University of Leeds

Aeronautical and Aerospace Engineering BEng

Individual Engineering Project

Final report

Rocket design and analysis in support of Leeds University Rocketry Association (LURA):

Flutter analysis and its prevention in rocket fins

Author:

Paula Perez Morgado

mn21ppm@leeds.ac.uk

paupemor@etsid.upv.es

Tutors:

External Tutor, UoL - Gregory de Boer

G.N.deBoer@leeds.ac.uk

Tutor, UPV - José Martínez Casas

jomarc12@upv.edu.es

Cotutor, UPV - Víctor Tomás Andrés Ruiz

vicanrui@upv.edu.es

MECH3890 Individual Engineering Project

Academic year: 2021/2022

Leeds, United Kingdom.

29th July 2022

Abstract

Within the field of the aeroelasticty, concretely in the dynamic one, the flutter is the most important phenomenon to consider during the first steps in a rocket design. Its prediction is not simple at all, but the catastrophic structural consequences it supposes make that it must be estimated and controlled as much as possible. That is why the phenomenon of flutter is chosen as the main subject to study throughout this final engineering project. Nonetheless, the divergence one is also presented since the structural failures it involves are not in the least less significant.

For that reason, a graphical user interface where it is possible to input the data of the fins that are being used in a rocket and get a first estimation of the velocities at which both phenomenons will occur is created. To this effect, the mathematical background of flutter is firstly worked out to reach their governing equations. Later on, those are implemented in MATLAB, creating a set of functions which will be finally used by the Simulink model, the MATLAB's toolbox where the user interface is developed. Lastly, a validation process is performed with the NACA Report No.685 and the well-known AeroFinSim simulator, where the inlfuence of different parameters is analysed.

As this project is intended to support the Leeds University Rocketry Association, the Simulink model is finally used with the data of the fins they are using on its current design. Thus, the flutter and divergence values they are considering at the moment are compared with the ones obtained in the Simulink model and the differences between them are discussed.

Contents

List of Figures

1 Introductory chapter

1.1 Introduction

Stability is one of the most crucial criteria to consider when designing a rocket. Apart from reaching high speeds during launch, its orientation and intended flight plan must be kept avoiding wobbling and tumbling. This is what is known as stability [8] and it must be conserved as much as possible, averting any threat that infringe upon it.

Fins are allocated on the rocket's tail to ensure stability. They try to guarantee the centre of pressure (green point) is always behind the gravity one (blue point), so that the generated aerodynamic forces act as restoring forces in response to any undesirable rotation the rocket can suffer from. Fig.1 [8] shows how those restoring forces acts depending on the locations of both centres.

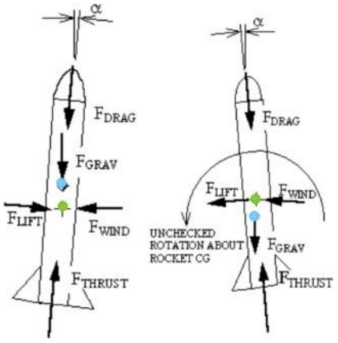
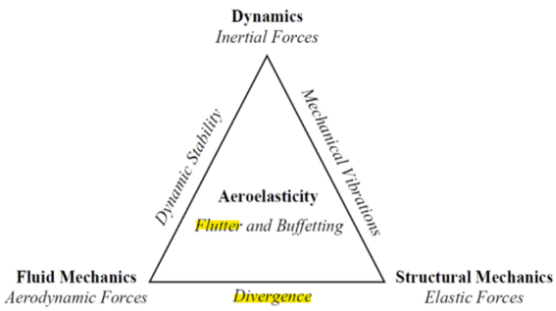


Figure 1: C_P and C_G

However, they are designed to work stably within a certain range of speeds. Beyond this range, fins can start to fiercely vibrate because of the coupling of two modes of vibration in the aeroelastic system. From this point, the air will act as damping reductor since the damping speed will no longer increase with flight velocity [6]. That would result in their permanent deformation and retaking the control would be practically impossible, leading to catastrophic structural failures [1].

Something similar happens when the divergence phenomenon occurs. In this case, the generated aerodynamic moments exceed the fin structural stiffness, being impossible to recover an equilibrium state then. The structure cannot withstand such aerodynamic loads and will be finally brought to the failure [11].

Because of that, the rocket fins flutter and divergence control must be a priority when dealing with stability, so aeroelasticity has to be seriously studied. To achieve this, the connection between the 3 main disciplines shown in the vertices of the Collar's triangle of forces of Fig.2 must be mathematically analysed, as it is performed in the next section.



It can be seen how the divergence stem from the combination of the aerodynamic and elastic forces, found within the static aeroelasticity. However, the flutter phenomenon also incorporates the inertial forces, involving the problem into the dynamic aeroelasticity [5].

Figure 2: *Collar's triangle of forces (Aerospace Engineering Blog, 2017)*

1.2 Aim of the project

Development of a complete aeroelastic simulator through the toolbox of Simulink which allows to obtain the vibratory response of the user-defined rocket fin to predict when the catastrophic flutter and divergence phenomena will occur. The flutter and divergence speeds and frequencies, as well as the evolution of the frequency of each mode of vibration when the flight speed is increased, will be the principal outcomes of that simulator.

1.3 Objectives

The principal objectives were updated at the beginning of the second semester, so small changes can be observed if they are compared with the ones written in the scoping and planning document. They are listed below, where the milestones are underlined:

1. Literature review
   (a) Introduction to aeroelasticity from literature.
   (b) Study of the differences between the static and the dynamic aeroelasticity, focusing on the dynamic one since flutter is found within it.
   (c) Investigate about the different possible approaches to solve the flutter problem.
2. Search of different numerical methods that could be used to get the implementation of the flutter governing equations into MATLAB, noting the possible limitations and the needed input values for each one.
3. Calculate the frequency of each of the vibration modes of the problem.
4. Identify the flutter and divergence speeds and frequencies.
5. Development of an aeroelastic simulator using the toolbox of Simulink which shows the vibratory response of the rocket fin that is being studied.
   (a) Think about the inputs that are needed for the simulation.
   (b) Determination of the outputs that the simulator will generate. They must be quite useful so that the user is able to get an idea of the velocity at which the fin will flutter and the divergence speed in a quick look.
6. Trade-off between the input parameters and the flutter results to find the most suitable conditions in terms of avoiding flutter effects.
7. Validation process with CFD simulations: design, mesh, and post-processing.

1.4 Report layout

The report starts with a brief introduction in order to make the reader aware of the importance of the fin flutter when designing a rocket. This is followed by the mathematical background where the flutter and divergence governing equations are reached. Next, they are implemented in a MATLAB code which is explained through a set of helpful flow diagrams. Then, the Simulink model is described, which results will be finally validated. Lastly, the flutter condition of the fins that LURA is currently using is deeply studied.

2 Mathematical background

During this chapter, all the essential governing equations of the flutter phenomenon are developed. For that, two of the most relevant books in the aeroelasticity field, [2] and [3], together with the lecture notes of an excellent university professor from the Universidad Politécnica de Valencia, [9] and [10], were studied in detail to finally be able to critically analyse and synthesize all the following mathematical developments.

The main objective is the study of the combination of the three kind of forces that come simultaneously into play in the vibration problem of flutter. For that, the energy method will be applied in an aerofoil simulating the fin of a rocket, leading to the Euler-Lagrange motion equations. First of all, the flutter phenomenon is represented through the two degrees of freedom system shown in Figure 3.

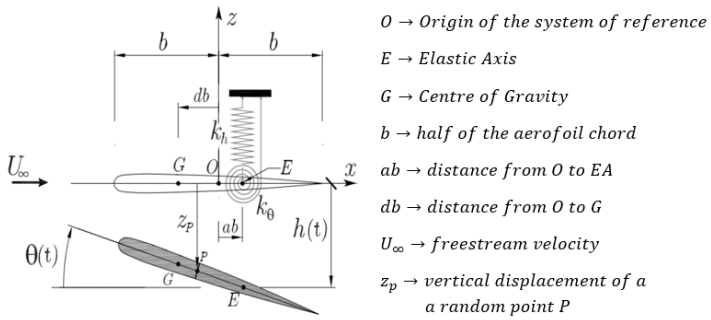


Figure 3: *Two degrees of freedom system used to represent the flutter phenomenon [9]*

It can be observed that the two degrees of freedom are the vertical displacement, $h(t)$, and the turn around the E, $\theta(t)$, being positive downwards and clockwise, respectively. Those represent the elastic of bending and torsion in real fins but in a simplified way. Thus, it is considered that both, the bending and torsion stiffness, k_h and k_θ, respectively, are concentrated in the E where the two elastic and lineal springs are drawn.

2.1 Equations of motion

The two degrees of freedom are the unknowns of the problem and they are bound together in a dimensionless vector $\boldsymbol{u} = \left\{h(t)/b, \theta(t)\right\}^T$. Now, the resulting differential system of Euler-Lagrange equations can be expressed as:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\boldsymbol{u}}}\right) + \frac{\partial D}{\partial \dot{\boldsymbol{u}}} + \frac{\partial U}{\partial \boldsymbol{u}} = \boldsymbol{Q}(t) \tag{1}$$

where T, U and D stand for the kinetic and potential energy, and Rayleigh dissipative potential, and potential energy, respectively.

2.1.1 Dynamic matrices

The matrices that appear in the analyses of free vibration in a mechanical system are usually known as dynamic matrices. They are the mass, stiffness, and damping matrices. However, the expression of the last one is not needed since the damping effect is not being considered in the analysis. The other two are shown below, in Eq.2 and Eq.3.

It can be seen how the mass ($\boldsymbol{M}$) and stiffness ($\boldsymbol{K}$) matrices are expressed as a function of two dimensionless matrices, $\boldsymbol{\mathcal{M}}$ and $\boldsymbol{\mathcal{K}}$, to ease further calculations. For that, the dimensionless parameters written to the right of their expressions are defined, being I_θ the radius of gyration about E, r_θ the dimensionless distance between G and E, and η the frequency ratio between the torsional (ω_θ) and the bending (ω_h) frequencies. In turn, I_E stands for the moment of inertia with respect to E. To see the process of how those matrices were obtained, please refer to Appendix A.1.

$$mass \rightarrow \boldsymbol{M} = mb^2 \begin{bmatrix} 1 & r_\theta \\ r_\theta & i_\theta^2 \end{bmatrix} = mb^2 \boldsymbol{\mathcal{M}} \qquad r_\theta = d - a,\ i_\theta = \sqrt{\frac{I_E}{mb^2}},\ r = \frac{r_\theta}{i_\theta} \quad (2)$$

$$stiffness \rightarrow \boldsymbol{K} = mb^2\omega_\theta^2 \begin{bmatrix} \eta^2 & 0 \\ 0 & i_\theta^2 \end{bmatrix} = mb^2\omega_\theta \boldsymbol{\mathcal{K}} \quad \eta = \frac{\omega_\theta}{\omega_h},\ \omega_\theta = \sqrt{\frac{k_\omega}{m}},\ \omega_h = \sqrt{\frac{k_h}{I_E}} \quad (3)$$

2.1.2 Generalised forces

Next step is analysing the external forces which depend on the system deformation according to the dynamic stability analysis that is being carried out. For that, the forces which are function of $\boldsymbol{u}$(t), their velocities $\dot{\boldsymbol{u}}$(t), and accelerations $\ddot{\boldsymbol{u}}$(t), must be studied.

The principle of virtual work or the virtual work method (Gavin, 2012) is applied. Thus, the generalised force associated to a determined degree of freedom is the virtual work performed by those external forces per unit or virtual variation. In such a way, it is possible to express the virtual work of a system as a linear combination like:

$$\delta\mathcal{W} = \{\delta h/b,\ \delta\theta\}\{Q_1, Q_2\}^T = \ \delta\boldsymbol{u}^T\boldsymbol{Q} \quad (4)$$

Hence, if the expression of the virtual work performed by the lift and the aerodynamic moment that the aerofoil experiences is obtained, it will be possible to solve for the generalised forces vector $\boldsymbol{Q}(t)$. The steps followed to reach this vector are not essential, so they are explained in App. A.2. Then, the final expression is shown below, where $\boldsymbol{Q}(t)$ is expressed as a function of $\boldsymbol{u}$, $\dot{\boldsymbol{u}}$(t), and $\ddot{\boldsymbol{u}}$(t):

$$\boldsymbol{Q}(t) = \pi\rho_\infty U_\infty^2 b^2 \boldsymbol{A}\boldsymbol{u}(t) + \pi\rho_\infty U_\infty b^3 \boldsymbol{B}\dot{\boldsymbol{u}}(t) + \pi\rho_\infty b^4 \boldsymbol{C}\ddot{\boldsymbol{u}}(t) \quad (5)$$

Eq.5, the coefficients of which are worked out in App.A.2, represents the most complete unsteady aerodynamic model where the next time dependency sources are considered:

- Relative motion between the aerofoil and the air flow.
- Unsteady nature of the pressure coefficient reflected in the terms related to the apparent mass.
- The eddies wake effect in the aerofoil through the Theodorsen function $\mathcal{C}(\mathcal{K})$, depending this on the value of the reduced frequency $\mathcal{K}$, which defines the degree of unsteadiness of the problem.

2.2 Solution of the dynamic problem

In order to address this dynamic problem, different assumptions can be made regarding the generalised forces. Then, different models of aerodynamic forces, which are deepened hereafter, arise. The key point is bearing the consequences of those assumptions in mind, and therefore the limitations of each model.

2.2.1 Steady aerodynamic forces

This is the simplest aerodynamic model since the terms related to the velocities and accelerations of the degrees of freedom are neglected. Consequently, the reduced frequency mus be $\mathcal{K} \ll 1$, what means that the oscillations must be quite slow with respect to the air flow velocity. Therefore, this model is ruled by the lineal differential system of equations shown in Equation 6, where only the first term of $\boldsymbol{Q}$ is conserved. Moreover, some initial conditions are considered for both degrees of freedom, $\boldsymbol{u_0}$, and their velocities,$\boldsymbol{v_0}$:

$$\begin{cases} \boldsymbol{M}\ddot{\boldsymbol{u}} + \boldsymbol{K}\boldsymbol{u} = \pi\rho_\infty U_\infty^2 b^2 \boldsymbol{A}\boldsymbol{u} \\ \qquad \boldsymbol{u}(0) = \boldsymbol{u_0}, \quad \dot{\boldsymbol{u}}(0) = \boldsymbol{v_0} \end{cases} \tag{6}$$

Looking for solutions of the kind $\boldsymbol{u}(t) = \bar{\boldsymbol{u}}e^{iwt}$, the eigenvalue problem shown in Eq.7 is reached. To see how to reach it in detail, refer to App.A.3. Next, solving the determinant of the expression between brackets for a wide range of dimensionless flight speeds V_∞, the dimensionless frequencies $\lambda = \Omega + ig$ of the torsion and bending modes, which may be complex, are obtained and can be plotted as it is shown in Fig.4.

$$\left[-\lambda^2\mathcal{M} + \boldsymbol{\mathcal{K}} - \frac{V_\infty^2}{\mu}\boldsymbol{A}\right]\bar{\boldsymbol{u}} = 0 \qquad where: \; V_\infty = \frac{U_\infty}{bw_\theta} \tag{7}$$

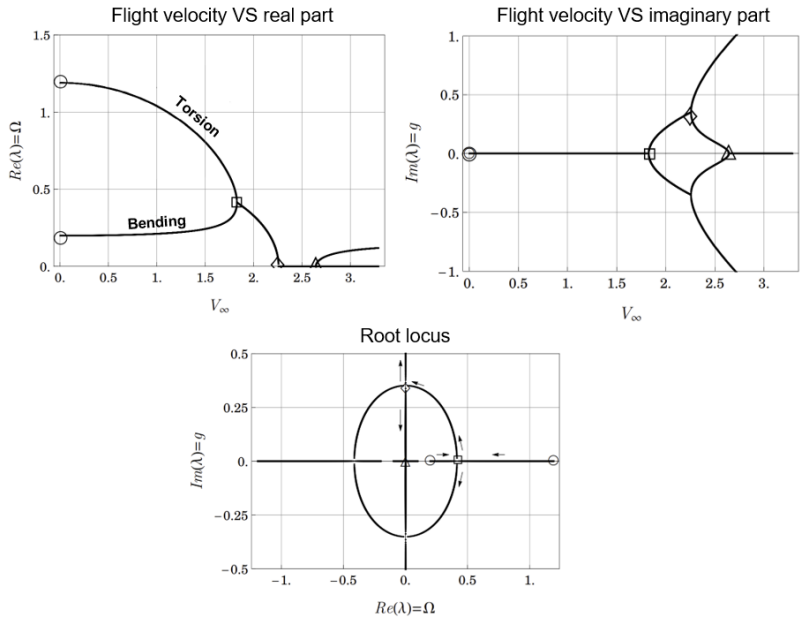


Figure 4: *Particular case solution for* $a = -0.2$, $\mu = 30$, $i_\theta = 0.611, d = 0, \eta = 0.2$ [9]

The flutter point (square mark), known in this model as the *coalescence* point, is found from the graph at the top right of Fig.4, just when the torsion and bending frequencies match. Before this point, the frequencies were real, making the system stable and being the torsion mode the one with highest frequencies. However, from this point the imaginary parts are no longer zero and, therefore, the instability begins. In turn, the divergence point (triangular mark) is located where the frequencies of both modes are zero, that is, when the system does not oscillate. Finally, just to remark that the natural frequencies are marked with circumferences. For a deeper analysis about the graphs, please read App.A.3.1.

