14

Unsteady Differentiation of Aerodynamic Coefficients: Methodology and Application

Carlo Necci and Nicola Ceresola
Alenia Aeronautica SpA
Italy

1. Introduction

The evaluation of aerodynamic forces and of the relevant coefficients is a fundamental task in aircraft design. A number of numerical methods have been investigated and some of them, the most reliable and cost effective, are daily applied in the Aerospace field for design, development and research purposes. Conversely, not so many methods have been considered reliable enough for computing aerodynamic derivatives, which are fundamental to evaluate sensitivities to aerodynamic or shape parameters. Despite the efforts deployed by former researchers, the basic mathematical complexity to evaluate derivatives strongly limits the application of several methods. Numerical differentiation is surely a task widely studied and most of the schemes which have been developed are based on the finite differences method. Finite differences are flexible and easily applicable, but they are just an algebraic transposition of the Taylor series expansion, and as such they are an approximation. As a matter of fact, the attempt to apply the mathematical theory of differentiation to computer programs has to be handled differently. Furthermore, while steady derivatives allow quantifying different sensitivities when flying in steady condition, dynamic derivatives are supposed to provide useful information on what happens in unsteady condition, but this introduces the extra complexity of the time dependency. It is hence again evident that the need of a solid mathematical basis is crucial when dealing with these kinds of problems.
A possible approach is the experimental use of physical models in wind tunnels; data acquired while simulating different steady flight conditions can be interpolated to compute steady derivatives. While unsteady derivatives can be quantified by both mathematically handling steady values or moving the model in the wind tunnel and then managing the data streams that have been acquired. The drawback of this method is its high cost, especially when complex motions are tested. Furthermore, experimental work is complex, since setup and test implementation require a wide experience both in preparing the system setup and in understanding results. An extensive work has been done so far in this field; some references (Almosnino, 1994; Altun & Iyigun, 2004; Anderson & Newmann, 1999; Anderson et al., 1984; Guglieri & Quagliotti, 1993; Guglieri et al., 1993; Murphy & Klein, 2001) can give just an idea of the huge effort so far deployed.
But in the latest years numerical approaches have been extensively developed and employed to fulfil the same results in a cheaper and faster way. The simplest algebraic way

to evaluate derivatives is, as said, the use of finite differences (Anderson et al., 1984) which simply requires evaluating the function of interest at two or more nearby states. This method can also be used as a validation technique and it is attractive since it does not require any modifications to existing source codes but it has the drawback to be sensitive to cancellation errors. It is important using a small step size to approximate the mathematical derivative, but the issue of subtractive cancellation of the terms in the numerator will always be experienced. An alternative to this is the Complex Variable Differentiation method (CVD), suggested by Anderson and other Authors (Anderson et al., 1999). Conceptually, CVD uses a series expansion of functions and a complex perturbation of terms, so in effect splitting up real and imaginary parts of the relevant function. For the first derivative, this method does not require subtracting the values computed in different points, so avoiding errors connected to cancellation; it has been successfully tested in computation of aero-structure sensitivity derivatives (Newmann et al., 1998) and works adequately. But it has the drawback to double the memory required for computation, since for each function both the real and the corresponding complex component have to be managed; runtime cost increases by a factor close to three.

Another possibility consists in using codes based on panel method. This method bases its applicability on the conceptual simplicity of panels but it showed to be unreliable in some particular conditions (Almosnino, 1994). In 2004 Green published a good work (Green et al., 2004) focused on the possibility to computationally separate dynamic and stability derivatives, which during the experimental testing are measured in combination. Their study was furthermore interesting for the introduction of the automatic differentiation technique, shortly identified as AD. This was the beginning for a new branch of numerical research, which gave remarkable results. In Europe, the French researcher Laurent Hascoet carried out a study that finally was formalised in a public paper (Hascoet, 2005) focused on parallelisation and differentiation techniques based on a strong mathematical background and algebraic adherence to the rules of mathematical analysis. The work was then formalised in a more extended research context when the French 'Institute National de Recherche en Informatique et en Automatique' (INRIA) developed a tool (Tapenade) completely devoted to differentiation of Fortran source codes (Hascoet & Pascual, 2004).

Industrial applications of Tapenade were carried out across Europe by different Aerospace Companies; it was showed that the AD approach can be applied for both aircraft sensitivities studies and for shape optimisation (Selmin, 2004). It was proved that the AD method was effective for differentiating both the only CFD solver and the complete computation chain (shape parameters + solver). Comparisons were carried out at numerical level and results were than compared again with finite differences computations; the outcomes were more than satisfactory.

In 2009 another work was published by the Authors et Al. (Necci, et al., 2009); its goal was to demonstrate the practical applicability of AD to industrial problems dealing with complex configurations. It was shown that automatic differentiation could be successfully used not only for research purposes but also to carry out a design activity on complex industrial configurations. Progress has been made from that point on and AD capabilities have been extended to compute stability and control dynamic derivatives for civil and fighter aircrafts. This paper deals with this. A first conceptual comparison between AD and other methods will be provided in order to explain AD peculiarities; then a deeper study of the AD numerical technology will be carried out. Finally some results will be given.

2. Automatic differentiation concept

Automatic differentiation is based on concepts which are extensively described in Ref. [10] and [11]. To summarise, let $\mathbf{X} \in \Re^n$ be a vector argument and $\mathbf{Y} = \mathbf{F}(\mathbf{X}) \in \Re^m$ a corresponding vector function. Equation $F_m = F_m(x_1, x_2,..., x_n)$ expresses the idea to have m functions depending on n variables. A computer program P able to evaluate Y can only evaluate simple functions; so we have to split up F_m into several sub-functions f_k, each one implemented by a corresponding instruction I_k. Function F will then be given by the composition $F = f_p \circ f_{p-1} \circ ... \circ f_1$ and the corresponding source code has to be a sequence like $P = \{I_1; I_2;...; I_p\}$. As a mathematical limitation, all the functions which are implemented in the code functions have to be differentiable, despite some exceptions (e.g. the square root); however, in general, functions implemented by arithmetic operations are indeed differentiable. The automatic differentiation tool evaluates derivatives by using the chain rule, i.e. it applies the following mathematical concept:

$$F'(X) = \underbrace{f_p'(X_{p-1}) \times f_{p-1}'(X_{p-2}) \times ... \times f_1'(X_0)}_{J} \tag{1}$$

Here $X_k = f_k(X_{k-1})$ is the value of a generic sub-function and $X_0 = X$ is its first value. J is clearly the Jacobean of F. Using the concept expressed by equation (1), derivatives can be translated back into a sequence of instructions I_k', the instructions will be coded and inserted back into a copy of the control program P. This new set of instructions, added to P, yields to program P', which now embeds instructions both for primitive and for derivative functions. Conceptually, the automatic differentiation works on some basic assumptions; firstly, P is considered simply as a run-time sequence of instructions and the AD tool differentiates this sequence. Secondly, each sequence is a composition of vector functions, each one assumed differentiable. Since each function is differentiable, we have the theoretical basis to differentiate according to the fundamental rule of mathematical analysis: we have to work not only on the sequence of functions but also on each single function. Furthermore one has to observe that the generic function $F_m = F_m(x_1, x_2,..., x_n)$ depends on n terms, so differentiation of a single function imposes computing n derivatives. Applying this concept to a vector function it is evident that equation (1) is a Jacobean and, as such, it requires multiplying matrices with matrices. This opens another issue: what is the best strategy to reduce the computational cost? Direct (tangent) or backwards (reverse) modes stem from this issue.

Physically, often we just need evaluating the sensibility of a quantity over a design parameter; an example is the sensibility of lift coefficient with respect to the angle of attack $(C_{L\alpha})$ or lateral force sensitivity to sideslip angle $(C_{Y\beta})$. For an aerodynamic coefficient, this means taking into account just its component related to a given aerodynamic parameter. Mathematically, this requires projecting a function (a total derivative) onto a given direction (a design parameter), as conceptually shown in Fig. 1.

Since our basis has n components, a generic directional derivative is the projection of the total derivative $\mathbf{F}'(\mathbf{X})$ along one component of the basis. Formally, we have to evaluate:

$$\mathbf{F}'(\mathbf{X}) \cdot \dot{\mathbf{X}} = \underbrace{f_p'(X_{p-1}) \times f_{p-1}'(X_{p-2}) \times ... \times f_1'(X_0)}_{J} \cdot \dot{X} \tag{2}$$

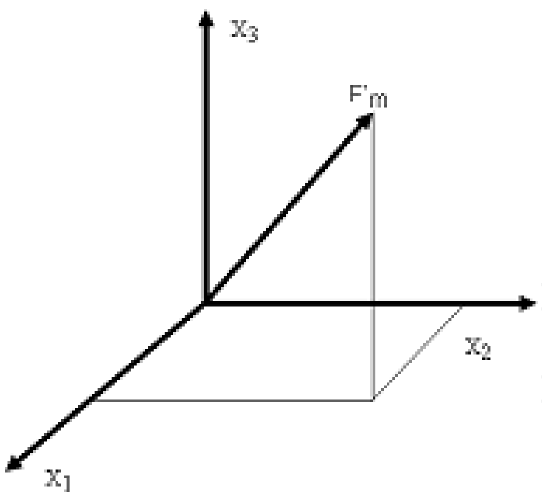

Fig. 1. Generic reference basis.

