Imperial College London

DEPARTMENT OF AERONAUTICS

Msc in Advanced Computational Methods for Aeronautics, Flow Management and Fluid-Structure interaction

External project at Dassault-Aviation
13/05/2008 – 16/09/2008

Prediction of aircraft aero-elastic stability including a nonlinear model of an actuator

By Camille Renvoisé

Supervisor at Dassault-Aviation : Gabriel Broux
Supervisor at Imperial College : Rafael Palacios

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

ACKNOWLEDGEMENTS

First of all, I would like to thank Gerard Menard, Eric Garrigues and Laurent Schmitt who gave me the opportunity to perform my master thesis at Dassault-Aviation.

Special thanks should be given to my supervisor at Dassault Aviation, Gabriel Broux, for his patience and his great help during my whole project.

Thanks are also due to Rafael Palacios for his support during these four months.

Finally, I would like to thanks the whole team of ASP for the pleasant atmosphere they keep up, especially Julie Malley, Gilbert Bouchardie and Khaled Khebache whom I share the office with.

PRESENTATION OF THE COMPANY

Dassault Aviation is one of the major players in the global aviation industry. A reasonably sized and financially secure private international group, with a presence in more than 70 countries across 5 continents, Dassault Aviation has been profitable ever since its creation in 1936. [1]

History

1928: creation of the "Société des Avions Marcel Bloch" (SAMB) company by Marcel Bloch.
1936: creation of Société Anonyme des Avions Marcel Bloch (SAAMB) which designed and developed prototypes that were mass produced by state-owned companies.
Due to the development of its business and its structure, the Dassault company was given various names from 1945 to 1960. It came to the forefront on a world-wide level and became able to supply combat aircrafts corresponding to French defense needs.
In 1966, the State decided to reorganise the aviation industry: Dassault was to concentrate on combat and business aircraft. Over the past 5 years, Falcons have represented on average 60% of sales realized.

Group Dassault-Aviation

Chairman and managing director: Charles EDELSTENNE

- Consolidated orders amount to 6.26 billion euros owing particularly to the sales of 212 FALCON, which sets an historical record in 2007.
- Consolidated 2007 sales amount to 4.08 billion euros.
- Consolidated operating profit is 503 million euros. It accounts for 12.3 % of consolidated sales. (2007)
- Consolidated net profit (total Group and net attributable Group profit) amounts to 382 million euros. It accounts for 9.4 % of consolidated sales. (2007)
- The net consolidated earnings per share is 37.8 euros. (2007)
- Staff of 11928 people (2006).

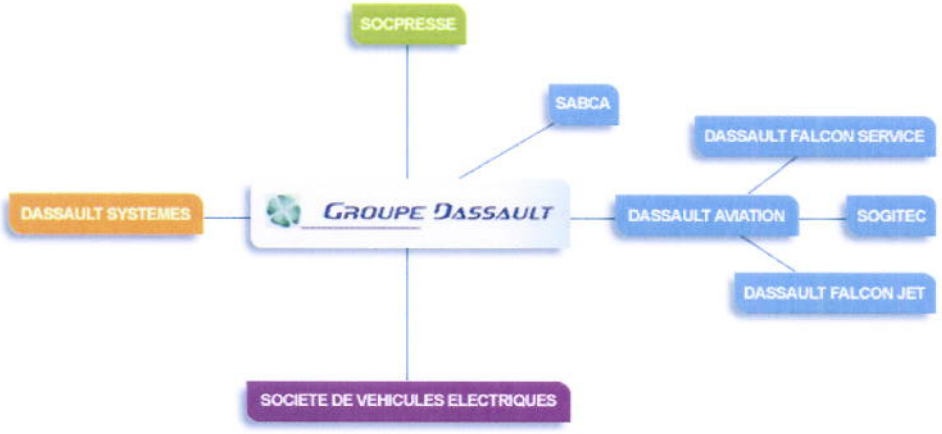


Figure 1: Group Dassault-Aviation

Planes

- ***Private jets: Falcon family***

- Flacon 50
- Falcon 2000DX / Falcon 2000LX
- Falcon 900EX/ Falcon 900DX
- Falcon 7X

- ***Military aircrafts***

- Mirage 2000

Put into service in 1984 in the French army, it is specialized in the air-air and air-surface missions. This polyvalent combat aircraft could be loaded up to 6,3T thank to several fixations.

About 600 Mirage 2000 and 2000-5 are operated worldwide and it equips nine armies.

- Rafale

This is a new generation of military aircraft. It is light a twin-engine with a high stealth. Thanks to its large range, its high payload, its shooting accuracy and its survivability, Rafale insures an amazing effective mission. It can reach mach 2 and 55 thousand feet.

- nEUROn

Dassault-Aviation has the leadership in this European project of fighting aircraft said "Unmanned".

It is an aircraft with a high stealth level and furnished with a internal storage for the air-surface shooting.

The first flight of the prototype is scheduled in 2011..

The sites

Dassault Aviation is present on several industrial sites in France. These are highly specialized. The US sites are dedicated to the aircraft furnishings and the support.

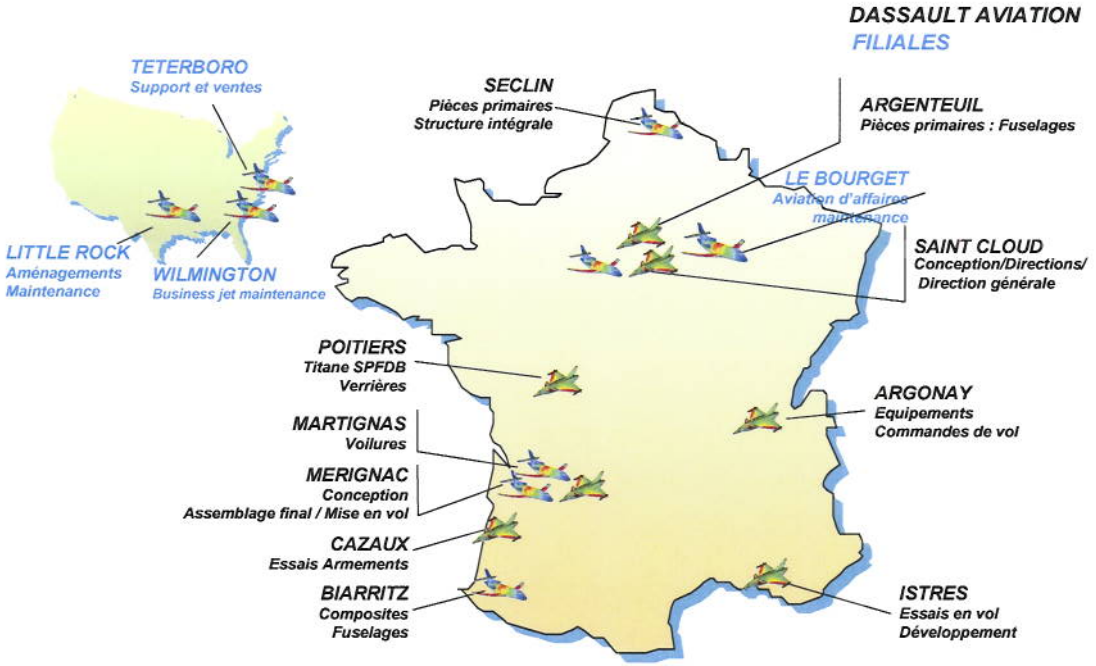


Figure 2: Sites of Dassault-Aviation

Saint-Cloud site

Saint Cloud site is situated in the west of Paris, close to the Seine. It regroups the design, research and development activities.

It is at the centre of the Dassault Aviation group, so it is also where we find the direction comity as well as the general secretarial activities.

Figure 3: Aerial view of Saint-Cloud site

Several directions take place there : the industry and sells direction (DGIA), the civil aircraft general direction (DGAC), the international general direction (DGI), The total quality general direction (DGQT), the military support general direction (DGSM), the technical general direction (DGT),the supply and purchase direction (DGAA), the financial and economics business direction (DAEF), the budget and general business direction, the commercial direction (DC) and eventually the commercial direction of the military business abroad (DCAME).

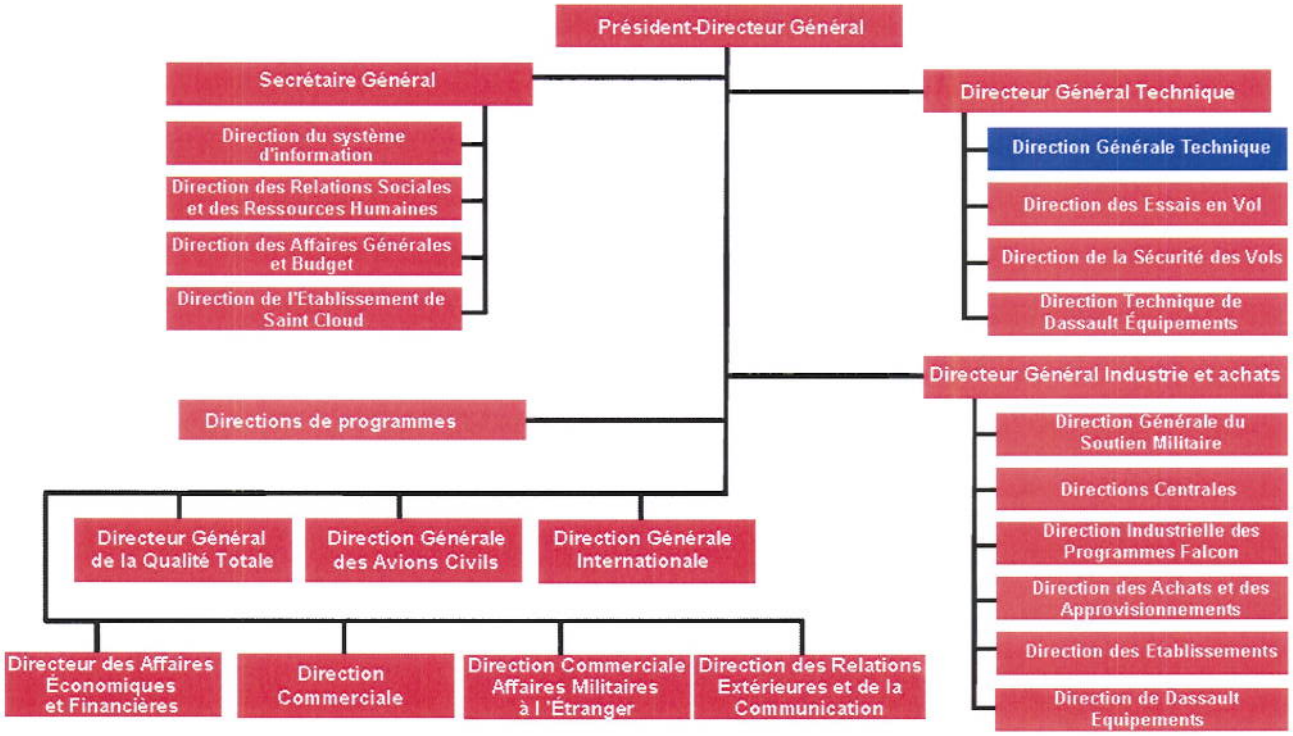


Figure 4: Organisation chart of Dassault-Aviation

DGT/DTAS/ASP

This project was carried out in the DGT/DTAS/ASP.

Mission of the direction DGT ("Technical General Direction"):

- Design and substantiation of the platform and system definition, from the feasibility phase to the operational phase during which DGT contributes to the product technical follow-up
- Program technical management
- Upstream research, scientific or technological studies and developments

Mission of the department DTAS ("Aero - Structure Design") :

- The preliminary and detailed design, including electrical circuits or hydraulic pipes
- The calculations of loading actions, flutter analysis, vibrations as well as acoustics.
- The definition and the methodology of the tools process for CFAO domains insuring the best synergy the other Dassault Aviation entities and partners.
- The identification, the development and the qualification of the new technologies

Mission of ASP ("Structure Platform Analysis") :

- The choice of the criterions, the design and the justification of the structure
- Analysis of loads and vibrations
- Analysis of aeroelasticity and internal acoustic
- Management of tests of structure in external laboratories(CEAT, CEPr, etc)

INTRODUCTION

The plane of my project is related to the Falcon 7X.

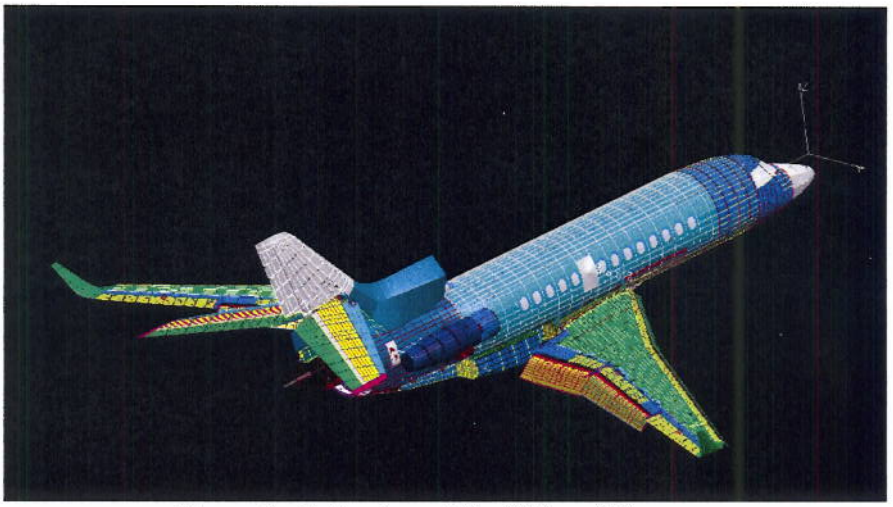


Figure 5: Catia view of the Falcon 7X

This project is related to the field of methodology and more precisely the equivalence between time simulations and flutter computations.

Such comparison between non-linear flutter in the frequency domain and time-marching solutions have already been studied for airfoils. For instance Ballhaus and Goorjian (1978) [2] calculated the aeroelastic response of a NACA 64A006 with a single-degree-of-freedom control surface at Mach number of 0.86. The developments were motivated by the wish to take into account the nonlinear effects of aerodynamic forces. The flutter of the same airfoil but with two-degrees-of-freedom was analysed by Yang (1979) with aerodynamic forces obtained by three different methods. After the flutter boundary was obtained, the response was confirmed near the flutter boundary by simultaneous time integration of the governing structural and aerodynamic equations. [11]

In our case, the nonlinearities come from a failure in the hydraulic circuit of one of the servo-actuators of an elevator. This study is related to the process of certification of the Falcon 7X.

Usually, the certification was made with flutter computations. The feedback was provided by another department of Dassault-Aviation. They computed the transfer functions and these tables of figures were taken into account when flutter boundaries were computed. The flutter boundary was not confirmed by time-marching solution.

The first purpose of this project is to be able to include a model of a feedback in the tool of Dassault Aviation for aeroelasticity (Elfini). The option chosen is a call to Matlab. This permits indeed, to change the model if necessary. Elfini needs to be able to call a Matlab program to include this feedback in the Time Response program. The Time Response program needs to be improved too to be able to use the result of the Matlab program.

The second purpose is to develop the programs needed to be able to do the comparison between time and flutter simulations. Time Response and Flutter were already developed in Elfini. The treatment of aerodynamic forces (Laplace Domain Reduction) for time simulations was developed, but had not really been tested. The feedback was coded for the flutter but not for the Time Response.

As a result, a first study has been made without feedback. Some other conclusions have to be brought about the equivalences between the different ways to take the aerodynamic forces into account in the computations.

Another objective is to study the behaviour of the plane, the changes in the curves of flutter and in time simulations with a non-linear feedback. The model of the hydraulic failure of the servo-actuator needs to be coded in Matlab. The study is made at a Mach number of 0.8. With such a non-linear feedback, limit cycle oscillations (LCO) can be observed at some dynamic pressure.

This thesis will begin by a preamble, in which some basic concepts used in this study will be exposed.
Then the part A will show a broad outline of the way to deal with aeroelasticity. Its structure will follow the definition of Collar, who sees aeroelasticity as the "mutual interaction" between elastic and inertial forces (the structure) and aerodynamic forces. Most of the concepts exposed in this part are part of the methodology used to compute aeroelasticity. Only the part of the model of the servo-actuator contains already some results obtained during the project.
The results of my simulations will be exposed in the part B, gathering time and flutter simulations. As some results are nor those expected, some reasons which may explain the gaps are put forward.

PREAMBLE

The purpose of this preamble is to offer diverse information about models which are used in the following study. Some hypotheses are made for example in the atmosphere model. The Runge-Kutta algorithm has been used in the model of the servo-actuator.

I - Atmosphere model

1.1 – Standard atmosphere

Each point M of the atmosphere can be defined by three parameters [3]:

- pressure $p(M,t)$ expressed in Pascal
- density $\rho(M,t)$ expressed in kilogram per meter cube
- temperature $T(M,t)$ expressed in Kelvin degree

The characteristics of the standard atmosphere are close to the characteristics of the real atmosphere. The hypothesis are the following ones:

1. The atmosphere remains unchanged with time. The characteristics do not depend on time.
2. The gas is perfect: $\frac{p}{\rho} = RT$, where $R = 287 Joule/(kg.K)$
3. The atmosphere has a weight and follows the Laplace law : $dp = -\rho g dh$
4. The temperature is linear with respect to the altitude (Toussaint's law - 1920) : $T = T_0 + T_h h$

T_h is constant in a certain range of altitudes :

Altitude h	Temperature gradient T_h
$0 \le h \le 11km$	$-6.5.10^{-3} K/m$
$11km < h \le 20km$	$0K/m$
$20km < h \le 32km$	$1.10^{-3} K/m$
$32km < h \le 47km$	$2.8.10^{-3} K/m$

Table 1: Temperature model

There are three law and four parameters (p, ρ, T, h). We need to know only one parameter, h for instance, to derive the other ones.
At the ground, the three parameters (p, ρ, T) are $p_0 = 101325Pa$, $T_0 = 288.16K$, $\rho_0 = 1.255kg/m^3$.

When the altitude increases, between $0km$ and $11km$:

$$T = T_0 + T_h h = 288 - 6.5.10^{-3} h$$

$$p = p_0\left(1 + \frac{T_h}{T_0} h\right)^{\frac{-g}{RT_h}} = 101325\left(1 - 22.6.10^{-6} h\right)^{5.26}$$

$$\rho = \frac{p}{RT} = \rho_0\left(1 + \frac{T_h}{T_0} h\right)^{\frac{-g}{RT_h}-1} = 1.225\left(1 - 22.6.10^{-6} h\right)^{4.26}$$

Note that the minus sign of T_h implies that under $11km$, the pressure, the density and the temperature decrease when the altitude h increases.

Up to $11km$, these laws can also be determined; they are more complicated.
With the following set of constants,

$$g = 9.80665 m.s^{-2}$$
$$R = 287.05 J.K^{-1}.kg^{-1}$$
$$\gamma = 1.4$$
$$P_0 = 101325 Pa$$
$$T_0 = 288.15 K$$
$$\rho_0 = \frac{P_0}{R.T_0}$$
$$a_0 = \sqrt{\gamma.R.T_0}$$

they are:

$0 \leq h \leq 11\ km$	$T(h) = T_0 - 6.5.10^{-3}.h$	$P(h) = P_0\left(1 - \frac{6.5.10^{-3}.h}{T_0}\right)^{\frac{g}{6.5.10^{-3}.R}}$
$11km < h \leq 20km$	$T = T(11000) = cste$	$P = P(11000).e^{\frac{-g(h-11000)}{RT(11000)}}$
$20km < h \leq 32km$	$T = T(20000) + 10^{-3}.(h - 20000)$	$P = P(20000)\left(1 + \frac{10^{-3}}{T(20000)}(h - 20000)\right)^{\frac{-g}{R.10^{-3}}}$

Table 2: Temperature and pressure laws

The speed of sound is $a = \sqrt{\gamma.R.T}$ and the density $\rho = \frac{P}{R.T}$.

In figure 6, the temperature is plotted in red for different altitudes. [4]

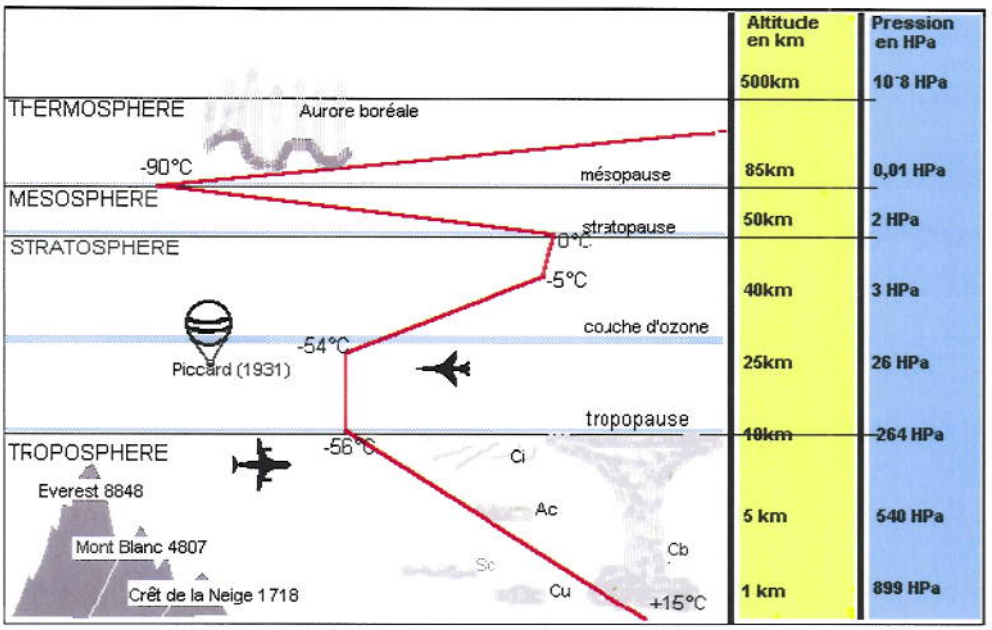


Figure 6: Atmosphere model

1.2 – Flight point

The true speed (with respect to air) is V. Then the Mach number is $M = \frac{V}{a}$.

Taking the variations of density in the air, the equivalent speed is $E_V = V.\sqrt{\frac{\rho}{\rho_0}}$ and the dynamic pressure $P_{dyn} = \frac{1}{2}.\rho_0.E_V^2$.

The total pressure is $P_i = P.\left(1 + \frac{\gamma - 1}{2}.M^2\right)^{\frac{\gamma}{\gamma-1}}$.

The conventional speed is V_C is the equivalent speed corrected from the effects of compressibility. It is also the speed displayed on the instrument panel in the plane.

Note that for $11km < h \leq 20km$, as the temperature is constant, the speed of the sound a is constant. At constant Mach number, the speed is then constant. For this reason, it is better to choose the dynamic pressure than the speed as parameter for the studies.

The speed introduces another degree of freedom. A flight point is then determined by two independent parameters between the altitude, the speed V, the equivalent speed E_V, the conventional speed V_C, the Mach number and the dynamic pressure.

All the studies in this project are done at a Mach number of 0.8. Alternatively the altitude or the dynamic pressure are the variable parameters. (Time simulations are done at several altitudes; the dynamic pressure is the parameter in flutter computations).

(See appendix A for the Chevallier graph)

II – Runge-Kutta algorithm

This algorithm is used to solve ordinary differential equations. Such problems can always be reduced to a set of first-order differential equations by a change of variables.

The problem which has to be solved is then:

$\frac{dy_i(x)}{dx} = f_i(x, y_1, ..., y_N)$ with boundary conditions $y_i(0) = y_{i0}$, $i = 1, ..., N$.

Runge-Kutta methods propagate a solution over an interval by combining the information from several Euler-style steps (each involving one evaluation of the right-hand f's), and using the information obtained to match a Taylor series expansion up to some higher order.

2.1 - Geometrical interpretation of a two-stage RK scheme [5]

In one dimension, the problem to be solved is $\frac{dy(x)}{dx} = f(x)$, $y(0) = y_0$.

The Euler formula is $y_{n+1} = y_n + h.f(x_n, y_n)$, which advanced a solution from x_n to $x_{n+1} = x_n + h$.

A typical Runge-Kutta scheme is:

$$k_1 = f(x_n, y_n)$$
$$k_2 = f(x_n + \alpha_2 \Delta x, y_n + \Delta x \alpha_2 k_1)$$
$$y_{n+1} = y_n + \Delta x(w_1 k_1 + w_2 k_2)$$

The coefficients α_2, w_1 and w_2 are obtained by matching the terms of y_{n+1} to Taylor series of k_2 (in two variables y_n and k_1) to the maximum accuracy:

$$k_2 = f|_n + \alpha_2 \Delta x \left.\frac{\partial f}{\partial x}\right|_n + \alpha_2 \Delta x k_1 \left.\frac{\partial f}{\partial y}\right|_n$$

$$y_{n+1} = y_n + \Delta x(w_1 + w_2) f|_n + \Delta x^2 w_2 \alpha_2 \left(\left.\frac{\partial f}{\partial x}\right|_n + f|_n \left.\frac{\partial f}{\partial y}\right|_n \right) = y_n + \Delta x f|_n + \frac{\Delta x^2}{2} \left(\left.\frac{\partial f}{\partial x}\right|_n + f|_n \left.\frac{\partial f}{\partial y}\right|_n \right)$$

$$\Rightarrow \quad \begin{cases} w_1 + w_2 = 1 \\ w_2 \alpha_2 = \frac{1}{2} \end{cases} \quad \Rightarrow \quad \begin{cases} w_1 = w_2 = \frac{1}{2} \\ \alpha_2 = 1 \end{cases}$$

The final Runge-Kutta scheme is then:

$$\boxed{\begin{cases} y_{n+1} = y_n + \frac{\Delta x}{2}(k_1 + k_2) \\ k_1 = f(x_n, y_n) \\ k_2 = f(x_n + \Delta x, y_n + \Delta x k_1) \end{cases}}$$

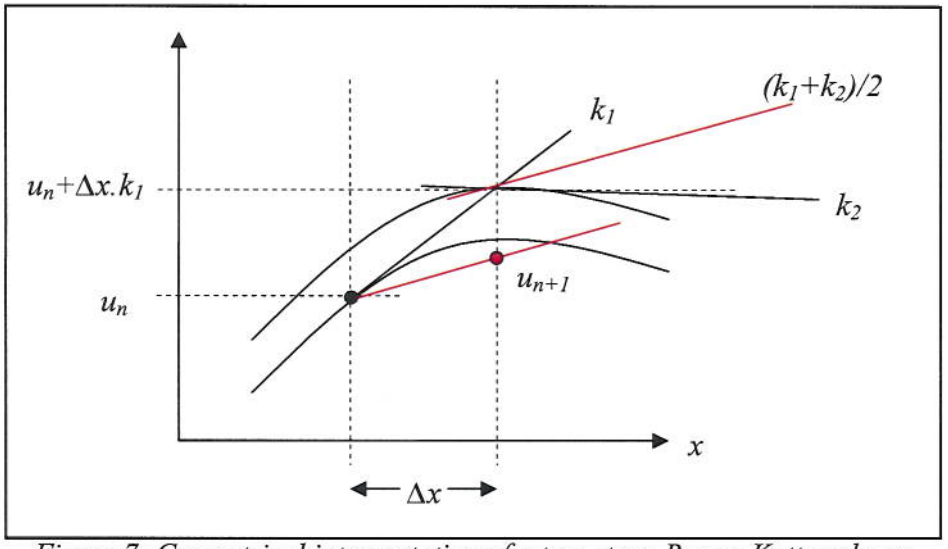


Figure 7: Geometrical interpretation of a two-stage Runge-Kutta scheme

Geometrically, the first step is to draw the tangent to the curve at u_n and determine the point belonging to this tangent at $t + \Delta t$. Then draw the curve $u_n + \Delta t.k_1$ and find the tangent of this curve at $t + \Delta t$. Take the average between the two tangents and find u_{n+1} by plotting this straight line from u_n.