Flutter velocity expression (V_f)

If the expression of the determinant of Eq.7, an incomplete 4^th order polynomial expression is obtained, which coefficients are:

$$D(\lambda) = \lambda^4 + R(V_\infty)\lambda^2 + S(V_\infty) \tag{8}$$

$$R = \frac{(1+2d)\frac{V_\infty^2}{\mu} - i_\theta^2(1+\eta^2)}{i_\theta^2(1-r^2)} \qquad S = \frac{i_\theta^2 - (1+2a)\frac{V_\infty^2}{\mu}}{i_\theta^2(1-r^2)}\eta \tag{9}$$

Then, just solving the quadratic function with the easy writing, one obtains:

$$\lambda_{1,2}^2 = \frac{-R \pm \sqrt{R^2 - 4S}}{2} \tag{10}$$

Both frequencies match at the flutter, so it can be stated that $\lambda_1 = \lambda_2$. Thus, from Eq. 10, the expression for the flutter frequency is obtained just making zero the term inside the square root:

$$\lambda_1 = \lambda_2 = \lambda_f \;\rightarrow\; R^2 - 4S = 0 \;\rightarrow\; \lambda_f = \sqrt{-\frac{R}{2}} = \sqrt{\frac{(1+2d)\frac{V_\infty^2}{\mu} - i_\theta^2(1+\eta^2)}{-2i_\theta^2(1-r^2)}} \tag{11}$$

Finally, the first order approximation theory will be used to find the flutter velocity associated to the previous flutter frequency. The resulting expression is noted below in Eq.12, but the whole development is found in App.A.3.2. It is remarked that this approximation gives enough accurate results within the range $0 \leq \eta \leq 0.5$ since it is being considered $\eta \ll 1$.

$$V_f \approx \sqrt{\mu\frac{i_\theta^2}{1+2d}\left[1 - 2\eta\frac{i_G}{i_\theta}\sqrt{\frac{2(d-a)}{1+2d}}\right]} \qquad where: \; i_G = i_\theta\sqrt{1-r^2} \tag{12}$$

Divergence velocity expression (V_D)

After the flutter condition, it arrives other flight speed for which the determinant $D(\lambda)$ becomes zero when the frequency is zero too ($\lambda = 0$, no oscillations). Expressing that in a mathematical way, the divergence velocity can be easily obtained solving the simplified determinant shown below in Eq.13:

$$det\left[\mathcal{K} - \frac{V_\infty^2}{\mu}\boldsymbol{A}\right] \quad \rightarrow \quad \frac{V_D^2}{\mu} = \frac{i_\theta^2}{1+2a} \tag{13}$$

2.2.2 Quasi-stationary aerodynamic forces

This model is improved with respect to the steady one since four new aerodynamic derivatives are introduced, which are considered to do not depend directly on the reduced frequency $\mathcal{K}$, but by means of the Theodorsen function. Moreover, the velocity terms of $\dot{h}$ are conserved in this case whereas the acceleration ones are neglected again. Thus, the system of equations that govern this model is the one shown below, where the matrix coefficients have slightly changed due to the incorporation of these new derivatives:

$$\begin{cases} \boldsymbol{M}\ddot{\boldsymbol{u}} + \boldsymbol{K}\boldsymbol{u} = \pi\rho_\infty U_\infty^2 b^2 \boldsymbol{\mathcal{A}}\boldsymbol{u} + \pi\rho_\infty U_\infty b^3 \boldsymbol{\mathcal{B}}\dot{\boldsymbol{u}} \\ \boldsymbol{u}(0) = \boldsymbol{u_0}, \quad \dot{\boldsymbol{u}}(0) = \boldsymbol{v_0} \end{cases} \tag{14}$$

To check the expressions of these new matrices $\mathcal{A}$ and $\mathcal{B}$ and the process of how obtaining the new aerodynamic derivatives ($C_{l_\theta}, C_{l_{\dot{\theta}}}, C_{m_\theta}, C_{m_{\dot{\theta}}}$), please refer to App.A.4.2.

Again, looking for solutions of the kind $\boldsymbol{u}(t) = \bar{\boldsymbol{u}}e^{iwt}$ and making some key arrangements:

$$\left[-\lambda^2\mathcal{M} + \mathcal{K} - \frac{V_\infty^2}{\mu}\mathcal{A} - \frac{i\lambda V_\infty}{\mu}\mathcal{B}\right]\bar{\boldsymbol{u}} = 0 \tag{15}$$

Calculating the determinant of the expression above for a wide range of speeds and equalling it to zero, the pursued set of eigenvalues (frequencies) is reached and, therefore, the three typical graphs can be represented. As it can be checked in Fig.5, the flutter speed is not determined by a *coalescence* point. In this model, flutter speed is found when the maximum velocity for which the imaginary part of both modes is positive is reached. Regarding the divergence one, it is located in the same way as before:

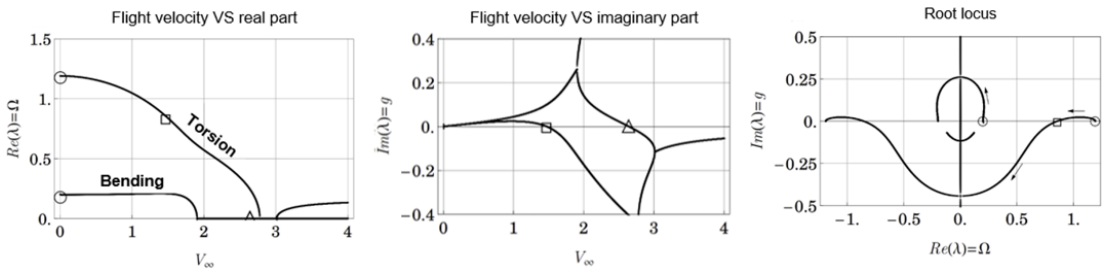


Figure 5: *Solution for the particular case of:* $a = -0.2$, $\mu = 30$, $i_\theta = 0.611, \eta = 0.2$, $d = 0$, $C_{l\theta} = 2\pi$, $C_{m\theta} = 0.3142$, $C_{l\dot{\theta}} = 6.254$, $C_{m\dot{\theta}} = -0.6235$ ***[9]***

Flutter velocity ($V_{f_{Theo}}$) - Theodorsen's method

This method is highly recommended when the degrees of freedom are less than four. Otherwise, miles of terms would appear and it would be inefficient. It is known that the flutter frequency will not have imaginary part, so it can be stated that $\lambda_f = \Omega_f$. Replacing that in Eq.15 and calculating the determinant expression, complex numbers are expected. Then, as $\bar{\boldsymbol{u}}$ cannot be zero, the imaginary and the real part of that expression must be zero. To get a mathematical perspective of it, please read App.A.4.3.

3 MATLAB code

This section is dedicated to the explanation of the different MATLAB functions which will be used to develop the user interface in Simulink. In order to avoid the reader get lost or needs to have some basic knowledge about programming language, those functions are easily explained through flow charts. Thus, it is possible to understand what and how has been programmed without having to read the code. To check the actual MATLAB scripts, please refer to Appendix B, where a screenshot of each function is included.

Before describing each function, it is essential to make the colour coding followed in the next flow diagrams clear:

- Rectangular blue blocks: input parameters. It is specified where they come from or if they are introduced by the user.
- Rectangular green blocks: output parameters achieved after a group of instructions.
- Diamond yellow blocks: represent conditional statements which are followed by a pair of instructions depending on whether the statement is true or false.
- Rectangular grey (white filled) blocks: instructions and mathematical operations.

3.1 Initial function - Dimensionless parameters

This function is in charge of calculating the dimensionless parameters presented in the green block based on the user input parameters (blue block). In the middle block, the expressions used to compute them are just shown since they have been already developed during the mathematical background of Section 2.

It can be observed the user will be able to choose which value to introduce, i_θ or I_E. If the first one is zero (the user does not introduce it), it will be calculated through the I_E value. By contrast, if a value of i_θ is indicated, it will be directly passed to the output section regardless the value of I_E since this will not be needed.

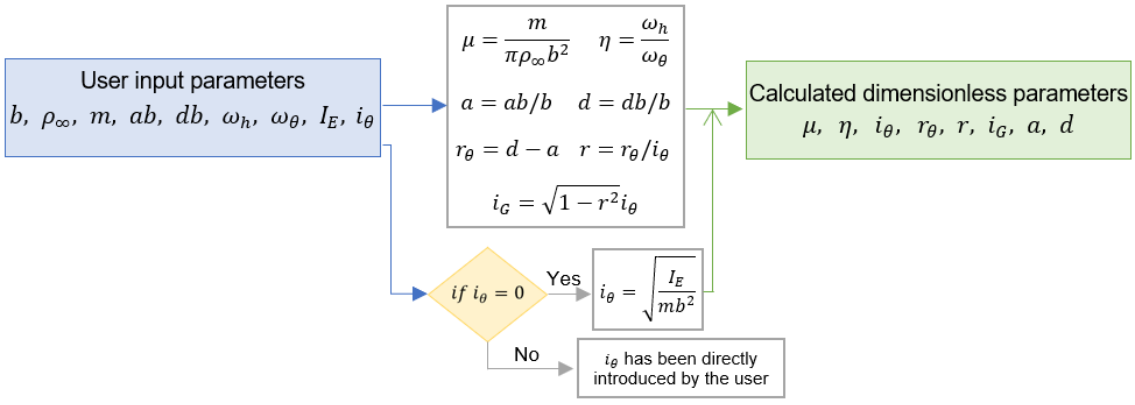


Figure 6: *Initial function flow chart. Dimensionless parameters calculation.*

3.2 Steady aerodynamics functions

3.2.1 Frequency vectors - Steady approach

Starting with the steady aerodynamic model, this function creates and fills the vectors with the modes of vibration of the problem to finally be able to plot them versus the dimensionless flight speed V_∞. Then, its purpose lies in generating the data needed to obtain the graphs of Fig. 4 and 5.

It can be seen in Fig. 7 that the input data are the dimensionless parameters obtained with the previous function (Sec. 3.1) and the upper bound of V_∞, introduced by the user. This last one must be changed when the graph is not fully represented because flutter occurs at higher speeds.

The first loop travels along the vector V_∞, calculating for each of its elements the values of R and S, and storing them in two vectors. Their expressions, which are clearly flight speed dependant, were shown in Eq. 9.

Once that first loop ends, R and S get completely filled, with the same length as V_∞. Thus, the coefficients for the polynomial equation p (Eq.8) are ready. Then, a nested for loop starts, involving two changing variables, i and j. It is time to compute the roots of p for each combination of $R(i)$ and $S(i)$. As it is a fourth-degree equation, there will be 4 solutions, one for each mode of vibration, which are supposed to always appear in the same order. Therefore, they are stored in two $4\text{x}(length(V_\infty))$ matrices, filling each row j with the j^{th} solution given by the instruction $roots(p)$.

As it is observed in the flow diagram below, two matrices are used because Simulink does not easily deal with complex numbers. As the solutions are expected to be complex, it is better that one matrix stores the real part and, the second matrix, the imaginary part. In such a way, complex numbers and any kind of error related to them are avoided.

The same happens when defining empty vectors. Simulink does not understand their size and stops running the corresponding MATLAB function. That is why vectors are defined through the instruction $zeros(rows, columns)$. Thus, instead of filling vectors, it is just a matter of replacing those zeros by the values wanted to be saved.

Coming back to the flow diagram, and being the nested loop already completed, the assignment of the frequencies stored in those matrices, $vector_{roots_{real}}$ and $vector_{roots_{imag}}$, to the 4 modes of vibrations is performed. These instructions are shown on the big grey rectangle on the right hand side of the chart. The real and the imaginary parts are kept separate to avoid coding errors, what will be finally advantageous to make the graphs.

It is known that, until reaching the coalescence point, the frequencies are real and of the kind $\{\Omega_1, -\Omega_1, \Omega_2, -\Omega_2\}$, and that the $roots$ function gives them in that order, being the first one the highest one. Therefore, as the torsion mode is the one with larger frequencies at the beginning, the first row of that matrices can be assigned to the first torsion mode. Consequently, the second row will be allocated to the second torsion mode, the third row to the first bending mode and, finally, the fourth row to the second bending mode. Thus, the 8 vectors containing the real and imaginary parts of the torsion and bending frequencies are reached.

However, sometimes the $roots$ function stops giving the first torsion frequency in the first position and the order is totally altered. For that reason, a control check should be carried out to those 8 vectors to detect any data discontinuity. The process of how to do it in detail is shown in the screenshot of the code of App.B.2, but it is basically based on locating the positions in which the data changes significantly in two or more vectors at the same time. Graphically, the user can realise about it since sudden vertical lines will appear in the plots. When this occurs, data must be interchanged between them in order to keep the frequencies of the modes in their corresponding vector.

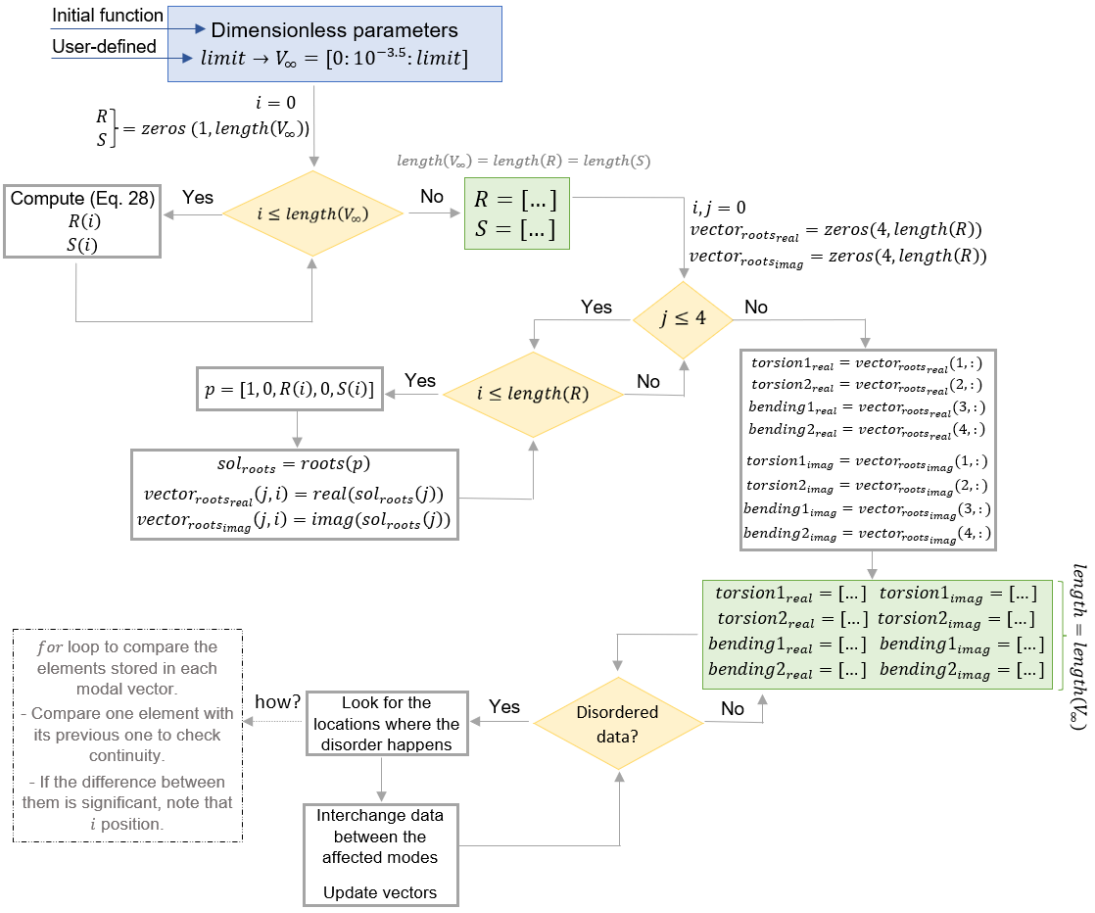


Figure 7: *Steady model 1^{st} function. Frequency vectors.*

3.2.2 Fin flutter speed value - Steady approach

The second function used for the steady aerodynamics model, which has been simplified through the flow chart shown in Fig. 8, consists in the resolution of the expression reached in Eq.11: $R^2 - 4S^2 = 0$.

The input data are, basically, the dimensionless parameters coming from the initial function (Sec.3.1) and the symbolic velocity variable $V_{\infty_{eq}}$. Both, R and S, depend on the square of the flight speed (Eq.9), so that equation will not have one single solution. However, only one of them will represent the flutter speed that is being looked for. It is remarked that the $double()$ instruction must be used after solving the equation in order to convert the resulting symbolic value to a MATLAB double precision variable.

The solution has to be real, positive, and the lowest one, avoiding the possible null velocities which correspond to the natural frequencies. Those are the conditions that must be checked in each element of $solutions$. If they are met, the velocity solution V_{sol} is updated. Otherwise, it is not modified. Thus, once the loop ends, the pursued flutter speed will be stored in V_{sol}. Finally, the value of the flutter frequency, λ_{sol}, is obtained just dividing the negative value of the R coefficient for that V_{sol} by 2.