$\mathbf{F}'(\mathbf{X})$ is a $\{m \times n\}$ matrix and the most efficient way to evaluate the quantity given in (2) is from right to left; this requires multiplying a matrix by a vector. We have to start the differentiation from the first instruction $f_1'(X_0)$ in the source code and then we have to progress on, until the last instruction is reached. This approach makes things easier from the point of view of program coding, as it allows merging the original source code with the lines of differenced functions immediately after each corresponding primitive instruction. Such a concept is the basis of direct (or tangent) differentiation.
Conversely, in optimization processes or inverse problems, it is required minimising a generic cost function with respect to a number of design parameters; a physical example of this can be the minimisation of drag index with respect to the angle of attack, to Mach number and to a number of shape parameters. This imposes the differentiation of a function with respect to a number of parameters and then to weight each component with a dedicated coefficient. Mathematically, this implies transforming $\mathbf{Y} = \mathbf{F}(\underline{\mathbf{X}})$ vector into a row (using transposition) and then multiplying this row by a weight vector $\overline{\mathbf{F}}(m)$, which clearly has to be an input. Formally, the product becomes $\mathbf{F}^{\mathbf{T}}(\mathbf{X}) \cdot \overline{\mathbf{F}}$ and its gradient is $\mathbf{F}^{'\mathbf{T}}(\mathbf{X}) \cdot \overline{\mathbf{F}}$; transposing $\mathbf{F}'$ means transposing the product $F' = f_p' \times f_{p-1}' \times \ldots \times f_1'$ and, remembering that $(\mathbf{A} \times \mathbf{B})^T = \mathbf{B}^T \times \mathbf{A}^T$, we have:

$$\mathbf{F}^{'T} = \left(f_p' \times f_{p-1}' \times \ldots \times f_1'\right)^T = f_1^{'T} \times f_2^{'T} \times \ldots \times f_p^{'T} \tag{3}$$

from which:

$$\frac{\partial \mathbf{F}}{\partial \mathbf{X}} = \mathbf{F}^{'T} \times \overline{\mathbf{F}} = \underbrace{f_1^{'T}(X_0) \times f_2^{'T}(X_1) \times \ldots \times f_p^{'T}\left(X_{p-1}\right)}_{J^T} \times \overline{F} \tag{4}$$

Here $\mathbf{F}^{'T}\{n \times m\}$ is the transposed Jacobean. Even in this case computation is more efficient from right to left; we can progress using a matrix by vector product, computationally cheaper than a matrix by matrix computation. In this second case, computation starts differencing the last p^{th} function and then going back to first function. The definition of backwards (or inverse) differentiation is due to this.

A general rule, easy to remember when choosing the best differentiation method comes from the following observation; the less number of rows in the multiplying vectors ($\dot{X}$ or $\overline{F}$) the less will be the operations. For direct mode, short $\dot{X}$ corresponds to small n , thus direct mode can be computationally effective when differencing a number of functions with respect to a few parameters. Conversely, for backwards mode, short $\overline{F}$ correspond to small m , and this means a few functions to differentiate; it is attractive when differencing with respect to a number of parameters, i.e. in case of optimisation problems.

3. Numerical approach

By taking into account a generic solution depending on the position vector, normal vectors and boundary conditions, the sensitivity of a generic aerodynamic solution with respect to a parameter β can be expressed as (Ref. [13]):

$$\frac{\partial \mathbf{U}}{\partial \beta} = \frac{\partial \mathbf{U}}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \beta} + \frac{\partial \mathbf{U}}{\partial \boldsymbol{\eta}_{ij}}\frac{\partial \boldsymbol{\eta}_{ij}}{\partial \beta} + \frac{\partial \mathbf{U}}{\partial \boldsymbol{\eta}_i^b}\frac{\partial \boldsymbol{\eta}_i^b}{\partial \beta} \tag{5}$$

If the automatic differentiation tool is used in order to evaluate the quantities expressed in (5), it will generate the total derivative with respect to the specified design parameter. But we use the automatic differentiation to evaluate the sensitivity to just some parameters, so the tool has to be tuned according to real needs, i.e. it has to compute just some derivatives, not the total derivative. For this reason, the differentiation procedure has to take into account the following issues:

- when computing the sensitivity with respect to a single parameter, the relevant parameter β has to be set to 'one' before entering the differentiation routines and all other parameters have to be set to 'zero'; this can be done directly into the code or using an external input file;
- in order to reduce computational costs, differentiation should be applied just after the primitive solution has converged, so to avoid computing also the iterations related to derivatives. In this way the application of differenced functions to a converged solution will imply only one further computation run.

However, despite the conceptual simplicity and elegance of such an optimized approach, the real implementation of automatic differentiation is very challenging from the software engineering side, especially for complex industrial tools.

The solver that includes differentiated functions, here defined 'augmented', will be discussed preliminary for the static differentiation, so limiting the action to its steady loop; dynamic differentiation will follow and will extend the action to the transient loop. Three different structures of the augmented algorithm will be studied; all of them will include some preliminary steps, i.e. the acquisition of initial values, check of grid metrics and acquisition of initial solution. These steps will not be subject to any differentiation, since just the core of computation has to be differenced and not all the modules of the solver. After these three steps, some differences among the structures will be discussed.

In structure 1 (Fig. 2), after some initial checks, the parameter with respect to which we want to differentiate is set to unity. Then the augmented algorithm is engaged and both primitive and differenced parameters are computed. This configuration is structurally simple but it requires computing derivatives for all iterations while the primitive solution is still reaching its convergence.