2.2 - Geometrical interpretation of a fourth-order RK scheme

The coefficients for the classical fourth-order Runge-Kutta scheme are:[6]

$$k_1 = f(x_n, y_n)$$

$$k_2 = f\left(x_n + \frac{\Delta x}{2}, y_n + \frac{k_1}{2}\right)$$

$$k_3 = f\left(x_n + \frac{\Delta x}{2}, y_n + \frac{k_2}{2}\right)$$

$$k_4 = f(x_n + \Delta x, y_n + k_3)$$

$$y_{n+1} = y_n + \Delta x\left(\frac{k_1}{6} + \frac{k_2}{3} + \frac{k_2}{3} + \frac{k_2}{3} + \vartheta(\Delta x^5)\right)$$

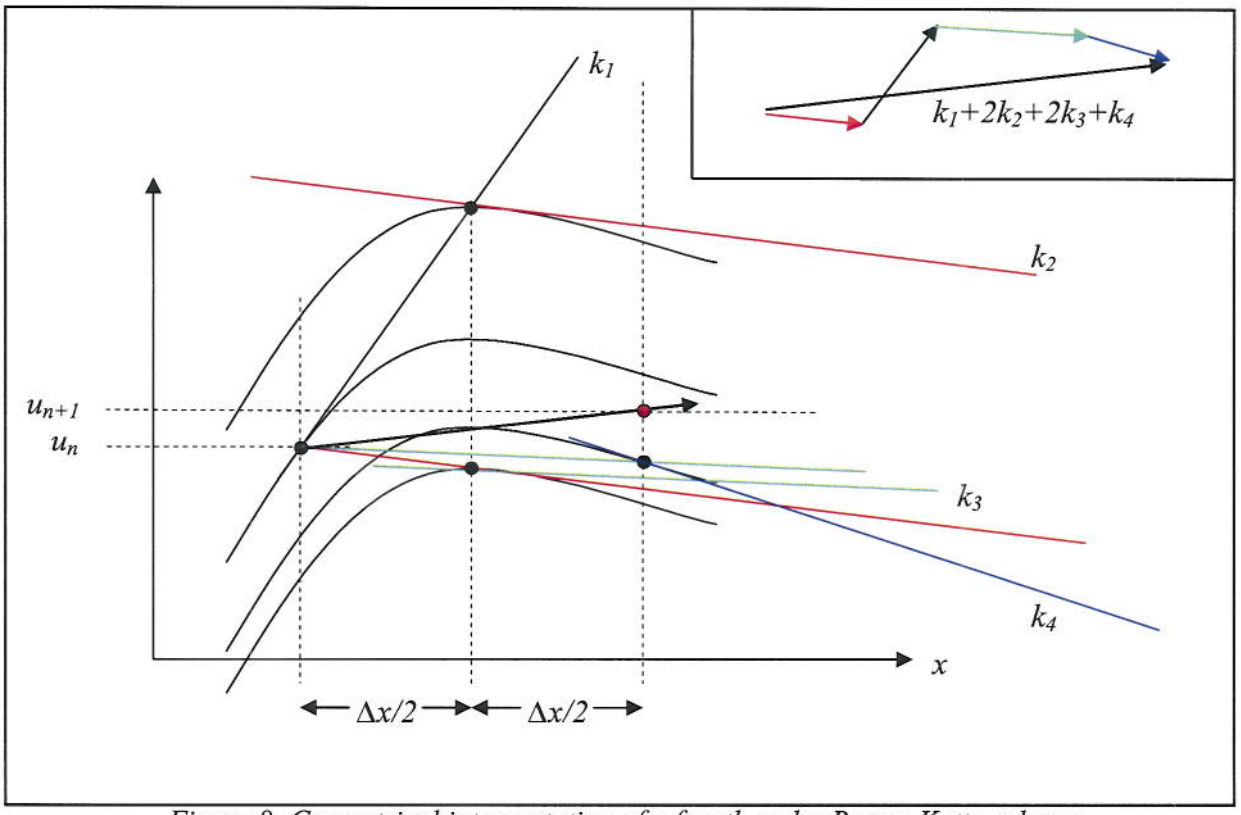


Figure 8: Geometrical interpretation of a fourth-order Runge-Kutta scheme

The method is the same as for a two-stage Runge-Kutta scheme. The more the step Δx can be reduced, the more accurate the scheme is.

The Runge-Kutta algorithm is used in the model of the servo-actuator. This scheme can be improved but as the accuracy of the results is sufficient *(see 2.2.3 – Validation of the model of the servo-actuator),* the improvements have not been developed in the project.

PART A

STRUCTURE – AERODYNAMICS – AEROELASTICITY

Aeroelasticity was defined by Collar in 1947 as *"the study of the mutual interaction that takes place within the triangle of the inertial, elastic, and aerodynamic forces acting on structural members exposed to an airstream, and the influence of this study on design." [10]*

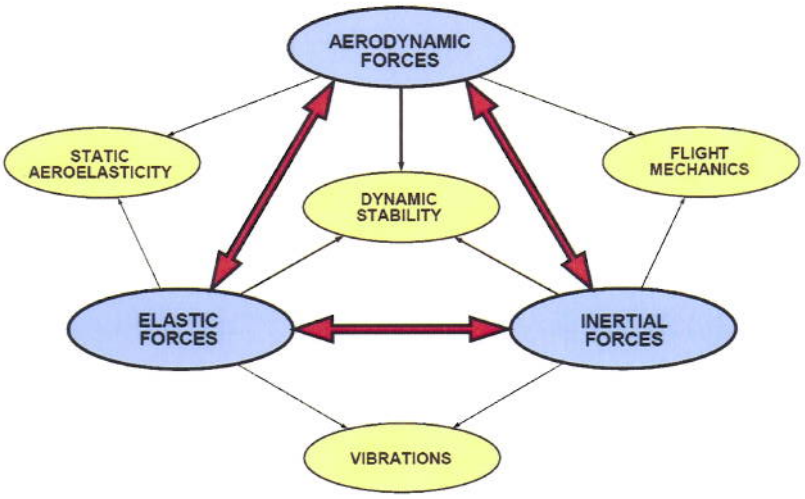


Figure 9: Collar Triangle

AI – STRUCTURE

A - 1.1 – Structure

A - 1.1.1 – Finite Element model

Consider a wing structure that has been modelled by the finite-element method. Let x be the vector of small displacement in the finite element model of aircraft in flight (more precisely at all element grid node). [7]

The general equation of motion of a N degrees of freedom is

$$\boxed{M\ddot{x} + C\dot{x} + Kx = F_{aero} + F_{other}} \qquad (1)$$

where M is the mass matrix,
C is the damping matrix,
K is the stiffness matrix,
They are $N \times N$ matrices.

A - 1.1.2 – How to compute the modes at ground?

In the frequency domain, the equation (1) becomes:

$$\left(-\omega^2 M + i\omega C + K\right)\left|x\right|e^{i\omega t} = \left(F_{aero} + F_{other}\right)e^{i\omega t}$$

The modes at ground can be found by considering the associated conservative system and by solving

$$\left(-\omega^2 M + K\right)\left|x\right| = 0$$

In practice, $\det\left(-\omega^2 M + K\right) = 0$ is solved.
ω is the natural frequency of the mode.
The eigenvectors form the modal basis and can be grouped in the matrix of modal shapes ϕ_S. ϕ_S is a $N \times N_m$ matrix, where N_m is the number of modes. There are rigid body modes, elastic and modes (from the resolution of the previous equation) and the rigid control surface modes. The rigid control surface modes are not solution of the equation $\det\left(-\omega^2 M + K\right) = 0$. They are displacements artificially added to the vector after the computation of the determinant.

The displacements in the modal basis are called the general displacements. Following Elfini nomenclature, we will refer to this vector as **GDISP** (X). Therefore the physical displacements and the modal displacements are linked by:

$$\boxed{x = \phi_S X}$$

Note that a multiplicative constant has to be chosen to define the modes (the matrix ϕ_S) : λ for instance. This constant is therefore in the modal mass, damping, stiffness matrix as λ^2 and in the GDISP as $1/\lambda$.

Multiply the equation (1) by ${\phi_S}^T$ on the left and the general equation of motion in the modal basis becomes:

$$\phi_S^T M \phi_S \ddot{X} + \phi_S^T C \phi_S \dot{X} + \phi_S^T K \phi_S X = \phi_S^T F$$

Set $M_S = \phi_S^T M \phi_S$, $C_S = \phi_S^T C \phi_S$, $K_S = \phi_S^T K \phi_S$, the modal mass matrix, the modal damping matrix and the modal stiffness matrix. They are $N_m \times N_m$ matrices.

$Q = \phi_S^T F$ is the forces in the modal basis.

$$\boxed{M_S \ddot{X} + C_S \dot{X} + K_S X = Q}$$

For a non-damped system: the mass matrix M_S, and the stiffness matrix K_S are diagonal in the natural modes base.

For lightly damped systems, Basil hypothesis is usually done: M_S, C_S and K_S are assumed to be diagonal (the other terms can be neglected in front of the diagonal ones) in the modal basis.

Experimentally, the modes can be found by the phase resonance method performed during ground tests. The input, a fixed-frequency sine, is delivered by shakers placed on the wing. Accelerometers measure the induced displacements of the structure.

In the case of Basil hypothesis, the modes are found when $-M_S\omega^2 + K_S = 0$ which is equivalent to $iC_S\omega q = Q$. The displacements of the structure are in phase quadrature with the excitation forces for this frequency. This frequency is a good estimate of the natural frequency of the mode.

In the case of an hydraulic failure of the servo-actuator, the structure is not lightly damped anymore. As a consequence, Basil hypothesis cannot be applied any more.

A – 1.1.3 – How to get the physical coordinates from the modal coordinates?

The matrix of modal shapes ϕ_S can be reduced by considering only some degrees of freedom. Then ϕ_S is a $N_S \times N_m$ matrix, where N_S is the number of selected modes. This reduced matrix is called the "MONitoring SET matrix" (**MONSET**) in Elfini. It allows to monitore some physical quantities (such as the displacements, speed, stresses…) at some selected points.

A – 1.1.4 – Forces

In equation (1), there are two sorts of forces applied on the structure F_{aero} and F_{other}. Hence, in the modal basis, there is $Q_{aero} = \phi_S^T F_{aero}$ and $Q_{other} = \phi_S^T F_{other}$. This matrix ϕ_S has undergone the operation of smoothing *(See 2.1 – Singularity method and aerodynamic shapes basis).*

F_{other} can be thrust forces, landing gear…

The aerodynamic forces F_{aero} applied on the plane can be expressed with the pressure coefficients.

$$F_{aero} = P_{dyn}.C_P(M,k)$$

With a Taylor series development:

$$F_{aero} = P_{dyn}.\left(C_P(M,k) + \frac{\partial C_P(M,k)}{\partial X} X \right)$$

The first term $P_{dyn}.\phi_S^T.C_P(M,k)$ arises from the rigid body motion, from the motion of control surfaces or from local turbulence. It is a $N_m \times 1$ matrix. It can be written $P_{dyn}.\phi_S^T.C_P(M,k) = GLOAD \times f(\omega)$ where $f(\omega)$ is the amplitude of the force. Some fake modes are artificially added to the basis for the storage of these additional forces.

The second term $P_{dyn}.\phi_S^T.\frac{\partial C_P(M,k)}{\partial X} X$ is the load caused by aeroelastic coupling forces (depending on structural elastic displacement X and its derivatives). It is a $N_m \times N_m$ matrix expressed on all the fixed modes.

These two terms are called **GLOAD** (Generalised Loads) in Elfini but they do not have exactly the same meaning. $P_{dyn}.\phi_S^T.C_P(M,k)$ is treated as an input whereas $P_{dyn}.\phi_S^T.\frac{\partial C_P(M,k)}{\partial X} X$ is treated as a stiffness matrix. However, they are stored in the same

matrix. In the modal basis, composed of rigid modes (R), elastic modes (E) and the control surface modes (CS), the "GLOAD" matrix can also be this matrix:

$$GLOAD = P_{dyn}.\phi_S^T.\left(\begin{array}{ccc|c} \text{R} & \text{E} & \text{CS} & \\ & \dfrac{\partial C_P(M,k)}{\partial X} & X & C_P(M,k) \\ & & & \\ & & & \end{array}\right)$$

inputs

The inputs of the system are composed of the modes added to store the $C_P(M,k)$ and the control surface modes.

In a temporal domain, the product of the derivative of the pressure coefficients and the displacements become a convolution product. The linear aeroelastic coupling forces in the temporal domain are then:

$$P_{dyn}\int_{-\infty}^{t}\frac{\partial C_P}{\partial X}(t-\tau)X(\tau)d\tau$$

In this report, the Elfini nomenclature will be used to appoint the GDISP, GLOAD, MONSET.

A - 1.1.5 – How to add feedback?

Flutter:

The feedback in a flutter computation is defined by:
- a MONSET to define the degree of freedom where the feedback takes its input
- a GLOAD to define the degree of freedom of the output of the feedback
- a transfer function

In our case, the transfer function is $H(\omega)=\dfrac{F(\omega)}{x(\omega)}$.

In the modal basis,

$$Q = \phi_S^T.F = \phi_S^T H(\omega)x(\omega) = \phi_S^T H(\omega)\phi_S X(\omega) = GLOAD \times f(\omega).$$

where

$$GLOAD = \phi_S^T$$

$$f(\omega) = H(\omega)\phi_S X(\omega) = H(\omega).MONSET.X(\omega)$$

In our case the MONSET and the $GLOAD^T$ are the same because the degree of freedom of the input of the feedback (the displacement) and the degree of freedom of its output (the force) are the same.

Time Response:

At each time step, a force induced by the displacement is put in the system.

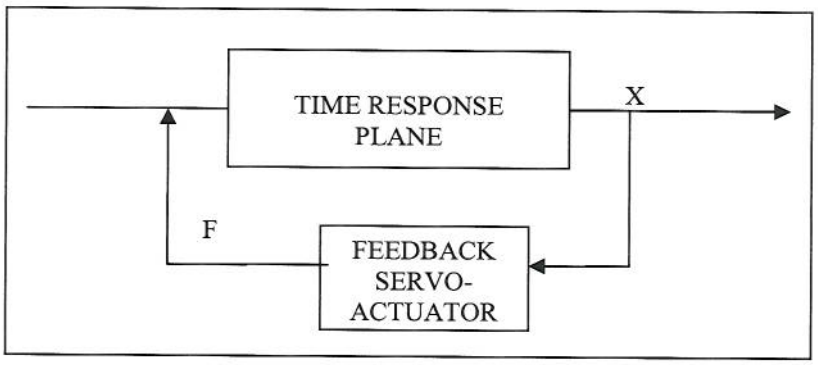


Figure 10: Feedback in Time Response

The displacement X which is expressed in the modal basis is combined to the same MONSET as in the flutter paragraph. It must indeed be a physical displacement to go to the Matlab Program coding the answer of the broken servo-actuator.

The result of the program Matlab (a double in this case) is multiplied by a GLOAD (to go to the modal basis) which is formed with the same MONSET as before. In the program coding the Time Response (TimeResponse.c) this vector is added to the equation which is solved at each time step *(See section 3.2 – Time Response)*.

A - 1.1.6 – How to add stiffness to the model?

In Elfini, it is possible to add stiffness to the model with the command "CRESTIFF". This will be often used to validate the results. For instance, to validate the call to Matlab from Elfini, the results must be the same by adding stiffness to the model or by adding a force $-K.x$ through Matlab.

The stiffness matrix we want is $K' = K + \Delta K$.
In the modal basis, the stiffness is written: $K'_S = \phi_S^T K \phi_S + \phi_S^T \Delta K \phi_S = K_S + \phi_S^T \Delta K \phi_S$.
The matrix ΔK is almost empty. It has only a Δk at the degree of freedom where stiffness has to be add.

$$\Delta K = \begin{pmatrix} & 0 & \\ 0 & & \\ & & \\ 0 & \Delta k & 0 \\ & 0 & \end{pmatrix}$$

This operation only need a MONSET, which represents the degree of freedom where stiffness is added. In our case, it is the MONSET DAXE *(See II – Model used for simulations with feedback and Appendix C for an example of the call of this program).*

A - 1.2 – Servo-actuator

On each elevator on the Falcon 7X, there are two servo-actuators.

The most critical failure case, concerning failure cases of the actuators of the elevators is the case where one servo-actuator is cut and the other one has an hydraulic failure. This case has a probability superior of 10^{-9} to happen. (when the plane takes off with a broken rod, what can not be detected at ground). This is the minimal probability of the cases Dassault-Aviation has to study to certify the plane.

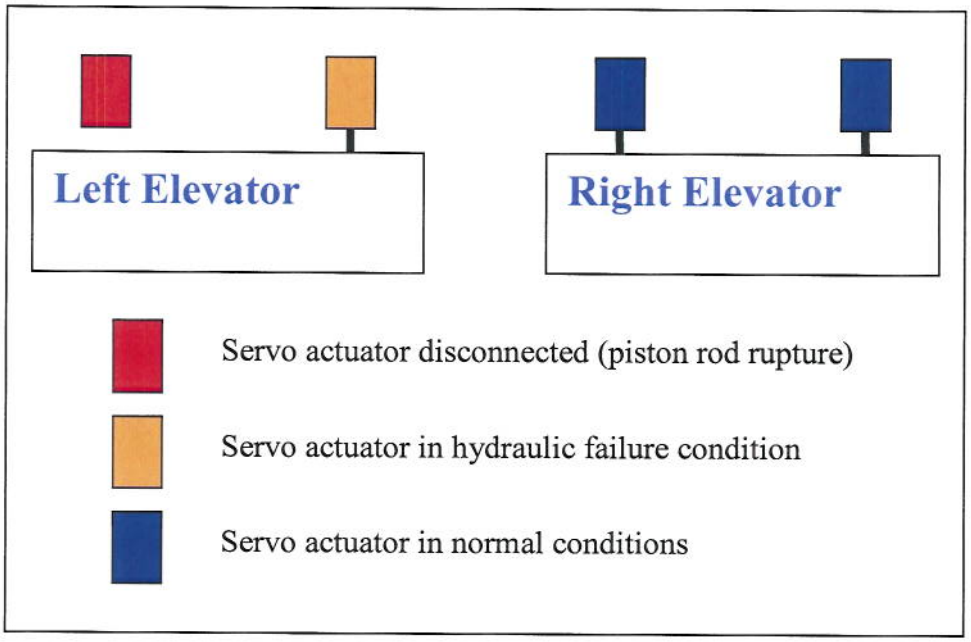


Figure 11: Most critical hydraulic failure case combined with another single failure for elevator

The method usually used to prove the stability is usually a flutter computation. The model of the servo-actuator has to be coupled with the structure through a feedback. In flutter computations, the feedback is added through a transfer function.

By simulating the system in time, we should find the same stability limit as with flutter computations.

Moreover, as the model of the servo-actuator is not linear, it must be possible to observe LCO (limit cycle oscillations) at some dynamic pressures at a given Mach number. In this way, temporal simulations could allow to extend the flight domain.

A - 1.2.1 - Model of the actuator [8]

In the case of a failure of the servo-actuator, a model of a damped-body is used. The fluid goes through a nozzle, causing non-linearities..

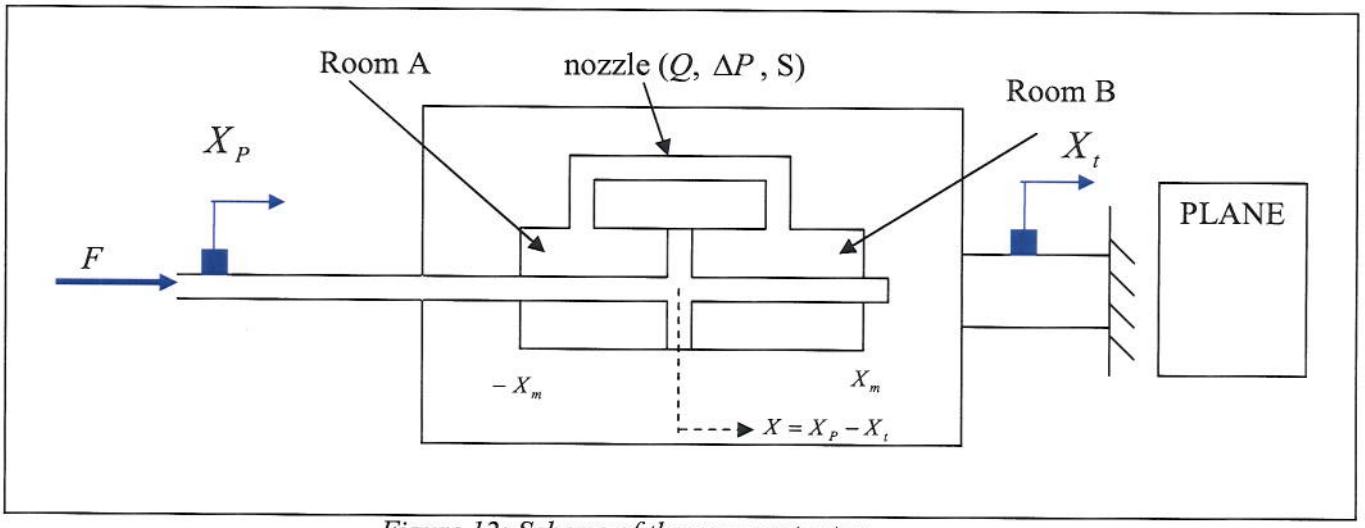


Figure 12: Scheme of the servo actuator

A - 1.2.1.1 - Nomenclature

ΔP is the difference of pressure between the rooms
S is the useful section of the piston
$2X_m$ is the highest possible displacement of the piston
$V \approx S \times X_m$ is the volume of a room (only valid for little displacements around the equilibrium point)
M_P is the mass of the piston
k is the hydraulic conductance of the nozzle

$X = X_P - X_t$ is the displacement of the piston with respect to the body ($-X_m \leq X \leq X_m$)

Assuming that the stiffness of the link plane-servo-actuator is infinite, $X_t = 0$.
As a result, $X = X_P$.

A - 1.2.1.2 - Equations

Let us apply the fundamental principle of dynamics to the piston:

$$\boxed{M_P \frac{d^2 X_P}{dt^2} = F - S.\Delta P}$$

The rate of flow through the nozzle depends on the velocity of the rod and on the difference of pressures between the two rooms:

$$\boxed{Q = S\frac{dX}{dt} - \frac{V}{2B}\frac{d\Delta P}{dt}}$$

It is only valid for little displacements around the equilibrium position $X << X_m$. It is also assumed that there is no leak in the damped body. The first term of this rate of flow expresses the effects of compressibility. The second term is the rate of flow caused by the displacements of the piston.

Let us make the hypothesis of a turbulent flow. The rate of flow is also:

$$\boxed{Q = k\sqrt{|\Delta P|}.\,\mathrm{sgn}(\Delta P)}$$

A - 1.2.1.3 – Resolution

These three equations are not coupled.
Combining the two equations of rate of flow, we obtain:

$$\frac{d\Delta P}{dt} = \frac{2B}{V}\left(S\frac{dX}{dt} - k\sqrt{|\Delta P|}.\,\mathrm{sgn}(\Delta P)\right)$$

At each time step, from a displacement, this equation of the form $\frac{dy(x)}{dx} = f(x,y)$ is solved with the Runge-Kutta algorithm, which has been introduced in section II.2.

Then the force is obtained with the first equation

$$F = M_P \frac{d^2 X_P}{dt^2} + S.\Delta P$$

At t, we have then the force $F(t)$ corresponding to the displacement $X(t)$.

See Appendix D for the program.

A - 1.2.3 - Validation of the model of the actuator

For its validation, the program is tested with an input of the form

$$X = X_0 \sin(2\pi ft) \qquad \text{with } X_0 = 1mm .$$

For some values of the frequency, the values of the modulus and phases are provided by another department of Dassault-Aviation (DED). They have to match with the results of the Matlab program. This study allows choosing the best method to compute the transfer function. Indeed, they could be computed with a temporal method, a method using Fourier transforms or Fourier series. To compare the three methods, some parameters have to be fixed:

$t_{max} = 1s$, $N = 20000$.

A - 1.2.3.1 - Initial data

The equipments department at Dassault-Aviation provided some the results. They cannot be displayed here for confidential reasons. They are obtained with the same equations and some frequency treatment and have been checked with experiments. This department also provided the values of the parameters of the equations of the model in order to run my program: the surface S, the mass of the piston M_P, the displacement X_P, the hydraulic conductance of the nozzle k, and the constant B expressed in bar.

A - 1.2.3.2 - Modulus :Temporal approach

The result of the Matlab program with the parameters provided by DED has the shape of figure 13 for f=5Hz:

A first way to get the modulus of F is to take the maximum of F.

(See Appendix D for the program)

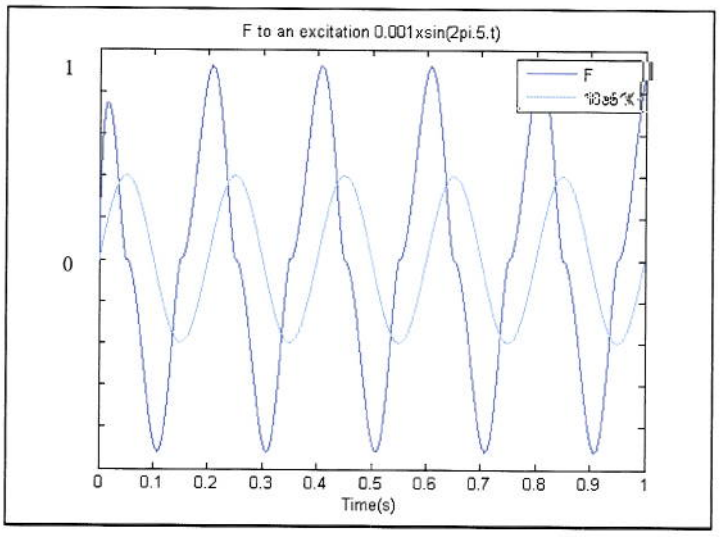


Figure 13: Normalised force created by a displacement in the servo-actuator

A - 1.2.3.3 - Modulus: Fourier series [9]

A function can be decomposed in a sum of cosine and sine at different frequencies, which are the harmonics of a signal. The coefficients A_P and B_P are computed separately and the function is reconstructed. Discrete series are derived from the theory of Fourier series.