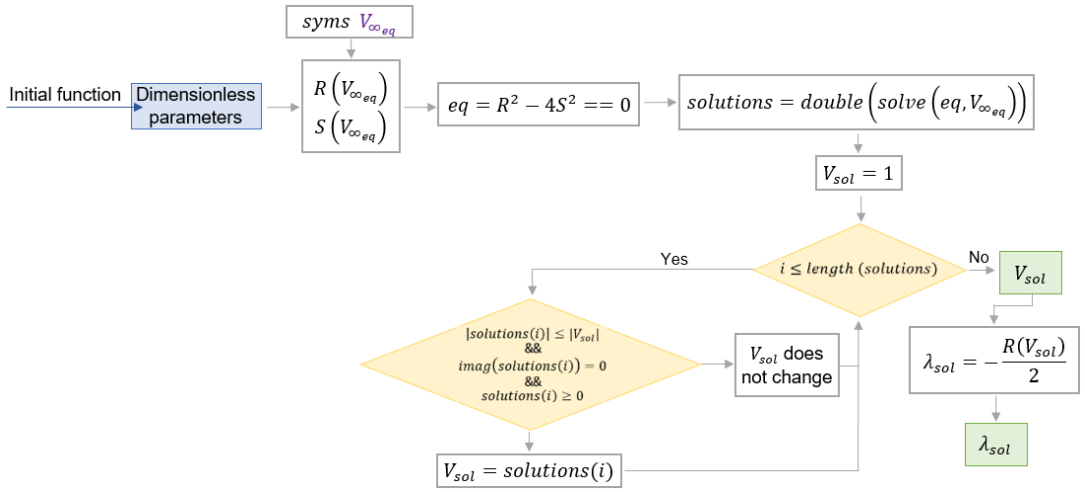


Figure 8: *Steady model 2^{nd} function. Value of the fin flutter speed*

3.3 Quasi-steady aerodynamics functions

3.3.1 Theodorsen function

Moving to the quasi-steady model, the first step is the calculation of the Theodorsen function (Eq.56) since it is essential to obtain the aerodynamic coefficients used in this model. It depends on a pair of Hankel functions, so the defined MATLAB function $besselh(order, kind, \mathcal{K}_f)$ is really helpful since it directly returns the value of the specified Hankel function. Finally, as it can be observed in the green block in Fig.9, its absolute value is taken since the derivatives must be real numbers.

It has not been mentioned anything about the input parameter, but it can be clearly seen in the diagram that it is the reduced flutter frequency $\mathcal{K}_f$, which is calculated from the flutter speed and frequency found in the steady model (Eq.56).

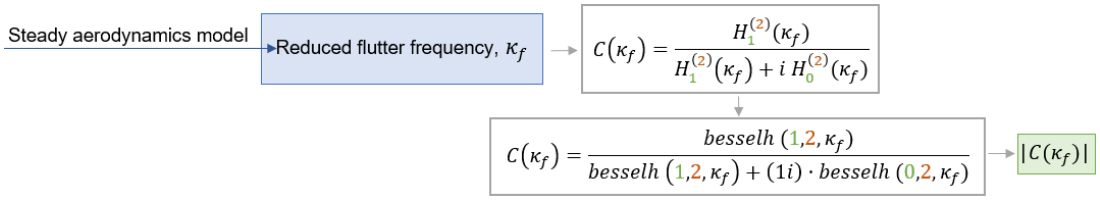


Figure 9: *Quasi-steady model 1^{st} function. Value of the Theodorsen function.*

3.3.2 Frequency vectors - Quasi-steady approach

As it is shown in the flow diagram of Fig.10, the process to fill the vectors with the frequency of the 4 modes of vibrations in the quasi-steady approach is really similar to the one followed in the steady approach (Sec.3.2.1). Due to this great similarity, this subsection only covers the differences between them. Otherwise, it would be redundant.

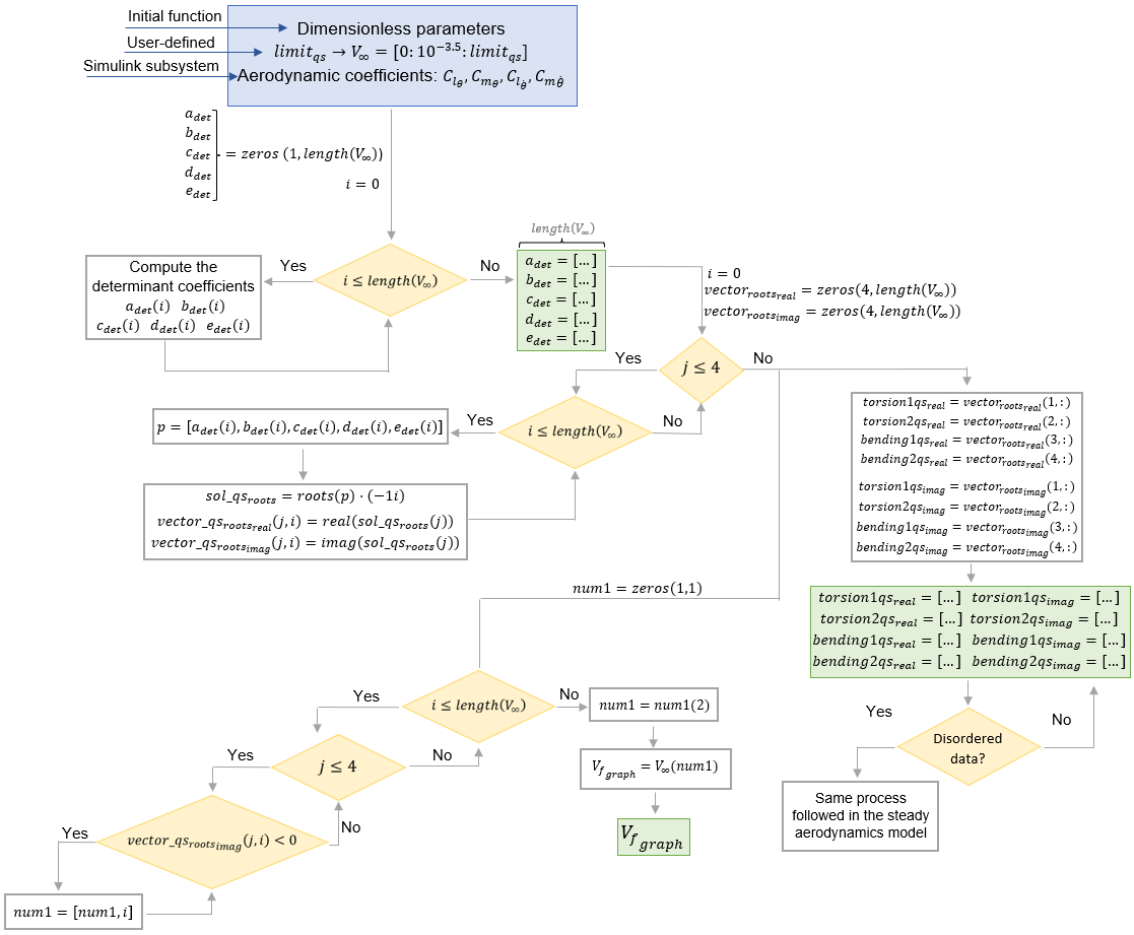


Figure 10: *Quasi-steady model 2^{nd} function. Frequency vectors, graphical flutter solution.*

The first difference is observed in the input data since the aerodynamic coefficients used in this quasi-steady model, obtained in a Simulink (App.C.2) subsystem, are needed.

Next, the first loop is in charge of calculating the determinant coefficients of the matrix in Eq.15. The imaginary unit appears in the last term, so the following substitution must be applied to be able to work out the determinant and its corresponding eigenvalues easily:

$$i\lambda = s \rightarrow \left| s^2\mathcal{M} + \mathcal{K} - \frac{V_\infty^2}{\mu}\mathcal{A} - \frac{sV_\infty}{\mu}\mathcal{B} \right| = p_{qs}(s) \tag{16}$$

A fourth-degree polynomial is found when the determinant written above is computed for each of the elements of the one-dimensional array of V_∞. Again, the user can choose the upper bound of it.

Nonetheless, the expression of the coefficients of that polynomial must be worked out before. For that, the determinant is calculated using symbolic variables, obtaining an expression of the kind $p_{qs} = a_{det}s^4 + b_{det}s^3 + c_{det}s^2 + d_{det}s + e_{det}$, being those coefficients:

$$a_{det} = i_\theta^2 - r_\theta^2 \qquad b_{det} = \frac{V_\infty}{\mu\pi}(-2C_{m_{\dot{\theta}}} + C_{l_\theta}i_\theta^2 - C_{l_{\dot{\theta}}}r_\theta + 2C_{m_\theta}r_\theta) \tag{17}$$

$$c_{det} = i_\theta^2(1+\eta^2) + \frac{V_\infty^2}{\mu^2\pi^2}(-2C_{l_\theta}C_{m_{\dot{\theta}}} + 2C_{l_{\dot{\theta}}}C_{m_\theta} - 2\pi C_{m_\theta}\mu - \pi C_{l_\theta}\mu r_\theta) \tag{18}$$

$$d_{det} = \frac{V_\infty}{\mu\pi}(-2C_{m_{\dot{\theta}}}\eta^2 + C_{l_\theta}i_\theta^2) \qquad e_{det} = \eta^2(i_\theta^2 - \frac{2C_{m_\theta}V_\infty^2}{\mu\pi}) \tag{19}$$

Then, once this first loop ends, those coefficients are ready for each value of V_∞ and, therefore, the next nested loop can start. The process is exactly the same as in the steady model, but the polynomial expression is obtained with those x_{det} coefficients.

Furthermore, it has to be pointed out that the solutions of the $roots(p_{qs})$ instruction must be multiplied by $(-1i)$. That is due to the substitution that was made in Eq. 16. The s values are not the eigenvalues, but the λ ones are, so: $s{\cdot}(-i) = i\lambda{\cdot}(-i) = -\lambda{\cdot}(i)^2 = \lambda$.

Finally, the frequency vectors are filled as it was done in Sec.3.2.1, also checking that the data has been stored orderly. However, this function also calculates the flutter speed from the matrix $vectorqs_{roots_{imag}}$ with another nested loop (Bottom-left of Fig.10).

As it was discussed in Sec.2.2.2, flutter occurs when the imaginary part of any of the vibration modes starts being negative. Thus, the imaginary part of the 4 modes of vibration is checked for each velocity to see if any of them is less than zero. If a negative value is detected, its location is added to the vector $num1$.

The first added location will be the pursued one since from that point the fin will start fluttering .As $num1$ was initialised with a zero because empty arrays do not work well in Simulink, the flutter location will be the second one. Just looking for this location in the velocity vector V_∞, the flutter speed, called $V_{f_{graph}}$ in this diagram, is found.

This value will not be as exact as the one obtained in the following subsection since V_∞ cannot be defined with infinity values. Then, its accuracy will depend on the number of elements of V_∞, also bearing in mind the increase in computational time when the number of elements is higher. Thus, the key point is a kind of trade-off between those.

3.3.3 Fin flutter speed (Theodorsen's method) - Quasi-steady approach

This function develops the Theodorsen's method explained in the mathematical background, at the end of Sec.2.2.2. For that, two symbolic variables are declared, V_f for the flutter speed, and Ω_f for the flutter frequency. When the flutter point arrives, the imaginary part of that frequency will be 0, so that is why Ω_f is treated as a real number.

Next, the expression of the real and imaginary parts of the determinant of Eq.57 are obtained, again, using symbolic variables, as it was done with the x_{det} coefficients in Sec.3.3.2. Those are quite long so, please, refer to Appendix B.6 to check them (7-24).

In order to find the value of V_f and Ω_f, both, the real and the imaginary parts, have to be equalled to zero. In such a way, a system of 2 equations is reached. It has not been commented but, as it can be observed in the chart below, the input parameters are the dimensionless parameters and the aerodynamic coefficients.

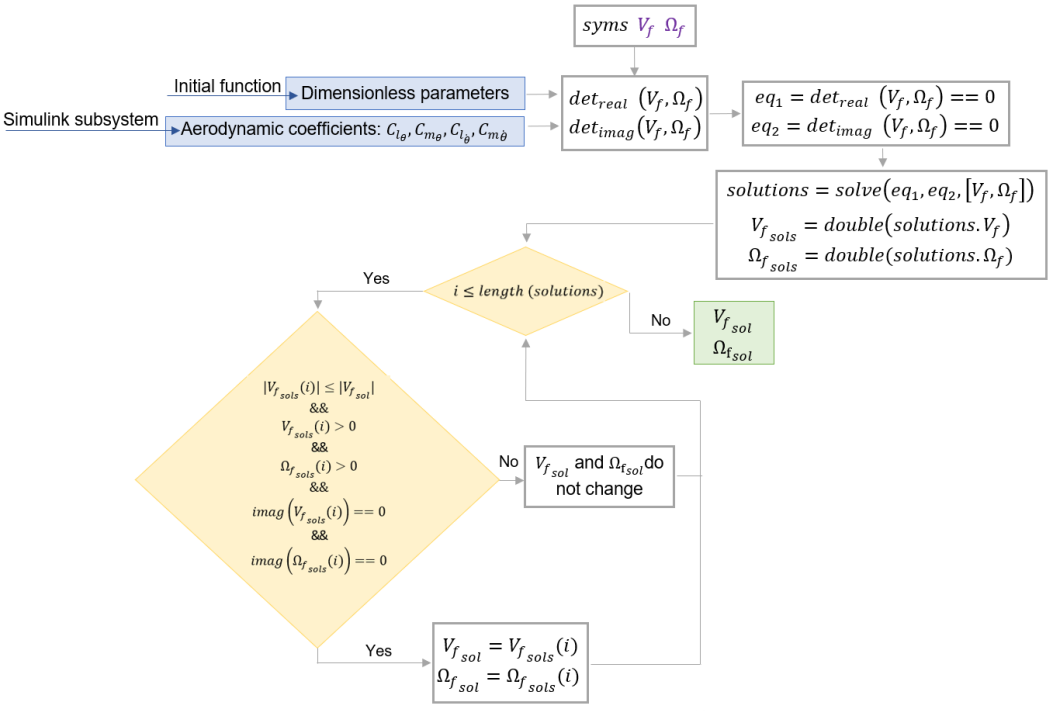


Figure 11: *Quasi-steady model 3^{rd} function. Flutter speed (Theodorsen's method).*

When solving the system, multiple solutions appear but only one combination of V_f and Ω_f is valid. Both of them have to be real, greater than 0, and the lowest velocity must be the chosen one (avoiding the null velocities which represent the natural frequencies).

This is what is checked with the final loop and the conditional statement in the yellow blocks. After going over the entire $solutions$ vector, the flutter speed and frequency are stored in the variables $V_{f_{sol}}$ and $\Omega_{f_{sol}}$ (green block).

Finally, it must be pointed out that this function, as well as the one developed in Sec.3.2.2, need to resort the $coder.extrinsic()$ strategy to declare them as extrinsic functions. That is because Simulink does not allow the direct use of the $solve()$ command in a MATLAB function block. To see how it is programmed in the code, please refer to App.32 and App.34.

3.4 Plotting function

The eight frequency vectors of Sec.3.2.1 and Sec.3.3.2 have been obtained to be able to plot the frequency curves and analyse the flutter point graphically. For that, the first step is to log the data to the MATLAB workspace from the Simulink model. As it will be seen in the next section, it is done through the block $out.variable\_name$.

To take the data coming from Simulink, the instruction $out.get(variable\_name)$ must be used. Next, 6 plots are arranged in grid a of 2x3, being the first row for the steady model, and the second row for the quasi-steady one. The 3 plots represented for both models are exactly the same as the ones shown in Fig.4 and Fig.5: the real part of the frequencies versus the dimensionless flight velocity, the imaginary part versus the dimensionless velocity, and the root locus (real versus imaginary part).

As it is indicated in the diagram below (Fig.12), the $hold\ on$ and $hold\ off$ instructions are essential to plot the 4 vibration modes all together with the flutter, the divergence and the natural frequencies marks in each plot.

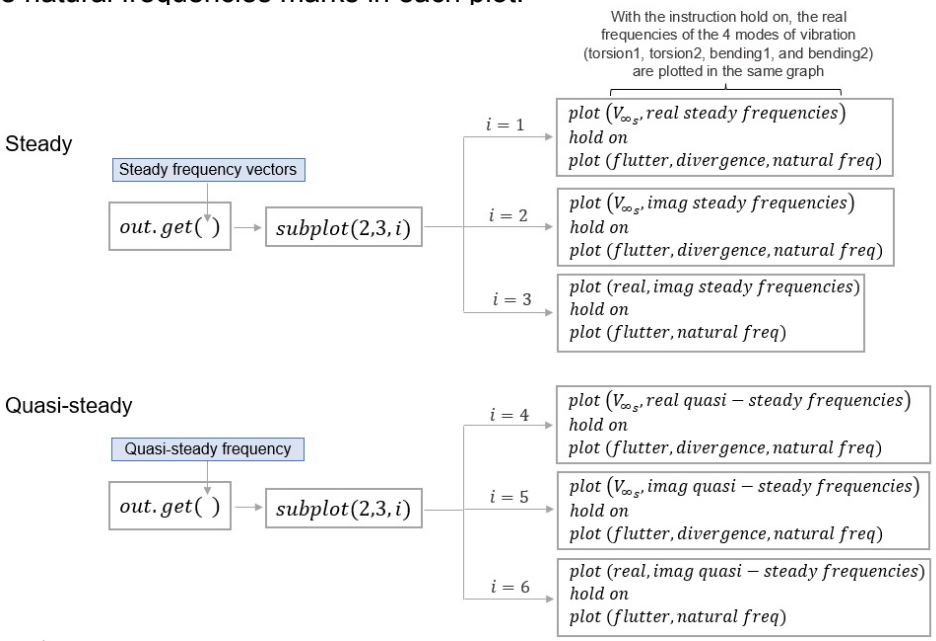


Figure 12: *Plotting function for the steady and the quasi-steady models*

4 Flutter simulator - Graphical user interface

As it was aimed at the beginning of the project, a flutter simulator like the one shown in Fig.13 is developed. Its main purpose is that the users are able to introduce the data of the fin they are studying and get a first idea about the velocity at which the flutter and divergence phenomena would occur. Moreover, the two different aerodynamic models, the steady and the quasi-steady one, are simulated so that the users can see the differences in the flutter frequency and speed depending on which approach is taken.