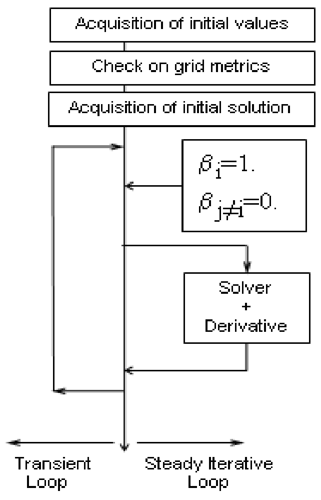

Fig. 2. Structure 1

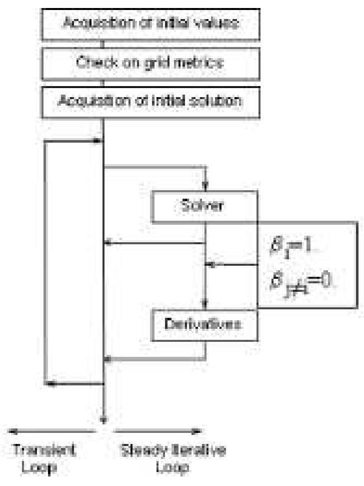

Fig. 3. Structure 2

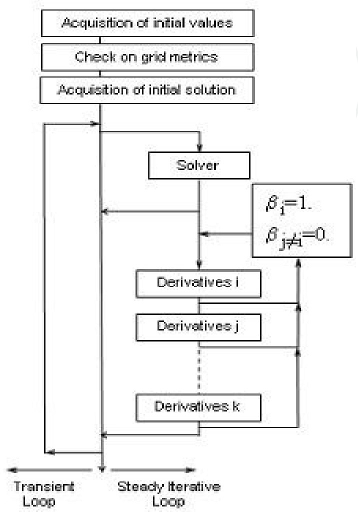

Fig. 4. Structure 3

This increases the runtime cost without any real benefit, unless the user is interested in studying the iterative evolution of derivatives. Structure 2 (Fig. 3) shows a variant; derivatives are computed just after the primitive solution has converged; only one extra run is required. Differentiation is paid with a minimal increase of computational cost and numerical efficiency has clearly increased. At this point, we can look at the problem from another point of view; we can remember that the AD tool provides the capability to differentiate a function with respect to one or more parameters. Surely, differencing with respect to several parameters involves a higher complexity of the augmented software because of the higher number of added instructions but, conceptually, structures 1 and 2 can be used indifferently to differentiate with respect to one or more parameters. An alternative structure of the augmented solver, suitable for differencing with respect to many parameters, is shown in structure 3 (Fig. 4). In this last case, a number of differentiation blocks can be cascaded, and each block computes the derivatives with respect to one only parameter. Then, iteratively, all the blocks can be engaged in sequence. It is necessary looping the cycle in such a way to reset to 'one' the correct β_i parameter before calling each block. The benefit comes from the capability to evaluate all the derivatives with respect to β_i with just one extra iteration for each parameter, like in Structure 2. Compared to structure 2, structure 3 has an advantage and a disadvantage. The advantage is that structure 2 manages a larger module involving several derivatives, and the use of only one block makes it easier linking such a block with the solver. The disadvantage is that if the user needs rearranging the differentiation with respect to one parameter, for any particular purpose, the use of structure 2 makes the work very hard. Structure 3 is simply a modular approach, which requires more initial work to integrate each differentiation module in the overall structure but that guarantees a higher flexibility in case of further reworking. It is evident that structure 3 has a practical difficulty; while in structures 1 and 2 the software has to be optimized for just one large block of augmented variables, in structure 3 more sub-blocks are involved, so optimization is longer and more complex. In this case, the modularity of the primitive code and software engineering skills of the user make the difference. Once the structure has been decided, a practical possibility from the software point of view is that all parameters that have to be set to 1. Then these can be read from an input file; then all different blocks, linked together with the relevant declaration files, are called by the program. These and a number of possible variants can be explored; it is just important to say that, whatever the user wants to do, he needs a complete control of the software. This means that the automatic differentiation technology cannot be applied to commercial tools, which are closed applications.
Moving now from static to dynamic differentiation, the only difference is in the type of parameters; static differentiation involves incidence angle, sideslip angle or geometric functions while dynamic differentiation involves roll, pitch and yaw velocities, i.e. it involves the time. Because of the dynamic nature of these parameters, AD impacts now the transient loop of the software. But in this case the good modularity of the solver that has been used in Alenia Aeronautica has shown its benefits, since the AD procedure has been used without any real problem and according to the rules described for the static differentiation.

4. Numerical applications and dynamic computing procedure

The results described in this section have been achieved using the tangent mode differentiation. Using the following expression:

$$H_d = \frac{\partial \mathbf{H}}{\partial \mathbf{B}} \tag{6}$$

we indicate with $\mathbf{H} = \{C_L, C_D, C_Y, C_l, C_m, C_n\}^T$ the aerodynamic coefficients vector and with $\mathbf{B} = (p, q, r)$ the design variables vector.
Because of the structure of the solver that has been used, the numerical sequence for computing a derivative requires three different steps:

1. the computation of a primitive steady solution (PS1);
2. the computation of a primitive unsteady solution (PS2);
3. the computation of the differentiated solution (DS).

Two tests cases have been investigated, a NACA0012 airfoil studied in viscid condition and a complete configuration, the DLR-F12, studied in inviscid condition. This second test case has been the real test bench, since the results that have been achieved have been compared with those collected by other Partners involved in the SIMSAC programme.

4.1 NACA0012 airfoil

NACA0012 has been studied for $M = 0.4$ and $\text{Re} = 6.2e + 06$. The grid is a C-type, 66880 nodes and 24924 elements, generated by an Alenia proprietary mesh generator. The skin has been meshed by placing 160 nodes on the upper and lower surfaces; the wake has been similarly meshed, by placing 50 nodes after the trailing edge. The skin mesh has then been extended in the outwards normal direction; the boundary layer has been filled up by building 57 parallel layers in the region between the skin and the area where the 99% of the freestream velocity is reached. Then, 200 more layers have been used in the farfield. Dimensions of the grid are as follows:

$$\Delta x = -0.15000e + 02; +0.16000e + 02$$

$$\Delta y = +0.0000e + 00; +0.10000e + 01$$

$$\Delta z = -0.15056e + 02; +0.15055e + 02$$

The mesh around both leading and trailing edges has been built by setting the initial spacing layer to 0.1e-02 chord units. Starting from the first layer, a geometrical progression has then been used; setting a growth factor 1.208, the mesh has been expanded in the normal outwards direction, until the final layer of the farfield has been reached. The airfoil chord length has been set to unity. The three-dimensions solver has been applied on this two-dimensions mesh by simply redefining the grid itself; the two-dimensions mesh has been generated on a vertical plane, which has then been doubled onto a second parallel plane. The two planes have then been joined together, so generating a three-dimensional structure. Fig. 5 shows how the two-dimensions mesh has been converted in three-dimensions.
As said earlier, the first step of the computation process has been the achievement of a stable primitive steady solution, for M=0.4 and α=2°; 15000 iterations have been used to completely minimise residuals and to reach an excellent solution. After this, two different dynamic analyses have been carried out, one involving some variable frequencies of oscillation and another involving some variable angles of attack. The laminar Prandtl number has been set to 0.72 and the turbulent to 0.90. CFL number has been set to 2.0 and no residual averaging has been necessary. The steady computation has required about 448 seconds on a parallel

Quadrics machine, while unsteady runs almost doubled this value. Thirty-two processors have been used, 28 for computing purposes and 4 for data traffic management.

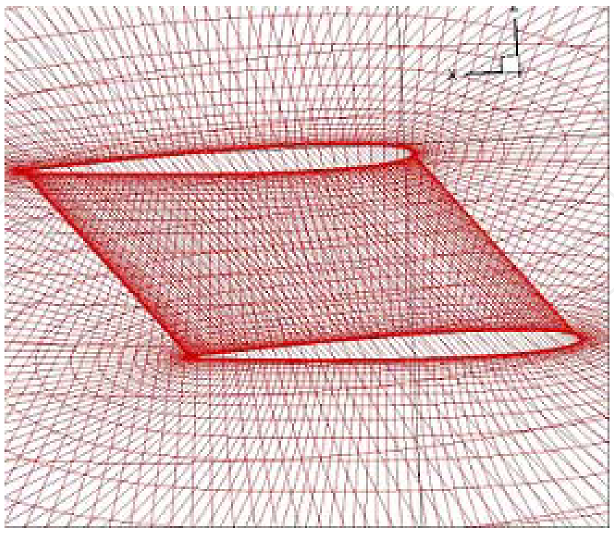

Fig. 5. Internal mesh – conversion to three-dimensions.

4.1.1 Variable frequency

The analysis for variable frequencies has required computing a number of primitive steady solutions, one for each frequency in the (used) range $0.025\,rad/s \le \omega \le 0.1\,rad/s$. For the sake of clarity, we can say that here we will just describe and discuss some details related to the numerical test case $M = 0.4, \alpha = 2°, \omega = 0.025\,rad/s$, remembering that the used process has been exactly the same for all the frequencies. After computing a stable primitive steady solution, the dynamic solver has been engaged. A pitch range $\Delta\alpha = \pm 1°$ has been imposed about the initial angle of attack and a forced oscillation has been applied. The rotation centre has been placed at $x = c/4$ and, in order to minimise the numerical transient due to the initial steady solution (PS1), three complete oscillations have been run. For each oscillation, 96 iterations have been computed for a total amount of 288 iterations; each one of these iterations required 200 sub-iterations to converge. Fig. 6 shows a graphical example.

Along the X axis is reported the number of iterations, while on y axis are shown both the corresponding values of the lift coefficient and of the angle of attack. It is evident that each time the local angle of attack increases, a corresponding increase of the lift coefficient is experienced as well. In order to show that each point in Fig. 6 corresponds to a converged value of the iteration process, Fig. 7 focuses the evolution of the solution for the lift coefficient during the very first step of the first oscillation. It is just the case to mention that this particular case has been chosen because the first unsteady iteration run after the initial (steady) solution is the most impacted from the numerical stability point of view. We can see that, when the sub-iteration process starts, the value of the lift coefficient is close to 0.24, which is the value provided by the steady solution PS1. After this, the unsteady computation begins and a slight fluctuation is experienced. Anyway, after about 150 sub-iterations, a convergence is reached. This last fact confirmed that the choice of 200 sub-iterations was successful, since the numerical fluctuation was almost completely eliminated. In order to prove this last sentence with numerical evidence, Table 1 provides maximum and minimum values for Cl during the forced three oscillations, as well as the relevant percentage fluctuations.