Fourier Series :

$$f(t) = \frac{A_0}{2} + \sum_{p=0}^{\infty} \left[A_p \cos(2\pi pft) + B_p \sin(2\pi pft) \right]$$

with $A_p = \frac{2}{T} \int_0^T y(t) \cos(2\pi pft) dt, p = 0,1,2,...$

and $B_p = \frac{2}{T} \int_0^T y(t) \sin(2\pi pft) dt, p = 1,2,...$

Discrete Series :

$$f(t_n) = \frac{A_0}{2} + \sum_{p=0}^{N} \left[A_p \cos(2\pi pft) + B_p \sin(2\pi pft) \right]$$

with $t_n = n \times dt$

$$A_0 = \frac{1}{N} \sum_{n=1}^{N} y(t_n);$$

$$A_{N/2} = \frac{1}{N} \sum_{n=1}^{N} y(t_n) \cos(n\pi);$$

$$A_p = \frac{2}{T} \sum_{n=1}^{N} y(t_n) \cos(2\pi pft_n), p = 1,2,...N/2+1$$

$$B_0 = B_{N/2} = 0;$$

$$B_p = \frac{2}{T} \sum_{n=1}^{N} y(t_n) \sin(2\pi pft_n), p = 1,2,..N/2+1.$$

In the program N is the number of samples, p is the order of the serial development. It cannot be higher than *N/2+1*: there are N inputs, so N variables A_p and B_p can be computed.

The function f is reconstructed with these coefficients. This operation is repeated for each frequency.

This method is quite long but very accurate.

A - 1.2.3.4 - Modulus: Comparison

The results of the temporal approach and a method using Fourier series are gathered in table 4. They are the relative percentages of errors of my results with respect to the DED results.

f	5	10	15	20	25	30
DED	$\|F_1\|$	$\|F_2\|$	$\|F_3\|$	$\|F_4\|$	$\|F_5\|$	$\|F_6\|$
Temporal approach	14%	5%	2%	1%	0.6%	0.4%
Fourier series	0.7%	0.2%	0.5%	0.9%	0.8%	0.7%

Table 3: Comparison of modulus from different approaches of the model of the servo-actuator

The two methods are accurate and follow the results provided by DED.

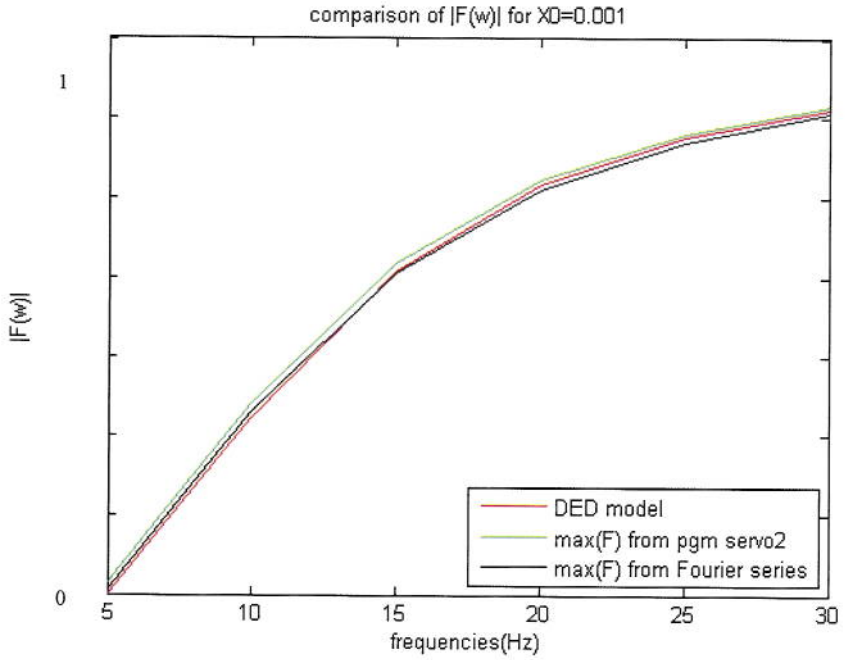
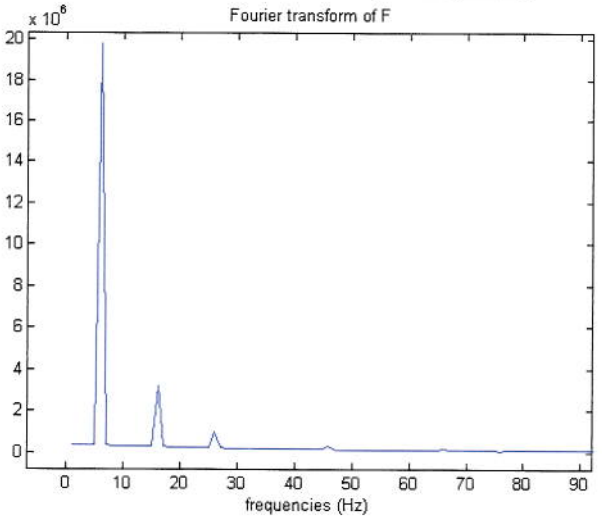


Figure 14: Normalised modulus of the force F(w)

Figure 15: Fourier transform of F(w)

As we can see in Figure 15, which is the Fourier transform of the modulus F, there is a main harmonic function in its decomposition, which corresponds to the frequency of the input.



The same result is obtained by plotting the coefficients of Fourier series.
Note that the main pike is at 5Hz, the frequency of the input, and the other ones are the odd coefficients of the sine (b3, b5).

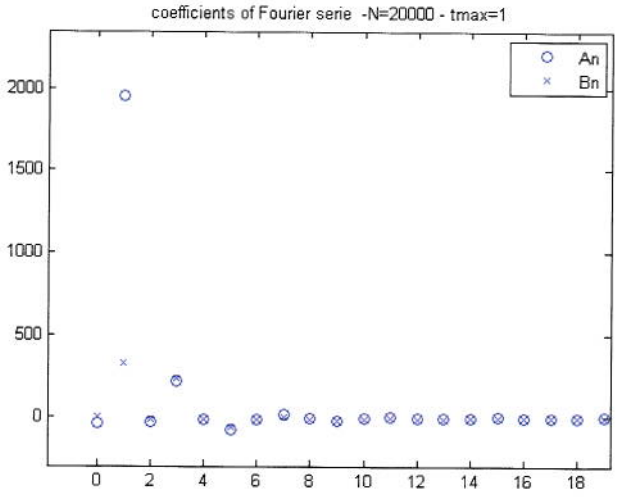


Figure 16: Coefficients of Fourier series of F(w)

Reconstructing the function only with this harmonic could faster the method using Fourier series. However, according to the curves of figure 17, this is not sufficient: the curve of the maximal harmonic (in blue) does not follow the three other ones.

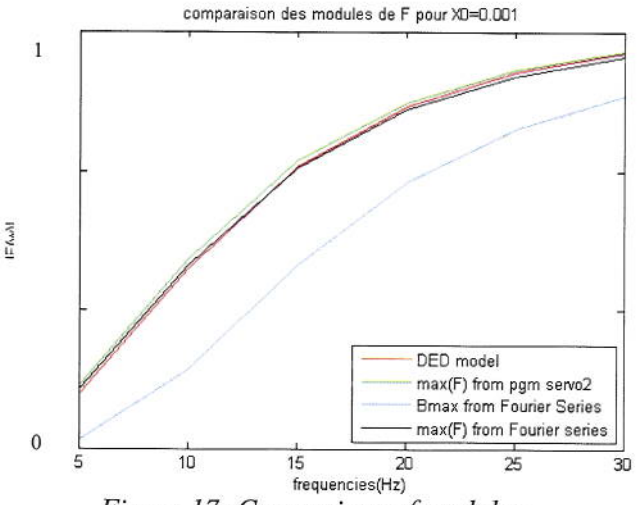


Figure 17: Comparison of modules

A - 1.2.3.5 - Phase

According to the previous graph showing the temporal behaviour of the force F and the relative displacement of the rod X, an experimental approach can be done to find the difference of phase between the force and the displacement. However, the precision remains poor.

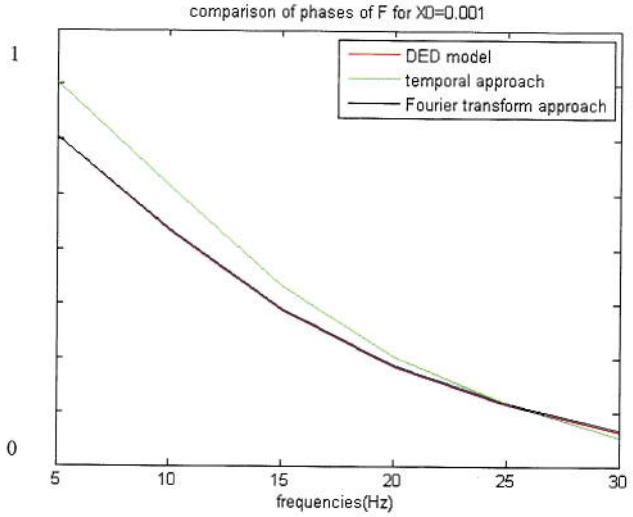


Figure 18: Normalised phases of F(w)

f	5	10	15	20	25	30
DED	$\lvert\varphi_1\rvert$	$\lvert\varphi_2\rvert$	$\lvert\varphi_3\rvert$	$\lvert\varphi_4\rvert$	$\lvert\varphi_5\rvert$	$\lvert\varphi_6\rvert$
Temporal approach	12%	12%	9%	5%	0.6%	4%
Fourier transform	0.1%	0.3%	0.5%	0.6%	0.9%	1%

Table 4: Comparison of phases from different approaches of the model of the servo-actuator

On the contrary, when the phase of the Fourier Transform is computed, the results are very close to the initial data. This is undoubtedly the more accurate and faster method for the phase. Even when zooming, the curves are very close.

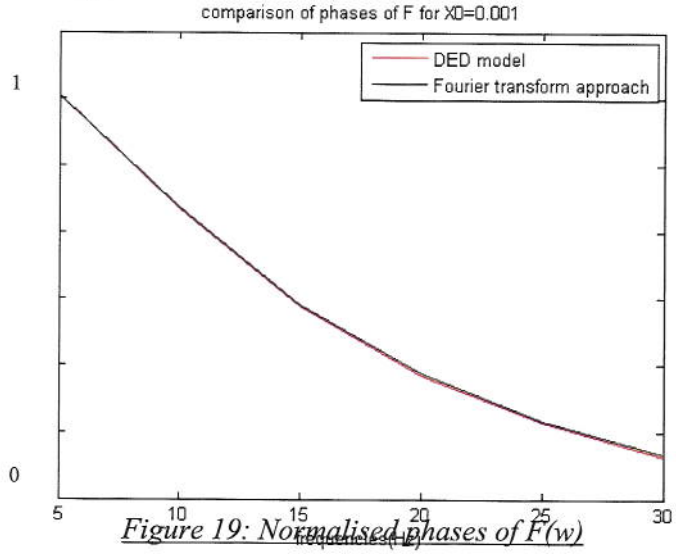


Figure 19: Normalised phases of F(w)

A - 1.2.3.6 - Conclusion

Whatever the method is for the modulus, the results are still accurate. A Fourier approach is better when the signal is not as smooth as our signal.
A Fourier Transform is also the best method to compute the phase of a transfer function

After this validation (at $X_0 = 1mm$), transfer functions can be computed for different amplitudes X_0. Indeed, a program which has to return the force induced by a displacement at each time step is needed for the time response and another one to compute transfer functions is needed for flutter computations.

A - 1.2.4 - Transfer functions

By exciting the model with a sine displacement $X(t,\omega) = X_0 \sin(\omega t)$, we can obtain the resultant force $F(t,\omega)$. Thus, in a steady harmonic motion, we can find a transfer function $H(j\omega) = \frac{F(j\omega)}{X(j\omega)}$.

This transfer function is complex and can be written as $H(j\omega) = |H(j\omega)| e^{j\varphi(\omega)} = |H(j\omega)| \cos(\varphi(\omega)) + j|H(j\omega)| \sin(\varphi(\omega)) = K_r(\omega) + jK_i(\omega)$.

In the previous equation, the modulus of the transfer function is $|H(j\omega)| = \frac{|F(j\omega)|}{|X(j\omega)|}$ and the difference of phases between the force and the displacement is $\varphi(j\omega) = \varphi_F(j\omega) - \varphi_X(j\omega)$.
Two constants appear: K_r has the dimension of a stiffness and K_i is a dissipation.

Transfer functions $H(X_0, \omega)$ are obtained. They will be used to be the feedback in flutter computations.

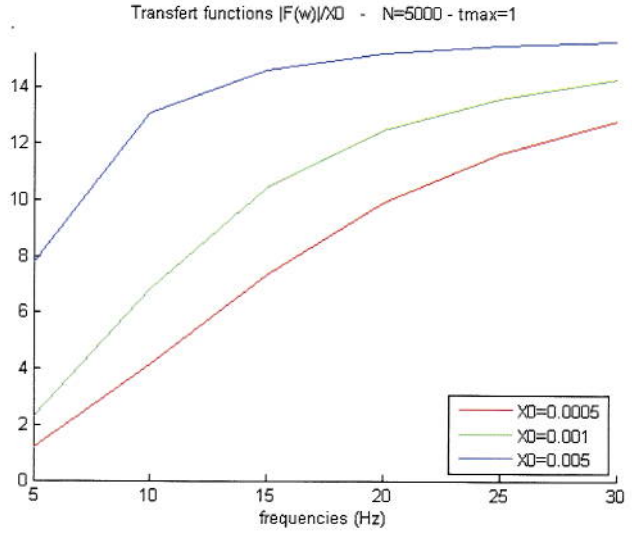


Figure 20: Modulus of Transfer functions for different amplitude of the input

A transfer function is put in serial with the stiffness of the rod of the finite element model:

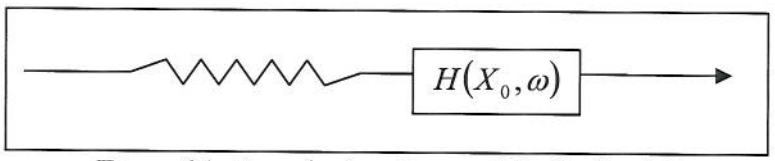


Figure 21: Transfer function used for feedback 1

The total transfer function is then: $\frac{H(X_0,\varpi).K}{H(X_0,\varpi)+K}$ as the rod is cut in our model. $H(X_0,\omega)$ has a real part and an imaginary part, denoted K_R and K_I.

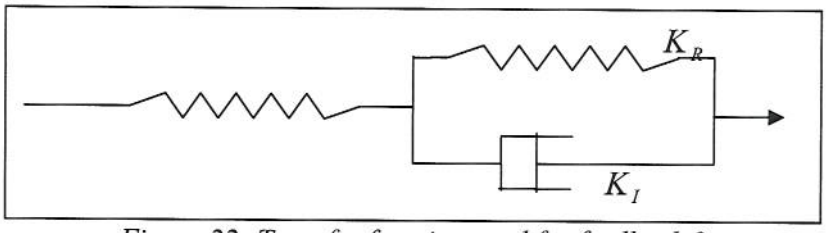


Figure 22: Transfer function used for feedback 2

$H(\omega)$, the transfer function computed in Matlab with the model of the servo-actuator, is computed in Elfini in the modal basis and becomes $G(\omega)$.

$G(\omega) = GLOAD \cdot g(\omega).MONSET$. *(See 1.1.5 - How to add feedback.)*

This MONSET is formed with the nodes of the rod where we want to apply the feedback. *(See II – Model used for simulations with feedback).*

The transfer functions can be expressed with normalised coordinates with respect to a normalised frequency

.

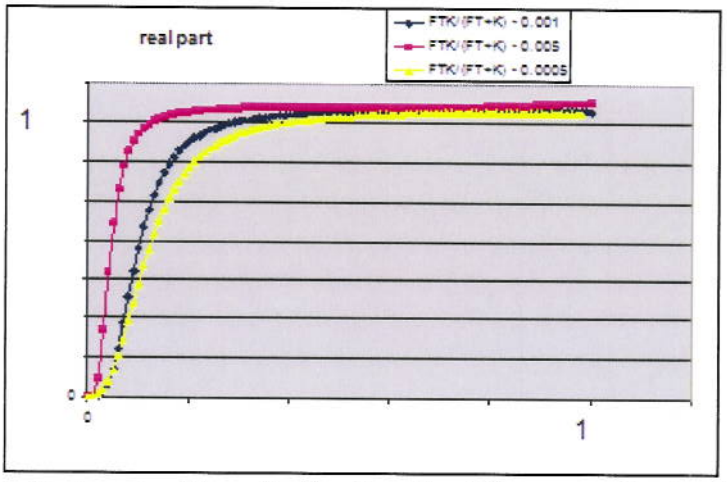


Figure 23: Real parts of Transfer functions for different amplitude of the input

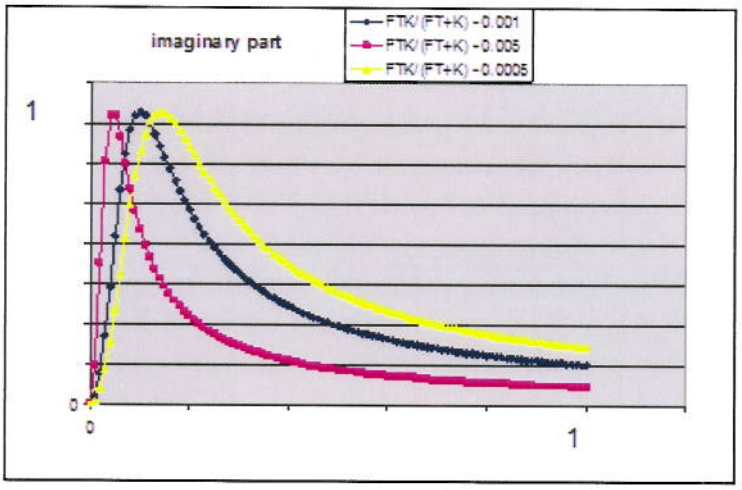


Figure 24: Imaginary parts of Transfer functions for different amplitude of the input

A - II – AERODYNAMICS

A - 2.1 – Singularity Method and aerodynamic shape basis

As it is more convenient to compute the aerodynamics loads independently of the structure in a first stage, the displacements can not be expressed in the modal basis. Otherwise they would depend on the mass distribution. [4]
Consequently, they are expressed in a basis of basic deformations of the aerodynamic surfaces denoted Γ and called aerodynamic shape functions. The shape functions are sequences of monomials. For example, on the horizontal tail plane:

$$\Gamma_{ij}(x,y) = (x - x_0)^i . (y - y_0)^j$$

where x and y are coordinates on the surface.

In the rudder, the coordinates are z and x.

The matrix Γ is formed with the Γ_{ij} :

$$\Gamma = \begin{pmatrix} \cdots & \begin{pmatrix} \Gamma_{ij} \end{pmatrix} & \cdots & \cdots \end{pmatrix}$$

number of aerodynamic degree of freedom

number of monomials

The physical displacements $\tilde{x}$ are then $\Gamma.\gamma$ where γ are the aerodynamic shape coordinates: the coordinates on the monomial basis.

The singularity method provides a solution for potential flow at subsonic speeds. The flow is assumed to be inviscid, potential, compressible, unsteady and irrotational. The pressure field $\frac{\partial C_P}{\partial \alpha}$ can be derived from this aerodynamic analysis (α is the local angle of attack of each aerodynamic element). On each local element of the aerodynamic mesh, $\frac{\partial C_P}{\partial \alpha_i}$ is computed, where α_i is the local angle of incidence. They are given in the Aerodynamic Influence Coefficient matrix (AIC).

In Elfini, they are stored in a matrix

$$\left[\frac{\partial C_P}{\partial \alpha}.\Gamma\right]$$

number of aerodynamic degree of freedom

number of monomials

The main advantage of this method is that it provides a simple mesh which permits fast computations.

As the aerodynamic mesh used in aeroelasticity and the finite element mesh are based on completely different theories, they do not match to each other.

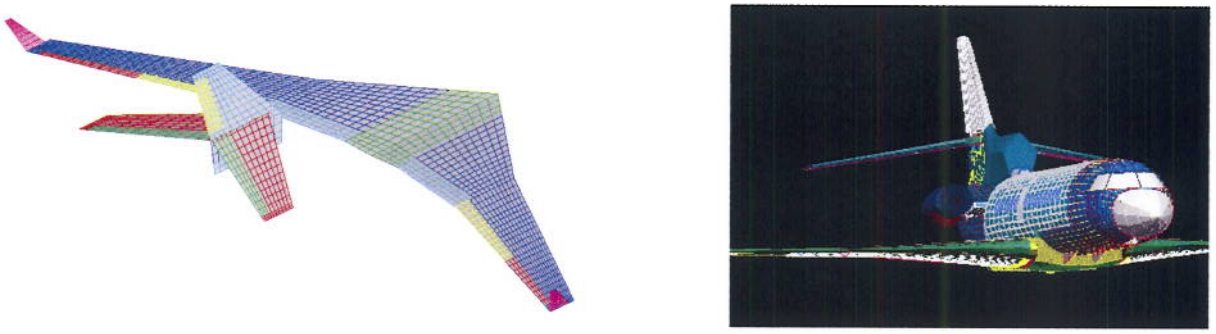


Figure 25: Aerodynamic mesh and structural mesh

It has to be possible to transfer the displacements of the structure on the aerodynamic field. An operation of smoothing has to minimise the difference between the displacements of the structure x in the finite element mesh and the displacements x in the aerodynamic mesh. By a least square minimizing method, $(x-\tilde{x})^T.(x-\tilde{x})$ has to be minimised.

This can be further be written as:

$$(x-\Gamma\gamma)^T.(x-\Gamma\gamma)$$

which is minimal when

$$\gamma = L.x$$

with

$$L=\left(\Gamma^T.\Gamma\right)^{-1}\Gamma^T$$

Note that the smoothing must be checked by comparing x and $\gamma.L.x$.

A - 2.2 – Rationalisation of the aerodynamic forces for time simulations

The aerodynamic loads are usually expressed with the C_P which depend on the Mach number and on frequency (actually the reduced frequency $k = \omega / V$).

For time simulations, the Laplace Domain Reduction (or rationalisation) is needed to include these forces in the computations.

In the modal basis, the generalised aerodynamic forces are usually given in the form: $FAG(k).X$.

The FAG can be approximated by the sum of a second-order polynomial and a rational function (the ratio of two polynomial functions):

$$FAG(k) = A_0 + A_1.k + A_2.k^2 + \sum_{i=1}^{N} \frac{k.A_i}{k - p_i}$$

where N is the order of the approximation

p_i are the poles of the interpolation

This operation takes place in the frequency domain, hence its name 'Laplace domain reduction".

The loads can be written in a matrix form as follow:

$$FAG(s).X = \left(A_0 + sP_{dyn}\frac{A_1}{V} + s^2 P_{dyn}\frac{A_2}{V^2} + sD(sI - P)^{-1}E \right).X$$

Set $\eta = s(sI - P)^{-1}E.X$

which is equivalent to $s\eta - P\eta = sE.X$

or in the temporal domain: $\dot{\eta} - P\eta - E.\dot{X} = 0$.

Hence the equations of the systems in the temporal domain are:

$$\begin{cases} FAG(t).X = A_0 X + P_{dyn}\frac{A_1}{V}\dot{X} + P_{dyn}\frac{A_2}{V^2}\ddot{X} + \bar{q}D\dot{\eta} \\ \dot{\eta} = P\eta + EX \end{cases}$$

These temporal loads can be easily added in the temporal equation of motion *(see 3.2 – Time response).*

A quasi-steady method does exist. Instead of interpolating the points for several frequencies with a rational function, the interpolation uses only two points: the FAG are approximated with a straight line.

$$FAG(s) = A_0 + \bar{q}\frac{A_1}{V}.s$$

At a fixed Mach number:

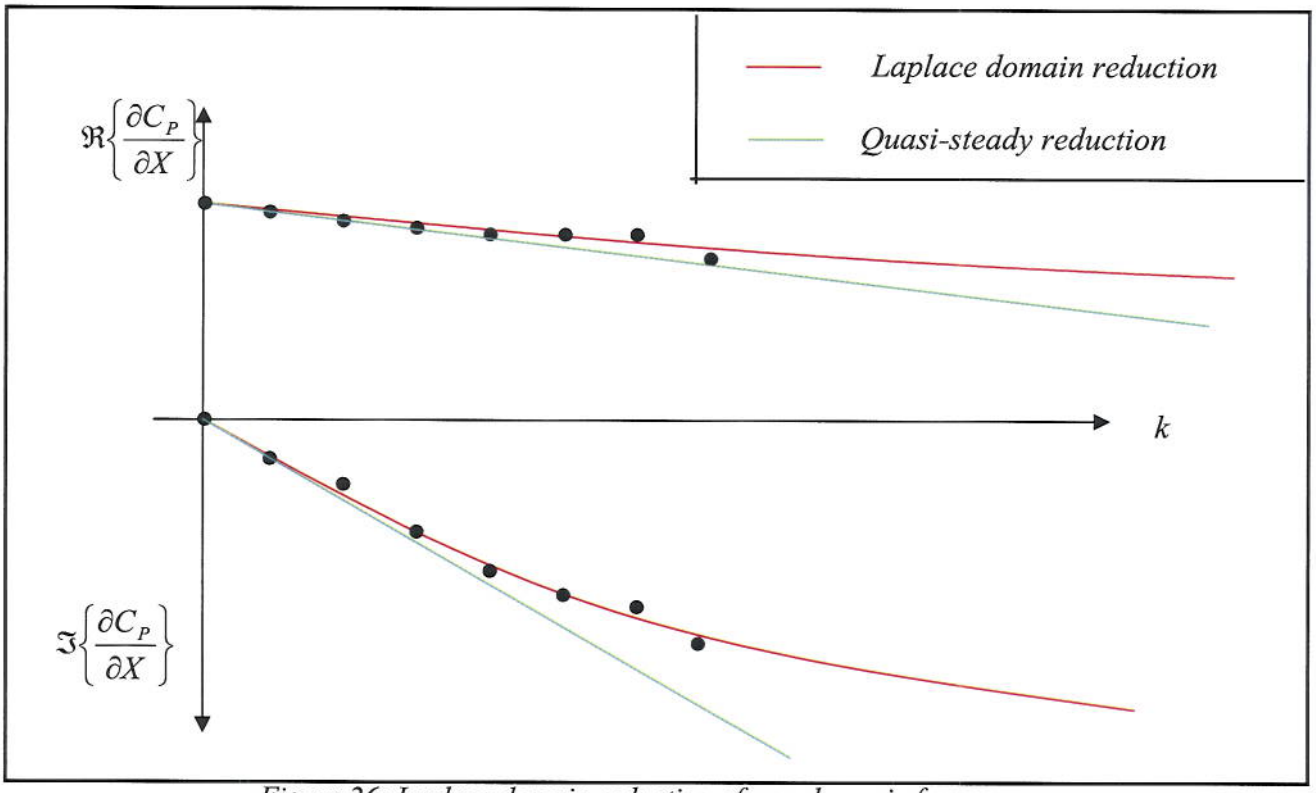


Figure 26: Laplace domain reduction of aerodynamic forces

This operation is done for each mode.

A - III – AEROELASTICITY

Aeroelasticity is the science which studies the interaction among inertial, elastic, and aerodynamic forces. Airplanes are not completely rigid and structure deformations may induce changes on aerodynamic forces. The additional force may cause instabilities. As an example, a flutter may arises when aerodynamic forces are coupled with one or several natural modes of the structure.

A - 3.1 – Flutter: the PK-Method

In flutter analysis we look for singularities of the aeroelastic equation.