As it is indicated in the outer view of the user interface shown below, it is mainly divided into 4 sections: user fin data, divergence speed, steady aerodynamics approach, and the quasi-steady one. Then, each of them is explained in detail throughout the next subsections, emphasising and referring to the MATLAB functions covered in Sec.3.

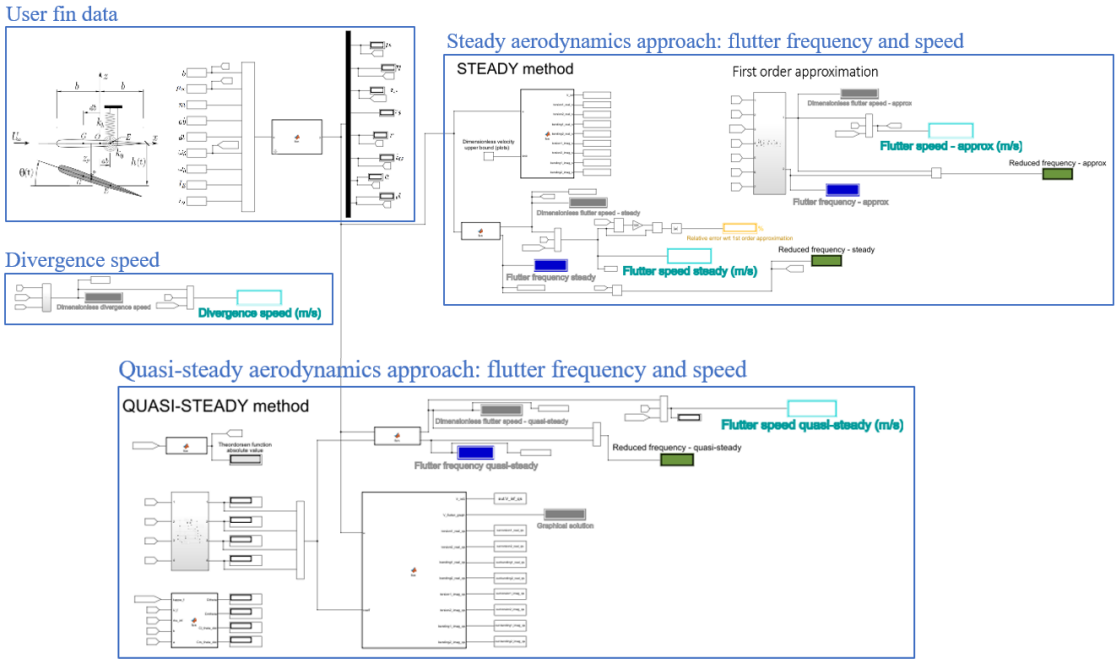


Figure 13: *Outer view of the graphical user interface*

4.1 User fin data

Fig.14 shows the user fin data section of the simulator zoomed in. It is based on the initial function which calculates certain dimensionless parameters from the user inputs. It was treated at length in Sec.3.1, specifying the formulas used for each parameter.

Although those parameters are not the principal output for the user, they are displayed so that the value of a determined parameter can be checked anytime. It may be the case that, for instance, the user is pursuing a concrete value for the mass ratio μ. Thus, it would be possible to check how this parameter varies when the input data is modified.

Finally, it must be mentioned that an sketch of the aerofoil dynamic model is shown on the left hand side so that the user does not lose sight of the problem that is being solved.

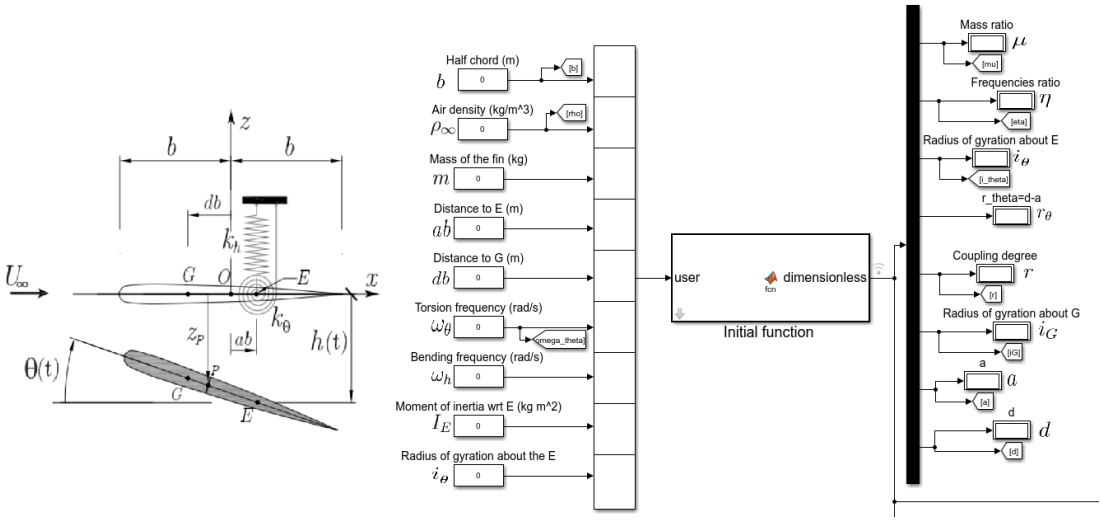


Figure 14: *User fin data zoom*

4.2 Divergence speed

This section of the user interface is dedicated to the calculation of the divergence velocity. As it can be seen, no MATLAB functions have been used to do it since the divergence formula is quite easy to implement. Therefore, the creation of a subsystem is done just taking the expression developed in Eq.13 and solving for V_D.

That subsystem is shown below, pointed with a grey arrow. The input parameters (a, i_θ, and μ) are taken from the dimensionless parameters obtained with the initial function covered in Sec.3.1. Next, just using the most common blocks, such as the add, the square or the product/divide one, the dimensionless divergence speed is reached and shown in the grey displayer. This value is sent to the MATLAB workspace with the $out.$ block instruction since the divergence point will be represented in the frequency graphs through the plotting function covered during Sec.3.4.

Lastly, if this dimensionless velocity is multiplied by the span b and the torsion frequency ω_θ (Eq.7), the actual divergence speed value is obtained and displayed in the biggest cyan rectangle, following the SI (m/s).

Regrading the divergence frequency, it has not been discussed or calculated either. That is because this phenomenon is characterised by not presenting oscillations, so the frequency can be directly set to 0, without working out any mathematical operation.

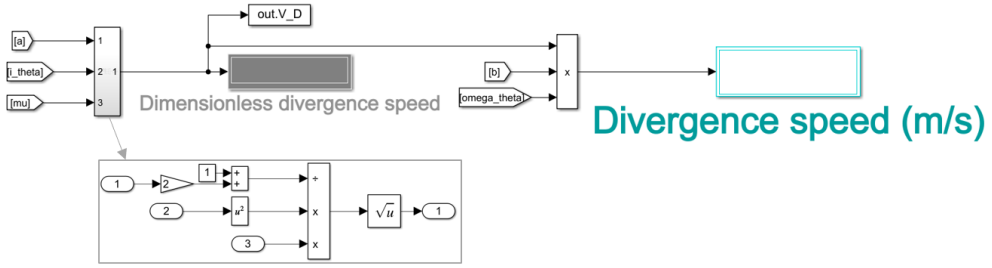


Figure 15: *Divergence speed section zoom*

4.3 Steady aerodynamics approach

Looking at Fig.16, it can be observed how the steady model 1^{st} and 2^{nd} functions are utilised. The input data for both of them is the dimensionless parameters, having the first one the additional velocity limit value commented in Sec.3.2.1 too.

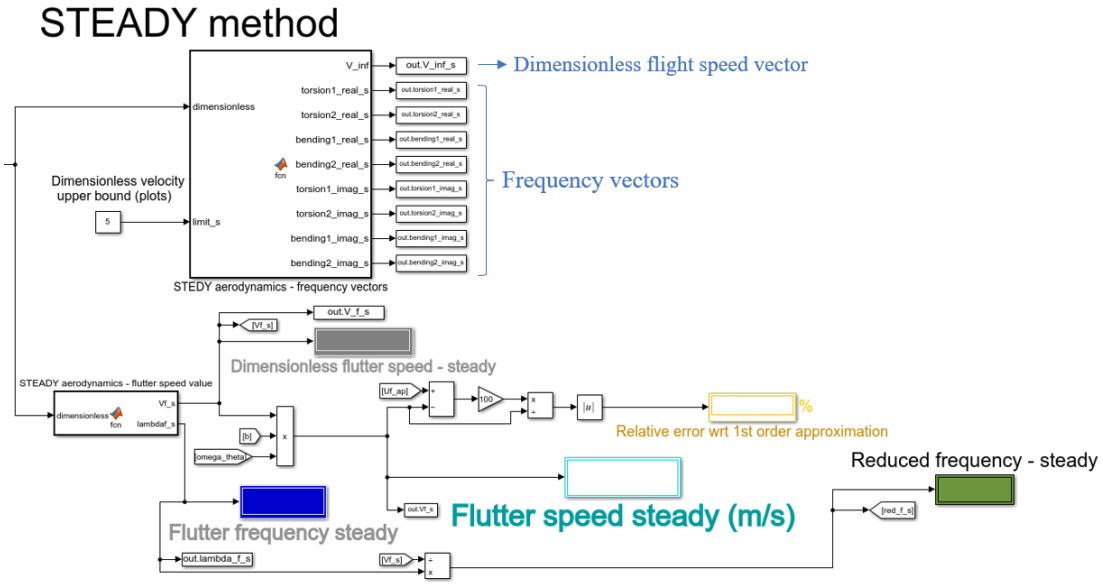


Figure 16: *Steady aerodynamics - frequency vectors, flutter speed and frequency values*

The first function, shown in the upper part of the model, returns the eight frequency vectors (there are only 4 modes of vibration, but as the real and imaginary parts had to be separated, this results in eight outputs) and the dimensionless flight velocity vector. They are transferred to the MATLAB workspace since both are essential for the frequency graphs implemented in the plotting function (Sec.3.4).

From the second function, just below the previous one, the dimensionless flutter speed (grey displayer) and the flutter frequency (drak blue displayer) values are obtained. Both of them are also logged to the MATLAB workspace with the $out.$ block since they are needed to represent the flutter point on the graphs. Furthermore, as well as in the divergence section, the actual flutter speed (biggest cyan displayer) is reached just using the product block to multiply the dimensionless one times the span and the torsion frequency, being those last two introduced by the user.

Moreover, the reduced frequency value is also calculated using a divide block, dividing the flutter frequency by the dimensionless flutter speed (Eq.56). The value obtained is shown in the dark green displayer, on the right hand side of the model.

It can be also observed a yellow displayer which seems to show an error value, in %. This values stands for the relative error between the exact solution coming from the direct solving of the steady model equations and the first order approximation.

Regarding this first approximation, its Simulink model is shown in the figure below:

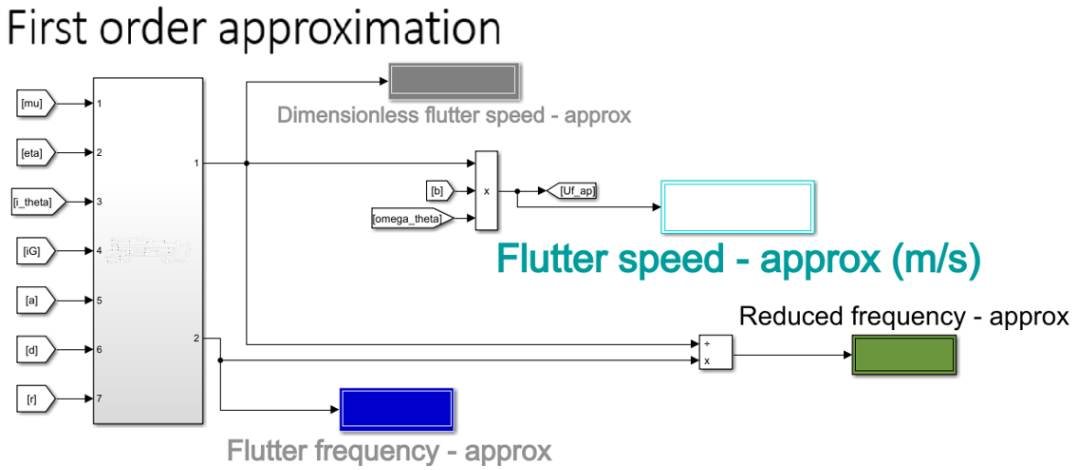


Figure 17: *Steady aerodynamics - First order approximation*

Looking at Fig.17, a new subsystem is clearly observed on the left hind side of the model, having as inputs all the dimensionless parameters excluding the r_θ value. The combination of blocks used for this subsystem is included in Appendix C.1 since it is quite long and it does not contribute to any new idea. It is just the set of blocks used to build the expression for V_f shown in Eq.12 and its corresponding flutter frequency (Eq.11).

As usual, the value of the reduced frequency is computed dividing the flutter frequency by the dimensionless flutter speed. The colour code for the displayers is exactly the same as before, so it is not repeated to avoid being redundant. Note that all the displayers are named with the word $approx$ at the end since, in this way, it is easier to differentiate them from the others when taking a quick look to the interface.

4.4 Quasi-steady aerodynamic approach

The Simulink model of the quasi-steady aerodynamics approach has been divided into two parts. Otherwise, it was impossible to appreciate all the blocks used in this section.

The first part is shown in Fig.18, just below. At the top left hand side of the model, the Theodorsen's function described in Sec.3.3.1 is implemented. As it can be seen, the input parameter is the reduced frequency obtained in the steady aerodynamics model. The output, as it can be expected, is the absolute value of the Theodorsen's function $\mathcal{C}$, which is needed for the calculation of the aerodynamic derivatives.

Next, the subsystem shown below the previous function is in charge of obtaining the values of the aerodynamic derivatives, which expressions were developed in Sec.A.4.2. Again, the fact of showing the block diagram of this subsystem would be useless since it does not help understanding the Simulink model that is being discussed in this section.

For that reason, it is included in Appendix C.2, just in case it is wanted to be checked.

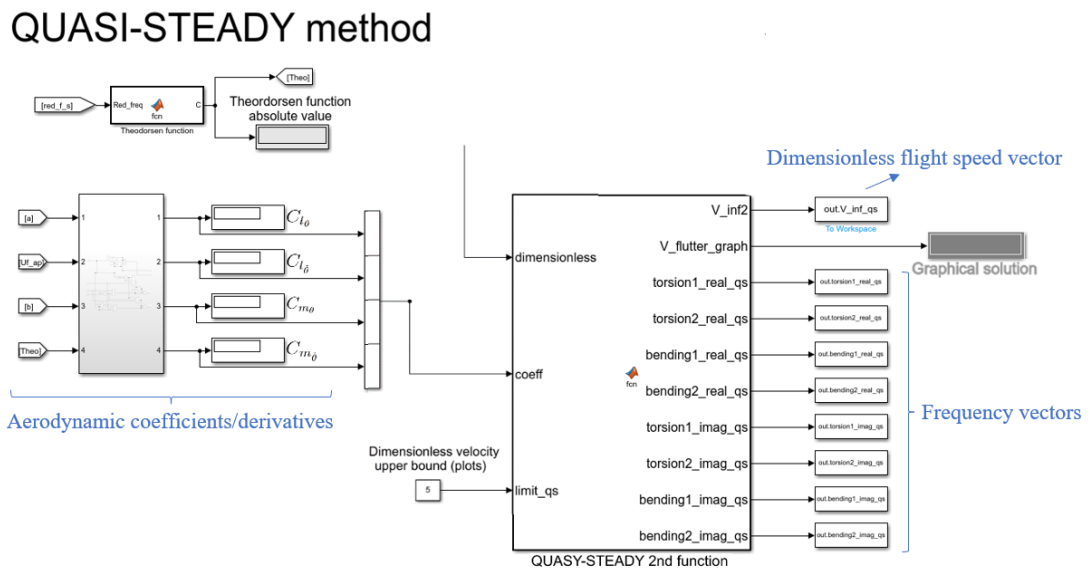


Figure 18: *Q-s model - Theodorsen's function, aerodynamic coefficients and frequency vectors*

Finally, the second function of the quasi-steady aerodynamics model is run, taking as inputs the dimensionless parameters (coming from the initial function of Sec.3.1) and those aerodynamic derivatives. The eight frequency vectors are obtained as outputs, as well as the dimensionless flight velocity vector. As well as in the first function of the steady model, all those vectors are logged to the MATLAB workspace to finally plot them through the final plotting function (Sec.3.4).

It has to be remarked that the graphical flutter solution, covered in Sec.3.3.2, is displayed on the big grey block. As it was commented, the accuracy of this flutter velocity will depend on the number of elements used to define the dimensionless flight speed vector. Again, it is reminded that the user can modify the upper bound of the dimensionless velocity just in case the flutter or the divergence points occurs later. That is why there is an extra input in the function, being a total of three.

Lastly, the flutter speed and frequency are obtained solving the system of equations reached with the quasi-steady aerodynamics assumptions, and transferred to the workspace. As expected, the inputs parameters are the dimensionless parameters and the aerodynamic derivatives. Again, the reduced frequency and the dimensional flutter speed are also computed. Regarding the displayers in Fig.19, the colour code used during all the Simulink model is followed once more time, so it is not repeated.

