predicted flutter speed and dimensionless wing tip displacement using the time domain approach

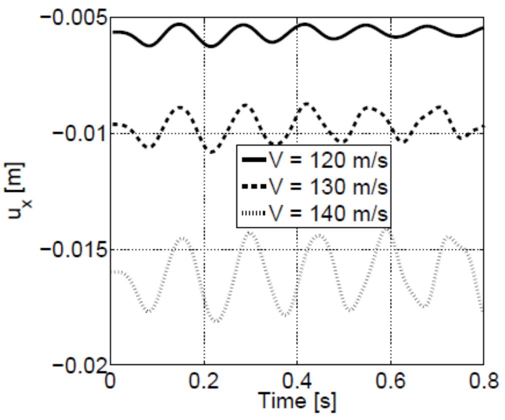
It is possible to observe that a trend emerges out of these findings: like in the frequency domain method, as the wing flexibility increases (as a consequence of the aspect-ratio increase), the flutter speed decreases. Additionally, as it happened in the frequency domain approach, flutter occurs for higher dimensionless tip displacements as the aspect-ratio increases. Again, this nonlinear behavior might be explained by the fact that the higher flexibility effect exceeds the reduction of the loads due to the lower airspeed.



(a) Plunge

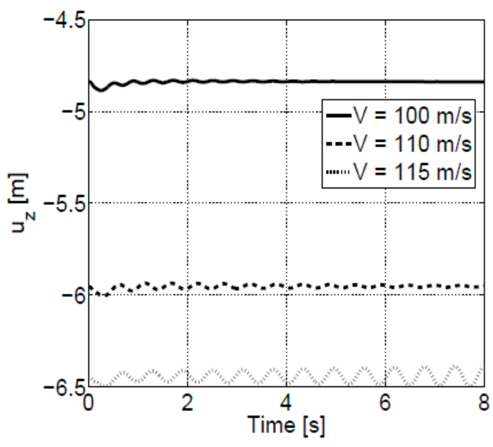


(b) Pitch

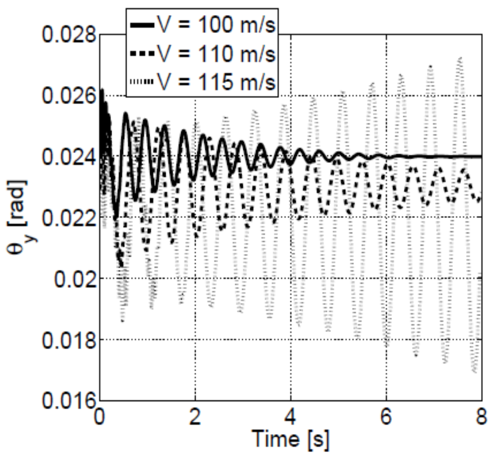


(c) Chord

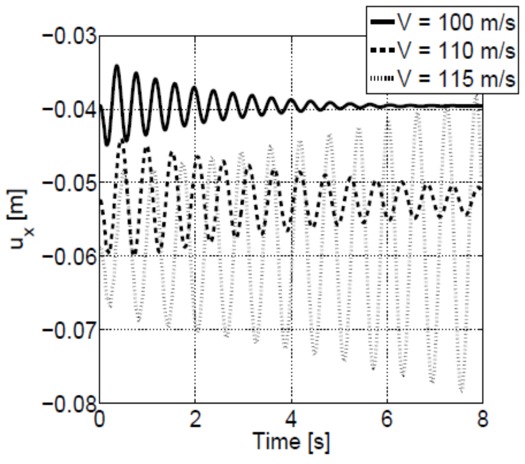
Figure 18 - Dynamic response of the 12 aspect-ratio wing measured at the wing tip in plunge, pitch and chord degrees of freedom for 3 different airspeeds (120 m/s, 130 m/s and 140 m/s) *[27]*