A - 3.1.1 - Eigenvalue problem

In the modal basis, the equation of motion governing the generalised displacements is:
$M\ddot{X} + C\dot{X} + KX = Q = q.X$

Assume little displacement around an equilibrium point, the displacements can be written as the sum of a constant displacement and a variable displacement $X = X_0 + \tilde{X}$.

$$\Rightarrow \quad M\ddot{\tilde{X}} + C\dot{\tilde{X}} + K\left(X_0 + \tilde{X}\right) = q_{aero}\left(X_0 + \tilde{X}\right)$$

In the frequency domain, the equation becomes:

$$-\omega^2 M\tilde{X} + i\omega C\tilde{X} + K\left(X_0 + \tilde{X}\right) = q_{aero}\left(X_0 + \tilde{X}\right)$$

The aerodynamic forces can be linearised to obtain an eigenvalue problem:

$$-\omega^2 M\tilde{X} + i\omega C\tilde{X} + K\left(X_0 + \tilde{X}\right) = q_{aero}\left(X_0\right) + \frac{\partial q_{aero}}{\partial \tilde{X}}\left(\frac{\omega}{V}, M\right)\tilde{X}$$

Consider the equilibrium of the static problem: $KX_0 = q_{aero}\left(X_0\right)$. The final equation which has to be solved is:

$$\Rightarrow \quad -\omega^2 M\tilde{X} + i\omega C\tilde{X} + K\tilde{X} - \frac{\partial q_{aero}}{\partial \tilde{X}}\left(\frac{\omega}{V}, M\right)\tilde{X} = 0$$

A - 3.1.2 - Input vibration

In an eigenvalue problem, the system is vibrating in a linear combination of its modes. The study then gives the frequency, the damping and the modal shape of each mode.

Assume the vibration is $X(P,t) = \sum_i \alpha_i Z_i(t),\ \alpha_i \in \boldsymbol{C}$,

$Z_i = \left[Z_i^j \right] e^{p_i t}$ which describes the vibration in the mode i, $\left[Z_i^j \right]$ is the modal shape (complex), $\Re\{p_i\}$ gives the damping of the mode and $\Im\{p_i\}$ its frequency:

$$p = \xi\omega_n + i\omega_n\sqrt{1-\xi^2} = \xi\omega_n + i\omega$$

where ω_n is the natural frequency of the mode.

The reduced frequency is $k = \frac{\omega}{V}$. (Some authors use $k = \frac{\omega L}{V}$ where L is a characteristic length of the system, but this length is usually not displayed by aeronautics engineers).

In the Laplace domain, the eigenvalue problem becomes:

$$\Rightarrow \left[p^2 M + pC + K - P_{dyn} \frac{\partial q_{aero}}{\partial \tilde{X}}(k, M) \right] \tilde{X} = 0$$

$$\Leftrightarrow \left[Z\left(P_{dyn}, \omega, \xi\right) \right] \tilde{X} = 0$$

A - 3.1.3 - Resolution with the modified PK-method (Brévan)

The purpose is to find the triplets $\left(P_{dyn}, \omega, \xi\right)$ for which $\left[Z\left(P_{dyn}, \omega, \xi\right) \right] \tilde{X} = 0$.
First, at $\left(P_{dyn\,0}, \omega_0, \xi_0\right)$, we write $Z_{000} = \left[Z\left(P_{dyn\,0}, \omega_0, \xi_0\right) \right]$.
$Z_{000} X_0 = 0$ which is the modal basis at $V_0 = 0$.

Loop on P_{dyn} V with a step dP_{dyn}:

We want to find Z_{111} and X_1 so that $Z_{111} X_1 = 0$
$P_{dyn_1} = P_{dyn\,0} + dP_{dyn}$

Loop on the modes:

- Compute $Z_{100} = \left[Z\left(P_{dyn_1}, \omega_0, \xi_0\right) \right]$.

For little displacements, $Z_{100} X_0 = \varepsilon X_0$, where ε is the smallest eigenvalue of Z_{100}.
We can also find ${}^t Y_0 Z_{100} = \varepsilon {}^t Y_0$.

By differentiating the equation $Z_{100} X_0 = \varepsilon X_0$, we obtain:

$$\frac{\partial Z_{100}}{\partial P_{dyn}} X_0 dP_{dyn} + \frac{\partial Z_{100}}{\partial \omega} X_0 d\omega + \frac{\partial Z_{100}}{\partial \xi} X_0 d\xi = 0.$$

At a first approximation, $Z_{100} = Z_{000} + \frac{\partial Z_{100}}{\partial P_{dyn}} dP_{dyn}$.

Hence, $Z_{100} X_0 - Z_{000} X_0 + \frac{\partial Z_{100}}{\partial \omega} X_0 d\omega + \frac{\partial Z_{100}}{\partial \xi} X_0 d\xi = 0$

$$\Rightarrow \boxed{Z_{100}X_0 + \frac{\partial Z_{100}}{\partial \omega} X_0 d\omega + \frac{\partial Z_{100}}{\partial \xi} X_0 d\xi = 0} \quad \boxed{A}$$

Loop on $d\omega$ and $d\xi$:

- Compute $d\omega$ and $d\xi$ with the complex equation above.
- Compute $Z_{111} = \left[Z\left(P_{dyn_1}, \omega_1, \xi_1\right)\right]$

$$\Rightarrow \quad Z_{111}X_1 = Z_{111}\left(X_0 + dX_0\right) = Z_{111}X_0 + Z_{111}dX_0 = Z_{111}X_0 + \left(Z_{000} + dZ_{000}\right)dX_0$$

$$\Rightarrow \quad Z_{111}X_1 = Z_{111}X_0 + dZ_{000}dX_0$$

$$\Rightarrow \quad dX_0 = {Z_{111}}^{-1} dZ_{000} X_0$$

- Compute X_1
- Compute $Z_{111}X_1$ and if it is still to big, go to $\boxed{A}$ until $Z_{111}X_1$ is small enough

If dP_{dyn} is small enough, the algorithm converges.

A - 3.1.4 - PK-method with feedback

The equation for each mode becomes

$$\boxed{\left[p^2M + pC + K + G(\omega) - P_{dyn}\frac{\partial q_{aero}}{\partial x}(k,M)\right]\tilde{X} = 0}.$$

$G(\omega)$ is the transfer function in the modal basis *(See 1.1.5 - How to add feedback?)*

We proceed in two steps.

Firstly solve

$$\det\left[p^2M + pC + K + G(\omega)\right] = 0$$

A PK-method is used as before: a number ε plays the role of the dynamic pressure and the transfer function plays the role of the dynamic forces. When ε is one, we have the modes with feedback but without aerodynamic forces.

Secondly proceed as above to find the modes with aerodynamic forces.

Graphically, if we plot $-\omega^2M + j\omega C + K$ and $G(\omega)$ for the mode 1 at the first P_{dyn} , the algorithm converges toward the intersection point:

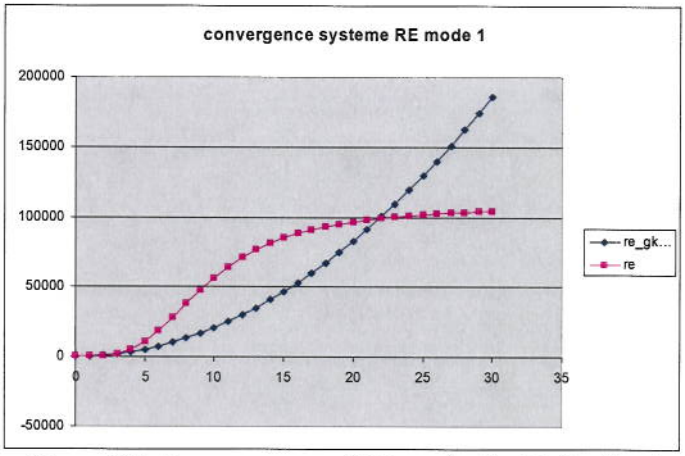


Figure 27: Convergence of PK-method with feedback

A - 3.2 – Time Response

A - 3.2.1 - Equations

In the modal basis, the equation which has to be solved is:

$$M_S\ddot{X} + C_S\dot{X} + K_S X = Q$$

With rationalised aerodynamic forces, the equations of the systems are:

$$\begin{cases} M\ddot{X} + C\dot{X} + KX = F(t) + A_0 X + P_{dyn}\dfrac{A_1}{V}\dot{X} + P_{dyn}\dfrac{A_2}{V^2}\ddot{X} + P_{dyn}D\dot{\eta} + F_{ig} \\ F_{ig} = -M_{ig}\ddot{X}_{ig} - C_{ig}\dot{X}_{ig} - K_{ig}X_{ig} + A_0 X_{ig} + P_{dyn}\dfrac{A_1^{ig}}{V}\dot{X}_{ig} + P_{dyn}\dfrac{A_2^{ig}}{V^2}\ddot{X}_{ig} \\ \dot{\eta} = P\eta + EX + E_{ig}X_{ig} \end{cases}$$

$$\begin{pmatrix} M - P_{dyn}\frac{A_2}{V^2} & 0 & M_{ig} - P_{dyn}\frac{A_2^{ig}}{V^2} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} \ddot{X} \\ \ddot{\eta} \\ \ddot{X}_{ig} \end{pmatrix} + \begin{pmatrix} K - P_{dyn}\frac{A_1}{V} & P_{dyn}D & C_{ig} - P_{dyn}\frac{A_1^{ig}}{V^2} \\ 0 & I & 0 \\ 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} \dot{X} \\ \dot{\eta} \\ \dot{X}_{ig} \end{pmatrix} + \begin{pmatrix} K - P_{dyn}A_0 & 0 & K_{ig} - P_{dyn}A_0^{ig} \\ -E & -P & -E_{ig} \\ 0 & 0 & I \end{pmatrix}\begin{pmatrix} X \\ \eta \\ X_{ig} \end{pmatrix} = \begin{pmatrix} F(t) \\ 0 \\ X_{ig} \end{pmatrix}$$

X_{ig} is the input due to the motion of the control surface. The motion of control surfaces leads to the creation of inertia, damping and stiffness forces. As a result, M_{ig}, K_{ig}, C_{ig} are the

matrices of the mass, stiffness and damping of the corresponding modes. K_{ig} and C_{ig} are often equal to zero.

Note that it is equivalent to add the lines of the aerodynamics in the matrices. Interpolating loads with a Laplace domain reduction is like adding some degree of freedom to the system, the degree of freedom of the aerodynamics.

The non linear feedback expressed as a force is added to $F(t)$.
Set these new matrices and vectors,

$$\widetilde{M} = \begin{pmatrix} M - P_{dyn}\frac{A_2}{V^2} & 0 & M_{ig} - P_{dyn}\frac{A_2^{ig}}{V^2} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Number of modes
Number of poles of the aero
Number of control surface modes

Number of degree | Number of poles of the aero modes | Number of control surface modes

$$\widetilde{C} = \begin{pmatrix} K - P_{dyn}\frac{A_1}{V} & P_{dyn}D & C_{ig} - P_{dyn}\frac{A_1^{ig}}{V^2} \\ 0 & I & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \widetilde{K} = \begin{pmatrix} K - P_{dyn}A_0 & 0 & K_{ig} - P_{dyn}A_0^{ig} \\ -E & -P & -E_{ig} \\ 0 & 0 & I \end{pmatrix},$$

$$\widetilde{X} = \begin{pmatrix} X \\ \eta \\ X_{ig} \end{pmatrix}, \quad \widetilde{F} = \begin{pmatrix} F \\ 0 \\ X_{ig} \end{pmatrix}$$

The new equation becomes,

$$\boxed{\widetilde{M}.\ddot{\widetilde{X}} + \widetilde{C}\dot{\widetilde{X}} + \widetilde{K}\widetilde{X} = \widetilde{F}} \qquad (2)$$

A - 3.2.2 - Resolution

There are quite a lot of time-marching schemes to solve the previous equation. At t_n, the derivatives of $\widetilde{X}$ are expressed with respect to $\widetilde{X}_i$, $i < n$. The equation (2) becomes:

$$P[0]\widetilde{X}_{n+1} = P[1]\widetilde{X}_n + P[2]\widetilde{X}_{n-1} + P[3]\widetilde{X}_{n-2} + P[4]\widetilde{X}_{n-3} + \widetilde{F}$$

The coefficients depend on the method. On Table 6, there are the coefficients for Houbolt, Gear and Newmark methods.

Method	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}
Houbolt	0	11dt/6	dt^2	-5	-3dt	0	4	1.5dt	0	-1	-dt/3	0
Gear	2	1.5dt	dt^2	-5	-2dt	0	4	0.5dt	0	-1	0	0
Newmark	1	μ_1	λ_1	-2	μ_2	λ_2	1	μ_3	λ_3	0	0	0

Table 5: Coefficients of time-marching schemes

For Newmark method: $\gamma = 1/2$, $\beta = 1/2$, $\lambda_1 = \beta.dt^2$, $\lambda_2 = (1/2 + \gamma - 2\beta).dt^2$, $\lambda_3 = (1/2 - \gamma + \beta).dt$, $\mu_1 = \gamma.dt$, $\mu_2 = (1 - 2\gamma).dt$, $\mu_3 = (\gamma - 1).dt$

HOUBOLT

$$P[0] = c_0 M + c_1 C + c_2 K$$
$$P[1] = -(c_3 M + c_4 C)$$
$$P[2] = -(c_6 M + c_7 C)$$
$$P[3] = -(c_9 M + c_{10} C)$$

NEWMARK

$$P[0] = c_0 M + c_1 C + c_2 K$$
$$P[1] = -(c_3 M + c_4 C + c_5 K)$$
$$P[2] = -(c_6 M + c_7 C + c_8 K)$$

GEAR

$$P[0] = c_0 M + c_1 C + c_2 K$$
$$P[1] = -(c_3 M + c_4 C)$$
$$P[2] = -(c_6 M + c_7 C)$$
$$P[3] = -(c_9 M)$$

Houbolt and Gear methods are second order methods. Newmark method is of order one.

$\tilde{X}_{n+1}$ is obtained by summing the second term and inversing the matrix $P[0]$.

A - 3.2.3 – Input of time simulations

One purpose of this project is a stability analysis. We want to see how the system reacts after a local perturbation. The perturbation must be pertinently chosen. It must excite a large range of frequencies (to have an influence on the maximum number of modes).

The following function is a good impulse :

$$input = 0.5 \times \frac{\sin\left(\omega\left(t - n\frac{t_0}{2}\right)\right)}{\left(\omega\left(t - n\frac{t_0}{2}\right)\right)}\left(1 - \cos\left(2\pi\frac{nt}{t_0}\right)\right) \times Hanning(t, 0, nt_0)$$

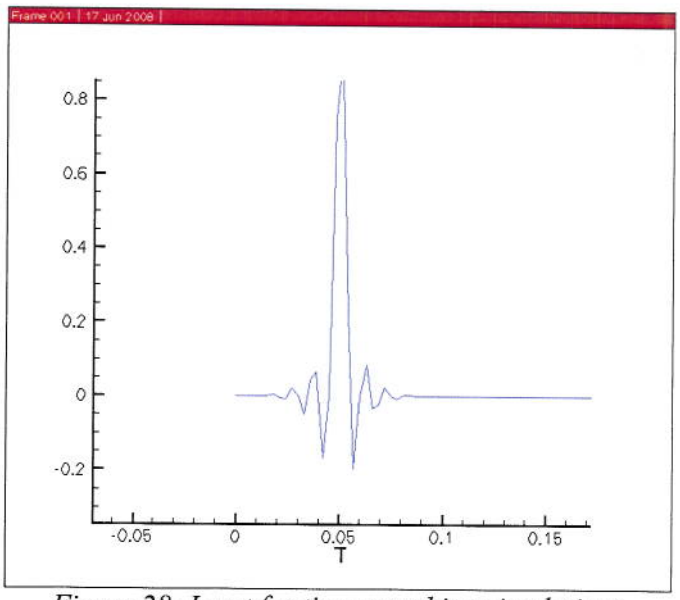


Figure 28: Input for time-marching simulations

According to the shape of its Fourier transform, this function has an influence on frequencies between 0 and 40 Hz :

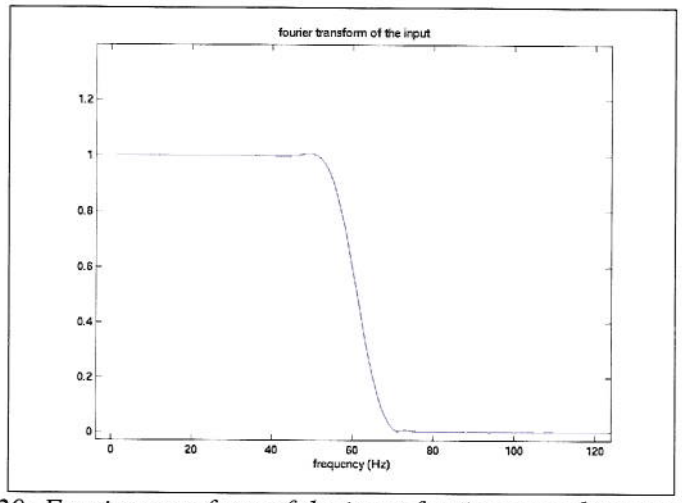


Figure 29: Fourier transform of the input for time-marching simulations

PART B

SIMULATIONS

B - I – PRELIMINART STUDY WITHOUT FEEDBACK

In this paragraph, we will show several characteristics and results we can have with flutter computations and time response. The purpose is to have conclusions on the methodology. The model used in this study is a simple one (not the one of the elevator we will use for the simulations with feedback).

B - 1.1 - Flutter

The first ten elastic modes are plotted on the graph below. This is an iso-mach computation. The ninth curve is going unstable at a dynamic pressure of 45000, which corresponds to an altitude of m.

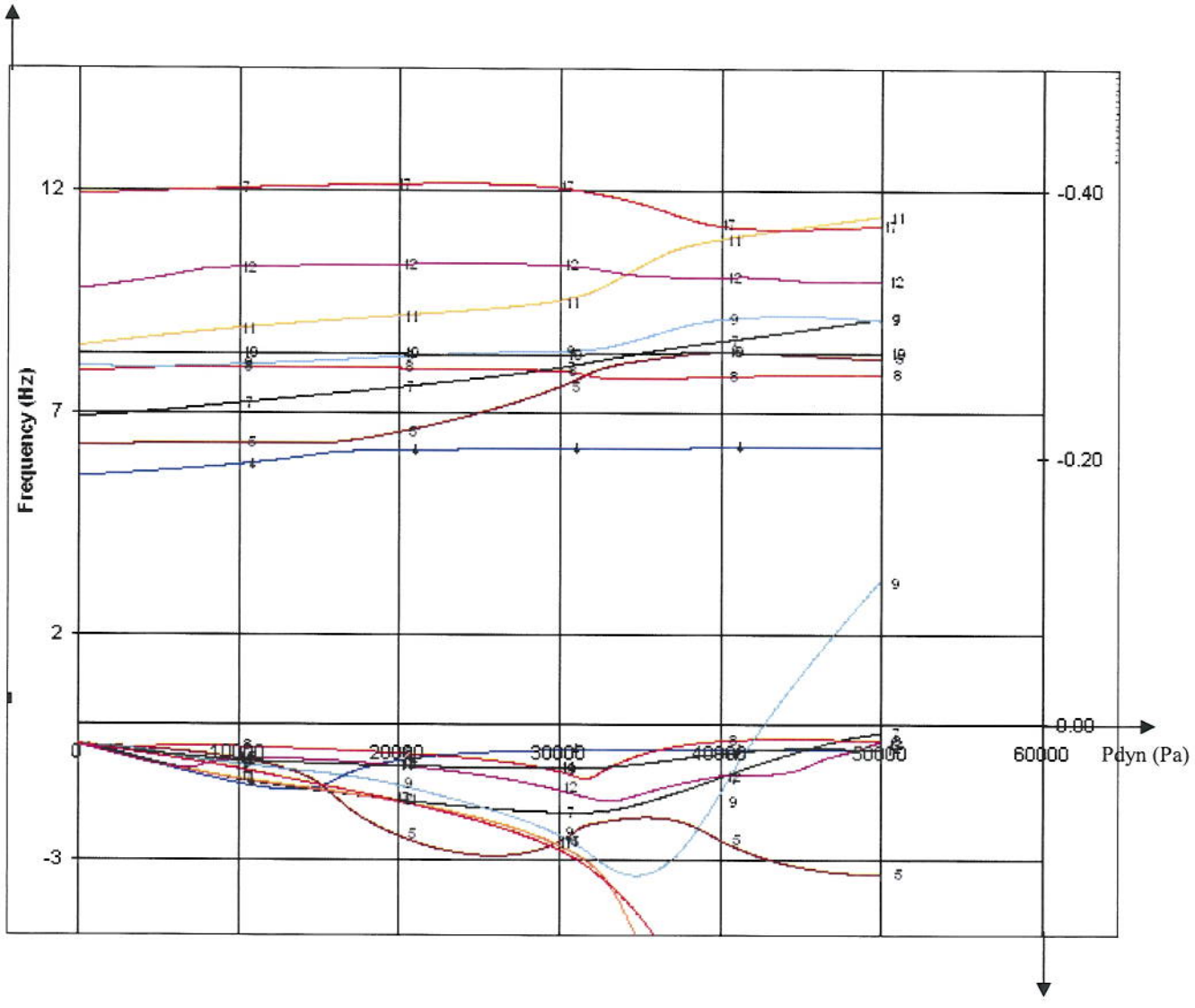


Figure 30: Flutter curve of the first ten modes of a study model

The frequency of the mode 9 is about 9Hz at the flutter point. We can plot the shape of the mode at this flutter point. In flight, the modes are complex, so there are a real part and an imaginary part:

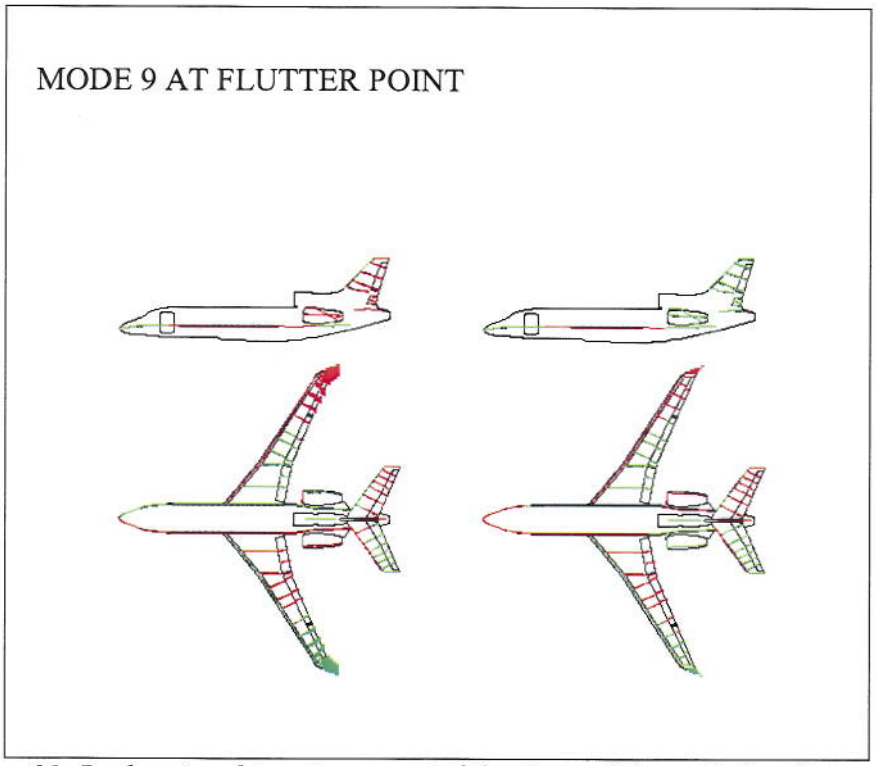


Figure 31: Real part and imaginary part of the shape of the mode 9 at flutter point

The instability is caused by a bending-torsion coupling. As the study is done on generalised displacements, we can not see explicitly the real displacements of the structure. It would have been possible with the transfer matrix between the modal basis and the physical displacements (with the MONSETs).

B - 1.2 - Time Response

Here the parameters chosen in the input function are:
$f_0 = 10$, $f_{\max} = 20$, $\omega = 2\pi f_0$, $t_0 = 1/f_0$, $SIZE\_T = 10000$, $TMAX = 30$
We want to analyse whether the time response will be stable or not at a certain dynamic pressure. The input is an impulse, as explained in section A-3.2.3.

If the dynamic pressure is higher than P_{dyn0} (or the altitude smaller than h_0), the time response is unstable. For instance for an altitude of 490m:

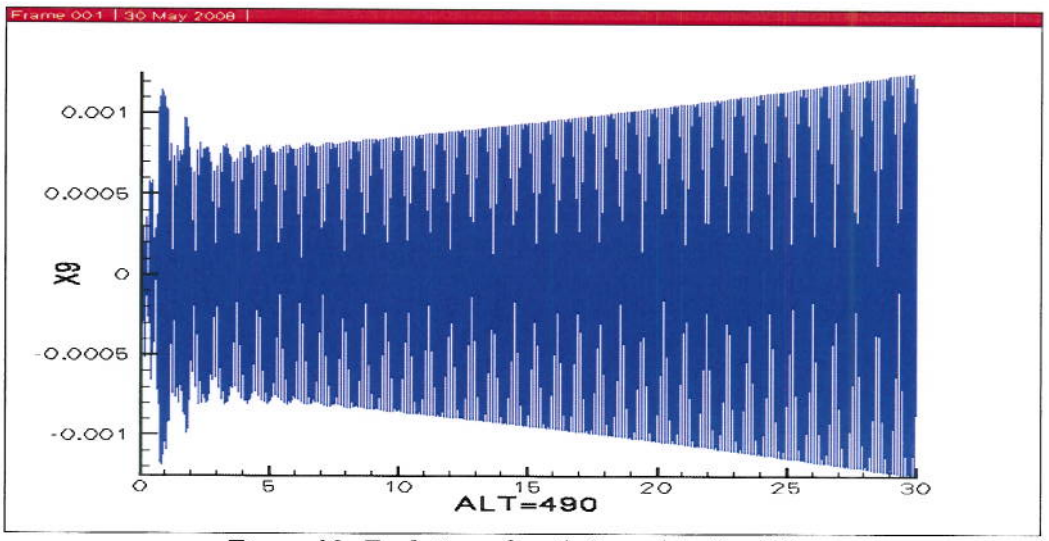


Figure 32: Evolution of mode 9 at altitude=490m

On the contrary, for higher altitudes the time response is stable (500m in the graph below). The simulation has been done in a larger period of time to be sure that it did not go unstable afterwards.

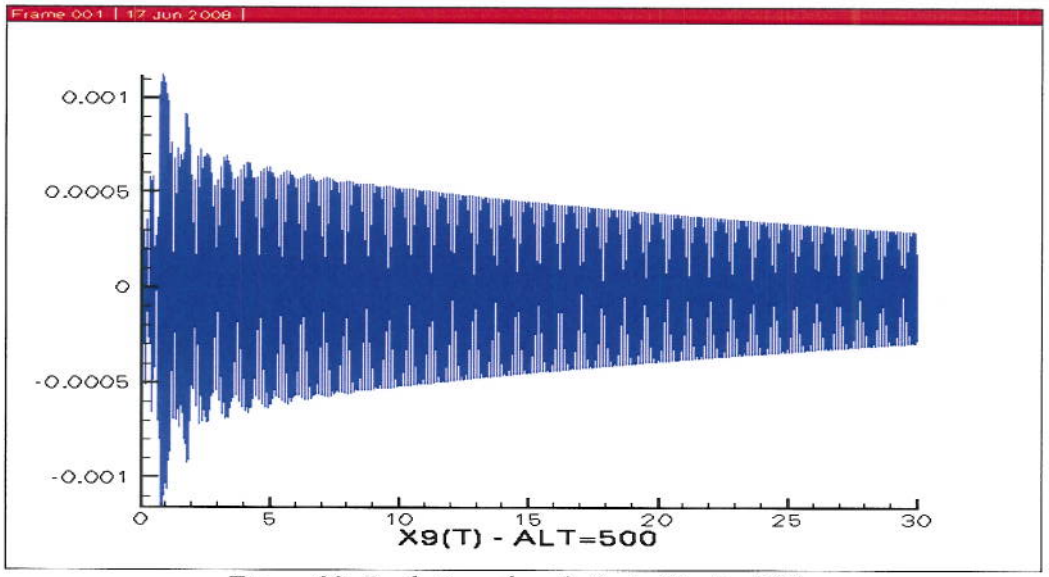


Figure 33: Evolution of mode 9 at altitude=500m

All the modes of the time response oscillate in phase at a frequency of 9Hz (the frequency of the mode 9 at the flutter point). It corresponds to a mode in flight, linear combination of the modes at ground (in another basis).