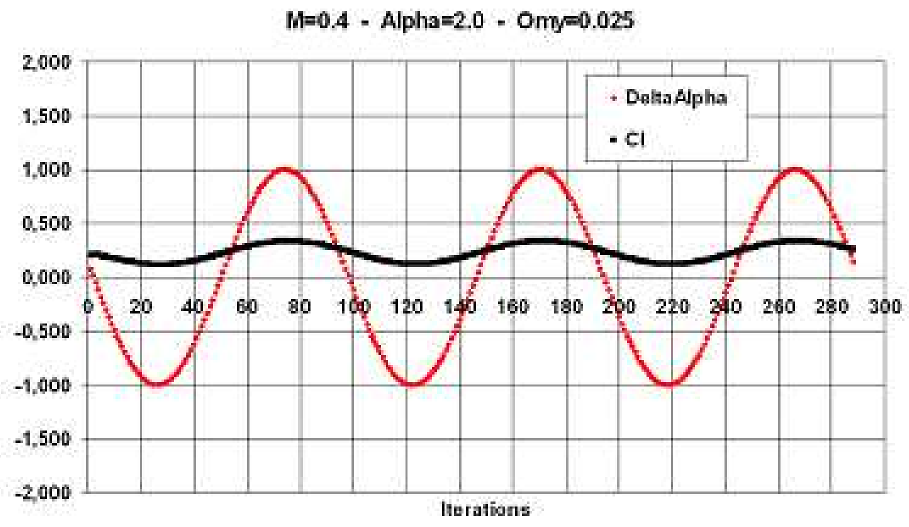

Fig. 6. NACA0012 oscillation evolution for $\left(M = 0.4, \alpha = 2°, \Delta\alpha = \pm 1°, \omega_y = 0.025\,rad/s\right)$.

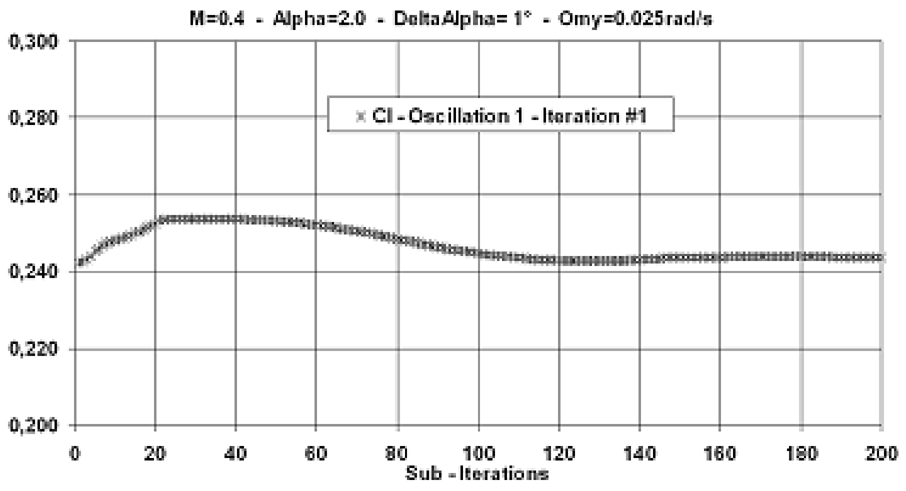

Fig. 7. NACA0012 - Lift coefficient evolution for unsteady computation
$\left(M = 0.4, \alpha = 2°, \Delta\alpha = 1°, \omega_y = 0.025\,rad/s\right)$

	Cl min	Cl max	% Fluctuation
Oscillation 1 - Sub-Iteration 1-100	2.4283e-01	2.6130e-01	7.068%
Oscillation 1 - Sub-Iteration 101-200	2.4302e-01	2.4787e-01	1.956%
Oscillation 2 - Sub-Iteration 1-100	2.4213e-01	2.5384e-01	4.613%
Oscillation 2 - Sub-Iteration 101-200	2.4356e-01	2.4376e-01	0.082%
Oscillation 3 - Sub-Iteration 1-100	2.4212e-01	2.5382e-01	4.609%
Oscillation 3 - Sub-Iteration 101-200	2.4356e-01	2.4375e-01	0.077%

Table 1. Cl convergence - values and errors

The table is arranged in such a way to allow an analysis for groups of sub-iterations. Each one of the three oscillations is split up in two phases; the first one focuses on sub-iterations 1 to 100 while the second focuses on sub-iterations 101 to 200. During oscillation 1, Cl values in the first phase range from 2.4283e-01 to 2.6130e-01 and the relevant percentage fluctuation is 7.068%; values in the second phase range from 2.4302e-01 to 2.4787e-01 and the corresponding percentage fluctuation drops down to 1.956%. This means that, at the end of oscillation 1, the solution has almost converged. Going on and computing the same values for oscillations 2 and 3, all percentage variations in the first phases become even smaller (4.613% and 4.609%) and almost disappear during the second phases (0.082% and 0.077%). This proofs that the unsteady solution PS2 has completely converged at the end of the third cycle. Stated this, we can study the evolution of our differenced solutions; in particular, it is interesting comparing the results provided by the augmented solver with those computed by using the finite differences method. Fig. 8 deals with this and it still provides information for the case $M = 0.4, \alpha = 2°, \omega = 0.025$.

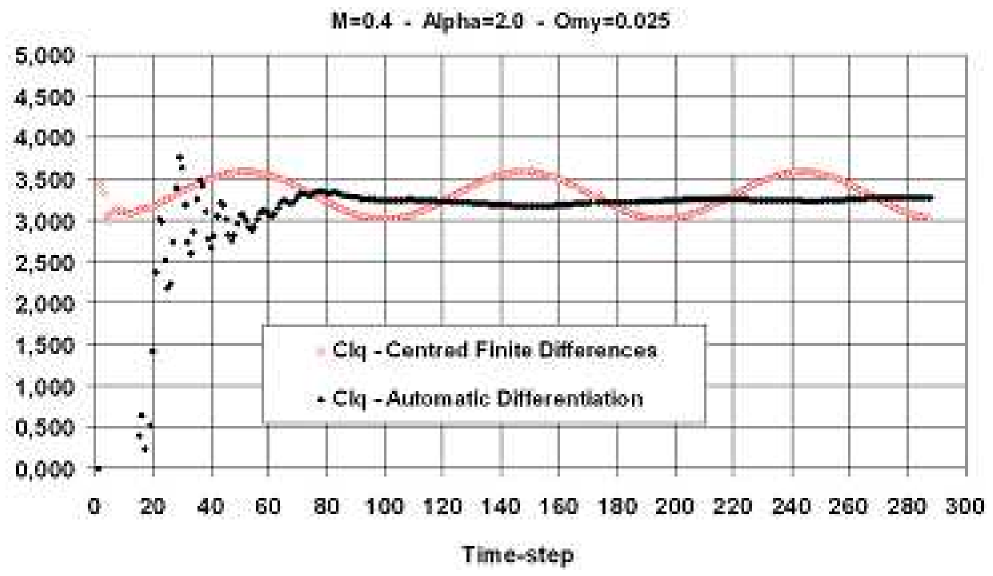

Fig. 8. Comparison between automatic differentiation and finite differences Clq derivatives $\left(M = 0.4, \alpha = 2°, \Delta\alpha = 1°, \omega_y = 0.025\, rad/s\right)$.

Finite differences have been computed by remembering that, from the analytical point of view, we have:

$$C_{l_{\dot{\alpha}}} = \partial C_l / \partial\left(\bar{c}\alpha/2U_\infty\right), \quad C_{m_q} = \partial C_m / \partial\left(\bar{c}q/2U_\infty\right) \tag{7}$$

Moving now from differential values to finite values, these expressions can be rewritten as:

$$C_{l_{\dot{\alpha}}} = \Delta C_l / \Delta\left(\bar{c}\alpha/2U_\infty\right), \quad C_{m_q} = \Delta C_m / \Delta\left(\bar{c}q/2U_\infty\right) \tag{8}$$

The curve related to finite differences fluctuates because of the fluctuating values of ΔC_l, computed by using the primitive solver. Conversely, the AD curve is much more stable; after an initial transient, the curve becomes almost steady and quickly converges. Table 2 shows both values and percentage errors between the two curves; it is evident that errors do vary mainly because of the finite differences fluctuations, while automatic differentiation numerical values show a good steadiness.