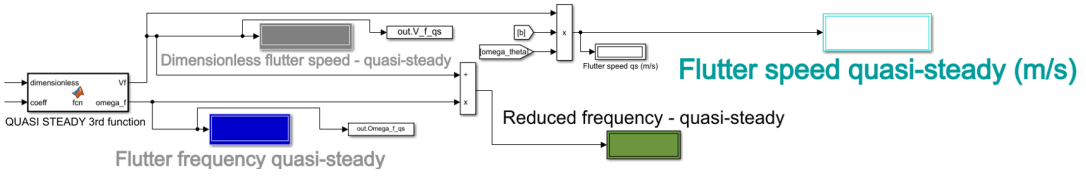


Figure 19: *Quasi-steady aerodynamics - Flutter speed and frequency values*

5 Validation process

Once the code and the Simulink model are covered, it is time to validate the flutter results obtained with the steady and the quasi-steady aerodynamic approaches. It is not enough to get values that make sense, they must be validated. For that, they are compared with the data published in the NACA Technical Report No. 685 [13], concretely with the graphs observed in the first case presented on it, the flexure-torsion one.

Before discussing the results, it must be remarked that this NACA's report express the mass ratio μ the other way round and use k to name it. However, please, bear in mind that $k = \frac{1}{\mu}$. Something similar with the radius of gyration i_θ, since it is written as r_θ.

5.1 Steady aerodynamics model

As it can be seen at the top of Fig.20, the chosen graph plots the dimensionless flutter speed versus the frequency ratio for different squared radius of gyration and mass ratios. Then, the curves from the third case, which have been zoomed below and where $k = 1/20$, are replicated immediately on its left with the data obtained from the steady aerodynamics approach, both the first order approximation (approx) and the steady model 2^{nd} function (2^{nd} function).

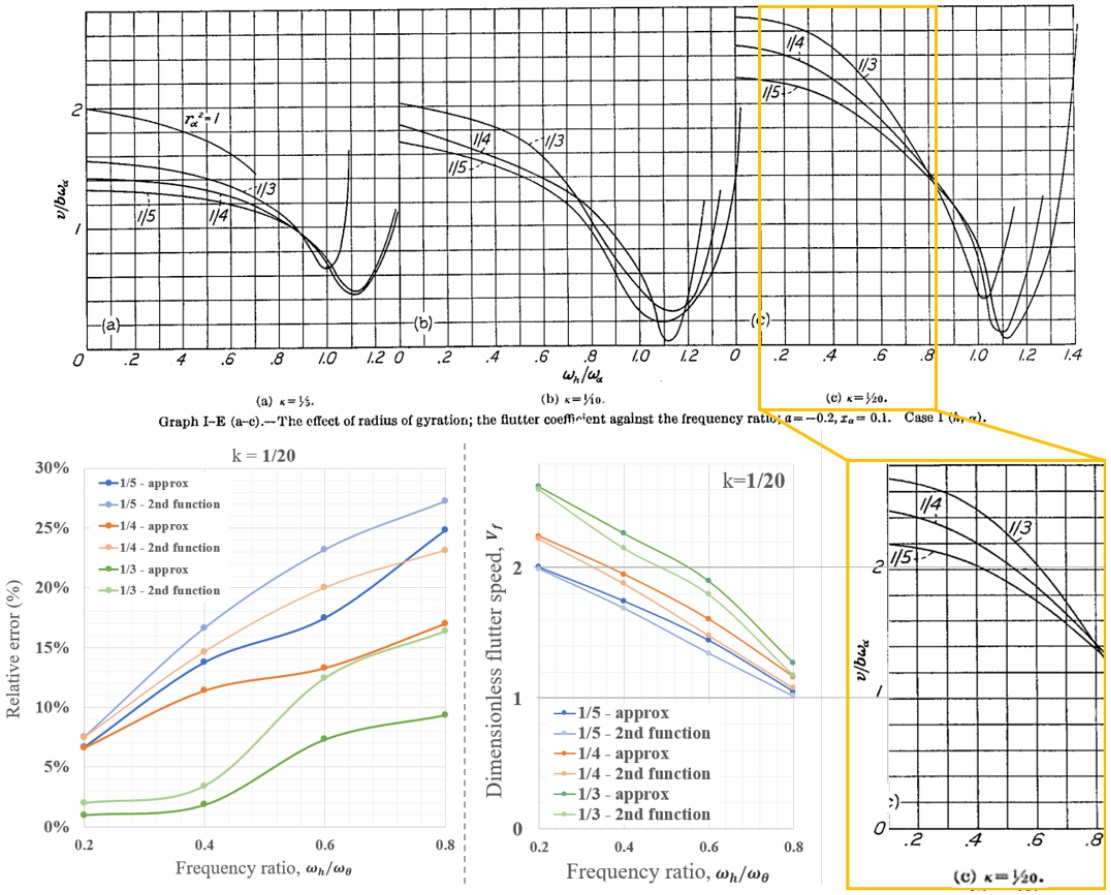


Figure 20: *Graph comparison between the steady model and the NACA Report No. 685.*

With respect to the tendency, it is checked that it is really similar. As the frequency ratio increases, the dimensionless flutter speed decreases regardless the value of the radius of gyration. Moreover, the point where the six curves come together at $\frac{\omega_h}{\omega_\theta} = \eta = 0.8$ is quite well represented with the steady approach. They are not as coincident as in the actual zoomed graph, but the intention to do it is clear. The curves have not been plotted beyond that point since, as it was said in the mathematical background, the steady approach and its first order approximation gives good results as long as $0 < \eta < 1$.

Besides that, it is also verified that the flutter speeds are higher as larger is the radius of gyration. However, there is no denying that the numbers are not exactly the same, they are always lower than the NACA's report ones, being even lower if the steady model second function (light coloured curves) is used instead of the first order approximation (dark coloured ones). This is completely reasonable since the steady model accounts neither for the velocity nor the acceleration terms whereas the NACA's report try to obtain the most possible accurate result. Thus, errors due to the approximations and simplifications taken in the steady approach were more than expected.

For that reason, the relative errors between the results of the first order approximation and the NACA's report ones are calculated. The same is done with the second function of the steady model, to finally plot them versus the frequency ratio. This relative errors graph is shown at the bottom left of Fig.20, following the same colour code. It can be appreciated how the errors are lower for small frequency ratios since it was indicated that the steady approach works rather well for $\eta << 1$. Furthermore, as the first order approximation results were higher in this case and therefore close to the NACA's report ones, their relative errors are smaller. To find the exact values of each method and their corresponding errors, please, refer to Appendix D, where a sample calculation is also done.

5.2 Quasi-steady aerodynamics model

As it can be seen in the table shown on Fig.21, the results of the quasi-steady approach are not accurate at all. The relative errors are huge, what means that the flutter speed values are rather far from the NACA's report ones. The model did not work as it should have and one potential problem could be perfectly found in the aerodynamic derivatives.

They have been obtained deriving the lift and moment coefficients with respect to $\dot{\theta}$, depending the resulting expressions on the Theodorsen's function (reduced frequency dependant) and the fin chord values. In the table below, the model was run for $i_\theta^2 = 1/3$

and for two different half chord values, $b = 0.9$ m and $b = 44$ m, being this last one extremely large and totally unfeasible in aircraft/rockets projects. Even so, the flutter speed values are really small in comparison with the steady model or the NACA's report ones. Moreover, it is pretty weird the fact that the flutter speeds are so insensitive to the frequency ratio variations. Unlike in the steady model, the changes in the flutter speed are almost unnoticeable when the frequency ratio increases.

b = 0.9 m

$i_\theta^2 = 1/3$	Quasi-steady approach	NACA 685	Relative error
ω_h/ω_θ	Vf	Exact	
0.2	0.9634	2.650	63.65%
0.4	0.9703	2.225	56.39%
0.6	0.9706	2.050	52.65%
0.8	0.9816	1.400	29.89%

b = 44 m

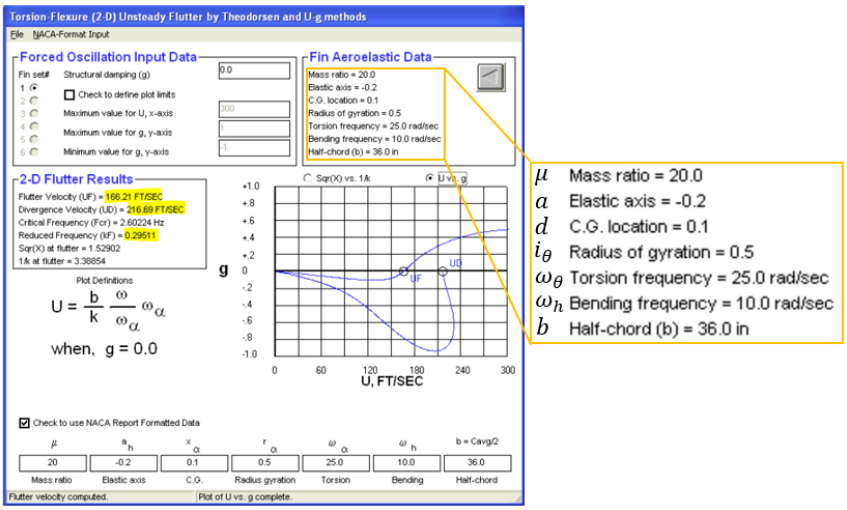
$i_\theta^2 = 1/3$	Quasi-steady approach	NACA 685	Relative error
ω_h/ω_θ	Vf	Exact	
0.2	1.209	2.650	54.38%
0.4	1.434	2.225	35.55%
0.6	1.478	2.050	27.90%
0.8	1.378	1.400	1.57%

Figure 21: *Relative errors between the quasi-steady model and the the NACA Report No. 685.*

As it is commented in the conclusion, specifically in Sec.7.4, it would be ideal that the future work undertake this problem and investigate where is the error.

5.3 AeroFinSim simulator

The AeroFinSim software is a really helpful app since it enables the user to obtain the critical flutter and divergence speeds from the input parameters zoomed on the right hand side of Fig.23. As it is indicated on its website [4], it uses the Theodorsen and U-g methods to solve the unsteady torsion-flexure problem. If their results are compared with the ones obtained with the steady model, the table shown below is reached. It can be seen that the reduced frequency and the divergence speed values are really close. Nevertheless, and as it was expected again, the flutter speed values are lower as it happened when comparing results with the NACA's report in Sec.5.1.



	AeroFinSim	First order approximation	Reltive error	Steady model 2nd function	Reltive error
Flutter velocity (ft/s)	**166.21**	143.86	13.45%	138.65	16.58%
Divergence velocity (ft/s)	**216.69**	213.02	1.69%	213.02	1.69%
Reduced frequency	**0.295**	0.2795	5.25%	0.3011	2.07%

Figure 22: *Relative errors between the steady model and the AeroFinSim 10 software*

6 Case of LURA's fins

As it is mentioned in the cover page and throughout the introduction, this project was born in order to help the LURA team with the flutter and divergence problem. The graphical user interface, as well as the code that makes it works, have been already explained. Thus, it is the time to input the data of the fins that LURA is currently using.

The CAD design (attached in Appendix E) and the OpenRocket Simulator [12] file which LURA is basing its fins design on were really helpful since all the data shown in the table below could be extracted:

Parameter	Symbol	Value
Root chord (m)	r_c	0.230
Tip chord (m)	t_c	0.030
Fin mass (kg)	m	0.728
Dimensionless elastic axis location (m)	a	0.0127
Dimensionless centre of gravity location (m)	d	0.0286

Figure 23: *LURA's fins parameters*

From those, the input parameters for the simulator developed in Sec.4 are almost ready. However, some calculations must be worked out before. It has been possible to take directly the values of the fin mass and the values of a and d, being the last two measured from half of mean aerodynamic chord. However, this mean aerodynamic chord, MAC, has had to be previously computed following the specific formula for tapered and delta fins shown in Eq.20 [7]. In such a way, the value of b is obtained as:

$$if\ t = \frac{r_t}{r_c} \rightarrow MAC = \frac{2}{3} r_c \frac{1 + t + t^2}{1 + t} = 0.153\ m \rightarrow b = \frac{1}{2} MAC = 0.0756\ m \tag{20}$$

No information was provided for the radius of gyration, so it has been left with one of the most common values used for aerospace purposes, which is $i_G = 0.5$. The same with the ratio of frequencies, which is set to $\eta = 0.2$, since it is indicated that the bending frequency is quite lower than the torsional one when real stiffness values are used, in the order of $\omega_h < 0.3\ \omega_\theta$ [9].

Then, the model is run and the flutter and divergence results obtained with the steady model are compared with the values that LURA is currently using, reaching the table shown in Fig.24. The results achieved with the quasi-steady approach are not being analysed since, as it has been seen in the previous section, they could not be validated.

	Steady		
(at sea level altitude)	1st order approx.	2nd function	LURA
Flutter speed (m/s)	571.3	560.1	578
Divergence speed (m/s)	601.3		367.2

Figure 24: *Comparison between the results of the steady model and the LURA's values*

It can be checked that the flutter speed values obtained with the steady model are enough close to the LURA's ones. As it happened in the validation process, the values

are slightly lower, being the first order approximation the nearest one. It has to be remarked the fact that the values of eta and i_G have been estimated, so the studied case may not be exactly the same as the one considered by LURA.

Regarding the divergence velocity, the steady model predict a value of 601.3 m/s, being higher than the flutter one. However, it happens precisely the opposite with the LURA's divergence speed since it is considerably lower, with a value of 367.2 m/s. That weird fact should be analysed in detail by the LURA's team since it may not be entirely reliable. In most of the cases, the flutter phenomenon occurs before the divergence, what makes totally sense since this last one implies the total failure of the structure due to the exceeding aerodynamic moment created. However, there is not any strict rule stating that flutter speeds must be always lower than the divergence ones.

Finally, the frequency graphs that have been commented throughout the project are shown in Fig.25. As the LURA's case is being analysed in this section, those graphs represent the behaviour of the frequency of their 4 modes of vibration. Then, the key point lies in knowing how to interpret them. For that, the flutter and divergence conditions for each aerodynamic approach should be remembered:

Steady approach: flutter dimensionless velocity is found from the coalescence point at which the bending and torsion real frequencies match. To get the actual or dimensional flutter speed, just multiply that value by half of the chord length and the torsion frequency. The divergence point can be detected when all the frequencies are 0 (no oscillations).

Quasi-steady approach: flutter dimensionless velocity is found at the point from which any vibration mode starts presenting negative imaginary parts. The divergence point works in the same way as in the steady approach. It must be pointed out that the natural frequencies (frequencies for flight speed = 0) are also marked in green on the graphs.

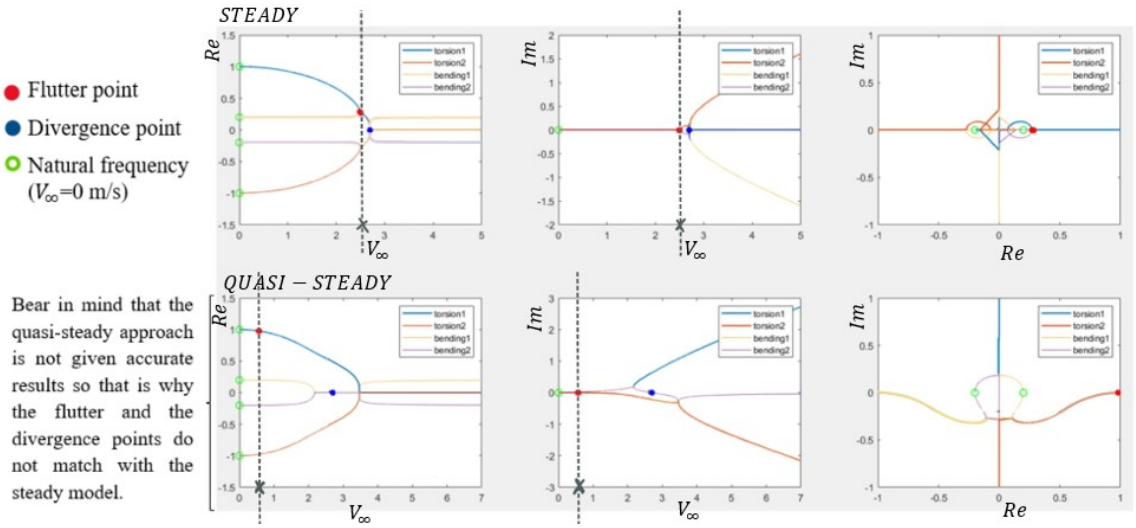


Figure 25: *Frequency graphs for the steady and the quasy-steady approaches*

7 Concluding chapter

To conclude this engineering project, all the achievements are presented one by one, comparing them with what was expected at the beginning of the semester. Moreover, the key points of every discussion are summarised, recommending where the future work may be undertaken.

7.1 Achievements

The following list collects all the accomplishments, making reference to the objectives established in Sec.1.3:

1. The literature review was shown in the scoping and planning document, where everything was perfectly referenced.
2. The different approaches (steady and quasi-steady aerodynamic models), as well as the numerical methods used to estimate the flutter and divergence conditions are developed in the mathematical background section, where all the assumptions taken in each of them are specified.
3. The frequency of each mode of vibration is achieved for both models through the frequency vectors functions of Sec.3.2.1 and Sec.3.3.2.
4. It has been possible to obtain the flutter and divergence speeds and frequencies too, thanks to the functions developed in Sec.3.2.2 and Sec.3.3.3.
5. The pursued aeroelastic simulator has been successfully built thanks to the toolbox of Simulink. It is a kind of user interface where the users can introduce the data of the fins they are using and obtain the flight speeds and frequencies at which flutter and divergence phenomena will take place. Regarding its structure, it has been seen in Sec.4 how it was divided into several sections clearly separated so that the users are sure about the values they are reading.

7.2 Discussion

In this subsection, the two last objectives stated in Sec.1.3 are discussed since they have not been achieved in the way they were defined. It is not weird at all that the objectives changes throughout the development of the project. It is not easy to foresee if one of the objectives will not be as essential as it seemed to be when it was written.