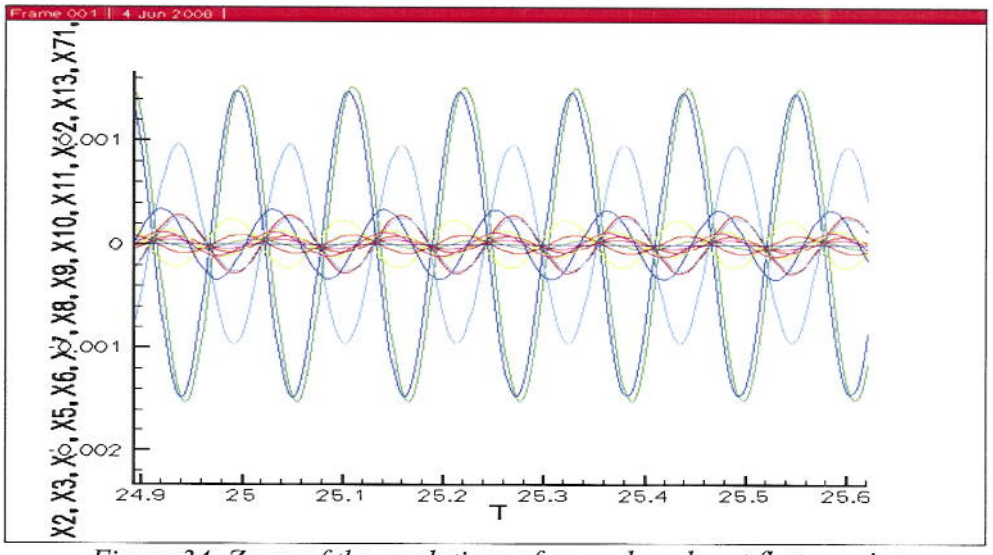


Figure 34: Zoom of the evolutions of several modes at flutter point

We want to obtain the same shape from this time response than the one we obtained from the flutter calculation. By scanning the time response at two different time separated by the quarter of the period, we should obtain the real part and the imaginary part of the previous graph.

Indeed, $\Re\{GDISP \times e^{i\omega t}\} = \Re\{GDISP\}\cos(\omega t) - \Im\{GDISP\}\sin(\omega t)$.
By scanning the time response of the mode 9 for different instants, we should have $\Re\{GDISP\}$ at t and $-\Im\{GDISP\}$ at $t+T/4$. A quite identical shape is found at 25s.

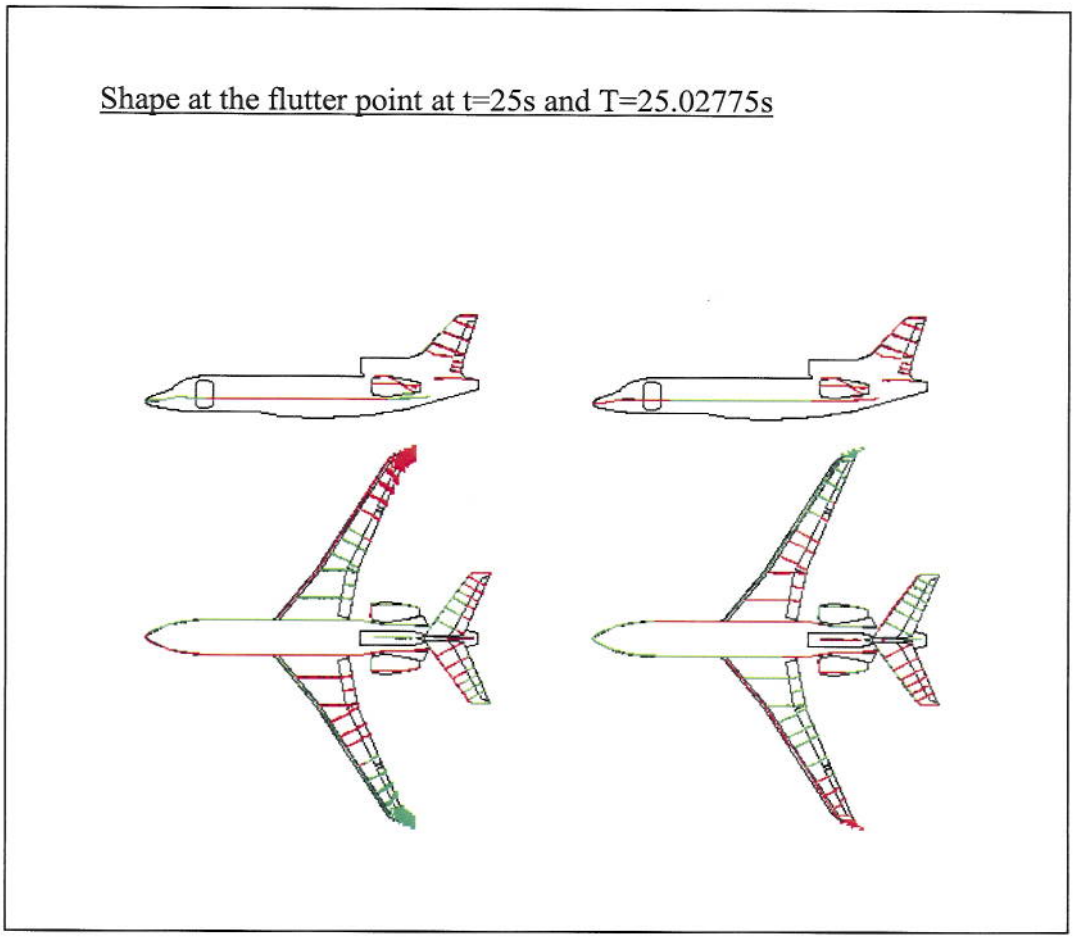
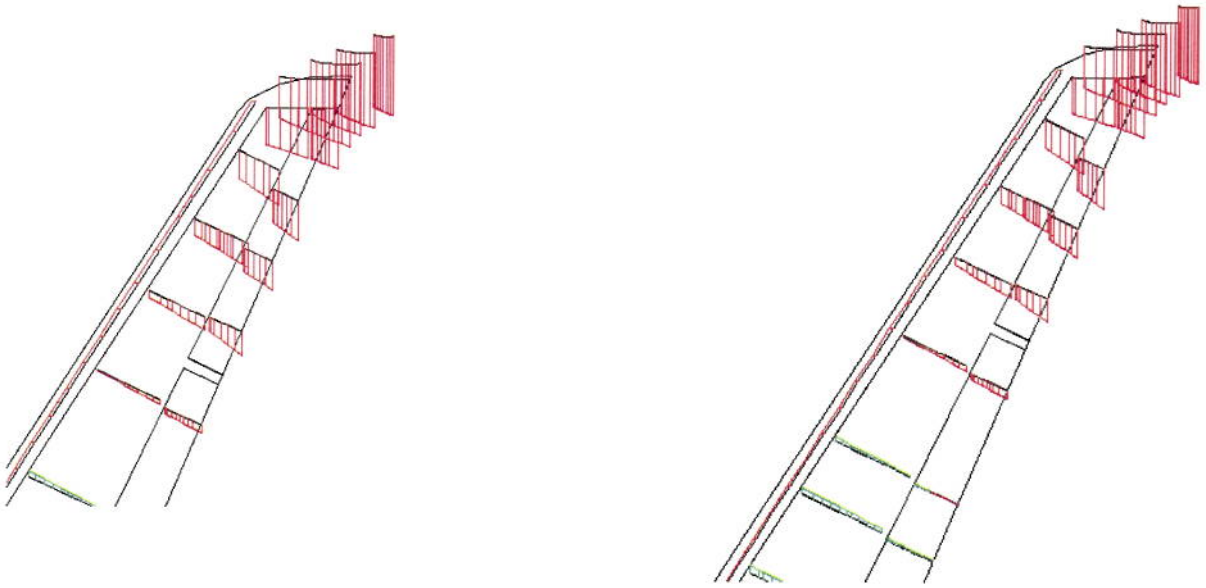


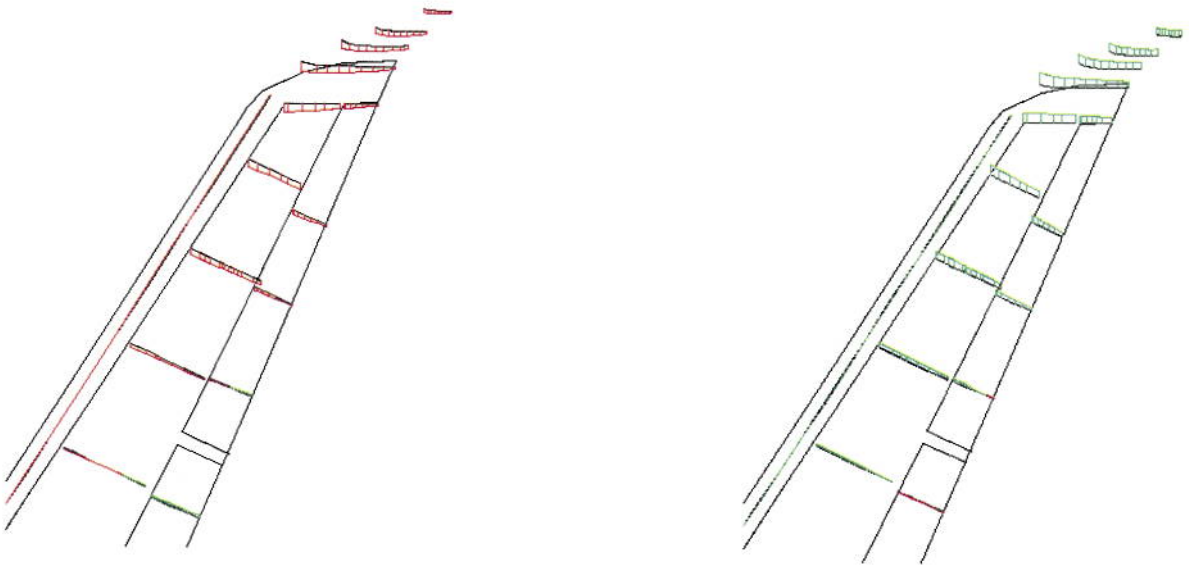
Figure 35: Shape at the flutter point from time simulations

Figure 36: Comparison of shapes at flutter point between time simulations and flutter simulations

ZOOM OF THE SHAPE OF THE REAL PART AND THE SHAPE AT t



ZOOM OF THE SHAPE OF THE IMAGINARY PART AND THE SHAPE AT t+T/4



Note the negative sign between the shape of the imaginary part and the shape at $t+T/4$, which we had in the equation.

B - II – MODEL USED FOR SIMULATIONS WITH FEEDBACK

B - 2.1 – Finite Element model

The study is done on a model of a Falcon 7X, more especially on the elevators.

The two rods which represent the two servo-actuators on the right elevator are cut.

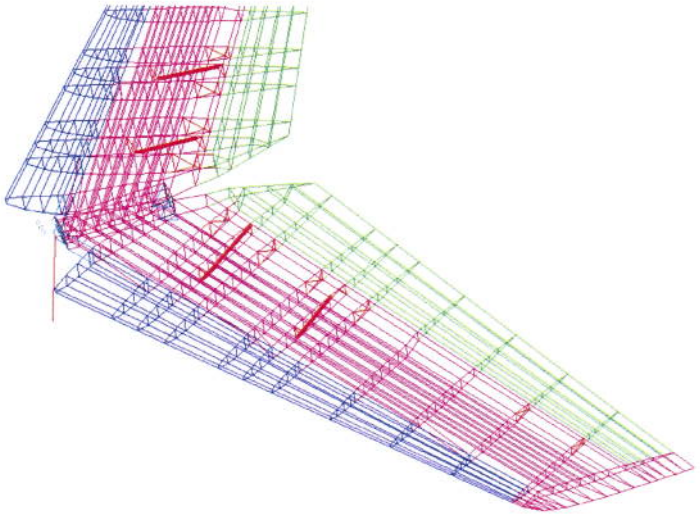


Figure 37: Catia view of the servo-actuators on the elevators

To impose the feedback on the correct points the coordinates of the nodes of the rod are taken from Catia.

B - 2.2 – Reduced model

A reduced model is derived from the finite element model. This reduced model has 6 rigid modes (translations, rotations), 150 flexible modes, 4 partial rigid modes and 5 loads.
We can see that there is a mode at a zero frequency in our model. To suppress the rods is equivalent to create a rigid mode. If we add stiffness equal to the original stiffness through Elfini ("CRESTIFF" command), there is no mode at a zero frequency any more but a mode at 15.543Hz instead. *(See Appendix 1 for the frequency and the damping of some flexible modes and loads.)*

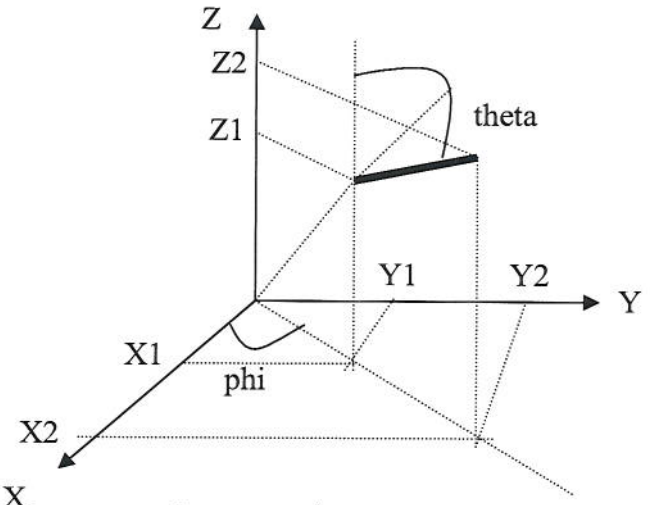
The loads are some modes which are manually added when the model is created. They correspond to the tractions applied on the rods. They are four rods in the model (two per elevator) so there are four loads added to the modal basis. A fifth load is artificially added to complete the basis. The frequencies of the loads are very high (more than 100Hz) and their damping are close to zero.

To put the feedback in the right place we have to build a MONSET which represents the rod. From the coordinates taken in Catia, we can derive the angles "phi" and "theta":

$$\cos(\varphi) = \frac{X2 - X1}{\sqrt{(X2 - X1)^2 + (Y2 - Y1)^2}}$$

$$\tan(\theta) = \frac{\sqrt{(X2 - X1)^2 + (Y2 - Y1)^2}}{Z2 - Z1}$$

We put in a MONSET the three translations and we project the MONSET on the axis of the rod. This projection is artificially put in the next line of the MONSET called "RX".



$$"RX" = (TX2 - TX1)\cos(\varphi)\sin(\theta) + (TY2 - TY1)\sin(\varphi)\sin(\theta) + (TZ2 - TZ1)\cos(\theta)$$

The stiffness of the rod can also be computed from some information taken in Catia: the area of the section of the rod and the Young modulus.

$$K = \frac{E \times S}{L} \quad , \quad S = 100.10^{-6} m^2 \quad , \quad E = 71000.10^6$$

In table 6 are the coordinates of the nodes used to build the MONSETs.

MONSET	Point	X (m)	Y (m)	Z (m)	L (m)	Theta (rad)	Phi (rad)	K
SER2	1	20.149	-0.841	2.289	0.515	1.319	0.453	1.38E+07
SER2	2	20.597	-0.622	2.417				
SERV	1	20.629	-1.756	2.171	0.515	1.319	0.453	1.52E+07
SERV	2	21.077	-1.5379	2.299				
DAXE		0.87049	0.4244	0.2491				

Table 6: Coordinates of the nodes used to build the transfer matrices

The input of the feedback is the MONSET DAXE created in the axis of the rod.
The output is a force which is on the same axis than the rod. A GLOAD is then created from the MONSET DAXE.
The values of the feedback are the values of the transfer function computed with Matlab.

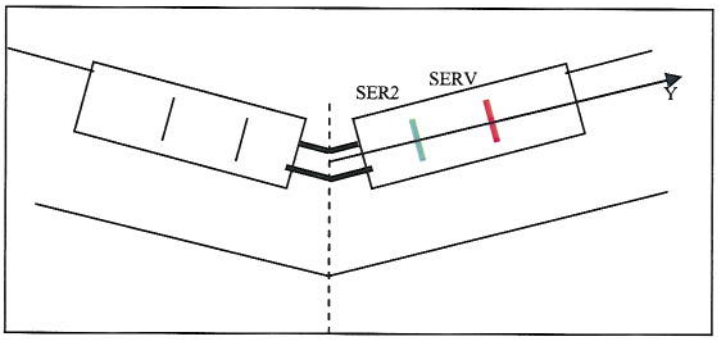


Figure 38: Scheme of the places of the MONSETs on the tail assembly

B - III – FLUTTER WITH FEEDBACK

This study in the frequency domain is performed with the previous model, in which the two rods of the left elevator have been cut. It is possible to add a feedback in the computations of flutter *(see PK-method section A-3.1)*. It can be added through a transfer function ("CREDATAFREQ" command). The values are computed with Matlab. For different amplitudes, the transfer functions are different *(see section A-1.2 for the Model of the servo-actuator)*. The effect on the flutter curves is to shift them towards the higher dynamic pressures (towards the right).

B - 3.1 - Model alone: the two rods are cut

In the finite element model which has been installed on the machine for this project with feedback, the two rods of the left elevator are cut. As a result, there are a lot of instabilities found with frequency domain simulations. They are of different sorts. Only the relevant modes have been plotted in the appendix.

In the flutter curves (also called V-g plots), the frequencies are at the top of the graph and the damping of the modes are at the bottom.

There are instabilities in which the damping is negative at low dynamic pressures and becomes positive again afterwards. It is the case for modes 4, 13, 6 and 9.

The modes 7, 10, 14 and 68 goes to flutter. They have the classical shapes of modes which are used as flutter boundaries for stability studies.

We can also see that the modes 37 and 46 are changing with each other in the damping curves. The same phenomenon occurs in the frequencies between mode 88 and mode 89. These changes are due to the convergence of the flutter algorithm.

(See flutter-appendix E1)

B - 3.2 - Test of the Laplace domain reduction of the aerodynamic forces

The aerodynamic forces are usually provided and stored as functions of frequencies. The Laplace domain reduction enable these forces to be taken into account in time domain simulations *(see section A-2.2 for the Laplace domain reduction)*. The Laplace domain reduction of the aerodynamic forces can be sampled at different frequencies in order to perform frequency domain simulations. The flutter computations can be done either with aerodynamic forces given in frequency or with the result of this sampling. As a result the comparison of the flutter curves allows the validation of the Laplace domain reduction without feedback.

(See flutter-appendix E2)

B - 3.3 - Comparison of the curves for different amplitudes in the transfer function

For different amplitude X_0, transfer functions are computed in Matlab and the corresponding flutter curves are plotted. These simulations take the stiffness of the rod into account. The curves of the modes, which does in flutter are plotted in the same graph *(See flutter-appendix E3)* . In figure 39, we see a zoom of the appendix E3, focusing on the first mode which has negative damping. As the amplitude used to compute the transfer function increases, the flutter point shifts towards higher dynamic pressures.

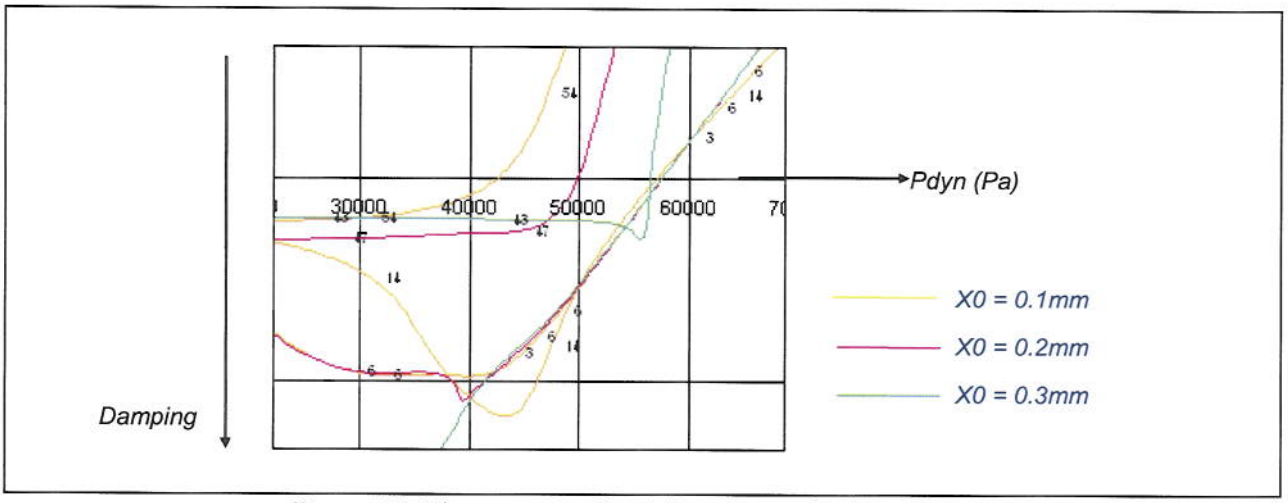


Figure 39:Flutter points for different transfer functions

Time domain simulations are going to be performed at several dynamic pressures around the flutter points. As there are some instabilities in the feedback with the model of the hydraulic failure of the servo-actuator, some limit cycle oscillations may appear. The main question is then: At a certain dynamic pressure, are the amplitudes of the LCO equal to the amplitudes used for the transfer functions?

B - IV – TIME RESPONSE WITH FEEDBACK

B - 4.1 - Validation of the feedback using Matlab

In the initial model, the two rods are cut. Stiffness can be added either initially on the model (using the command "CRESTIFF", *see 1.1.5 – How to add stiffness to the model*) or with a Matlab program which returns a force -*KX* for a given displacement.

On figure 38, the result is the displacement of the nodes of the rods (MONSET DAXE).
The result using a "CRESTIFF" is in green. The result using a call to Matlab is in red.

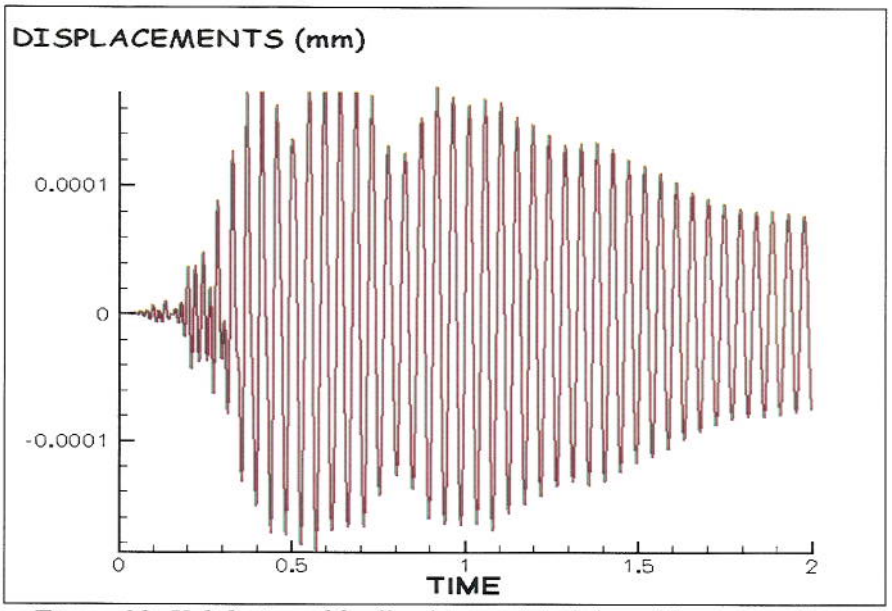
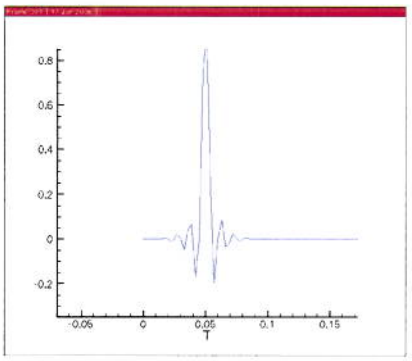


Figure 38: Validation of feedback using Matlab : additional stiffness

As the two curves are the same, this simulation validates the call to Matlab and the treatment of the results of Matlab in Elfini.

B - 4.2 - Time Response with feedback



For time domain simulations, the input is the same than the one used without feedback:
As previously, for low dynamic pressure, the time response converges and for high dynamic pressures, it diverges.
In a small range of values of the pressure the divergence or the convergence is very slow. This is a limit cycle oscillation (LCO) after a transient period.

Different aerodynamic forces can be used for time simulations. In the quasi-steady method, we subtract only $\bar{q}A_0$ to the stiffness matrix and $\bar{q}A_1$ to the damping matrix. It is equivalent to consider that the aerodynamic forces are straight lines (taking the two first points of the rational function). This method is not as accurate as the method where rational functions are used. Numerical divergences happen for different altitudes and it is difficult to determine the physical convergence or divergence of the simulation.

As previously, before putting a non-linear feedback, a study can be realised by adding stiffness directly to the model. This is all the more pertinent that the simulation with a non-linear feedback last longer than a simulation with a CRESTIFF.

B - 4.2.1 - Additional stiffness and Quasi Steady Aero

The divergence is observed clearly at 2000 and 3000 meters. At 5000 and 6000 meters, the simulations diverge with a numerical divergence. Small oscillations follow the sine and make the simulation diverge. At 10000 meters the numerical divergence happens at the end of the 10s.