(a) Plunge

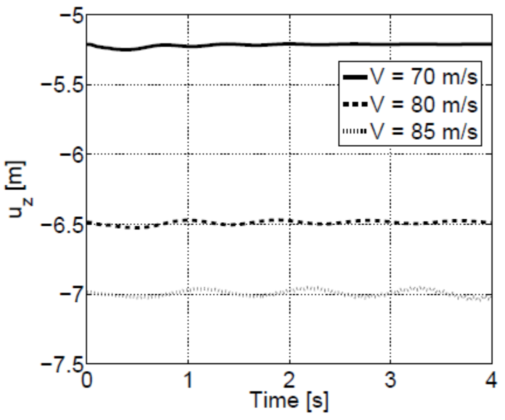


(b) Pitch

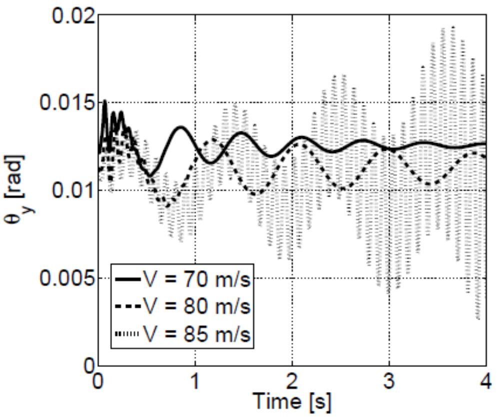


(c) Chord

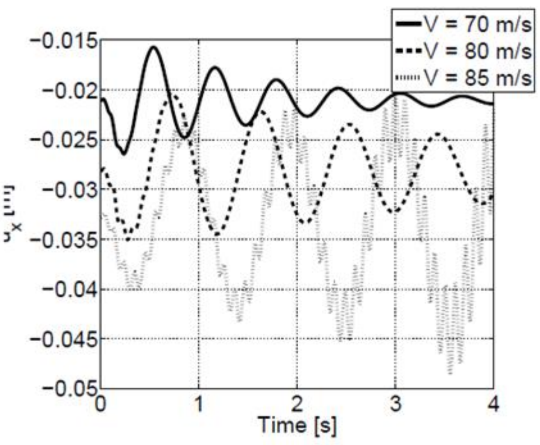
Figure 19 - Dynamic response of the 20 aspect-ratio wing measured at the wing tip in plunge, pitch and chord degrees of freedom for 3 different airspeeds (100 m/s, 110 m/s and 115 m/s) *[27]*



(a) Plunge



(b) Pitch



(c) Chord

Figure 20 - Dynamic response of the 28 aspect-ratio wing measured at the wing tip in plunge, pitch and chord degrees of freedom for 3 different airspeeds (70 m/s, 80 m/s and 85 m/s) *[27]*

4.4. Comparison Between Frequency and Time Domain Methods

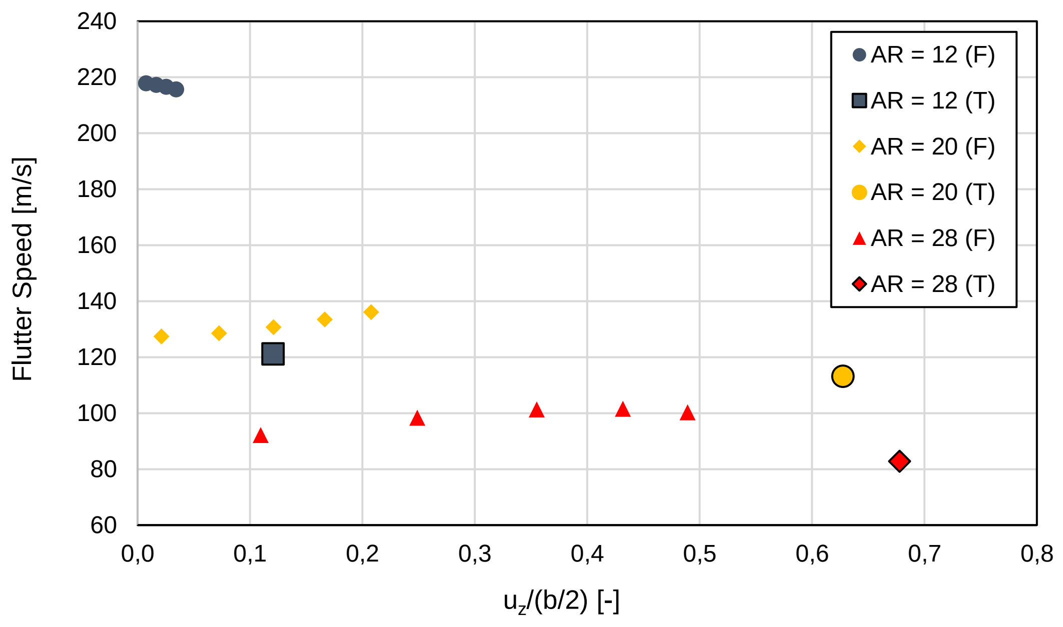


Figure 21 - Flutter speeds and corresponding dimensionless vertical wing tip displacement obtained through frequency and time domain methods

In Figure 21, the flutter results obtained using both frequency (F) and time (T) domain methods are depicted, where one can see the flutter speed variation as a function of the dimensionless vertical wing tip displacement for the wings with aspect-ratios of 12, 20 and 28.