Time-step	Clq FD	Clq AD	Error %
0	3.43E+00	2.76E+01	-704.96%
20	3.18E+00	2.37E+00	25.52%
40	3.52E+00	2.81E+00	20.18%
60	3.53E+00	3.05E+00	13.61%
80	3.20E+00	3.33E+00	-3.97%
100	3.02E+00	3.25E+00	-7.59%
120	3.25E+00	3.23E+00	0.74%
140	3.56E+00	3.19E+00	10.51%
160	3.48E+00	3.19E+00	8.55%
180	3.13E+00	3.22E+00	-2.68%
200	3.03E+00	3.25E+00	-7.09%
220	3.26E+00	3.25E+00	0.21%
240	3.58E+00	3.24E+00	9.68%
260	3.42E+00	3.26E+00	4.77%
280	3.08E+00	3.27E+00	-6.04%

Table 2. Clq comparison - AD vs FD - values and errors M=0.4, α=2.0°, Δα=1°, ωy=0.025 rad/s

Similar considerations can be repeated for C_{m_q} (Fig. 2). In this case percentage errors are initially very high but convergence is fast and the error becomes smaller than 5%

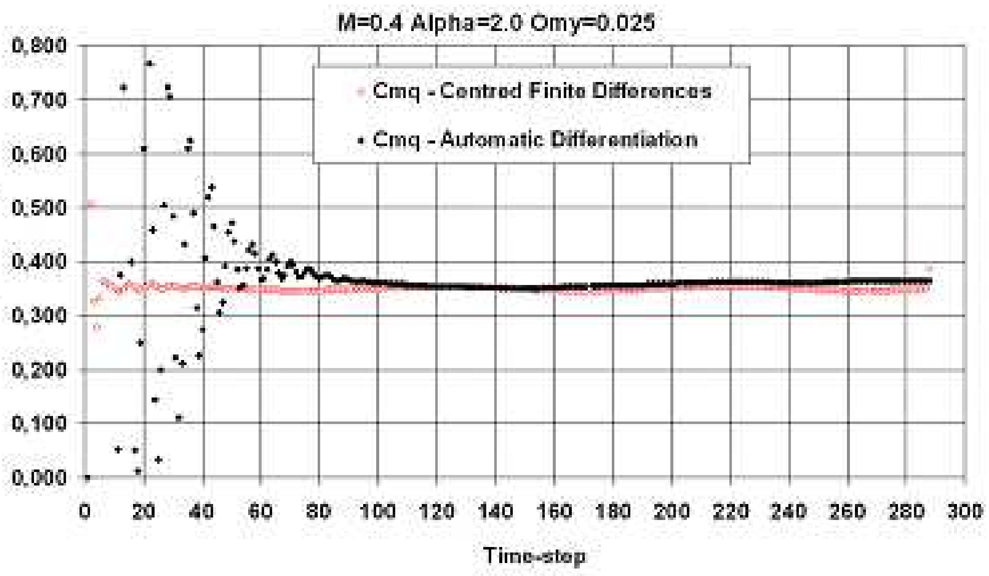

Fig. 9. Comparison between automatic differentiation and finite differences Cmq derivatives $\left(M = 0.4, \alpha = 2°, \Delta\alpha = 1°, \omega_y = 0.025\, rad/s\right)$.

The following figures (Fig. 10 and 11), show the values of derivatives for the same flight condition, but with an oscillation frequency $\omega = 0.05$.

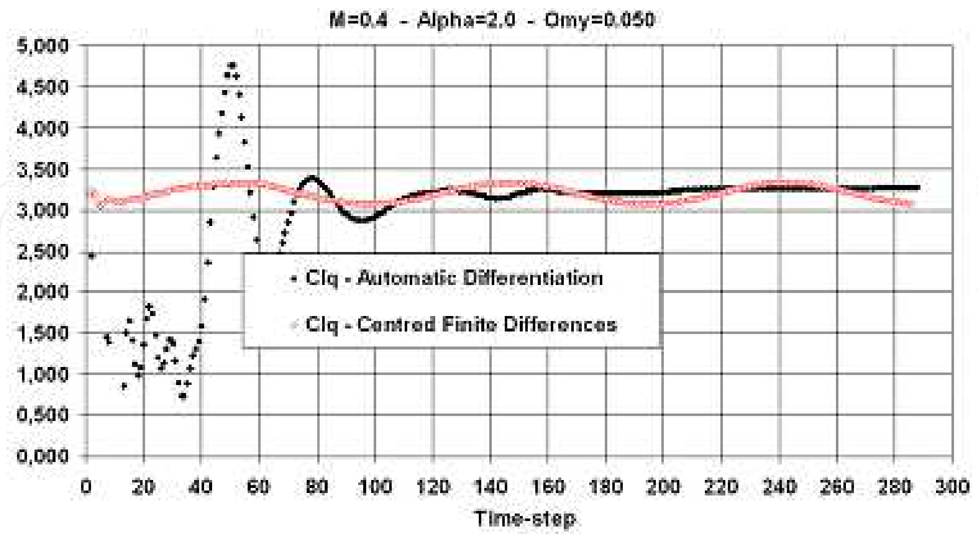

Fig. 10. Comparison between automatic differentiation and finite differences Clq derivatives $\left(M = 0.4, \alpha = 2°, \Delta\alpha = 1°, \omega_y = 0.050\, rad/s\right)$.

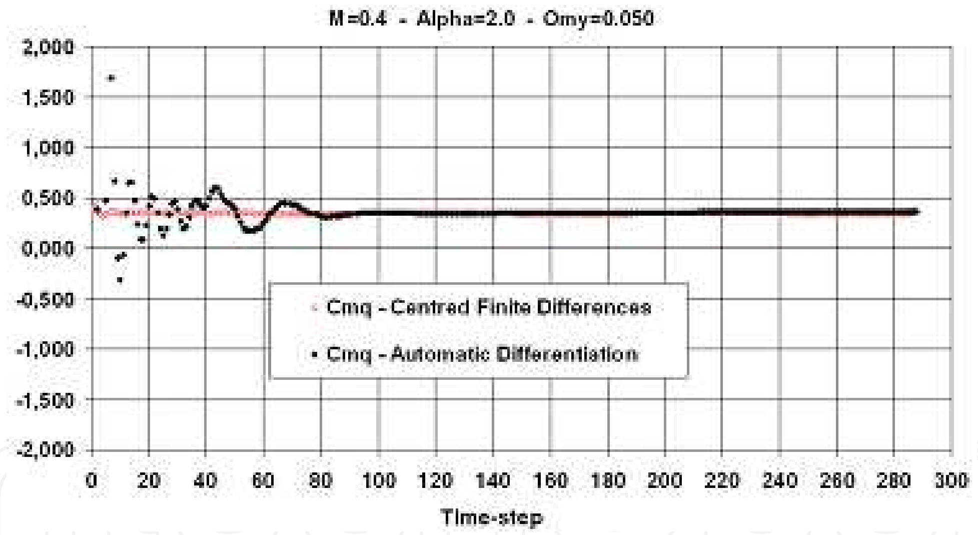

Fig. 11. Comparison between automatic differentiation and finite differences Cmq derivatives $\left(M = 0.4, \alpha = 2°, \Delta\alpha = 1°, \omega_y = 0.050\, rad/s\right)$

The overall progression of results is good, despite some local peaks. As a general behaviour the automatic differentiation provides results much more stable, because of its direct computation method based on a mathematical management of functions and not on a latter data handling with FD. Anyway some initial fluctuations have to be expected also in automatic differentiation, since anyway the relevant derivatives are managed by the augmented solver as normal functions which need some time to converge.
Finally, figures Fig. 12 and 13 show the evolution of C_{lq}, C_{dq}, C_{mq} as functions of frequencies.

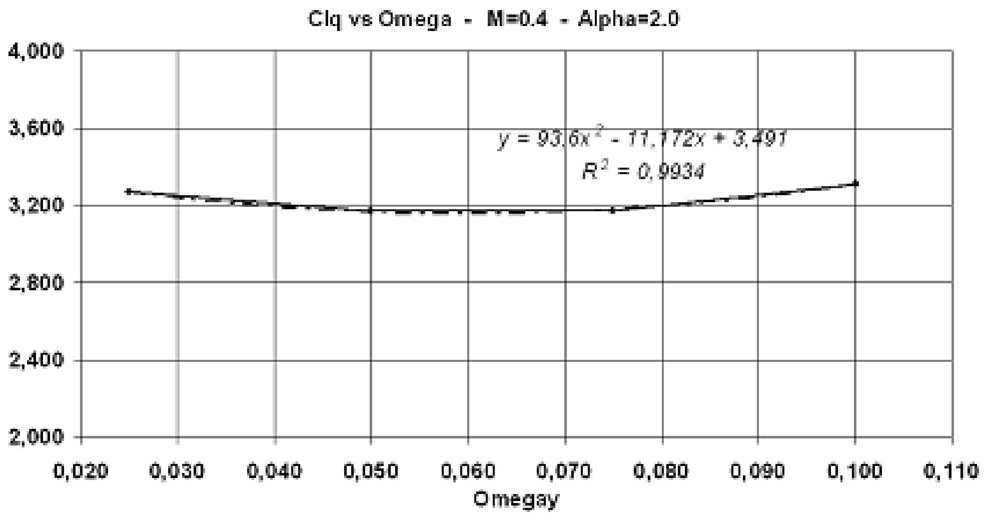

Fig. 12. Automatic differentiation Clq vs frequency progression $(M = 0.4, \alpha = 2°, \Delta\alpha = 1°)$

The relevant interpolating second order curves have been reported, as well as the least squares values computed as follows:

$$R = \sum_{i=1}^{n} \left[y_{i-AD} - f(x_i) \right] \tag{9}$$

The value of y_{i-AD} value is computed by using the AD, while $f(x_i)$ is computed by using the tendency function.