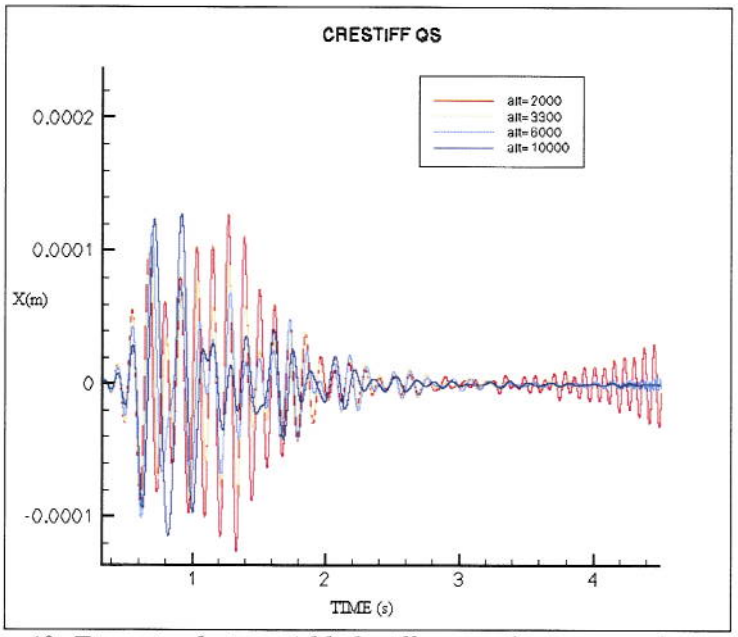


Figure 40: Time simulation: Added stiffness and quasi-steady aero

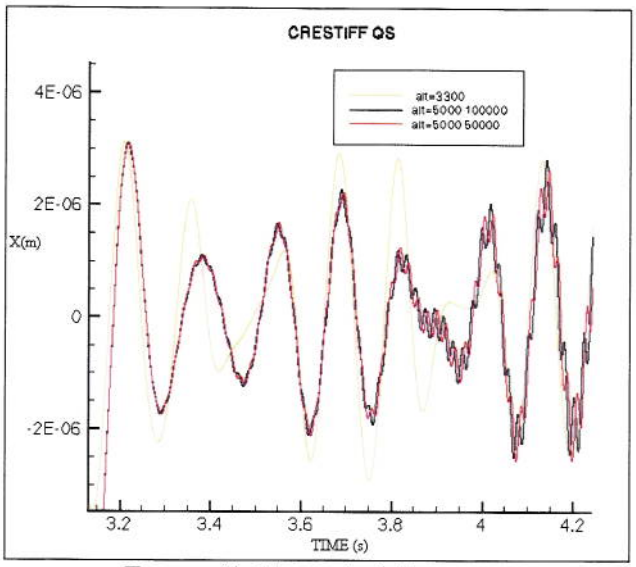


Figure 41: Numerical divergence

This numerical divergence, which is frequently observed with quasi-steady aero, remains present when the sampling is doubled.

B - 4.2.1 - Additional stiffness and Rationalised Aero

The simulation is stable for 10000, 5000, 4000, 3000 meters. It diverges for 2000 meters.

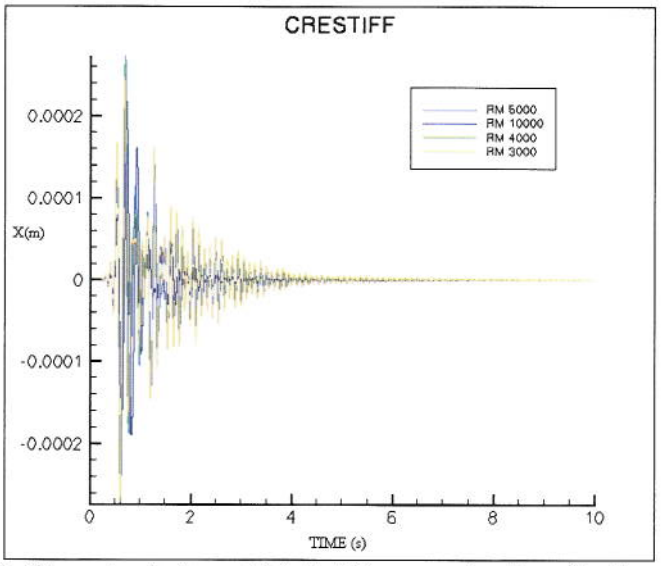


Figure 42: Time simulation: Added stiffness and rationalised aero 1

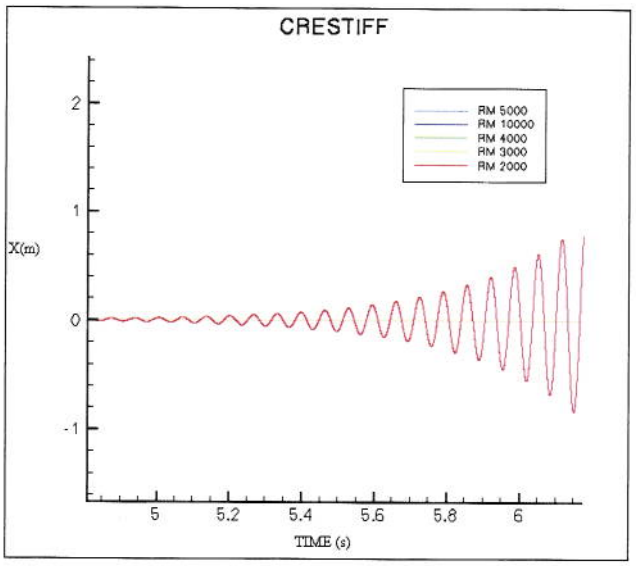


Figure 43: Time simulation: Added stiffness and quasi-steady aero 2

This sort of divergence is due to some poles which are complex conjugates, with a positive real part.

It can be noted that the higher the altitude is, the faster the convergence is.

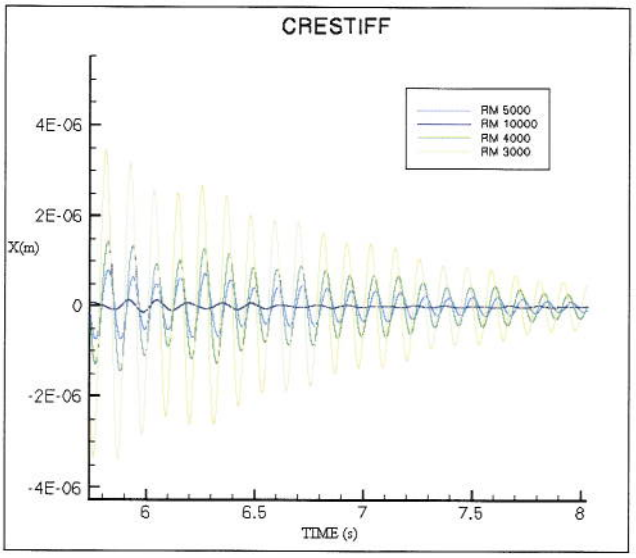


Figure 44: Time simulation: Added stiffness and quasi-steady aero 3

The curves, which converge, have oscillations of the same frequency (the frequency of the mode at the flutter point).

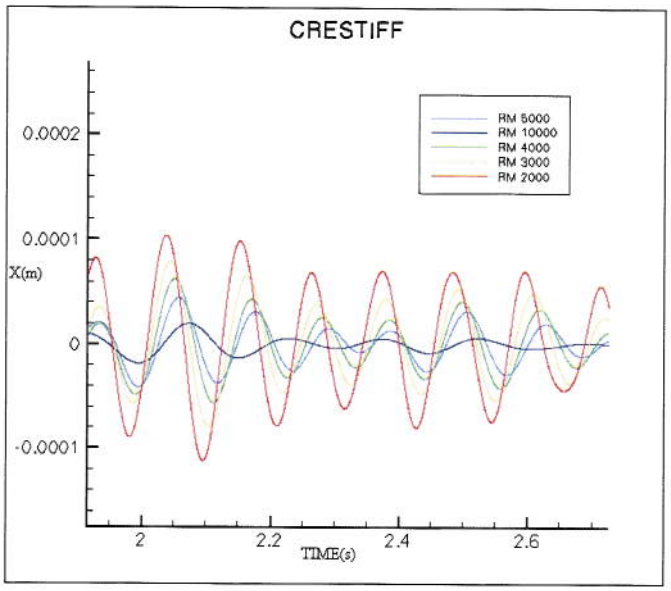


Figure 45: Time simulation: Added stiffness and quasi-steady aero 4

B - 4.2.3 - Comparison between Rationalised Aero and Quasi Steady Aero

This study with a "CRESTIFF" allows a comparison between the two ways of taking the aerodynamic forces into account. At the beginning, the oscillations have roughly the same shapes at an altitude of 10000 meters.

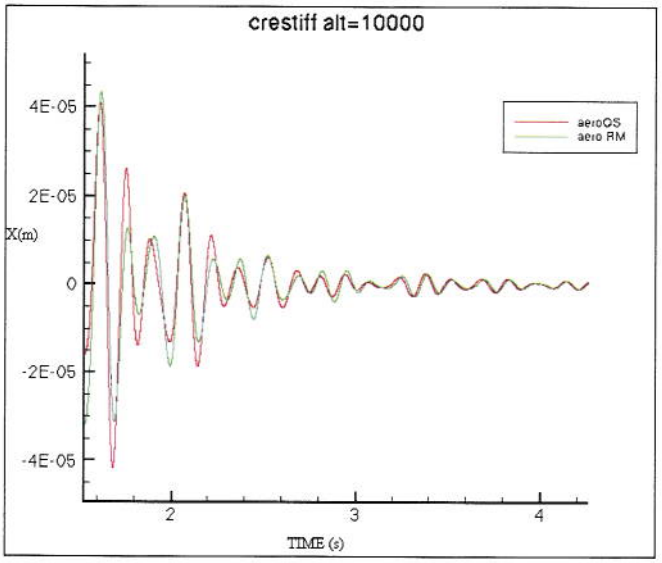


Figure 46: Time simulation: comparison between the two methods of the aero rationalisation

The simulation is more likely to diverge (numerical divergence) when QS aero is used.

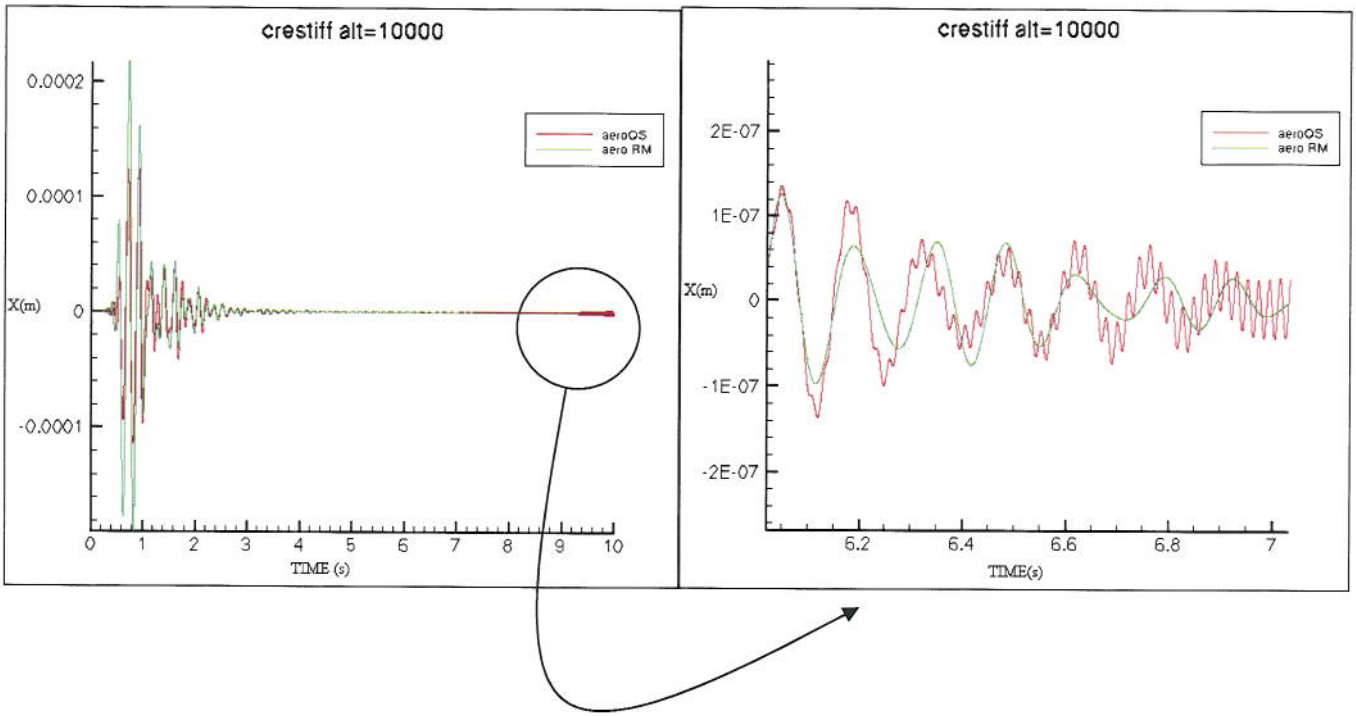


Figure 47: Time simulation: comparison between the two methods of the aero rationalisation

B - 4.2.4 - Non Linear Feedback and Quasi Steady Aero

To avoid numerical divergence, the simulations need better sampling. Therefore we picked 100000 points for 10s. There is a LCO at 3300, 4000 and 5000 meters. They are possible thanks to the non-linearity of the feedback.

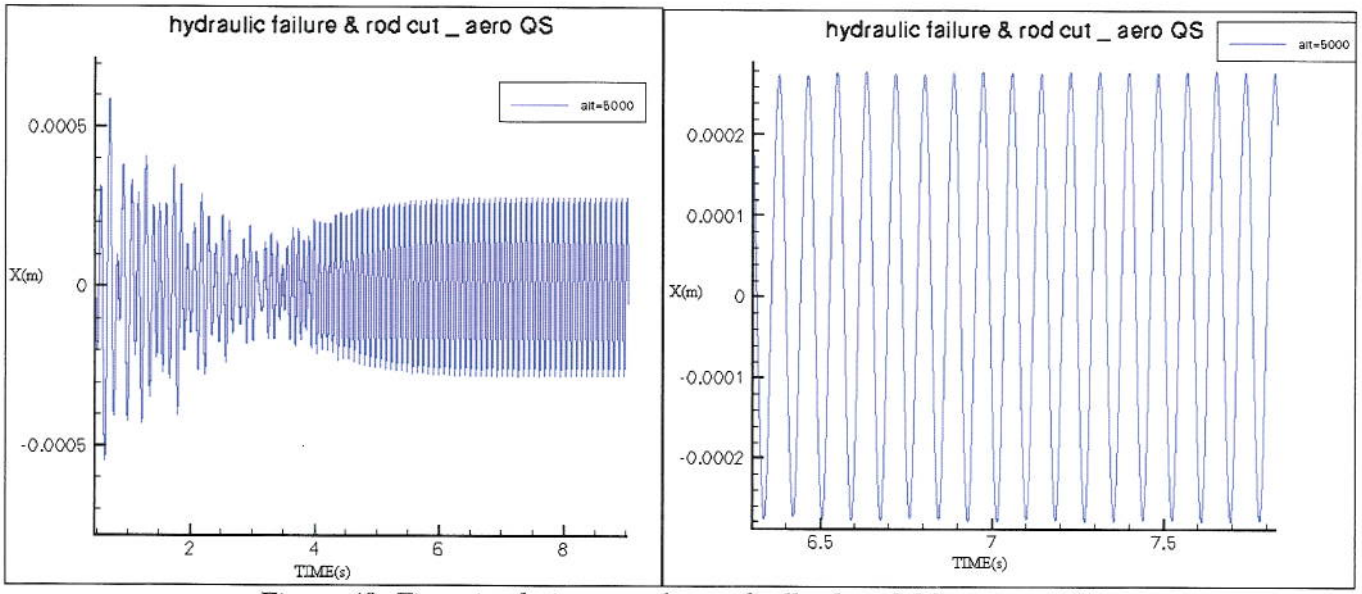


Figure 48: Time simulation: non-linear feedback and QS aero at 5000 m

At 3000 meters, the time response diverges :

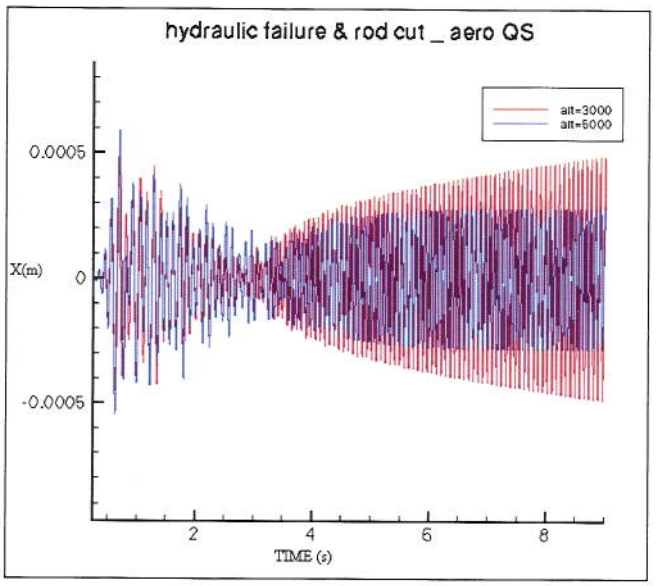


Figure 49: Time simulation: non-linear feedback and QS aero at 3000 and 5000 m

B - 4.2.5 - Non Linear Feedback and Rationalised Aero

After a transcient period, there are LCO of different amplitudes for the different altitudes. They look like sine of the same frequency. A Fourier analysis will confirm this hypothesis.

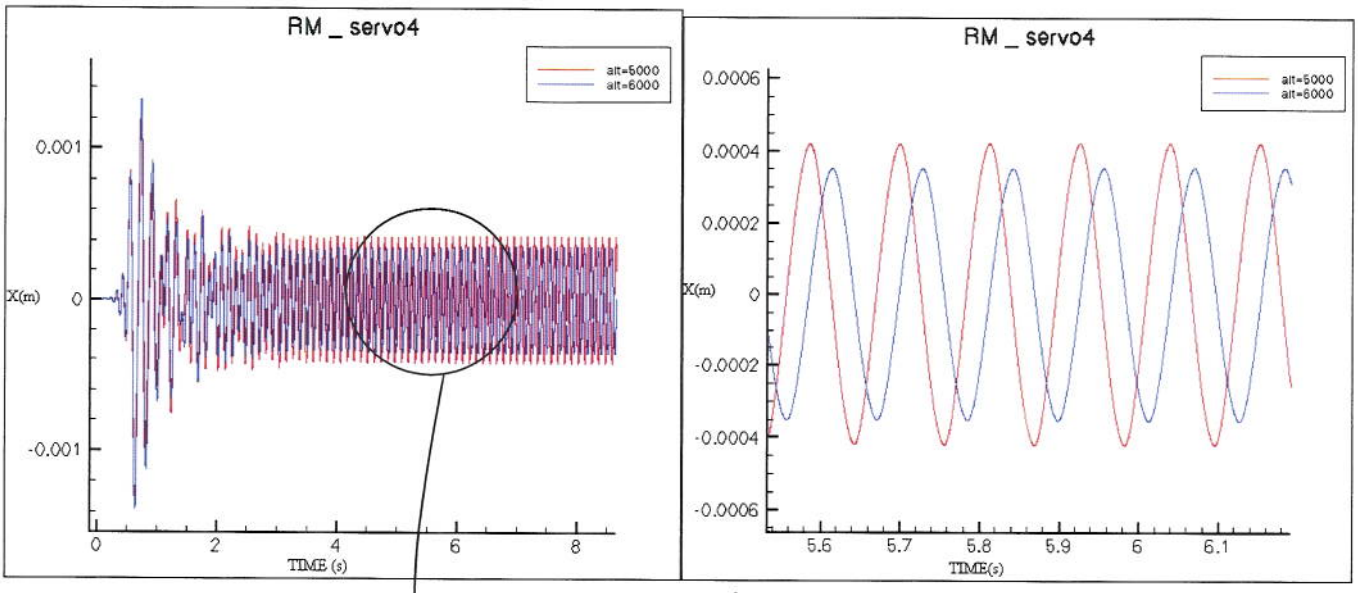


Figure 50: Time simulation: non-linear feedback and rationalised aero at 5000 and 6000m

The Fourier transforms are computed in such a way that the ordinate is the amplitude of the sine. For the altitudes of 5000 meters and 6000 meters the frequency is the same (8.8 Hz).

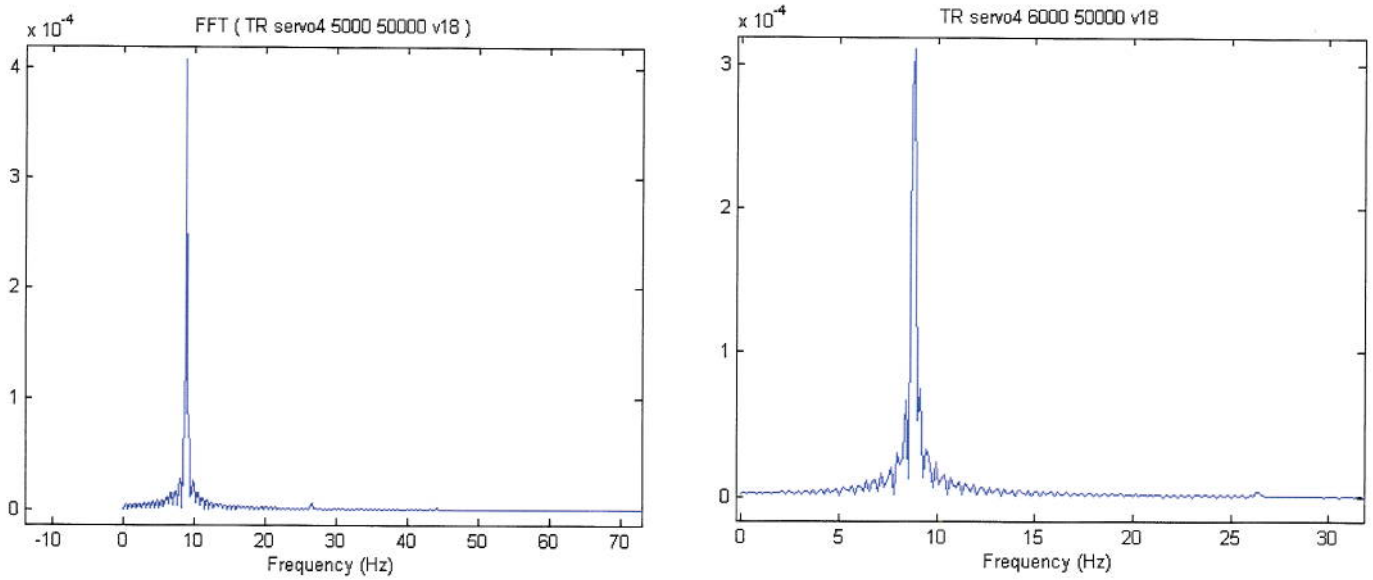


Figure 51: Fourier transforms of time response with non-linear feedback and RM aero at 5000 and 6000 m

```
NFFT = 2^nextpow2(25000); % Next power of 2 from length of y
L=length(TIME); %length of the signal
Fs=50000/10; %sampling frequency
Y = fft(DAXE,NFFT)/L;
f = Fs/2*linspace(0,1,NFFT/2);
figure;plot(f,2*abs(Y(1:NFFT/2)))
title('FFT ( TR servo4 50000 50000 v18 )')
xlabel('Frequency (Hz)')
```

Figure 52: Matlab program for the Fourier transform

A sine can be plotted in the same graph and be the same curve:

$$output = 3.53.10^{-4}.\sin(2\pi.8.8(t-0.017))$$

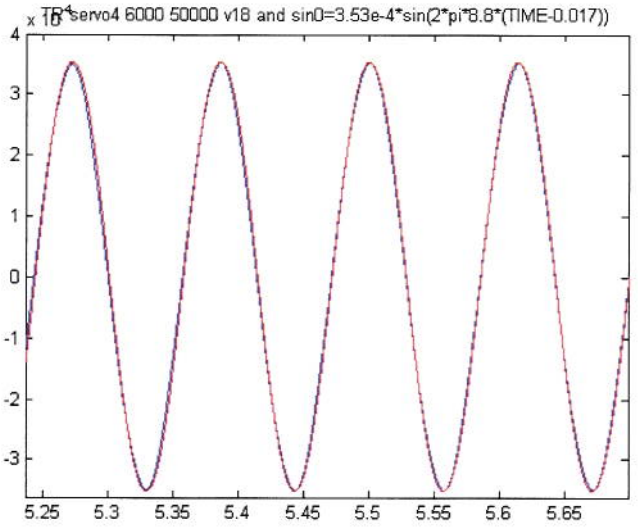


Figure 53: Time simulation: non-linear feedback and RMaero at 6000 m

LCO can be observed at 3000, 4000, 5000, 6000 meters high.

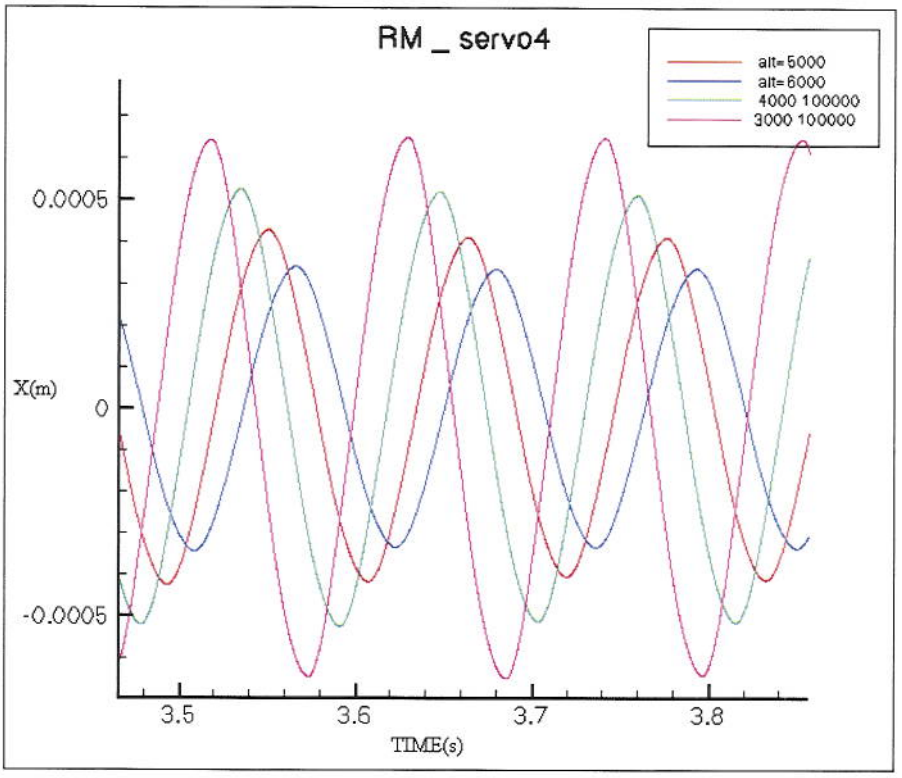


Figure 54: Time simulation: non-linear feedback and RM aero at 3000, 4000, 5000 and 6000 m

B - V – COMPARISON BETWEEN FLUTTER BOUNDARIES AND TIME-MARCHING SCHEMES

On Figure 55, the blue points represent the amplitudes of the LCOs found for simulations at different dynamic pressures. The correspondence between dynamic pressure and altitude has been made with the atmosphere model. The pink points are the flutter boundaries for the simulations made with transfer functions of different altitudes.

The match between flutter boundaries is not as good as expected. Although the variations are the same, there is a gap between the results of the time simulations and the boundaries of the transfer functions.