As it was discussed in the previous sections, some trends behave alike in both the frequency and the time domain methods of calculation: the predicted flutter speed decreases as the wing flexibility increases (as a consequence of the aspect-ratio increase); flutter occurs at increasing dimensionless tip displacements as the aspect-ratio increases; the critical flutter mode seems to be the first torsion for all the considered wing models; and it could be said that the predicted flutter mechanisms are the same in both methods.

However, despite the flutter speed estimated using the frequency domain becoming closer to the one predicted by time domain method as the wing deformation increases, it was not possible to obtain exactly the same flutter speed in both approaches. Moreover, by analyzing Figure 21, it seems these predictions using the frequency domain approach may be less conservative than the ones obtained by means of the time domain method.

5. Concluding Remarks

As introduced in the first chapter, the importance of a fast and accurate method to estimate flutter speed in optimization and preliminary aircraft design problems has been a key driver in the use of frequency domain approaches to estimate the flutter speed. However, this approach needs to be carefully employed, especially when dealing with highly flexible high aspect-ratio wings, where geometrical nonlinearities such as large deformations are presented. Since these large wing deflections change the aeroelastic behavior, it is thus of paramount importance to adequately account for them on the flutter speed prediction.

In this work, a frequency domain method, where the wing aeroelastically static equilibrium state is used as an input instead of the undeformed wing, was presented and compared with a time domain method for the flutter speed prediction of high aspect-ratio wings. It could be concluded that some trends seemed to converge and were identified as behaving alike in both cases. However, despite the solutions obtained with the two methods becoming closer as the wing deformation increases, it was not possible to predict exactly the same flutter speed result in both techniques.

One aspect that might have helped to draw a better comparison between methods is if we would have been able to extract more data in the time domain method. Then, there would be more information in the graph of Figure 21, which could have helped us to better understand the behavior of the results in the time domain. Unfortunately, such was not possible due to numerical instabilities and convergence could not be reached above the presented angle of attack.

Overall, it could be concluded that the developed frequency domain tool managed to produce consistent results at a low computational cost, and it can prove useful in situations where its low computational cost would be relevant. For example, in optimization problems, the frequency domain tool could be used to evaluate relative predicted flutter speeds in a comparison between different wing models.

A suggestion for the future could be to refine the structural component of the frequency domain tool. For example, a change that could be interesting and easy to implement would be to replace the Euler-Bernoulli beam model with the Timoshenko one. A more complex upgrade would be to change the beam model for a plate model. The most laborious improvement to the process would probably be to generate experimental data in order to validate the flutter speed and flutter mechanism predictions.

References

[1] I. H. Abbott and A. E. Von Doenhoff, Theory of wing sections: including a summary of airfoil data, Mineola: Dover Publications, 1959.

[2] J. E. Cooper and M. Y. Harmin, “Aeroelastic behaviour of a wing including geometric nonlinearities,” *Aeronautical Journal,* vol. 115, no. 1174, pp. 767-777, 2011.

[3] J. R. Wright and J. E. Cooper, Introduction to aircraft aeroelasticity and loads, Chichester, UK: John Wiley & Sons, 2007.

[4] P. Mahamuni, A. Kulkarni and Y. Parikh, “Aerodynamic study of blended wing body,” *International Journal of Applied Engineering Research,* vol. 9, no. 24, pp. 29247-29255, 2014.

[5] J. Wolkovitch, “The joined wing: An overview,” *Journal of Aircraft,* vol. 23, no. 3, p. 161–178, 1986.

[6] E. Ting, K. Reynolds, N. Nguyen and J. Totah, “Aerodynamic analysis of the Truss-Braced wing aircraft using Vortex-Lattice superposition approach,” in *32nd AIAA Applied Aerodynamics Conference*, Atlanta, USA, 2014.