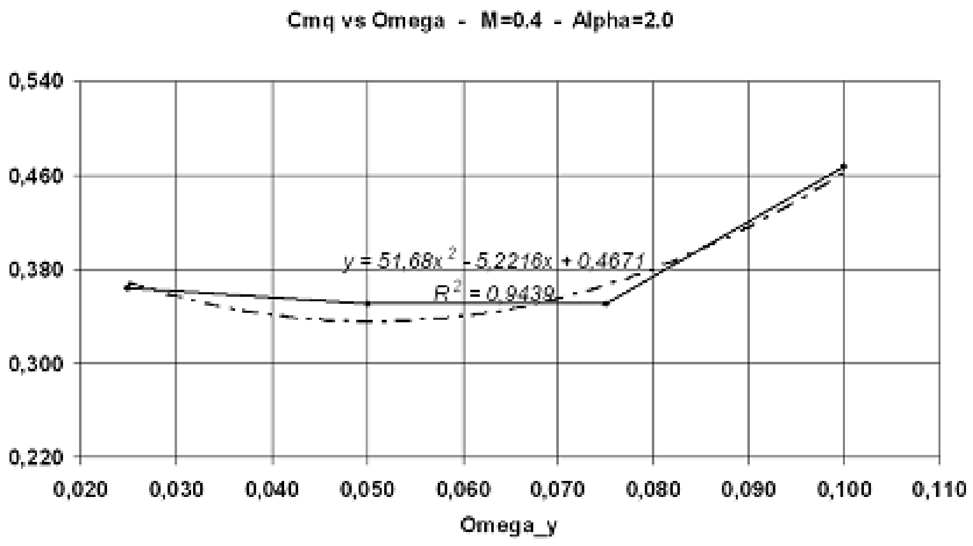

Fig. 13. AD Cmq vs frequency progression $(M = 0.4, \alpha = 2°, \Delta\alpha = 1°)$.

It is shown that all curves can be reduced to polynomials; however the physical evolution is well modelled by polynomials just in some cases, e.g. for Clq and Cmq, while in other cases (Cdq) such a reduction cannot be really accepted since curves are not smoothed and a polynomial reduction is only an algebraic manipulation

4.1.2 Variable incidence

The same approach described for studying the variable frequency has been applied to variable angles of attack. The most challenging dynamic behaviour is for $\omega = 0.1$, so some curves (Fig. 14-17) and tables are here discussed for such a frequency. The angle of attack ranges from $\alpha = 5°$ to $\alpha = 8°$.

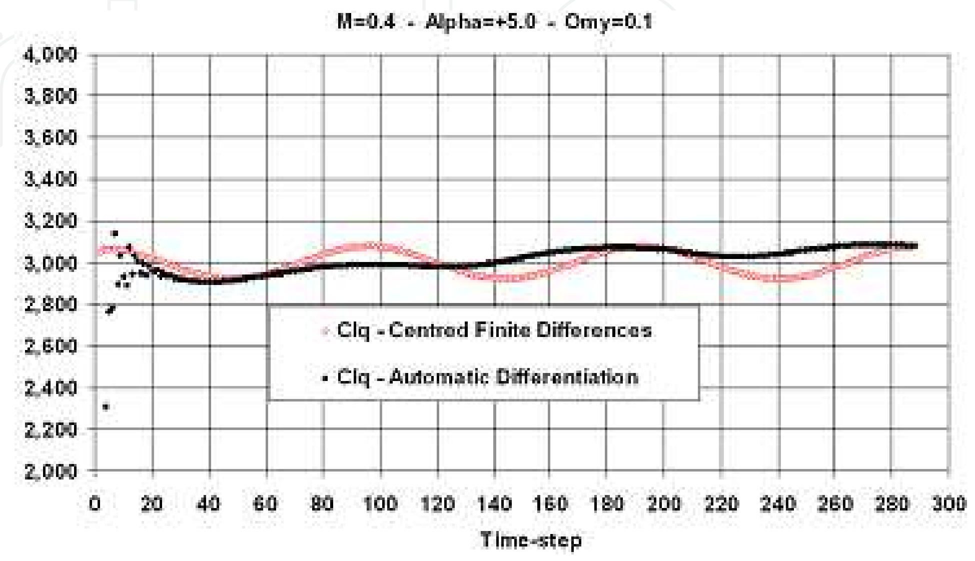

Fig. 14. Comparison between automatic differentiation and finite differences Clq derivatives $\left(M = 0.4, \alpha = 5°, \Delta\alpha = 1°, \omega_y = 0.100\, rad/s\right)$

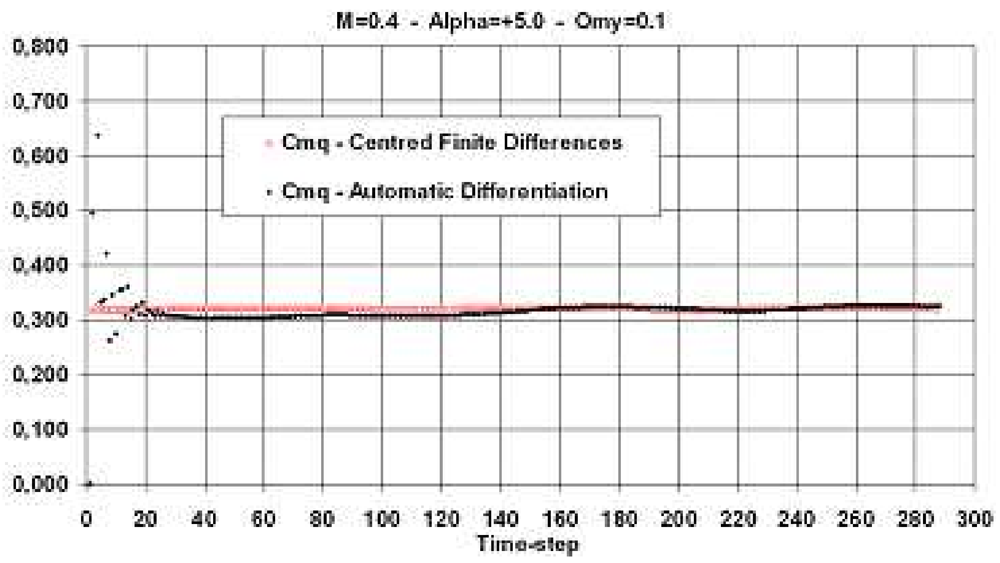

Fig. 15. Comparison between automatic differentiation and finite differences Cmq derivatives $\left(M = 0.4, \alpha = 5°, \Delta\alpha = 1°, \omega_y = 0.100\, rad/s\right)$.

For $\alpha = 5°$, the values of percentage errors for C_{l_q} show a strong variability, due to the high frequency. However, when the solution is stabilised, the percentage error between AD and

FD derivatives is not higher than 4.16 % (time-step 240) and in some points it is close to 0%. For C_{m_q} , the error reached after stabilisation is 2.03% (time step 260) so showing to be even more stable than C_{l_q} . Repeating the same analysis for $\alpha = 8°$, both C_{l_q} and C_{m_q} show a similar behaviour and anyway the percentage error between the two methods (AD and FD) is acceptable.

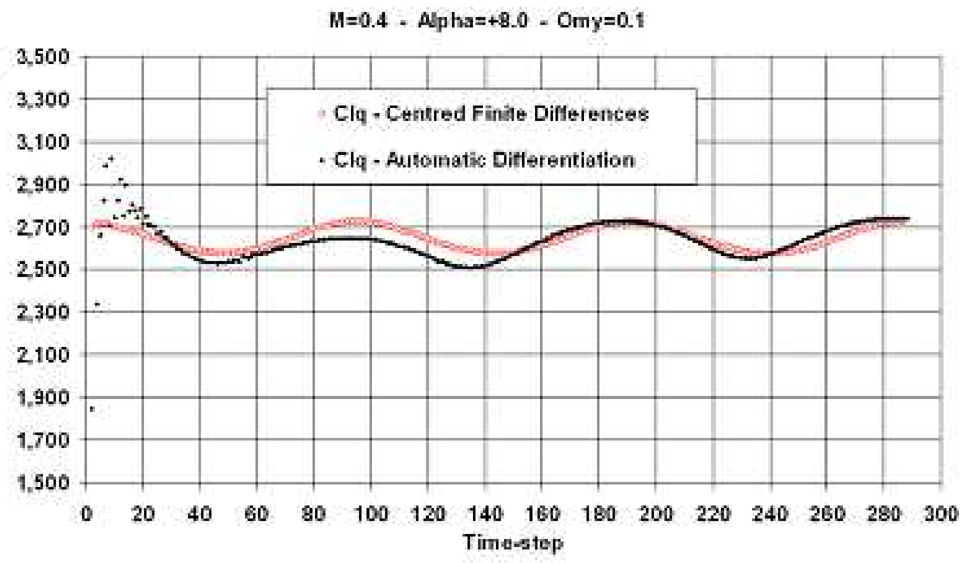

Fig. 16. Comparison between automatic differentiation and finite differences Clq derivatives $\left(M = 0.4, \alpha = 8°, \Delta\alpha = 1°, \omega_y = 0.100\, rad/s\right)$