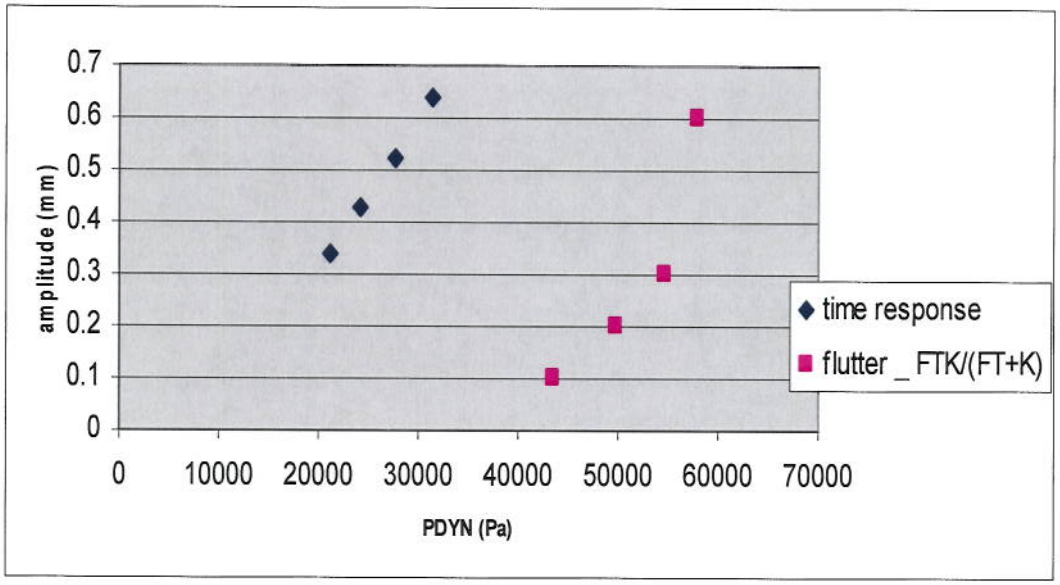


Figure 55:Amplitudes with respect to dynamic pressures

In order to explain the gap between the results obtained in the frequency domain and the results obtained in the time domain, some hypothesis were put forward:

- In the time simulations, the rod has been neglected whereas it has been taken into account for flutter computations.
- The Laplace Domain reduction may explain the differences as it remains an approximation

CONCLUSION

One aim of my project within Dassault was a computing one. Some programs have been created, as the Matlab program modelling the servo-actuator in hydraulic failure, or the C program coding the feedback; and some programs have been improved, as the program using the Laplace domain reductions. Most of this part has been validated with the simulations without feedback. The Matlab program has been validated with data from the Equipments department. The model used for simulations without feedback was a classical model with every rods in it, whereas these rods have been cut in the model I installed for simulations with feedback.

Without feedback, the equivalence between time domain and frequency domain simulations is clearly shown. The time domain simulations find the same flutter boundary than the frequency domain simulations. This result is illustrated by extracting the generalised displacements and plotting the shapes of the plane.

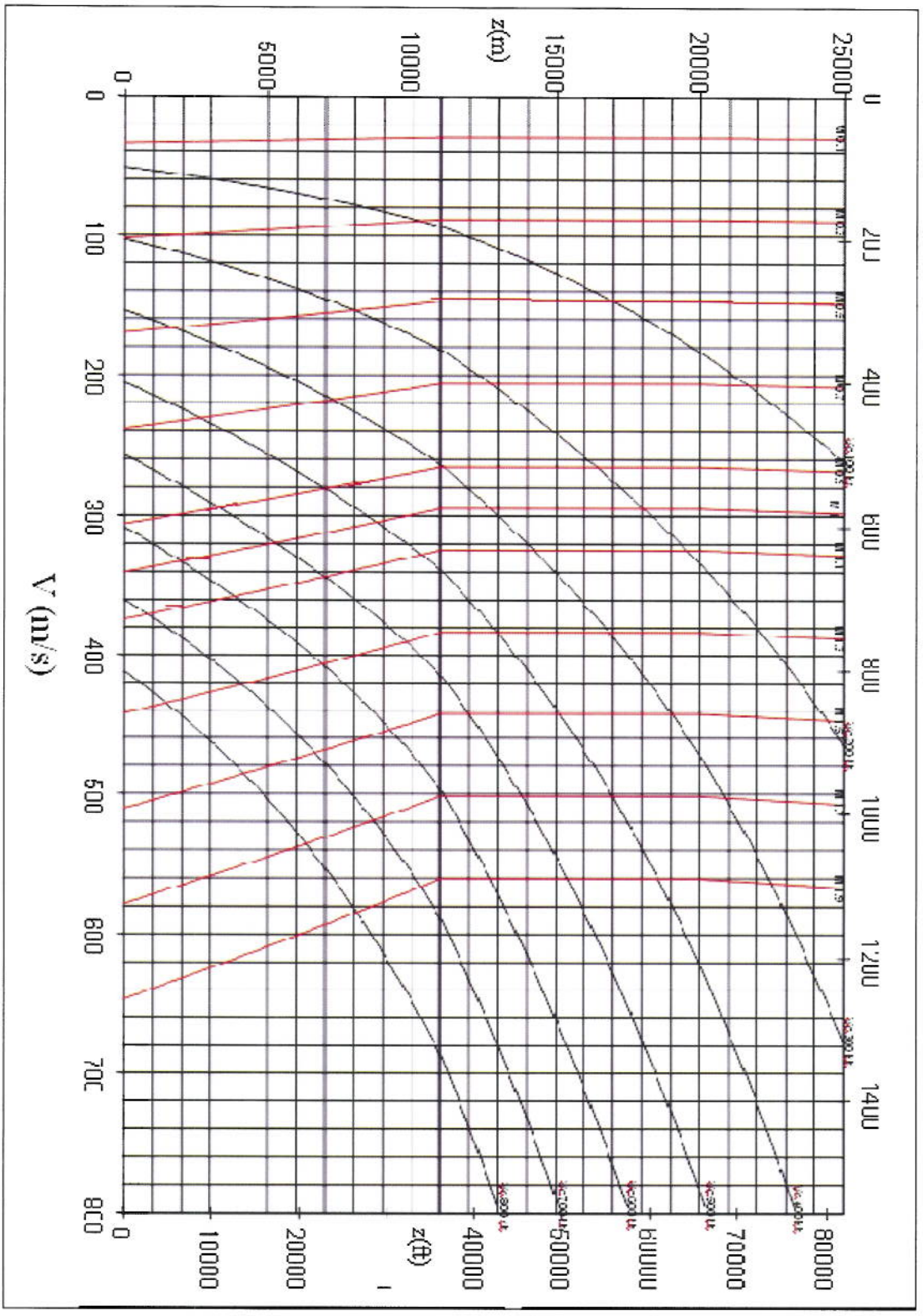
With feedback, the results are not so good. Although the amplitudes of the LCO with respect to the corresponding dynamic pressure have the same shape the amplitudes of the transfer functions with the dynamic pressure of the corresponding flutter curve, there is a gap between flutter points and the dynamic pressure used to compute the LCO of this altitude. This result can be explained by the different hypothesis made all through the project.

To improve the study, this should be further explored. Concerning the risk of errors due to the approximation of the Laplace domain transform, flutter curves may be computed with the aerodynamic forces of the time simulations. Indeed, aerodynamic forces can be sampled and then be expressed again with respect to the frequency. The second source of errors might lie in the fact that the stiffness of the existing rod at the place of the feedback is neglected in time simulations. It is more difficult to put a stiffness in serial in time domain than to transform the transfer function. In its actual form, the Elfini Time Response program can only add a feedback in parallel. One solution could be to change the Matlab program. It might be improved by saving the force and the displacement at each time step and using X-F/K as the displacement for the next time step. Another solution would be to generate another model, where an extremity of the rod where the hydraulic failure model is applied is duplicated. This should permit to put the force resulting from the displacement of the rod in serial with the model of the hydraulic failure.

This project at Dassault Aviation allowed me getting a new experience before the end of my formation. The work was really interesting and largely enhanced my knowledge about aircraft design.

APPENDICES

APPENDIX A - CHEVALIER GRAPH



DASSAULT AVIATION

APPENDIX B - MODES (FREQUENCIES AND MASS)

MODEL

MODE 1 - FREQ = 0.386 - MASS = 5.241E+00
MODE 2 - FREQ = 2.945 - MASS = 4.223E+00
MODE 3 - FREQ = 3.713 - MASS = 3.202E+01

.....

MODE 9 - FREQ = 8.229 - MASS = 1.171E+02
MODE 10 - FREQ = 8.805 - MASS = 1.830E+02
MODE 11 - FREQ = 8.852 - MASS = 1.174E+01
MODE 12 - FREQ = 9.084 - MASS = 2.741E+02
MODE 13 - FREQ = 10.038 - MASS = 5.856E+01
MODE 14 - FREQ = 10.580 - MASS = 1.773E+02
MODE 15 - FREQ = 11.349 - MASS = 7.392E+01
MODE 16 - FREQ = 11.545 - MASS = 1.379E+01
MODE 17 - FREQ = 11.736 - MASS = 2.167E+02
MODE 18 - FREQ = 12.559 - MASS = 9.468E+01
MODE 19 - FREQ = 14.361 - MASS = 2.658E+02
MODE 20 - FREQ = 15.123 - MASS = 1.849E+01
MODE 21 - FREQ = 15.600 - MASS = 1.533E+02
MODE 22 - FREQ = 15.977 - MASS = 1.101E+01

....

WITH "CRESTIFF"

MODE 1 - FREQ = 2.944 - MASS = 1.327E+02
MODE 2 - FREQ = 3.707 - MASS = 1.038E+01
MODE 3 - FREQ = 5.149 - MASS = 3.770E-01

...

MODE 9 - FREQ = 8.714 - MASS = 6.709E-02
MODE 10 - FREQ = 8.820 - MASS = 3.892E+00
MODE 11 - FREQ = 9.079 - MASS = 7.039E+00
MODE 12 - FREQ = 9.949 - MASS = 3.895E-02
MODE 13 - FREQ = 10.576 - MASS = 5.932E-01
MODE 14 - FREQ = 11.345 - MASS = 1.178E+00
MODE 15 - FREQ = 11.540 - MASS = 2.921E-01
MODE 16 - FREQ = 11.736 - MASS = 2.815E+00
MODE 17 - FREQ = 12.558 - MASS = 1.393E+01
MODE 18 - FREQ = 14.359 - MASS = 6.945E-01
MODE 19 - FREQ = 15.122 - MASS = 6.905E-01
MODE 20 - FREQ = 15.453 - MASS = 2.504E-03
MODE 21 - FREQ = 15.609 - MASS = 4.017E-02
MODE 22 - FREQ = 15.981 - MASS = 2.393E-01

...

APPENDIX C - ELFINI CARDS

Appendix C1 - How to add stiffness on the model?

```
%MIXMODEL
 MODEL=${MODEL}
 EIGENSOLVE MONSET-DIM=TRRA
 CRESTIFF NEW-STIFF=ST12
        ADD MONSET=DAXE VALUE=${K}
 EIGENSOLVE STIFF=ST12 MASS="=STD" GDISP=MS12 MONSET-DIM=TRRA
%MIXMODEL
```

Appendix C2 - How to add feedback?

```
%MIXMONSET
 MODEL=${MODEL}
        MONSET=${MONSET_PANNE}
                DEFINE MATRIX=TX1 NODE=18501 TYPE=TX
                DEFINE MATRIX=TX2 NODE=18818 TYPE=TX
                DEFINE MATRIX=TY1 NODE=18501 TYPE=TY
                DEFINE MATRIX=TY2 NODE=18818 TYPE=TY
                DEFINE MATRIX=TZ1 NODE=18501 TYPE=TZ
                DEFINE MATRIX=TZ2 NODE=18818 TYPE=TZ
                ADD SELECT=SER2 COORD=0 0 0 KIND=DISP TYPE=TX NAME=DIFX
                CHANGE NAME=DIFX NEW-COEF = (TX2 -TX1)
                FILTER NAME=DIFX
                WRITE=DIFX
        MONSET=DIFX
                ADD SELECT=SER2 COORD=0 0 0 KIND=DISP TYPE=TY NAME=DIFY
                CHANGE NAME=DIFY NEW-COEF = (TY2 -TY1)
                WRITE=DIXY
        MONSET=DIXY
                ADD SELECT=SER2 COORD=0 0 0 KIND=DISP TYPE=TZ NAME=DIFZ
                CHANGE NAME=DIFZ NEW-COEF = (TZ2 -TZ1)
                WRITE=DXYZ
        MONSET=DXYZ
                ADD SELECT=SER2 COORD=0 0 0 KIND=DISP TYPE=RX NAME=DAXE
                CHANGE NAME=DAXE NEW-COEF=(TZ2 -TZ1)*cos(theta)+(TX2 -TX1)*cos(phi)*sin(theta)+(TY2 -TY1)*sin(phi)*sin(theta)
                 WRITE=DIF1
        MONSET=DIF1
                 FILTER TYPE=RX
                 WRITE=DAXE
%MIXMONSET
```

Appendix C3 - Flutter

```
#define FT30=/Projet/PEA_AEROELAS/CAMILLE/FT30_NOMI_N22 READ-ONLY
#define FT36=/Projet/FNX/CS/scdtasl/DIR_F7X/Camille/Simu/FT/FT36_bis
#define FT31=/Projet/FNX/CS/scrdmmm/FT/WLET/F31_WLET_SAF_SAT READ-ONLY

#define MODEL=MN22
#define MONSET_PANNE = SER2
#define MONSET_RUPT = SERV
#define MODEL = MN22
#define GLOAD=SI80

#define theta=1.319
#define phi=0.453

#define X1_SER2=20.149
#define Y1_SER2=-0.841
#define Z1_SER2=2.289
#define X2_SER2=20.597
#define Y2_SER2=-0.622
```

```
#define Z2_SER2=2.417

#define E=71000E6
#define AREA=100E-6
#define L=#evald(sqrt((X2_SER2-X1_SER2)^2+(Y2_SER2-Y1_SER2)^2+(Z2_SER2-Z1_SER2)^2))
#define K=#evald(E*AREA/L)

%ALLOC
        FT30=${FT30}
        FT31=${FT31}
        FT36=${FT36}
%ALLOC

%READ-FT30
        MODEL=${MODEL}
                AERO GLOAD=${GLOAD}
                MONSET=${MONSET_PANNE}
                 MONSET=${MONSET_RUPT}
                MONSET=TRRA
%READ-FT30

%MIXMODEL
        MODEL=${MODEL}
                EIGENSOLVE FIX=1-6;162-165
                        CREDAMPING
                                MODAL DEGREE=1-150 VALUE=0.015
%MIXMODEL
```

```
*********************************************************
* DEFINITION OF THE MONSET *
*********************************************************

%MIXMONSET
 MODEL=${MODEL}
        MONSET=SER2
                DEFINE MATRIX=TX1_PANNE NODE=18501 TYPE=TX
                DEFINE MATRIX=TX2_PANNE NODE=18818 TYPE=TX
                DEFINE MATRIX=TY1_PANNE NODE=18501 TYPE=TY
                DEFINE MATRIX=TY2_PANNE NODE=18818 TYPE=TY
                DEFINE MATRIX=TZ1_PANNE NODE=18501 TYPE=TZ
                DEFINE MATRIX=TZ2_PANNE NODE=18818 TYPE=TZ
                ADD SELECT=SER2 COORD=0 0 0 KIND=DISP TYPE=TX NAME=DIFX
                CHANGE NAME=DIFX NEW-COEF = (TX2_PANNE-TX1_PANNE)
                FILTER NAME=DIFX
                WRITE=DIFX
        MONSET=DIFX
                ADD SELECT=SER2 COORD=0 0 0 KIND=DISP TYPE=TY NAME=DIFY
                CHANGE NAME=DIFY NEW-COEF = (TY2_PANNE-TY1_PANNE)
                WRITE=DIXY
        MONSET=DIXY
                ADD SELECT=SER2 COORD=0 0 0 KIND=DISP TYPE=TZ NAME=DIFZ
                CHANGE NAME=DIFZ NEW-COEF = (TZ2_PANNE-TZ1_PANNE)
                WRITE=DXYZ
        MONSET=DXYZ
                ADD SELECT=SER2 COORD=0 0 0 KIND=DISP TYPE=RX NAME=DAXE
                CHANGE NAME=DAXE NEW-COEF=(TZ2_PANNE-TZ1_PANNE)*cos(theta)+(TX2_PANNE-
TX1_PANNE)*cos(phi)*sin(theta)+(TY2_PANNE-TY1_PANNE)*sin(phi)*sin(theta)
                WRITE=DIF1
        MONSET=DIF1
                FILTER TYPE=RX
                WRITE=DAXE
%MIXMONSET

%MIXMODEL
 MODEL=${MODEL}
 DEFLOAD GLOAD=GL01
```

```
        FORCE NAME=GL01 MONSET=DXYZ F=cos(${phi})*sin(${theta}) sin(${phi})*sin(${theta}) cos(${theta})
%MIXMODEL

#include DATA

%ALAMO
 READ DATA-FREQ=out/in
 EXPORT FILE=DF.plt FORMAT=TECPLOT VARTYPE=NORM
 EXPORT FILE=DFphase.plt FORMAT=TECPLOT VARTYPE=PHASE
 EXPORT FILE=DFimag.plt FORMAT=TECPLOT VARTYPE=IMAG
%ALAMO

%MIXMODEL
        MODEL=${MODEL}
                DEFFEEDBACK NAME=fct_transfert
                INPUT NAME=in MONSET=DAXE
                OUTPUT NAME=out GLOAD=GL01
                DEFINITION      DATA-FREQ=out/in
%MIXMODEL


************************************************
*  ADDED STIFFNESS TO EF MODEL     *
************************************************
%MIXMODEL
 MODEL=${MODEL}
 EIGENSOLVE MONSET-DIM=TRRA
 CRESTIFF NEW-STIFF=ST12
        ADD MONSET=DAXE VALUE=${K}
 EIGENSOLVE STIFF=ST12 MASS="=STD" GDISP=MS12 MONSET-DIM=TRRA
%MIXMODEL

*********************
*  FLUTTER *
*********************

%FLUTTER
  MODEL=${MODEL} FLUTTER=FLUT COMPUTE-DEGREE=1-100
    AERO-CONDITION KIND=ISO-MACH MACH=0.8 SIZE=200 PDYN-MAX=80000
                STRUCT-CONDITION DAMPING="=STD"
*               STRUCT-CONDITION DAMPING="=STD" STIFF=ST12
                SYSTEM-CONDITION SIZE=200
    AERO KIND=FREQUENCY
      USE ALL
   FEEDBACK NAME=fct_transfert
%FLUTTER

%WRITE-FT36
 MODEL=${MODEL}
        FLUTTER=FLUT TITLE="COURBE DE FLUTTER _ feedback FT 0.0001 _ 1-100" X-AXIS=PDYN
%WRITE-FT36

%ANALYSE-FLUTTER
  MODEL=${MODEL} FLUTTER=FLUT NAMESPACE=FLUT
     EXTRACT-POINT KIND=FLUTTER
*                BUILD-GDISP GDISP=GD00
%ANALYSE-FLUTTER

******** TRACE DU MODE DE FLUTTER
%TMR
        FILE=TMR_FLUT
                ORIGINE-VUE1 = 0 16.67
        ORIGINE-VUE3 = 0 0
                MODEL=${MODEL} GDISP=GD00
                MONSET=TRRA VARTYPE=REAL NAME=REAL COLOR=RED-GREEN LEGEND="$CHARAC
MODELE=MN22" HEIGHT=10 POS-LEGEND=5 5
                MONSET=TRRA VARTYPE=IMAG NAME=IMAG COLOR=RED-GREEN
                TITRE LOCAL = "DEFORMEE VOL DU MODE DE FLUTTER - 0.0005 - M=0.8" HEIGHT=20
                AUTOSCALE=2. PER-FRAME=YES
```

```
                DECALAGE
                        NAME=IMAG DX=30
                        CURVE NAME=REAL
                                #include ../calcul/F7X_2D.dtl
                        CURVE NAME=IMAG
                                #include ../calcul/F7X_2D.dtl
%TMR
```

Appendix C4 - Time Response

```
************************************************
*                TIME-RESPONSE                 *
************************************************

#DEFINE ALT=3300

%TIME-RESPONSE
        MODEL=${MODEL} TR=TRnlfeed
        METHODE=HOUBOLT
                STRUCT-CONDITION DAMPING="=STD"
!               STIFF=ST12
        MODULATION SIZE=${SIZE_T} TIME=${TMAX}
         AERO-CONDITION KIND=FLIGHT
                 DEFINE MACH=0.80 ALT=${ALT}
         AERO MACH=0.8
        INPUT DEGREE-NAME=PR_DLG COEF=1 X=${FUNC} START=0 END=TMAX
                INPUT DEGREE-NAME=PR_DLD COEF=1 X=${FUNC} START=0 END=TMAX
                NLFEEDBACK NAME=nlfeed
%TIME-RESPONSE
```

APPENDIX D - MATLAB PROGRAMS

Appendix D1 - Program used to obtain F(t) from X(t) at each time step.

```
function [yout,F]=servo4(X,X_der,X_der2,h,itime)

i=itime-1;
B=8000e5;k=4.6269E-9;Xm=37.5E-3;S=3.76E-4;M=0.85;
V=S*Xm;F=0;dydt=0; Raid=1.381e7;

if (i==1)
  yout(1)=0;%c le deltaP0
  F0(1)=0;
else
      load('servo_temp.mat','yout','F0');
end

X=-X; X_der=-X_der;

dydt=derivs(yout(i),X_der,B,k,V,S,M);
yout(i+1) = RK(yout(i),dydt,h,X_der,B,k,V,S,M);
F0(i+1) = -M*X_der2+S*yout(i+1);
F=F0(i+1);

save ('servo_temp.mat','yout','F0')

function [yout] = RK(y,dydt,h,INPUT,B,k,V,S,M)
    hh=h*0.5; h6=h/6.0;yt=0;
    yt=y+hh*dydt;
    dyt=derivs(yt,INPUT,B,k,V,S,M);
    yt=y+hh*dyt;
    dym=derivs(yt,INPUT,B,k,V,S,M);
    yt=y+h*dym;
    dym=dym+dyt;
    dyt=derivs(yt,INPUT,B,k,V,S,M);
    yout=y+h6*(dydt+dyt+2.0*dym);

function[dydt] = derivs(y,X_der,B,k,V,S,M)
    dydt=-2.*B.*k./V.*sqrt(abs(y))*sign(y)+2.*B.*S./V*X_der;
```

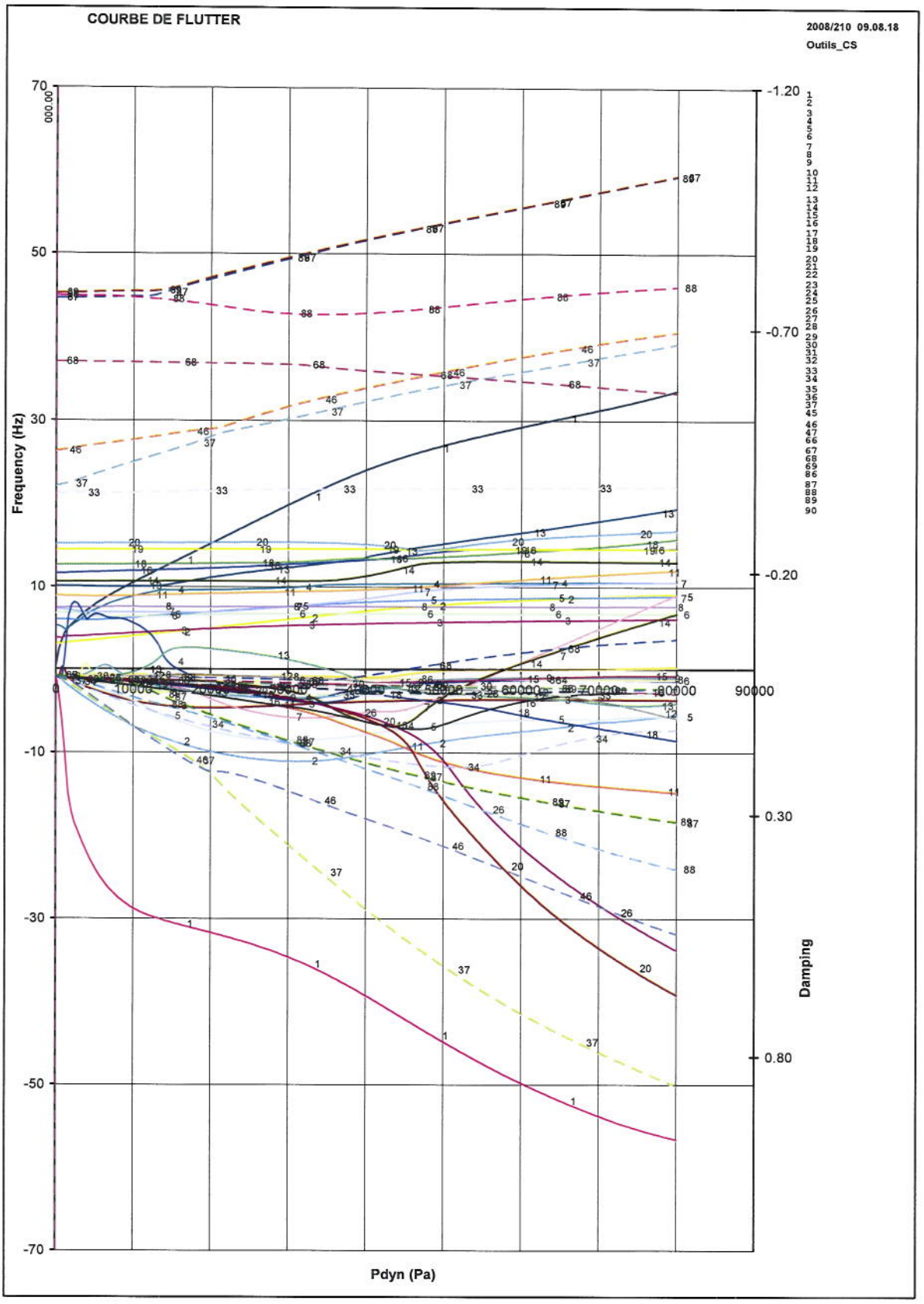
Appendix D2 - Program used to obtain transfer functions

```
function [res]=transfer_function()
close all

%VARIABLES A CHANGER
X0_tab=[ 0.0001 ];
w0=1;

%VARIABLES
tmax=1; N=5000; dt=1/N; df=1/dt;
t = 0:dt:tmax-dt;
B=8000E5;k=4.6269E-9;Xm=37.5E-3;S=3.76E-4;M=0.85;V=S*Xm; h=dt;
freq_tab=1:100;
Raid=1.381E+07

module(1:length(freq_tab))=0;
max_Ap(1:length(freq_tab))=0;
max_Bp(1:length(freq_tab))=0;
mod_serie(1:length(freq_tab))=0;
colors={'r' 'g' 'b' 'm' 'c' 'k'};

%LOOP ON X0
for g=1:length(X0_tab)
X0=X0_tab(g)

    %%LOOP ON FREQUENCIES
    for w=1:length(freq_tab)'
        freq=freq_tab(w);

        %INPUT is the speed
        dep = X0*sin(2*pi*freq*t);
        INPUT(1)=(dep(2)-dep(1))/h;
        for j=2:N
            INPUT(j) = (dep(j)-dep(j-1))/h;
        end

        %INITIALISATION OF deltaP0
        y=0;
        x(1:N)=0;

        %LOOP ON TIME
        for j=1:N
            dydx=derivs(x(j),y,INPUT,B,k,V,S,M,j);
            yout=RK(y,dydx,x,h,INPUT,B,k,V,S,M,j);
            y=yout;
            OUTPUT(j)=yout;
            x(j+1)=x(j)+h;
        end

            F(j) = M*(INPUT(j)-INPUT(j-1))/h+S*OUTPUT(j);
        End

           %COMPUTE F
        F(1)=M*(INPUT(2)-INPUT(1))/h+S*OUTPUT(1);
        for j=2:N
            F(j) = M*(INPUT(j)-INPUT(j-1))/h+S*OUTPUT(j);
        end
```

```
        %TIME APPROACH
        module(w)=max(F);

        %FOURIER TRANSFORM
        fft_F=fft(F);
        fft_dep=fft(dep);
        phase1=(angle(fft_F)-angle(fft_dep));


        freq*tmax+1;
        phase(w)=phase1(floor(freq*tmax+1));
        FT(w)=unwrap(module(w)/X0*exp(i*phase(w)));

    end

    modFT=module/X0;
    phaseRAD=unwrap(phase);
    modFT_w0=module(w0)/X0;
    phase_w0=phase(w0*tmax)*180/pi;
    FT

    %Graph of transfer function
    hold on
    subplot(2,1,1); plot(freq_tab,modFT)
    title('transfert functions '); xlabel('frequencies (Hz)')

end

legend('X0=0.001');text(w0,modFT_w0,'\leftarrow module(w0,X0)/X0',...
     'HorizontalAlignment','left')
hold on
plot(w0,modFT_w0,'marker','square','markersize',6,...
                 'markeredgecolor','k','markerfacecolor',[.6 0 .6],...
                 'linestyle','-','color','r','linewidth',2); hold off

subplot(2,1,2); plot(freq_tab,phase)

figure; subplot(2,1,1); plot(freq_tab,real(FTajust_serie))
subplot(2,1,2); plot(freq_tab,imag(FTajust_serie))



%_____________________________________________________________________

%SUBROUTINES

function [yout] = RK(y,dydx,x,h,INPUT,B,k,V,S,M,j)
    hh=h*0.5; h6=h/6.0;yt=0;
    xh=x(j)+hh;
    yt=y+hh*dydx;
    dyt=derivs(xh,yt,INPUT,B,k,V,S,M,j);
    yt=y+hh*dyt;
    dym=derivs(xh,yt,INPUT,B,k,V,S,M,j);
    yt=y+h*dym;
    dym=dym+dyt;
    dyt=derivs(x+h,yt,INPUT,B,k,V,S,M,j);
    yout=y+h6*(dydx+dyt+2.0*dym);

function[dydx] = derivs(x,y,INPUT,B,k,V,S,M,j)
   dydx=-2.*B.*k./V.*sqrt(abs(y))*sign(y)+2.*B.*S./V*INPUT(j);
```