As it can be checked in Sec.5, the validation process was finally carried out with the NACA Report No.685 (Theodorsen and Garrik, 1940) and the wwll-known software of AeroFinSim Simulator (Cipolla, 2022). The CFD simulations that were thought at the beginning would have been so useless. First, in order to match the dimension order, it would have had to be a 2D simulation since 3D effects have not been considered in

any moment. Moreover, the CAD design would have been uncertain since choosing an aerofoil would have not been trivial. There is not much information about flutter analysis performed with CFD. Even LURA has not defined any aerofoil for their fins yet either.

In the case of the mentioned trade-off, it can be said that it has been done somehow by means of some trends obtained in the validation process of Sec.5. Thus, the principal conclusion drawn from it, considering the studied range $0 < \eta < 1$, is that: flutter speed increases with higher radius of gyration, and decreases when the frequency ratio is increased. Therefore, whenever possible, it will be always preferable to use high radius of gyration and small frequency ratios. In such a way, the flutter speed will be significantly higher and, consequently, the speed operating range of the rocket wider.

7.3 Conclusions

After considering the project overall, it has been more than demonstrated that the flutter prediction is not simple at all. Even though, it has been possible to build the pursued aeroelastic simulator, which works quite well with the steady method, hoping it results helpful for LURA. That is why the values obtained from this model were validated easily, giving the first order approximation the closest results to the ones stated by NACA.

Just the opposite happens with the quasi-steady flutter values, which are significantly far from those. Thus, it has been impossible to validate them. However, it is highly likely that the problem/error is due to the way the Theodorsen's function and, therefore, the aerodynamic coefficients are obtained.

Finally, it has to be pointed out the fact that the divergence velocity that LURA is obtaining is lower than the flutter one, as it was discussed in Sec.6. It would be recommended that this value is checked once more time since it is not usual that the total structural failure occurs before the flutter phenomenon.

7.4 Future work

This last subsection is dedicated to suggest the reader about how this project could be undertaken for future work.

As it has been mentioned before, it is needed that another method to get the aerodynamic derivatives is investigated because they are causing the quasi-steady inaccuracy almost certainly. Moreover, the non-steady aerodynamic model could be also incorporated to the simulator to take into account the acceleration terms of the equations too, instead of neglecting them. Lastly, to better approximate the real case of the fins, the flutter and divergence analysis could be performed in 3D, considering the whole span of them.

References

[1] Benson, T. 2021. *Rocket Stability*. [Online]. [Accessed 18 November 2021]. Available from: https://www.grc.nasa.gov/www/k-12/rocket/rktstab.html

[2] Bisplinghoff, R. L., Ashley, H. and Halfman, R. L. 2013. Aeroelasticity. Newburyport: Dover Publications.

[3] Bisplinghoff, R. L. and Ashley, H. 2013. Principles of Aeroelasticity. New York: Dover Publications.

[4] Cipolla, J. 2022. *AeroFinSim 10*. [Online]. [Accessed 25 March 2022]. Available from: http://www.aerorocket.com/finsim.html

[5] Collar, A.R. 1978. The first fifty years of aeroelasticity. *Aerospace*. 5(2), pp.12-20.

[6] Fung, Y.C. 1993. Introduction. In: Dover Publications, Inc. ed. *An Introduction to the THEORY OF AEROELASTICITY*. 2nd ed. London: Constable and Company Ltd, pp. 11-13.

[7] Johnson, P.K. 2003. *Finding the Mean Aerodynamic Chord (MAC)*.[Online]. [Accessed 3 April 2022]. Available from: https://acortar.link/WHZNlh

[8] Nakka, R. 2001. *Fins for Rocket Stability*. [Online]. [Accessed 23 November]. Available from: https://www.nakka-rocketry.net/fins.html

[9] Navarro, M.L. 2009. *Aeroelasticidad dinámica. Flameo del perfil en régimen incompresible*. Valencia: Escuela Técnica Superior de Ingeniería del Diseño, UPV.

[10] Navarro, M.L. 2009. *Aerodinámica no-estacionaria de perfiles*. Valencia: Escuela Técnica Superior de Ingeniería del Diseño, UPV.

[11] Nithin, S. and Vijayalakshmi, B.K. 2019. REVIEW ON AEROELASTICITY. *International Journal of Engineering Applied Sciences and Technology*. 4 (8), pp 271-274.

[12] OpenRocket. 2022. *Build better rockets*. [Online]. [Accessed 15 March 2022]. Available from: https://openrocket.info/index.html

[13] Theodorsen, T. and Garrick, I.E. 1940. *Report No.685 - MECHANISM OF FLUTTER A THEORETICAL AND EXPERIMENTAL INVESTIGATION OF THE FLUTTER PROBLEM*. Washington D.C: National Advisory Committee for Aeronautics.

A Mathematical developments

A.1 Process to obtain the dynamic matrices

Mass matrix (M)

The mass matrix is found through the kinetic energy of the system due to the velocities of the points on the aerofoil. Nevertheless, the oscillations are usually enough small ($\theta << 0$) to be able to neglect horizontal velocities and consider only the vertical ones. In such a way, all the points at a constant x distance will have the same vertical speed, $\dot{z_p}(x,t)$, which must be written as a function of the dimensionless vector $\boldsymbol{u}$. So, attending to the geometry in Figure 3:

$$z_p(x,t) = -h(t) - \theta(t)(x - x_E) = \left\{-b,\ -(x - x_E)\right\} \left\{h/b,\ \theta\right\}^T = \boldsymbol{d}^T(x)\boldsymbol{u}(t) \tag{21}$$

$$\dot{z_p}(x,t) = \boldsymbol{d}^T(x)\dot{\boldsymbol{u}}(t) \tag{22}$$

Then:

$$T = \frac{1}{2}\int_{-b}^{b} \dot{z_p}^2(x,t)\, dm \ = \frac{1}{2}\int_{-b}^{b} \dot{z_p}^T(x,t)\dot{z_p}(x,t)\, dm \ = \frac{1}{2}\dot{\boldsymbol{u}}^{\boldsymbol{T}} \int_{-b}^{b} \boldsymbol{d}(x)\boldsymbol{d}^T(x)\, dm\ \dot{\boldsymbol{u}} \tag{23}$$

$$T = \frac{1}{2}\dot{\boldsymbol{u}}^{\boldsymbol{T}}\boldsymbol{M}\dot{\boldsymbol{u}} \ \rightarrow \ \boldsymbol{M} = \begin{bmatrix} b^2 m & bS_E \\ bS_E & I_E \end{bmatrix} \tag{24}$$

being $m = \int_{-b}^{b} dm$ the aerofoil mass, $I_E = \int_{-b}^{b} (x - x_E)^2\, dm$ the moment of inertia with respect to the E, and $S_E = \int_{-b}^{b} (x - x_E)\, dm$ the static moment or first moment of area with respect to the E too.

Next, some extra parameters are defined in order to make the matrices dimensionless. Those are the dimensionless turn radius, i_θ, the distance between the E and G, r_θ, and the ratio between them, r:

$$i_\theta = \sqrt{\frac{I_E}{mb^2}} \qquad r_\theta = \frac{S_E}{mb} = \frac{x_G - x_E}{b} = d - a \qquad r = \frac{r_\theta}{i_\theta} \tag{25}$$

Thus, the M defined in Equation 24 is written as a function of a dimensionless matrix denoted by $\mathcal{M}$:

$$\boldsymbol{M} = \begin{bmatrix} b^2 m & bS_E \\ bS_E & I_E \end{bmatrix} = mb^2 \begin{bmatrix} 1 & r_\theta \\ r_\theta & i_\theta^2 \end{bmatrix} = mb^2 \boldsymbol{\mathcal{M}} \tag{26}$$

Stiffness matrix

In this case, the stiffness matrix is obtained through the expression of the potential energy of the system. Therefore, assuming that the aerofoil is in its resting position lying on the x axis, the deformation energy is:

$$\mathcal{U} = \frac{1}{2}k_h h^2 + \frac{1}{2}k_\theta \theta^2 = \frac{1}{2}\{h/b,\ \theta\} \begin{bmatrix} k_h b^2 & 0 \\ 0 & k_\theta \end{bmatrix} \{h/b,\ \theta\}^T = \frac{1}{2}\boldsymbol{u}^T \boldsymbol{K} \boldsymbol{u} \tag{27}$$

being the stiffness matrix $\boldsymbol{K}$:

$$\boldsymbol{K} = \begin{bmatrix} k_h b^2 & 0 \\ 0 & k_\theta \end{bmatrix} \tag{28}$$

In the same way as with the mass matrix, it is preferable to work with dimensionless matrices. Thus, besides the dimensionless turn radius defined in Equation 25, three more dimensionless parameters are used. Those are the frequencies ω_θ and ω_h created from the stiffness values of the springs. The first one is the frequency of the system considering the aerofoil has a fixed joint in the E, whereas the second one is the frequency of the system assuming the aerofoil cannot turn. Finally, η represents the ratio between them:

$$\omega_h = \sqrt{\frac{k_h}{m}} \qquad \omega_\theta = \sqrt{\frac{k_\theta}{I_E}} \qquad \eta = \frac{\omega_h}{\omega_\theta} \tag{29}$$

Finally, rearranging $\boldsymbol{K}$ (Equation 28) and introducing those new parameters, the dimensionless stiffness matrix $\boldsymbol{\mathcal{K}}$ is reached, which only depends on the frequencies ratio and on the dimensionless turn radius:

$$\boldsymbol{K} = \begin{bmatrix} k_h b^2 & 0 \\ 0 & k_\theta \end{bmatrix} = m b^2 \omega_\theta^2 \begin{bmatrix} \eta^2 & 0 \\ 0 & i_\theta^2 \end{bmatrix} = m b^2 \omega_\theta \boldsymbol{\mathcal{K}} \tag{30}$$

A.2 Intermediate steps for the generalised forces vector

When a virtual variation $\delta\boldsymbol{u}$ occurs, the lifting surface changes its geometry from z_p to $z_p + \delta z_p$. Then, the pressure acting on a point x performs a work of $\Delta p(x,t)\ dx\ \delta z_p(x,t)$. Thus, the whole pressure distribution over the aerofoil do a virtual work of:

$$\delta\mathcal{W} = \int_{-b}^{b} \Delta p(x,t)\delta z_p(x,t)\, dx \tag{31}$$

Again, it is assumed that the aerofoil is deformed as a solid rigid, moving downwards $h(t)$ and rotating an angle θ with respect to the E. Then, recalling the expression for z_p in Equation 21, substituting it in Equation 31, and rearranging the integrals:

$$\delta\mathcal{W} = \int_{-b}^{b} \left(-\delta h(t) - \delta\theta(t)(x - x_E)\right)\ \Delta p(x,t)\, dx \tag{32}$$

$$\delta\mathcal{W} = \delta h \left(-\int_{-b}^{b} \Delta p(x,t)\, dx \right) + \delta\theta \left(-\int_{-b}^{b} (x - ab)\Delta p(x,t)\, dx \right) \tag{33}$$

Paying attention to the integrals that follow δh and $\delta\theta$, the lift and the aerodynamic moment in the E ($x = ab$) are found, respectively. Therefore, writing Equation 33 as a function of $\delta\boldsymbol{u}$, the generalised forces vector is reached:

$$\delta\mathcal{W} = \delta h(-L) + \delta\theta M_a = \{\delta h/b,\ \delta\theta\}\{-Lb,\ M_a\}^T = \delta\boldsymbol{u}^T\boldsymbol{Q} \tag{34}$$

The unsteady aerodynamics linearised in incompressible regime allows an analytical solution for the lift and the aerodynamic moment in a concrete point [10]. Both expressions, shown in Equation 35, contain terms related to the circulatory nature (subindex Q) and the apparent mass (subindex A), being the first ones multiplied by the Theodorsen function $\mathcal{C}(\mathcal{K})$.

$$L = 2\pi\rho_\infty U_\infty b\, \mathcal{C}(\mathcal{K}) \left[\dot{h} + U_\infty\theta + b\left(\frac{1}{2} - a\right)\dot{\theta} \right] + \pi\rho_\infty b^2 \left(\ddot{h} + U_\infty\dot{\theta} - ab\ddot{\theta} \right) \tag{35}$$

$$L = \mathcal{C}(\mathcal{K})L_Q + L_A \tag{36}$$

$$M_a = 2\pi\rho_\infty U_\infty b^2\, \mathcal{C}(\mathcal{K}) \left(\frac{1}{2} - a\right) \left[\dot{h} + U_\infty\theta + b\left(\frac{1}{2} - a\right)\dot{\theta} \right] + \tag{37}$$

$$+\pi\rho_\infty b^2 \left[ab\ddot{h} - U_\infty b\left(\frac{1}{2} - a\right)\dot{\theta} - b^2\left(\frac{1}{8} + a^2\right)\ddot{\theta} \right]$$

$$M_a = \mathcal{C}(\mathcal{K})M_Q + M_A \tag{38}$$

It must me mentioned that $\mathcal{K}$ refers to the reduced frequency of the system, which is calculated as $\mathcal{K} = \frac{\omega\, b}{U_\infty}$ and controls the lifting effect due to the eddies distribution along the wake in the aerofoil. It will help to define the unsteadiness degree of the problem.

Finally, the generalised forces vector can be written as a function of $\boldsymbol{u}$ in an easy way as follows:

$$\boldsymbol{Q}(t) = \pi\rho_\infty U_\infty^2 b^2 \boldsymbol{A}\boldsymbol{u}(t) + \pi\rho_\infty U_\infty b^3 \boldsymbol{B}\dot{\boldsymbol{u}}(t) + \pi\rho_\infty b^4 \boldsymbol{C}\ddot{\boldsymbol{u}}(t) \tag{39}$$

being:

$$\boldsymbol{A} = \mathcal{C}(\mathcal{K}) \begin{bmatrix} 0 & -2 \\ 0 & 1 + 2a\theta \end{bmatrix} \qquad \boldsymbol{C} = \begin{bmatrix} -1 & a \\ a & -a^2 - 1/8 \end{bmatrix} \tag{40}$$

$$\boldsymbol{B} = \mathcal{C}(\mathcal{K}) \begin{bmatrix} -2 & -1 + 2a \\ 1 + 2a & 1/2 - 2a^2 \end{bmatrix} + \begin{bmatrix} 0 & -1 \\ 0 & -1/2 + a \end{bmatrix} \tag{41}$$

A.3 Steady aerodynamics model - Intermediate steps

The solution to the system written in Eq.6 will be the lineal combination of the set of solutions of its homogeneous equation. Looking for solutions of the kind $\boldsymbol{u}(t) = \bar{\boldsymbol{u}}e^{iwt}$, the following eigenvalue problem is reached:

$$[-\omega^2 \boldsymbol{M} + \boldsymbol{K} - \pi\rho_\infty U_\infty^2 b^2 \boldsymbol{A}]\ \bar{\boldsymbol{u}} = 0 \tag{42}$$

Introducing now the expressions of the dimensionless mass and stiffness matrices obtained in Equation 26 and 30:

$$[-\omega^2 mb^2 \boldsymbol{\mathcal{M}} + mb^2\omega_\theta^2 \boldsymbol{\mathcal{K}} - \pi\rho_\infty U_\infty^2 b^2 \boldsymbol{A}]\ \bar{\boldsymbol{u}} = 0 \tag{43}$$

Dividing by mb^2, using $\mu = m/\pi\rho_\infty b^2$ as the aerofoil mass coefficient, defining the dimensionless frequency as $\lambda = \omega/\omega_\theta$ and the dimensionless flight velocity as $V_\infty = U_\infty/b\omega_\theta$, the typical eigenvalue problem appears, referring $\mathcal{K}_{eq}$ to the equivalent stiffness matrix:

$$\left[-\lambda^2\mathcal{M} + \boldsymbol{\mathcal{K}} - \frac{V_\infty^2}{\mu}\boldsymbol{A}\right]\bar{\boldsymbol{u}} = \left[-\lambda^2\mathcal{M} + \boldsymbol{\mathcal{K}_{eq}}\right]\bar{\boldsymbol{u}} = 0 \tag{44}$$

Just calculating the determinant of the expression between brackets, $D(\lambda)$, and equalling it to zero, the values of λ are obtained.

A.3.1 Frequency plots - Extended discussion

Before the flight, when $V_\infty = 0$, the frequencies are found in the real axis. Those are the natural frequencies (no flight speed, no aerodynamic forces acting on the aerofoil) and they are represented with a circumference in the three graphs of Figure 4.

Looking at Equation 7, it is noticeable that $\mathcal{K}$ starts decreasing its values as flight velocity increases. Paying special attention to the non-null terms in $\boldsymbol{A}$, which is multiplied by V_∞, they are directly connected to the torsion mode since those multiply θ. Thus, the torsion frequency presents a larger variation than the bending one. This last one is also affected since both are coupled (both vibration modes depend on h and θ), but to a lesser degree.

In such a way, the frequencies become closer to each other along the positive real axis. Before reaching the same value, the solution is stable, corresponding to the case of free vibrations without dissipative damping. It is remarked that only one pair of frequencies is plotted. The conjugated ones would do the same but in the negative part.

The point where the frequencies match is known as $coalescense$. From this point on, real part Ω will be the same for both modes but they will come into the complex plane, drawing an ellipse in the root locus.

Now, the solution becomes unstable since some terms start depending on a positive time dependant exponential function (right part of Equation 45) that will cause the structural failure:

$$\boldsymbol{u}(t) = e^{-g\omega_\theta t}(C_1\psi e^{i\Omega\omega_\theta t} + C_4\psi^* e^{-i\Omega\omega_\theta t}) + e^{g\omega_\theta t}(C_3\psi e^{-i\Omega\omega_\theta t} + C_2\psi^* e^{i\Omega\omega_\theta t}) \tag{45}$$

being ψ the eigenvectors associated to the eigenvalues λ and C_x the constants that will be determined through the initial conditions noted in Equation 6.