[7] F. Afonso, J. Vale, E. Oliveira, F. Lau and A. Suleman, “A review on the non-linear aeroelasticity of high aspect-ratio wings,” *Progress in Aerospace Sciences,* vol. 89, pp. 40-57, 2017.

[8] C. E. S. Cesnik and E. L. Brown, “Modeling of a high aspect ratio active flexible wings for roll control,” in *Proceedings of 43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Denver, Colorado, USA, 2002.

[9] M. J. Patil and D. H. Hodges, “Flight dynamics of highly flexible flying wings,” *Journal of Aircraft,* vol. 43, no. 6, p. 1790–1799, 2006.

[10] Z. Wang, P. Chen, D. Liu and D. Mook, “Nonlinear-aerodynamic/nonlinear-structure interaction methodology for a high-altitude-long-endurance wing,” *Journal of Aircraft,* vol. 47, no. 2, p. 556–566, 2010.

[11] J. Murua, R. Palacios and J. M. R. Graham, “Applications of the unsteady vortex-lattice method in aircraft aeroelasticity and flight dynamics,” *Progress in Aerospace Sciences,* vol. 55, p. 46–72, 2012.

[12] F. Petrini, F. Giuliano and F. Bontempi, "Comparison of time domain techniques for the evaluation of the response and the stability in long span suspension bridges.," *Computers & Structures,* no. 85, p. 1032–1048, 2007.

[13] L. Salvatori and C. Borri, “Frequency- and time-domain methods for the numerical modeling of full-bridge aeroelasticity,” *Computers & Structures,* no. 87, pp. 675-687, 2007.

[14] L. Salvatori and P. Spinelli, “Effects of structural nonlinearity and along-span wind coherence on suspension bridge aerodynamics: some numerical simulation results,” *Journal of Wind Engineering and Industrial Aerodynamics,* vol. 94, no. 5, p. 415–430, 2006.

[15] A. Suleman, F. Afonso, J. Vale, E. Oliveira and F. Lau, “Non-linear aeroelastic analysis in the time domain of high-aspect-ratio wings: Effect of chord and taper-ratio variation,” *The Aeronautical Journal,* vol. 121, no. 1235, pp. 21-53, 2017.

[16] F. Afonso, G. Leal, J. Vale, E. Oliveira, F. Lau and A. Suleman, “The effect of stiffness and geometric parameters on the nonlinear aeroelastic performance of high aspect ratio wings,” *Proc IMech E Part G: J Aerospace Engineering,* vol. 231, no. 10, pp. 1824-1850, 2017.

[17] S. J. Hulshoff, Aeroelasticity (version 11.1): AE4930, Delft: TU Delft, Faculty of Aerospace Engineering, 2011.

[18] H. J. Hassig, “An approximate true damping solution of the flutter equation by determinant iteration,” *Journal of Aircraft,* vol. 8, no. 11, p. 885–889, 1971.

[19] J. N. Reddy, An introduction to the finite element method, 3rd. ed. ed., New York: McGraw-Hill Education, 2005.

[20] S. Raghu, Finite Element Modeling Techniques in MSC.NASTRAN and LS/-DYNA, CreateSpace Independent Publishing Platform, 2010.

[21] E. Albano and W. P. Roden, A Doublet-Lattice method for calculating lift distributions on oscillating surfaces in subsonic flows, Hawthorne, USA: Northrop Corporation, Norair Division,, 1969.

[22] J. P. Giesing, T. P. Kalman and W. P. Rodden, "Part I, Vol. I: Direct Application of the Nonplanar Doublet-Lattice Method," in *Subsonic unsteady aerodynamics for general configurations. Air Force Flight Dynamics Laboratory Report No. AFFDL-TR-71-5*, Vols. 1, Parte I, 1971.

[23] MSC Software, MSC.NASTRAN Version 68 Aeroelastic Analysis User’s Guide, MSC Software, 2004.

[24] MSC Software, DMAP programmer's guide, MSC Software.

[25] MSC Software, Getting Started with MSC NASTRAN - User’s Guide, MSC Software, 2012.

[26] MSC Software, MSC NASTRAN 2012 - Quick Reference Guide, MSC Software, 2012.

[27] F. Afonso, J. Vale, E. Oliveira, R. Correia, F. Lau and A. Suleman, “Comparison between frequency and time domain approaches for estimation of flutter speed in high aspect-

ratio wings," in *International Conference on Structural Engineering Dynamics*, Ericeira, Portugal, 2017.