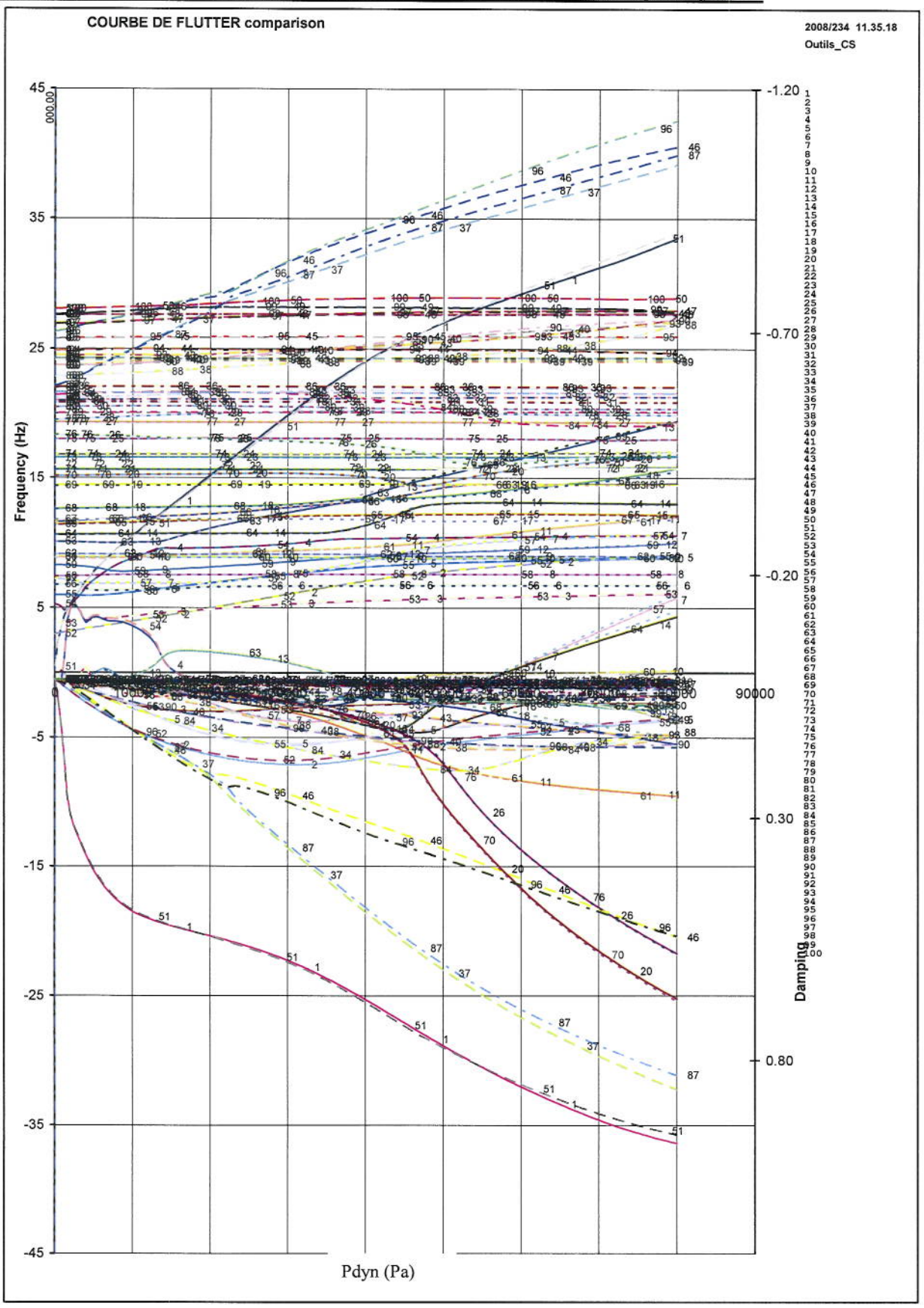
APPENDIX E - FLUTTER

Appendix E1 - Model



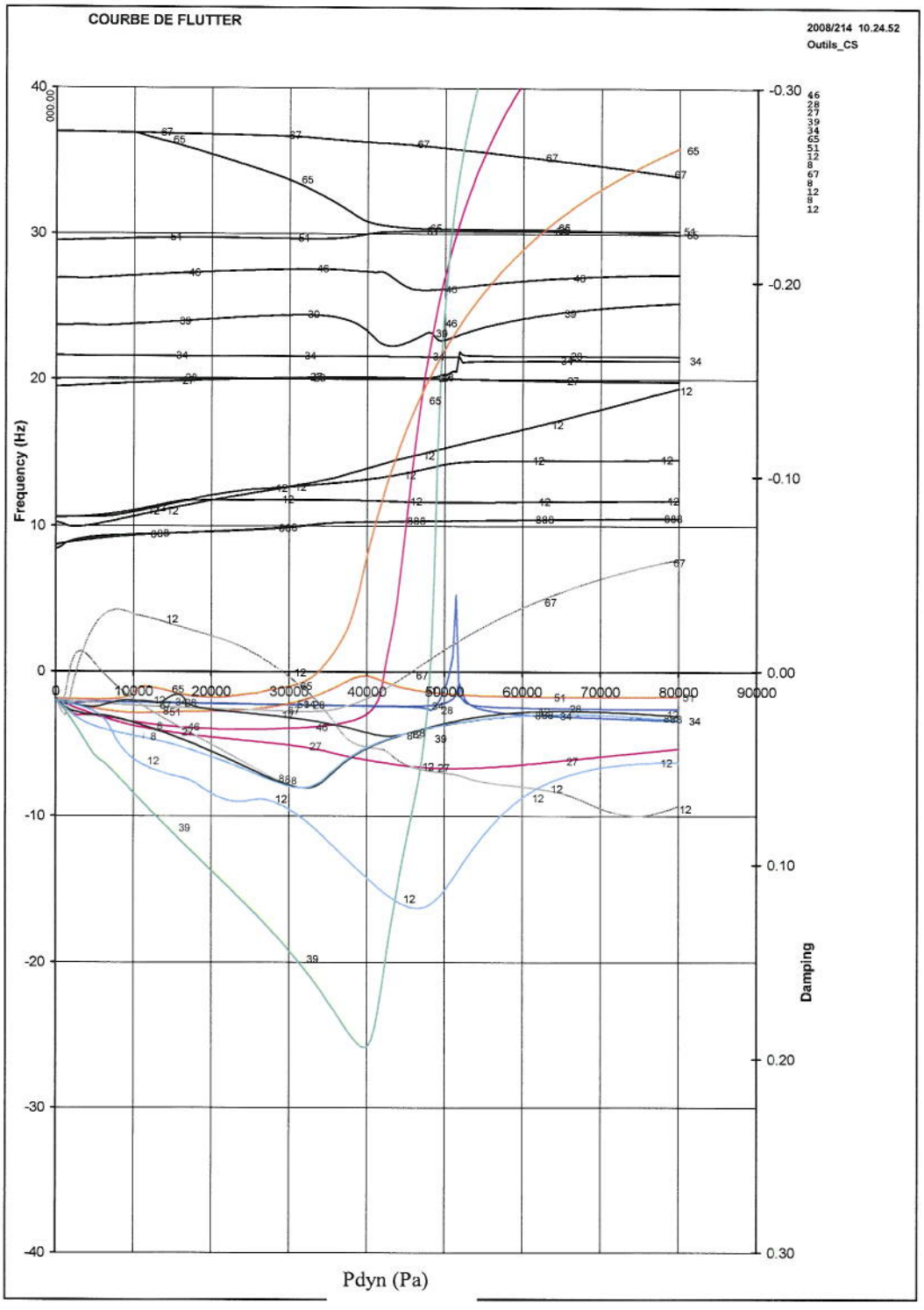
DASSAULT AVIATION

Appendix E2- Comparison between rationalised aero and frequency aero



DASSAULT AVIATION

Appendix E3- Flutter curves for different transfer functions



Legend

- 0.25 K
- 0.5K
- K
- Transfer function with the amplitude $\delta = 0.1mm$
- Transfer function with $\delta = 0.2mm$
- Transfer function with $\delta = 0.3mm$

APPENDIX F - ELFINI SUBROUTINE

```
/*
 * ----------------------------------------------------------------
 *
 * TR_Feedback
 *
 * Traitement des CS_Feedback
 *
 * Copyright (c) 2003-2004 Dassault Aviation
 *
 * $Id$
 *
 * $Log$
 *
 * ----------------------------------------------------------------
 */

#include "engine.h"

/* Declaration des variables */

static int sizefb;
static CS_Feedback * fb = (CS_Feedback *) NULL;
static CS_NLFeedback * nlfb = (CS_NLFeedback *) NULL;
/*static CS_Monset **tmp_monset = (CS_Monset**) NULL;*/
/*static double **tmp_monset = (double**) NULL;*/
/*static CS_Model * model = (CS_Model *) NULL;*/
static Engine * ep;
static CS_Monset * tmp_monset = (CS_Monset*) NULL;
static FILE * file_buffer = (FILE*) NULL;

/* Declaration des fonctions public */
static int InitFeedback(DataList * lineList,CS_Model * model,CS_TR * tr);
static void MKCFeedback(void);
static void ClearFeedback(void);

/*
 *----------------------------------------------------------------
 * InitFeedback : Initialisation d'un Feedback
 *----------------------------------------------------------------
 */
int InitFeedback(DataList * inList,CS_Model * model,CS_TR * tr)
{
        char * texterror = "Error in InitFeedback";
        char error[1024];
         int i,j,nfb,ordre;
        int * ipole = (int*) NULL;
        char * tmp_s = (char*) NULL;
        char * file = (char*) NULL;
        DataList * sysList = (DataList*) NULL;
        DataList * matList = (DataList*) NULL;
        DataList * tmpList = (DataList*) NULL;
        DataList * fbList = (DataList*) NULL;
        CS_Feedback * new_fb = (CS_Feedback*) NULL;
        CS_NLFeedback * new_nlfb = (CS_NLFeedback*) NULL;


        /* Récupération de la DataList SYSTEM */
        GetDataListFromDataList(tr->infoList,"SYSTEM",&sysList);
        fbList = DataListCopy(inList);
```

```
printf("Traitement de InitFeedback\n\n");
/* Récupération du Feedback */
PrintDataList(inList);
for(tmpList=inList;tmpList;tmpList=tmpList->next)
{
        PrintDataList(inList);
        if(IsDataList(tmpList,"FEEDBACK"))
        {
                if(GetDataStringFromDataList(tmpList,"NAME",&tmp_s))
                        PrintError("Error reading NAME on FEEDBACK line")

                printf("1\n");
                if(GetCS_FeedbackFromCS_Model(model,tmp_s,&new_fb))
                        PrintError("Error in GetCS_FeedbackFromCS_Model")
                if(tmp_s) DLfree(tmp_s); tmp_s=(char *) NULL;

                if(!new_fb)
                {
                        sprintf(error, "Error: fb is empty\n");
                        PrintError(error)
                }
        }

        if(IsDataList(tmpList,"NLFEEDBACK"))
        {
                if(GetDataStringFromDataList(tmpList,"NAME",&tmp_s))
                        PrintError("Error reading NAME on NLFEEDBACK line")
                if(GetCS_NLFeedbackFromCS_Model(model,tmp_s,&new_nlfb))
                        PrintError("Error in GetCS_NLFeedbackFromCS_Model")
                if(tmp_s) DLfree(tmp_s); tmp_s=(char *) NULL;
                if(!new_nlfb)
                {
                        sprintf(error, "Error: nlfb is empty\n");
                        PrintError(error)
                }

                if(IsDataOnDataList(tmpList,"FILE-BUFFER"))
                {
                        if(GetDataFileoutFromDataList(tmpList,"FILE-BUFFER",&file))
                                PrintError("Error reading FILE-BUFFER on NLFEEDBACK line")
                                DeleteDataOnDataList(tmpList,"FILE-BUFFER");
                        if(!(file_buffer = fopen(file,"w")))
                        {
                                sprintf(error,"Error writing file %s",file);
                                PrintError(error)
                        }
                        if(file) DLfree(file); file=(char *) NULL;
                }
        }
}


if(new_nlfb->mf){
        if (!(ep = engOpen("\0"))) {
        sprintf(error, "\nCan't start MATLAB engine\n");
        PrintError(error)}
}


/* Combinaison et projection sur le GDISP */
if(new_fb){
        new_fb = CombineCS_FeedbackWithCS_GDISP(new_fb,gdisp);
        if(!new_fb) PrintError("Error in CombineCS_FeedbackWithCS_GDISP")
        }


if(new_nlfb){
        new_nlfb = CombineCS_NLFeedbackWithCS_GDISP(new_nlfb,gdisp);
        if(!new_nlfb) PrintError("Error in CombineCS_NLFeedbackWithCS_GDISP")
```

```
		}

	if(new_nlfb->mf)
	{
		tmp_monset=CS_MonsetAlloc("tmp_monset");
		if(!tmp_monset) PrintError("Error in CS_MonsetAlloc")
		if(CS_MonsetMemAlloc(tmp_monset,new_nlfb->ninput,3*new_nlfb->monset->sizebase,0))
		PrintError("Error in CS_MonsetMemAlloc")
		for(i=0;i<new_nlfb->ninput;i++)
		{
			for(j=0;j<new_nlfb->monset->sizebase;j++)
			{
			MAT(tmp_monset->r_value,i,j+new_nlfb->ikind[i]*new_nlfb->monset->sizebase,3*new_nlfb-
			>monset->sizebase)=MAT(new_nlfb->monset->r_value,i,j,new_nlfb->monset->sizebase);

			}
		}
	}


	/* Ajout dans la liste */

	if(IsCS_FeedbackOnChain(fb,new_fb->name)||IsCS_NLFeedbackOnChain(nlfb,new_nlfb->name))
	{
		sprintf(error,"Feedback already defined");
		PrintError(error)
	}
	if(new_fb) {
		AddCS_FeedbackOnChain(&fb,new_fb);
		if(!fb) PrintError(error)}
	if(new_nlfb){
		AddCS_NLFeedbackOnChain(&nlfb,new_nlfb);
		if(!nlfb) PrintError(error)}


	/* Introduction de nouvelles variables */
	if(nlfb->iprig[0]!=0){
		 sizem += nlfb->noutput;
		 sizemax += nlfb->noutput;}
	if(fb->iprig[0]!=0) {
		sizem += fb->noutput;
		sizemax += fb->noutput;}

	printf("fin de InitFeedback\n");

	return 0;
	ERR_OUT:
		return 1;
}

/*
 *-----------------------------------------------------------------
 * MKCFeedback : Modification des matrices MKC
 *-----------------------------------------------------------------
 */
void MKCFeedback(void)

{
	int i,j,p;
	if(nlfb && nlfb->iprig[0]!=0)
	{

		double v0,v1,v2;
		v0 = pdyn; v1 = v0/speed; v2 = v1/speed;

		for(i=0;i<nmode;i++)
		{
			for(j=0;j<nlfb->noutput;j++) {
				MAT(M,i,j+sizem-ndeg,sizem) = MAT(mass[0]->value,i,nlfb->iprig[j],ndeg);
```

```
                                MAT(K,i,j+sizem-ndeg,sizem) = MAT(stiff[0]->value,i,nlfb->iprig[j],ndeg);
                                if(damp) MAT(C,i,j+sizem-ndeg,sizem) = MAT(damp[0]->value,i,nlfb-
                                >iprig[j],ndeg);
                        }
                }

                for(i=0;aeror && i<nlfb->noutput;i++) MAT(M,i+ndeg+aeror->npole,i+nmode+aeror->npole,sizem) = 1;
                for(i=0;!aeror && i<nlfb->noutput;i++) MAT(M,i+ndeg,i+nmode,sizem) = 1;

        }
}


/*
 *---------------------------------------------------------------
 * ComputeLoadNLFeedback : Construction des loads pour la
 * modélisation des systemes non linéaire
 *---------------------------------------------------------------
 */
int ComputeLoadNLFeedback()
{
        #define  BUFSIZE 256
        char buffer[BUFSIZE+1];
         char * texterror = "Error in ComputeLoadNLFeedback";
        char error[1024];
        int i,j,k,size_fun,size_input,size_path,NStructElems;
        char * INPUT = (char*) NULL ,*PATH = (char*)NULL,*FUNCTION = (char*)NULL;
        mxArray *OUTPUT = (mxArray *) NULL;
        double *pr,*pi;
        double * tmp_tr=(double*) NULL;
        double * input=(double*) NULL;

        CS_GDISP * tmp_GDISP = (CS_GDISP*) NULL;
        CS_GDISP * GDISP_SORTIE = (CS_GDISP*) NULL;

        if(nlfb)
        {
                int i,j,nfb;
                CS_NLFeedback * nlfbi = (CS_NLFeedback*) NULL;

                buffer[BUFSIZE]='\0';
                engOutputBuffer(ep, buffer, BUFSIZE);


                /* Allocation des tableaux */
                input = (double *) malloc((nlfb->ninput)*sizeof(double));
                if(!input) PrintErrMem


                /* Allocation des GDISP de travail */
                tmp_GDISP=CS_GDISPAlloc("tmp_GDISP");
                if(!tmp_GDISP) PrintError("Error in CS_GDISPAlloc")
                if(CS_GDISPMemAlloc(tmp_GDISP,1,3*ndeg,0)) PrintError("Error in CS_GDISPMemAlloc")


                /* Remplissage du GDISP de travail*/
                for(j=0;j<ndeg;j++){
                        MAT(tmp_GDISP->r_value,0,j,tmp_GDISP->sizebase)=MAT(x,itime-1+ordre,j,sizemax);
                }
                if(itime==1)
                {
                        for(j=ndeg;j<2*ndeg;j++){
                        MAT(tmp_GDISP->r_value,0,j,tmp_GDISP->sizebase)=mcoef[1]/(dt*dt)*MAT(x,itime-
                        1+ordre,j-ndeg,sizemax)
                        +mcoef[4]/(dt*dt)*MAT(x,itime-2+ordre,j-ndeg,sizemax)+mcoef[7]/(dt*dt)*MAT(x,itime-
                        3+ordre,j-ndeg,sizemax);

                        MAT(tmp_GDISP->r_value,0,j+ndeg,tmp_GDISP->sizebase)=mcoef[0]/(dt*dt)*MAT(x,itime-
                        1+ordre,j-ndeg,sizemax)
```

```
            +mcoef[3]/(dt*dt)*MAT(x,itime-2+ordre,j-ndeg,sizemax)+mcoef[6]/(dt*dt)*MAT(x,itime-
            3+ordre,j-ndeg,sizemax);
            }
}
else
{
            for(j=ndeg;j<2*ndeg;j++){
            MAT(tmp_GDISP->r_value,0,j,tmp_GDISP->sizebase)=mcoef[1]/(dt*dt)*MAT(x,itime-
            1+ordre,j-ndeg,sizemax)
            +mcoef[4]/(dt*dt)*MAT(x,itime-2+ordre,j-ndeg,sizemax)+mcoef[7]/(dt*dt)*MAT(x,itime-
            3+ordre,j-ndeg,sizemax)
            +mcoef[10]/(dt*dt)*MAT(x,itime-4+ordre,j-ndeg,sizemax);

            MAT(tmp_GDISP->r_value,0,j+ndeg,tmp_GDISP->sizebase)=mcoef[0]/(dt*dt)*MAT(x,itime-
            1+ordre,j-ndeg,sizemax)
            +mcoef[3]/(dt*dt)*MAT(x,itime-2+ordre,j-ndeg,sizemax)+mcoef[6]/(dt*dt)*MAT(x,itime-
            3+ordre,j-ndeg,sizemax)
            +mcoef[9]/(dt*dt)*MAT(x,itime-4+ordre,j-ndeg,sizemax);
            }
}

/* Remplissage des inputs*/
if(tmp_monset->sizebase==3*ndeg )
{
            CS_Monset * input_monset = (CS_Monset *) NULL;
            input_monset=CombineCS_MonsetWithCS_GDISP(tmp_monset,tmp_GDISP);
            if(!input_monset) PrintError("Error in CombineCS_MonsetWithCS_GDISP");
            for(i=0;i<nlfb->ninput;i++) input[i] = input_monset->r_value[i];

            if(input_monset) CS_MonsetFree(&input_monset);
}
else{
            sprintf(error,"tmp_monset->sizebase %d is not equal to 3*ndeg %d\n",tmp_monset-
            >sizebase,ndeg);
            PrintError(error)
}

/* Entrées Matlab */
size_path=10+strlen(nlfb->mf->path);
PATH=(char *) malloc(size_path*sizeof(char));
sprintf(PATH,"cd('%s');",nlfb->mf->path);
size_input=15+15*nlfb->ninput;
INPUT=(char *) malloc(size_input*sizeof(char));
sprintf(INPUT,"INPUT=[%15.7E,%15.7E]",input[0],input[1]);
size_fun=strlen(nlfb->mf->function)+40+15*nlfb->ninput;
FUNCTION=(char *) malloc(size_fun*sizeof(char));
sprintf(FUNCTION,"[yout,F]=%s(%15.7E,%15.7E,%15.7E,%15.7E,%d);",nlfb->mf-
>function,input[0],input[1],input[2],dt,itime);
if(file_buffer) fprintf(file_buffer,"%s",FUNCTION);
engEvalString(ep,PATH);
if(file_buffer)
{
            if(!(buffer+2=='\0') ) fprintf(file_buffer,"\t%s\n",buffer+2);
}
engEvalString(ep,FUNCTION);
if(file_buffer)
{
            if(!(buffer+2=='\0') ) fprintf(file_buffer,"\t%s\n",buffer+2);
}
OUTPUT=engGetVariable(ep,"F");
pr=mxGetPr(OUTPUT);
NStructElems = mxGetNumberOfElements(OUTPUT);
if(OUTPUT) mxFree(OUTPUT); OUTPUT=(mxArray *) NULL;
if(NStructElems!=nlfb->noutput) PrintError("Error in Matlab (Invalid output)");

/* Résultat sys_load*/
if(nlfb->gload){
            if(!sys_load) sys_load = malloc(ndeg*sizeof(double));
            if(!sys_load) PrintErrMem
```

```
                        for(j=0;j<ndeg;j++){
                                sys_load[j]=0;
                                for(i=0;i<NStructElems;i++){
                                        sys_load[j] += pr[i]*MAT(nlfb->gload->r_value,i,j,nlfb->gload->sizebase);
                                        /* printf("sys_load[%d] %f\n",j,sys_load[j]);*/
                                }
                        }
                }

                /* Sortie en gdisp*/
                if(nlfb->iprig[0]!=0){

                        if(!sys_disp) sys_disp =  malloc((nlfb->noutput)*sizeof(float*));
                        if(!sys_disp) PrintErrMem

                        for (j=0;j<nlfb->noutput;j++)sys_disp[j]=pr[j];
                }
                if(INPUT) free(INPUT);INPUT = (char *) NULL;
                if(PATH) free(PATH);PATH = (char *) NULL;
                if(FUNCTION) free(FUNCTION);FUNCTION = (char *) NULL;
                if(tmp_tr) free(tmp_tr); tmp_tr = (double*) NULL;
                if(input) free(input); input = (double*) NULL;
                if(pr) free(pr); pr = (double*) NULL;
                if(tmp_GDISP) CS_GDISPFree(&tmp_GDISP);tmp_GDISP = (CS_GDISP*) NULL;
                if(GDISP_SORTIE) CS_GDISPFree(&GDISP_SORTIE);GDISP_SORTIE = (CS_GDISP*) NULL;
        }

        return 0;
  ERR_OUT:
    return 1;
}

/*
 *---------------------------------------------------------------
 * ClearFeedback : Suppression des Feedback
 *---------------------------------------------------------------
 */
void ClearFeedback(void)
{

        /* Fermeture de Matlab */
        engClose(ep);

        if(tmp_monset) CS_MonsetFree(&tmp_monset); tmp_monset=(CS_Monset *) NULL;
        CS_FeedbackChainFree(&fb);

}
```

TABLE OF REFERENCES

[1] Internal website of Dassault Aviation
[2] Ballhaus, W. F. and Goorjian, P. M., "Implicit Finite Difference Computations of Unsteady Transonic Flows About Airfoils", AIAA Journal 15 (1977), 1728-35
[3] SupAero course notes, by Jean-Luc BOIFFIER – Dynamique de Vol 2
[4] Elfini v8 documentation
[5] Finite Difference approximation of ODEs, Dr Peiro, Imperial College London
[6] Numerical Recipes in C, The Art of Scientific Computing, Second Edition, William H. Press, Saul A. Teukolsky, William T. Vettering, Brian P. Flanney, Cambridge University Press
[7] Courses notes, Spring 2008, by R. Palacios, Imperial College London.
[8] *resultats_simu.doc by Jerôme Labi, Dassault-Aviation*
[9] Fourier Analysis of Time Series, by Dr. R. L. Herman, UNC Wilmington.
[10] MDO Technology needs in aeroelastic structural design, by H.G. Hönlinger, J. Krammer, M. Stettner, German Aerospace Center (DLR) (Göttingen, Germany), Daimler-Benz Aerospace AG (Munich, Germany)
[11] Studies in Nonlinear Aeroelasticity, Earl H. Dowell, Marat Ilgamov

A part of the knowledge about Elfini has been transmitted in oral. For this reason, there is no source for parts as "Laplace Domain Reduction of aerodynamic forces" or "PK-method" for example.