Therefore, it can be said that the point represented with a square is the last stable one, characterised by a velocity $V_f = U_f/b\omega_\theta$ which is known as flutter velocity, and a frequency $\lambda_f = \Omega_f$, called flutter frequency. Thus, it is essential to fly always at lower velocities than this flutter velocity. Otherwise, the structure will dramatically and irrevocably fail.

A.3.2 First order approximation - Complete development

The first order approximation theory is used with the term V_∞/μ to find the flutter velocity associated to the previous flutter frequency. The fact that the decoupled bending and torsion frequencies fulfil that $\eta = \omega_/\omega_\theta \ll 1$. Thus, S can be consider to be zero and consequently, according to Equation 11, R must be also zero. In this way:

$$R(\eta = 0) = \frac{(1+2d)\frac{V_\infty^2}{\mu} - i_\theta^2}{i_\theta^2(1-r^2)} = 0 \quad \rightarrow \quad \frac{V_\infty^2}{\mu}(\eta = 0) = \frac{i_\theta^2}{1+2d} \tag{46}$$

Deriving $R^2 - 4S$ with respect to η and assuming it to be zero again:

$$\frac{V_\infty}{\mu}'(\eta = 0) = \left(-\frac{\frac{\partial R}{\partial \eta} - 2\frac{\partial S}{\partial \eta}}{\frac{\partial R}{\partial \frac{V_\infty^2}{\mu}} - 2\frac{\partial S}{\partial \frac{V_\infty^2}{\mu}}} \right)_{\eta\to 0, \frac{V_\infty^2}{mu} = \frac{V_\infty^2}{mu}(\eta=0)} = -2\frac{V_\infty^2}{\mu}(\eta = 0)\sqrt{\frac{2r_\theta(1-r^2)}{1+2d}} \tag{47}$$

Finally, just applying the first order approximation, the expression for flutter velocity is found:

$$\frac{V_\infty^2}{\mu}(\eta) \approx \frac{V_\infty^2}{\mu}(\eta = 0) + \frac{V_\infty^2}{\mu}'(\eta = 0)\eta \tag{48}$$

$$\frac{V_f^2}{\mu} \approx \frac{i_\theta^2}{1+2d}\left[1 - 2\eta\frac{i_G}{i_\theta}\sqrt{\frac{2(d-a)}{1+2d}}\right] \tag{49}$$

being $i_G = i_\theta\sqrt{1-r^2}$. It must be beard in mind that the expression shown in Equation 49 gives enough accurate results within the range $0 \leq \eta \leq 0.5$ since it has been considered $\eta \ll 1$. Besides that, the reduced frequency $\mathcal{K}$ has been also considered to be really small and therefore the velocity and acceleration terms of $\boldsymbol{u}$ were neglected.

However, it is not usually that the flutter $\mathcal{K}$ is enough small to make such assumption. That is why this unsteady model should only be used as a first approximation of the flutter velocity value. It basically gives information about how the locations of E and G affect the stability of the aerofoil, but it does not consider effect of the aerofoil velocity and acceleration in the aerodynamic pressures.

A.4 Quasi-steady aerodynamics model - Intermediate steps

A.4.1 New matrix coefficients

The matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, which were used in the steady approach, are substituted by other similar ones, $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$, which are adapted to include the previous aerodynamic derivatives/coefficients to remove the direct influence of $\mathcal{C}(\mathcal{K})$. Thus, in order to keep the structure of the generalised forces vector $\boldsymbol{Q} = \{-Lb,\ M_a\}$, those new matrices are:

$$\boldsymbol{\mathcal{A}} = \begin{bmatrix} 0 & -C_{l\theta}/\pi \\ 0 & 2C_{m\theta}/\pi \end{bmatrix} \qquad \boldsymbol{\mathcal{B}} = \begin{bmatrix} -C_{l\theta}/\pi & -C_{l\dot{\theta}/\pi} \\ 2C_{m\theta}/\pi & 2C_{m\dot{\theta}/\pi} \end{bmatrix} \tag{50}$$

A.4.2 Expressions of the aerodynamic derivatives

The value of C_{l_θ} is given by the thin aerofoil theory for a cambered aerofoil. It corresponds to the value of the slope $\frac{\partial C_l}{\partial \theta}$, which is 2π. From this, the value of C_{m_θ} can be calculated as follows:

$$C_{m_\theta} = \frac{(\frac{1}{2} + a)C_{l_\theta}}{2} = \left(\frac{1}{2} + a\right)\pi \tag{51}$$

Regarding $C_{m_{\dot{\theta}}}$ and $C_{m_{\dot{\theta}}}$, their expressions are obtained deriving the next approximations for the lift and the aerodynamic moment (per unit span), which are function of them:

$$L \approx \frac{1}{2}\rho_\infty U_\infty^2 2b \left[C_{l_\theta}\left(\theta + \frac{\dot{h}}{U_\infty}\right) + C_{l_{\dot{\theta}}}\left(\frac{b\dot{\theta}}{U_\infty}\right)\right] \tag{52}$$

$$M_a \approx \frac{1}{2}\rho_\infty U_\infty^2 (2b)^2 \left[C_{m_\theta}\left(\theta + \frac{\dot{h}}{U_\infty}\right) + C_{m_{\dot{\theta}}}\left(\frac{b\dot{\theta}}{U_\infty}\right)\right] \tag{53}$$

Knowing these expressions, it is just needed to solve for the still missing derivatives, $C_{l_{\dot{\theta}}}$ and $C_{m_{\dot{\theta}}}$, and differentiate both with respect to $\dot{\theta}$. In such a way:

$$C_L \approx \frac{L}{\rho_\infty {U_\infty}^2 b} \rightarrow \frac{\delta C_L}{\delta \dot{\theta}} = C_{l_{\dot{\theta}}} = \frac{\pi b}{U_\infty}(1 + 2{\cdot}|\mathcal{C}(k)|(0.5 - a)) \tag{54}$$

$$C_m \approx \frac{M_a}{2\rho_\infty {U_\infty}^2 b^2} \rightarrow \frac{\delta C_m}{\delta \dot{\theta}} = C_{m_{\dot{\theta}}} = \frac{\pi b(0.5 - a)}{U_\infty}(-0.5 + |\mathcal{C}(k)|(0.5 + a)) \tag{55}$$

They are calculated with the Theodorsen function $\mathcal{C}(k)$ using the value of the flutter reduced frequency obtained through the steady model:

$$K_f = \frac{\omega_f b}{U_\infty} = \frac{\omega_f/\omega_\theta}{\frac{U_\infty}{b\omega_\theta}} = \frac{\lambda_f}{V_f} \quad \rightarrow \quad \mathcal{C}(k_f) = \frac{H_1^{(2)}(k_f)}{H_1^{(2)}(k_f) + iH_0^{(2)}(k_f)} \tag{56}$$

As it can be seen in Equation 56, $\mathcal{C}(\mathcal{K})$ depends on the Hankel functions of the second kind, also known as Bessel function of the third kind. Those can be directly obtained through the MATLAB function $(H = besselh(nu, Z))$, so it is not worthwhile to spend time and space developing them.

A.4.3 Theodorsen's method - Mathematical perspective

It is easily noticeable that the determinant of the expression between brackets of the equation reached after substituting λ by ω_f in Eq.15 will give complex numbers of the form $\mathcal{P} = \mathcal{P}_R + i\mathcal{P}_I$, which will depend on V_f and Ω_f.

$$\left[-\lambda^2\mathcal{M} + \mathcal{K} - \frac{V_\infty^2}{\mu}\mathcal{A} - \frac{i\lambda V_\infty}{\mu}\mathcal{B}\right]\bar{\boldsymbol{u}} = 0 \tag{57}$$

Thus, in order to fulfil Equation 57, the real and the imaginary part of $\mathcal{P}$ must be zero.

Finally, just solving the system of equations shown in Equation 58, the values of the flutter frequency and velocity are obtained.

$$\begin{cases} \mathcal{P}_R(\Omega_f, V_f) = 0 \\ \mathcal{P}_I(\Omega_f, V_f) = 0 \end{cases} \tag{58}$$

It has to be taken into account that multiple solutions will appear, though the lowest real and positive values will be the ones that will represent the beginning of the flutter phenomenon.

B MATLAB scripts

B.1 Initial function code

```
function dimensionless = fcn(user)
b = user(1); %half of the chord
rho_inf = user(2); %freestream density
m = user(3);  %fin mass
mu = m/(pi*rho_inf*b^2); %mass ratio

omega_theta = user(6); %frequency of the system if it's
                       %suppossed there's a fix joint on the E
omega_h = user(7); %frequency of the system if it's
                   %suppossed it cannot turn
eta = omega_h/omega_theta; %decoupled frequency ratio

IE = user(8);
i_theta = user(9); %radius of gyration

if i_theta==0
    i_theta = sqrt(IE/(m*b^2));
end

a = user(4)/b; %dimensionless distance to E
d = user(5)/b; %dimensionless distance to G
r_theta = d - a; %dimensionless distance between E and G
r = r_theta/i_theta; %mesures the coupling degree
iG = sqrt(1 - r^2)*i_theta;

dimensionless = [mu eta i_theta r_theta r iG a d]; %outputs
```

Figure 26: *Initial function code - Dimensionless parameters*

B.2 Steady model 1st function code. Frequency vectors.

```
function [V_inf, torsion1_real_s, torsion2_real_s, bending1_real_s, bending2_real_s,
          torsion1_imag_s, torsion2_imag_s, bending1_imag_s, bending2_imag_s]
          = fcn(dimensionless, limit_s)
mu2 = dimensionless(1);
eta2 = dimensionless(2);
i_theta2 = dimensionless(3);
r_theta2 = dimensionless(4);
r2 = dimensionless(5);
iG2 = dimensionless(6);
a2 = dimensionless(7);
d2 = dimensionless(8);

V_inf = [0:10^(-3.5):5];
len = length(V_inf);

R = zeros(1,len);
S = zeros(1,len);

% Create vectors for R and S
for i=1:length(V_inf)

    R(1,i) = ((1+2*d2)*V_inf(i)^2/mu2 - i_theta2^2*(1+eta2^2))/(i_theta2^2*(1-r2^2));
    S(1,i) = (i_theta2^2 - (1+2*a2)*(V_inf(i)^2/mu2))*eta2^2/(i_theta2^2*(1-r2^2));

end

```

Figure 27: *Steady model 1st function code - Part 1*

```
% Create a vector for the roots
vector_roots_real = zeros(4,len);
vector_roots_imag = zeros(4,len);
for j=1:4
    for i=1:length(V_inf)
        p = [1, 0, R(i), 0, S(i)];
        raices = roots(p);
        vector_roots_real(j,i) = real(raices(j));
        vector_roots_imag(j,i) = imag(raices(j));
    end
end

torsion1_real_s = vector_roots_real(1,:);
torsion2_real_s = vector_roots_real(2,:);
bending1_real_s = vector_roots_real(3,:);
bending2_real_s = vector_roots_real(4,:);

torsion1_imag_s = vector_roots_imag(1,:);
torsion2_imag_s = vector_roots_imag(2,:);
bending1_imag_s = vector_roots_imag(3,:);
bending2_imag_s = vector_roots_imag(4,:);

values1 = zeros(1,1);
values2 = zeros(1,1);
values3 = zeros(1,1);
values4 = zeros(1,1);

```

Figure 28: *Steady model 1st function code - Part 2*

```
%Organise vectors, locate possible discontinuities
for i=2:length(vector_roots_real)
    if abs(torsion1_real_s(i)-torsion1_real_s(i-1))>0.1
        values1 = [values1, i];
    end
    if abs(torsion2_real_s(i)-torsion2_real_s(i-1))>0.1
        values2 = [values2, i];
    end
    if abs(bending1_real_s(i)-bending1_real_s(i-1))>0.1
        values3 = [values3, i];
    end
    if abs(bending2_real_s(i)-bending2_real_s(i-1))>0.1
        values4 = [values4, i];
    end
end
if length(values1)>1
   for i=values1(2):length(vector_roots_real)
       torsion1_real_s(i)=vector_roots_real(2,i);
       torsion1_imag_s(i)=vector_roots_imag(2,i);
       torsion2_real_s(i)=vector_roots_real(1,i);
       torsion2_imag_s(i)=vector_roots_imag(1,i);
   end
   if length(values2)>1
       for i=(values2(3)):length(vector_roots_real)
           torsion1_real_s(i)=vector_roots_real(3,i);
           torsion1_imag_s(i)=vector_roots_imag(3,i);
           bending1_real_s(i)=vector_roots_real(2,i);
           bending1_imag_s(i)=vector_roots_imag(2,i);
       end
```

Figure 29: *Steady model 1st function code - Part 3*

```
   end
elseif length(values2)>1
    for i=values2(2):length(vector_roots_real)
           torsion2_real_s(i)=vector_roots_real(3,i);
           torsion2_imag_s(i)=vector_roots_imag(3,i);
           bending1_real_s(i)=vector_roots_real(2,i);
           bending1_imag_s(i)=vector_roots_imag(2,i);
    end
end
```

Figure 30: *Steady model 1st function code - Part 4*

B.3 Steady model 2nd function code. Value of the fin flutter speed

```
function [Vf_s, lambdaf_s] = fcn(dimensionless)

coder.extrinsic('solveSyms');
Vf_s = 0;
lambdaf_s = 0;
[Vf_s, lambdaf_s] = solveSyms(dimensionless);

```

Figure 31: *Steady model 2nd function code - Part 1*

```
function [Vf_s, lambdaf_s]  = solveSyms(u)

mu = u(1);
eta = u(2);
i_theta = u(3);
r = u(5);
a = u(7);
d = u(8);

syms V_inf_eq

R_eq = ((1+2*d)*V_inf_eq^2/mu - i_theta^2*(1+eta^2))/(i_theta^2*(1-r^2));
S_eq = (i_theta^2 - (1+2*a)*(V_inf_eq^2/mu))*eta^2/(i_theta^2*(1-r^2));

eq1 = R_eq^2 - 4*S_eq == 0;
solution = double(solve(eq1, V_inf_eq))
% Choose the minimum and positve value
V_sol = 100;
for i=1:length(solution)
    if (abs(solution(i))<= (abs(V_sol)) && (solution(i)>=0))
        V_sol = solution(i);
    end
end
lambda_flutter = sqrt(-((1+2*d)*V_sol^2/mu - i_theta^2*(1+eta^2))/(2*i_theta^2*(1-r^2)));
Vf_s = double(V_sol);
lambdaf_s = double(lambda_flutter);
end
```

Figure 32: *Steady model 2nd function code - Part 2*

B.4 Theodorsen's function code

```
function C = fcn(Red_freq)
C = abs((besselh(1, 2, Red_freq))/(besselh(1, 2, Red_freq) + (1i)*besselh(0, 2, Red_freq)));
end
```

Figure 33: *Quasi-steady model 1st - Theodorsen's function code*

B.5 Quasi-steady model 2^{nd} function code. Frequency vectors.

```
function [V_inf2, V_flutter_graph, torsion1_real_qs, torsion2_real_qs, bending1_real_qs,
          bending2_real_qs, torsion1_imag_qs, torsion2_imag_qs, bending1_imag_qs, bending2_imag_qs]
          = fcn(dimensionless, coeff, limit_qs)
Cltheta = coeff(1);
Cmtheta = coeff(2);
Cltheta_dot = coeff(3);
Cmtheta_dot = coeff(4);

mu4 = dimensionless(1);
eta4 = dimensionless(2);
i_theta4 = dimensionless(3);
r_theta4 = dimensionless(4);
r4 = dimensionless(5);
iG4 = dimensionless(6);
a4 = dimensionless(7);
d4 = dimensionless(8);
V_inf2 = [0:10^(-3.5):limit_qs];
len = length(V_inf2);

%determinant_curves = det(s^2*M_bar + K_bar - V_inf^2*A/mu - s*V_inf*B/mu);

% a = (i_theta^2*mu^2*pi^2 - mu^2*r_theta^2*pi^2)/(mu^2*pi^2)
% b = (- 2*pi*Cmtheta_dot*V_inf*mu + pi*Cltheta*V_inf*i_theta^2*mu -
%pi*Cltheta_dot*V_inf*mu*r_theta + 2*pi*Cmtheta*V_inf*mu*r_theta)/(mu^2*pi^2)
% c = (- 2*Cltheta*Cmtheta_dot*V_inf^2 + 2*Cltheta_dot*Cmtheta*V_inf^2 + i_theta^2*mu^2*pi^2
%+ eta^2*i_theta^2*mu^2*pi^2 - 2*pi*Cmtheta*V_inf^2*mu - pi*Cltheta*V_inf^2*mu*r_theta)/(mu^2*pi^2)
% d = (- 2*pi*Cmtheta_dot*V_inf*eta^2*mu + pi*Cltheta*V_inf*i_theta^2*mu)/(mu^2*pi^2)
% e = (eta^2*i_theta^2*mu^2*pi^2 - 2*pi*Cmtheta*V_inf^2*eta^2*mu)/(mu^2*pi^2)