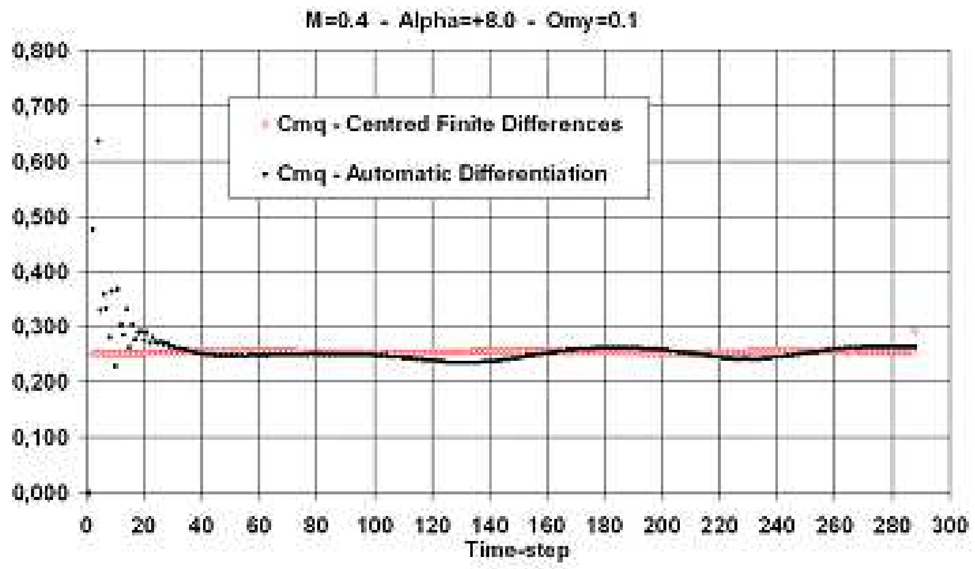

Fig. 17. Comparison between automatic differentiation and finite differences Cmq derivatives $\left(M = 0.4, \alpha = 8°, \Delta\alpha = 1°, \omega_y = 0.100\, rad/s\right)$.

Finally, Fig. 18 to Fig. 20 show the evolutions of C_{lq}, C_{dq}, C_{mq} as functions of the angle of attach, ranging in the interval $\alpha \in [-2°; +8°]$. The evolutions are well smoothened and their regularity is evident.

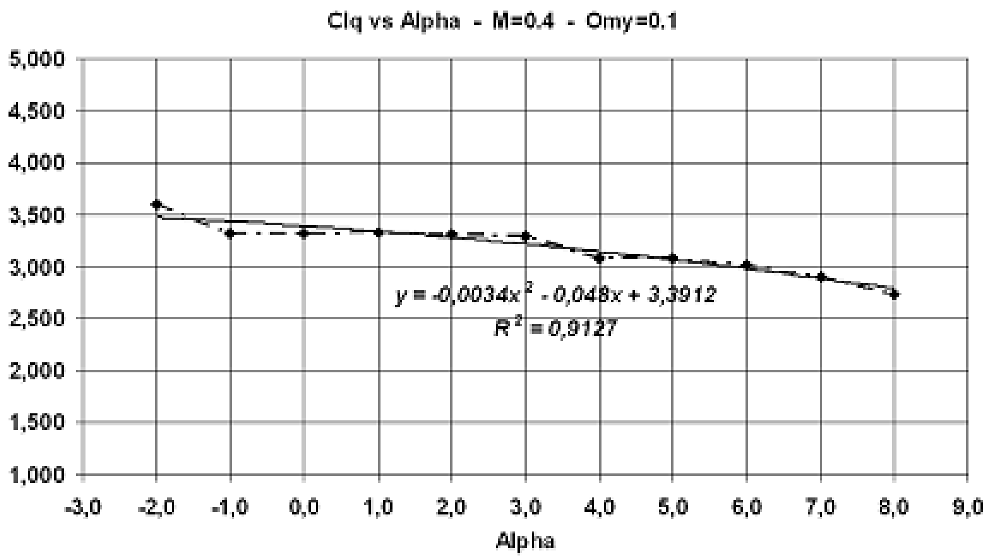

Fig. 18. Automatic differentiation Clq vs Alpha $\left(M = 0.4, \Delta\alpha = 1°, \omega_y = 0.100\, rad/s\right)$

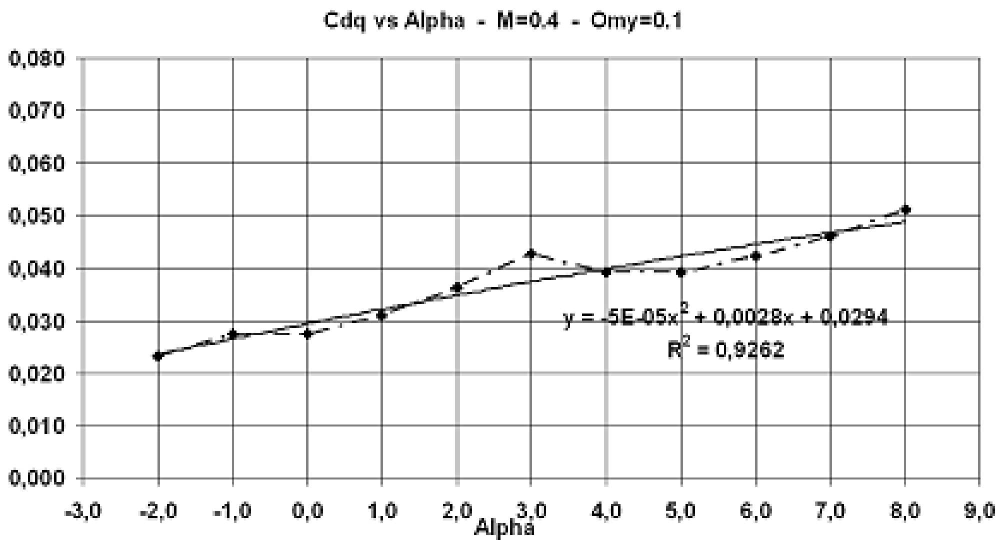

Fig. 19. Automatic differentiation Cdq vs Alpha $\left(M = 0.4, \Delta\alpha = 1°, \omega_y = 0.100\, rad/s\right)$.

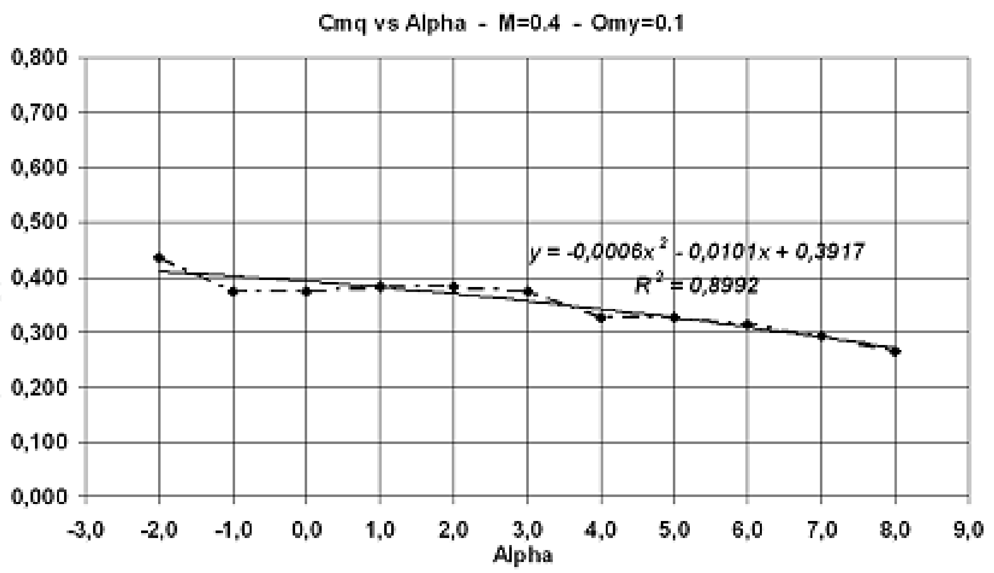

Fig. 20. Automatic differentiation Cmq vs Alpha $\left(M = 0.4, \Delta\alpha = 1°, \omega_y = 0.100\, rad/s\right)$.