```

Figure 34: *Quasi-steady model 2^{nd} function code - Part 1*

```
a = zeros(1,len);
b2 = zeros(1,len);
c = zeros(1,len);
d = zeros(1,len);
e = zeros(1,len);

% Create vectors for R and S
for i=1:len
 a(i) = (+ i_theta4^2*mu4^2*pi^2 - mu4^2*r_theta4^2*pi^2)/(mu4^2*pi^2);
 b2(i) = (- 2*pi*Cmtheta_dot*V_inf2(i)*mu4 + pi*Cltheta*V_inf2(i)*i_theta4^2*mu4 -
 pi*Cltheta_dot*V_inf2(i)*mu4*r_theta4 + 2*pi*Cmtheta*V_inf2(i)*mu4*r_theta4)/(mu4^2*pi^2);
 c(i) = (- 2*Cltheta*Cmtheta_dot*V_inf2(i)^2 + 2*Cltheta_dot*Cmtheta*V_inf2(i)^2
 + i_theta4^2*mu4^2*pi^2 + eta4^2*i_theta4^2*mu4^2*pi^2 - 2*pi*Cmtheta*V_inf2(i)^2*mu4
 - pi*Cltheta*V_inf2(i)^2*mu4*r_theta4)/(mu4^2*pi^2);
 d(i) = (+ pi*Cltheta*V_inf2(i)*i_theta4^2*mu4 - 2*pi*Cmtheta_dot*V_inf2(i)*eta4^2*mu4)/(mu4^2*pi^2);
 e(i) = (+ eta4^2*i_theta4^2*mu4^2*pi^2 - 2*pi*Cmtheta*V_inf2(i)^2*eta4^2*mu4)/(mu4^2*pi^2);
end

% Create a vector for the roots
vector_roots_realqs = zeros(4,len);
vector_roots_imagqs = zeros(4,len);
for j=1:4
    for t=1:len
        p_qs = [a(t), b2(t), c(t), d(t), e(t)];
        raices_qs = roots(p_qs)*(-1i);
        vector_roots_realqs(j,t) = real(raices_qs(j));
        vector_roots_imagqs(j,t) = imag(raices_qs(j));
    end
end
```

Figure 35: *Quasi-steady model 2^{nd} function code - Part 2*

```
59
60 -  number1 = zeros(1,1);
61 -  for i=2:len
62 -      for j=1:4
63 -          if (vector_roots_imagqs(j,i)<0)
64 -              number1 = [number1, i];
65            end
66        end
67    end
68 -  number1 = number1(2);
69 -  V_flutter_graph = V_inf2(number1);
70
71 -  torsion1_real_qs = vector_roots_realqs(1,:);
72 -  torsion2_real_qs = vector_roots_realqs(2,:);
73 -  bending1_real_qs = vector_roots_realqs(3,:);
74 -  bending2_real_qs = vector_roots_realqs(4,:);
75
76 -  torsion1_imag_qs = vector_roots_imagqs(1,:);
77 -  torsion2_imag_qs = vector_roots_imagqs(2,:);
78 -  bending1_imag_qs = vector_roots_imagqs(3,:);
79 -  bending2_imag_qs = vector_roots_imagqs(4,:);
80
81 -  values1 = zeros(1,1);
82 -  values2 = zeros(1,1);
83 -  values3 = zeros(1,1);
84 -  values4 = zeros(1,1);
```

Figure 36: *Quasi-steady model 2nd function code - Part 3*

```
85    % Order vectors
86 -  for i=2:length(vector_roots_realqs)
87
88 -      if abs(torsion1_real_qs(i)-torsion1_real_qs(i-1))>0.1
89 -          values1 = [values1, i];
90        end
91
92 -      if abs(torsion2_real_qs(i)-torsion2_real_qs(i-1))>0.1
93 -          values2 = [values2, i];
94        end
95
96 -      if abs(bending1_real_qs(i)-bending1_real_qs(i-1))>0.1
97 -          values3 = [values3, i];
98        end
99
100 -     if abs(bending2_real_qs(i)-bending2_real_qs(i-1))>0.1
101 -         values4 = [values4, i];
102       end
103
104   end
105
106 -  if length(values1)>1
107 -      for i=values1(2):length(vector_roots_realqs)
108 -          torsion1_real_qs(i)=vector_roots_realqs(2,i);
109 -          torsion1_imag_qs(i)=vector_roots_imagqs(2,i);
110 -          torsion2_real_qs(i)=vector_roots_realqs(1,i);
111 -          torsion2_imag_qs(i)=vector_roots_imagqs(1,i);
112       end
113
```

Figure 37: *Quasi-steady model 2nd function code - Part 4*

```
114 -      if length(values2)>1
115 -          for i=(values2(3)):length(vector_roots_realqs)
116 -              torsion1_real_qs(i)=vector_roots_realqs(3,i);
117 -              torsion1_imag_qs(i)=vector_roots_imagqs(3,i);
118 -              bending1_real_qs(i)=vector_roots_realqs(2,i);
119 -              bending1_imag_qs(i)=vector_roots_imagqs(2,i);
120           end
121       end
122 -   elseif length(values2)>1
123 -       for i=values2(2):length(vector_roots_realqs)
124 -              torsion2_real_qs(i)=vector_roots_realqs(3,i);
125 -              torsion2_imag_qs(i)=vector_roots_imagqs(3,i);
126 -              bending1_real_qs(i)=vector_roots_realqs(2,i);
127 -              bending1_imag_qs(i)=vector_roots_imagqs(2,i);
128       end
129   end
```

Figure 38: *Quasi-steady model 2nd function code - Part 5*

B.6 Quasi-steady model 3rd function code. Flutter speed (Theodorsen's method)

```
function [Vf, omega_f] = fcn(dimensionless, coeff)

coder.extrinsic('solveSyms_qs');
Vf=0;
omega_f=0;
[Vf, omega_f]= solveSyms_qs(dimensionless, coeff);

```

Figure 39: *Quasi-steady model 3rd function code - Part 1*

```
function [Vf_qs, omega_f] = solveSyms_qs(u, coeff)
mu = u(1); i_theta = u(3); eta = u(2); r_theta = u(4);
r = u(5); iG = u(6);a = u(7); d = u(8);
Cltheta = coeff(1); Cmtheta = coeff(2); Cltheta_dot = coeff(3); Cmtheta_dot = coeff(4);

syms Vf_qs omega_f
real_det = (eta^2*i_theta^2*mu^2*pi^2 + 2*Cltheta*Cmtheta_dot*Vf_qs^2*omega_f^2
- 2*Cltheta_dot*Cmtheta*Vf_qs^2*omega_f^2 - i_theta^2*mu^2*omega_f^2*pi^2
+ i_theta^2*mu^2*omega_f^4*pi^2 - mu^2*omega_f^4*r_theta^2*pi^2 - eta^2*i_theta^2*mu^2*omega_f^2*pi^2
- 2*pi*Cmtheta*Vf_qs^2*eta^2*mu + 2*pi*Cmtheta*Vf_qs^2*mu*omega_f^2
+ pi*Cltheta*Vf_qs^2*mu*omega_f^2*r_theta)/(mu^2*pi^2);
imag_det = (pi*Cmtheta_dot*Vf_qs*mu*omega_f^3*2 - pi*Cltheta*Vf_qs*i_theta^2*mu*omega_f^3*1
- pi*Cmtheta_dot*Vf_qs*eta^2*mu*omega_f*2 + pi*Cltheta*Vf_qs*i_theta^2*mu*omega_f*1
+ pi*Cltheta_dot*Vf_qs*mu*omega_f^3*r_theta*1 - pi*Cmtheta*Vf_qs*mu*omega_f^3*r_theta*2)/(mu^2*pi^2);

solutions = (solve(real_det==0, imag_det==0, [omega_f, Vf_qs]));
omega_f_sol = double(solutions.omega_f); Vf_sol = double(solutions.Vf_qs);

Vflutter = 100; omega_flutter = 100; %High numbers to do the first comparison
for i=1:length(omega_f_sol)

    if ((abs(Vf_sol(i))<=abs(Vflutter))&& Vf_sol(i)>0&&imag(Vf_sol(i))==0&&omega_f_sol(i)>=0&&imag(omega_f_sol(i))==0)
        Vflutter = Vf_sol(i);
        omega_flutter = omega_f_sol(i);
    end
end
Vf_qs = double(Vflutter); omega_f = double(omega_flutter);
end
```

Figure 40: *Quasi-steady model 3^{r}d function code - Part 2*

C Simulink subsystems

C.1 First order approximation (steady model) subsystem

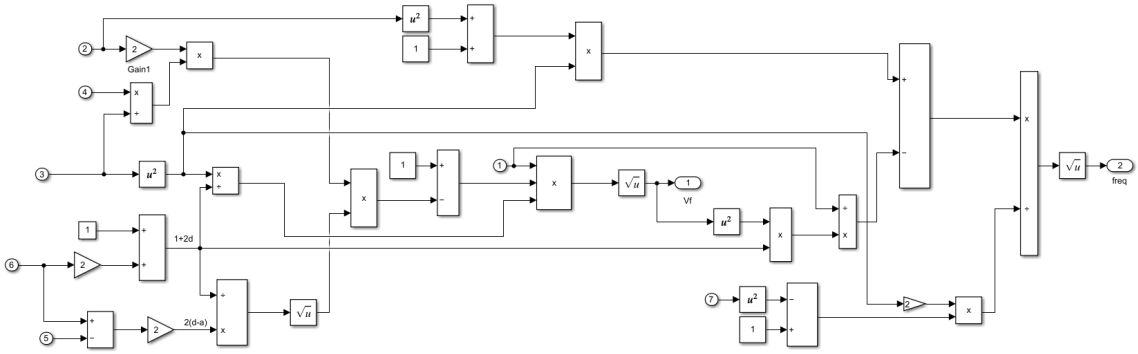


Figure 41: *First order approximation subsystem*

C.2 Aerodynamic derivatives (quasi-steady model) subsystem

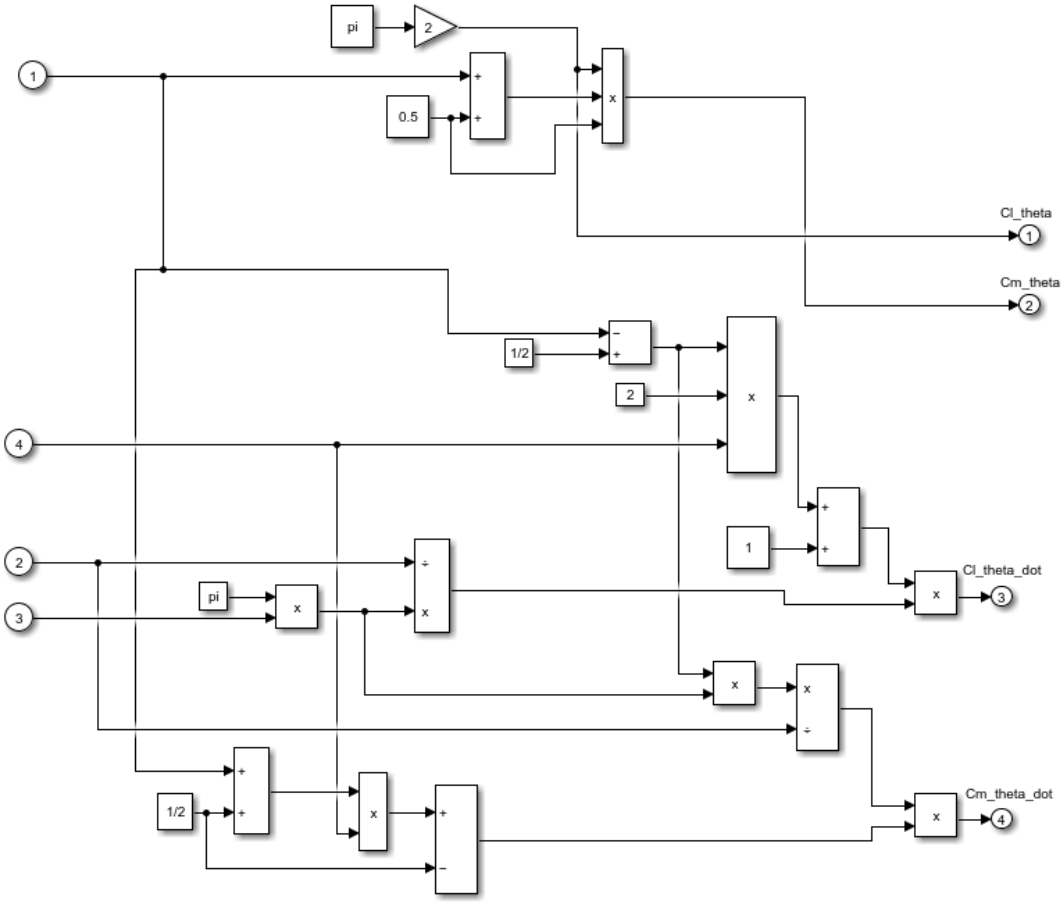


Figure 42: *Aerodynamic derivatives subsystem*

D Validation process - Relative error values

Just in case there is any confusion regarding how the relative errors are obtained, a sample calculation for the first error shown on the upper table is left below:

$$i_\theta^2 = 1/5,\ \ \omega_h/\omega_\theta = 0.2 \rightarrow Relative\ error = \frac{|V_f - V_{fNACA}|}{V_{fNACA}} \cdot 100 = 6.70\% \qquad (59)$$

$i_\theta^2 = 1/5$	NACA 685	First order approximation		Relative error	Steady model - 2nd function		Relative error
ω_h/ω_θ	Vf Exact	k_f	Vf		k_f	Vf	
0.2	2.150	0.1753	2.006	6.70%	0.1819	1.989	7.49%
0.4	2.025	0.3081	1.746	13.78%	0.3302	1.688	16.64%
0.6	1.750	0.4896	1.444	17.49%	0.5395	1.344	23.20%
0.8	1.395	0.8240	1.049	24.80%	0.8561	1.015	27.24%

$i_\theta^2 = 1/4$	NACA 685	First order approximation		Relative error	Steady model - 2nd function		Relative error
ω_h/ω_θ	Vf Exact	k_f	Vf		k_f	Vf	
0.2	2.400	0.1564	2.241	6.62%	0.1626	2.221	7.46%
0.4	2.200	0.2751	1.949	11.41%	0.2964	1.878	14.64%
0.6	1.850	0.4380	1.604	13.30%	0.4900	1.48	20.00%
0.8	1.400	0.7411	1.162	17.00%	0.8089	1.076	23.14%

$i_\theta^2 = 1/3$	NACA 685	First order approximation		Relative error	Steady model - 2nd function		Relative error
ω_h/ω_θ	Vf Exact	k_f	Vf		k_f	Vf	
0.2	2.650	0.1352	2.624	0.98%	0.1407	2.598	1.96%
0.4	2.225	0.2379	2.266	1.84%	0.2576	2.149	3.42%
0.6	2.050	0.3793	1.9	7.32%	0.4311	1.795	12.44%
0.8	1.400	0.6456	1.269	9.36%	0.7445	1.171	16.36%

Figure 43: *Relative errors between the steady model and the NACA Report No.685*

E LURA's fins CAD design

The geometry of the fins that the LURA's team is using in its current design can be observed in the figure below, being all the measures and tolerances in millimetres.

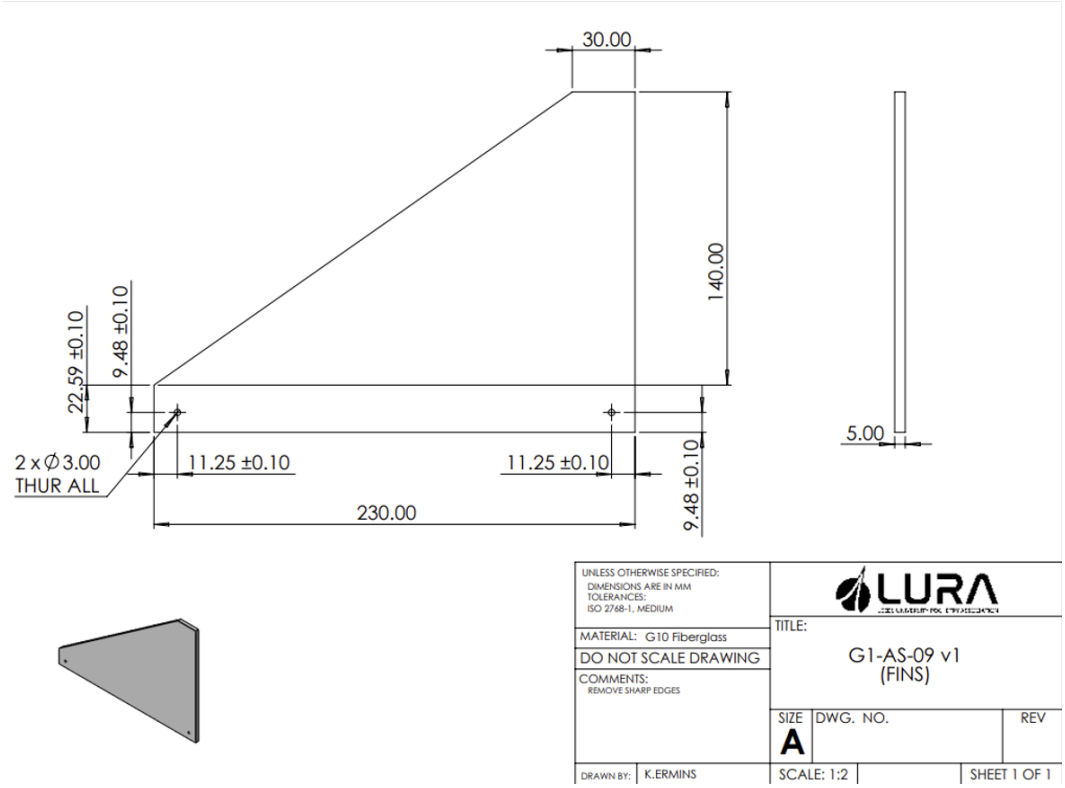


Figure 44: *CAD design of the current fins used by LURA*