4.2 F12

The second sequence of tests has been carried out on the DLR-F12 configuration, working in inviscid condition at $M = 0.202$ $\left(u = 70\, m/s\right)$. As already said, F12 has been one of the test benches used in SimSAC; within such a context, DLR and Cerfacs/ONERA generated two sets of meshes, unstructured and structured respectively, to be used by Partners in order to have a common computing basis. The Euler grid used by Alenia was the unstructured full configuration, 3802374 nodes, and 25263927 segments. The dimensions of the grid are the following:

$$-4.951e+02 < \Delta x < 7.048e+02$$

$$-5.998e+02 < \Delta y < 5.998e+02$$

$$-6.034e+02 < \Delta z < 5.969e+02$$

Figure 21 show two views of the overall mesh.

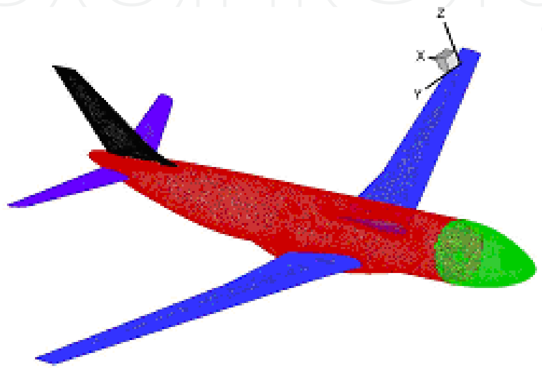

Fig. 21. F12 overall mesh.

An extensive numerical testing has been carried out by running 15000 iterations to reach the primitive steady solution PS1. After this, the unsteady solution has been computed by engaging the AD solver and running it for 15000 more iterations.
Computation has been run considering normal environmental conditions; in particular, the solver has been set to simulate a flight at $p = 1Atm, T = 25°C, u = 70\,m/s$. Numerical data have been compared to those acquired during the wind tunnel testing; the German-Dutch Wind Tunnels premise DNW-NWB, sited in Braunschweig, has been used for the acquisition of experimental data. Static testing included α- and β-sweeps. Figures 22 to 24 show the evolutions of computed lift, pitch moment and drag coefficients as functions of the angle of attack, compared to the experimental values. It is evident a large data spread, from different Partners, for angles of attack up to 8°, especially for pitch moment and drag. This is mainly due to the different numerical approaches used by different Partners. Fig. 25 to Fig. 27 show roll, pitch and yaw moments as functions of sideslip angle; in these cases, the data spread is evident just for pitch, while for roll and yaw moments data are much more similar with each other despite different numerical approaches. Figure 28 shows the sensitivity to the angle of attack; it is evident that, for an angle of attack 0°, the numerical result exactly overlaps the experimental one. This is the last result, computed in static condition, showed here. Figures 29 and 30 report dynamic sensitivities with respect to the pitch rate in the longitudinal plane, respectively for lift and pitch moment. Even in this case different numerical approaches lead to a wide range of results for the same configuration. Dynamic derivatives with respect to roll and yaw rates are shown in figures 31 to 36. Numerical data are in the latest cases much closer to experimental values and their spread is reduced to a minimum.

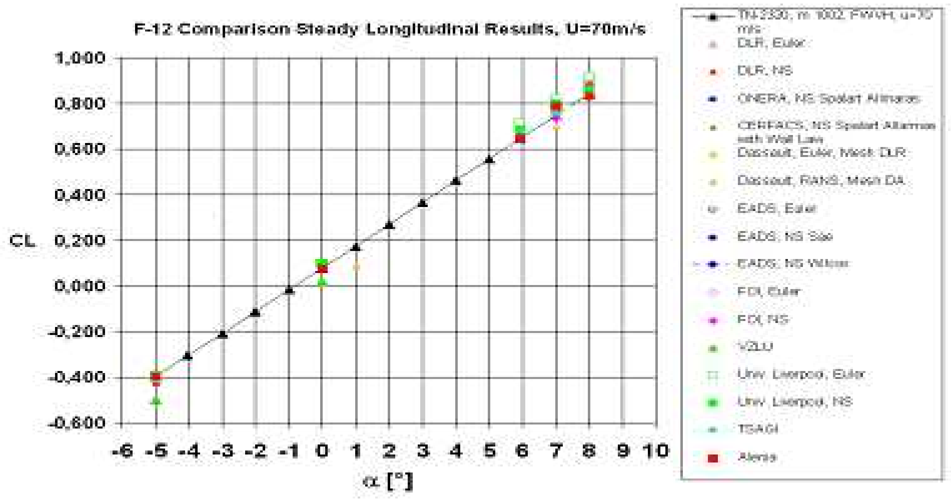

Fig. 22. CL Comparison – Numerical vs. Experimental Results.

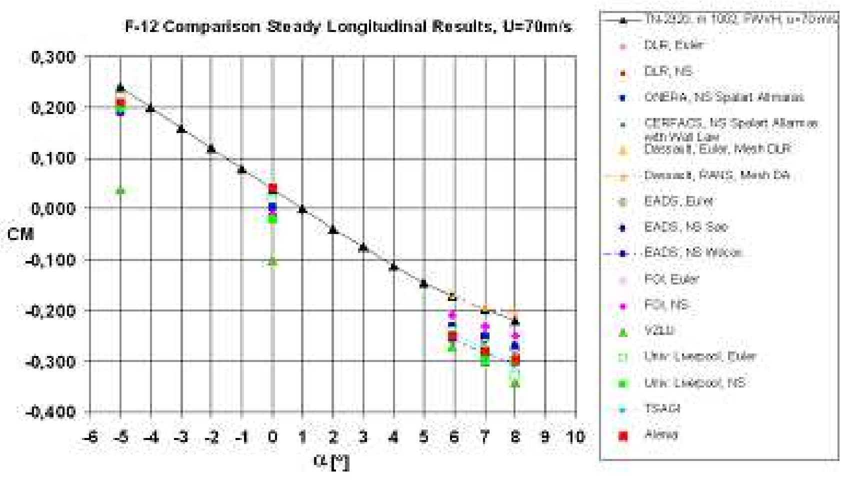

Fig. 23. CM Comparison – Numerical vs. Experimental Results.

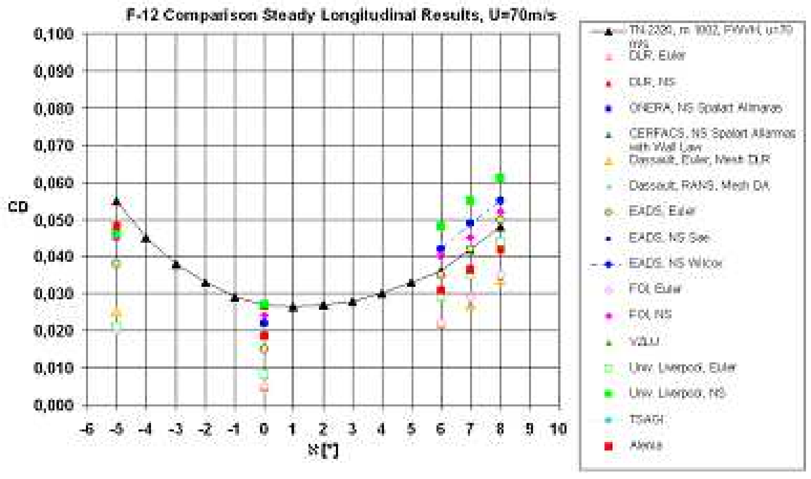

Fig. 24. CD Comparison – Numerical vs. Experimental Results.

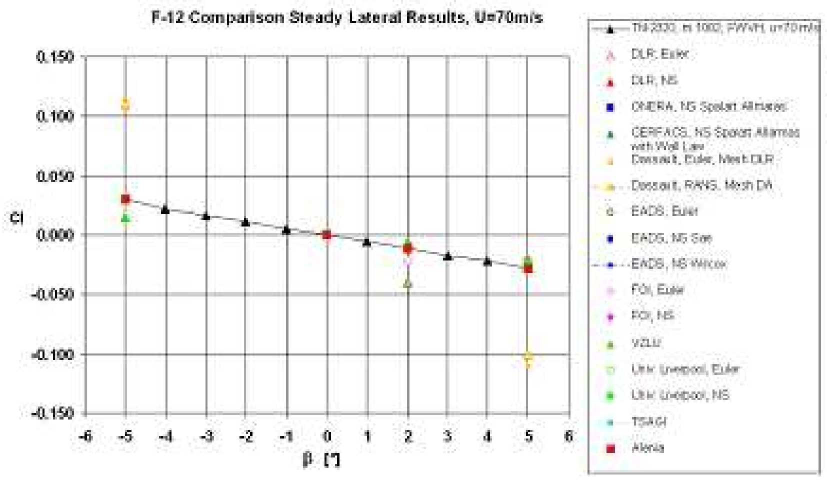

Fig. 25. Cl Comparison – Numerical vs. Experimental Results.

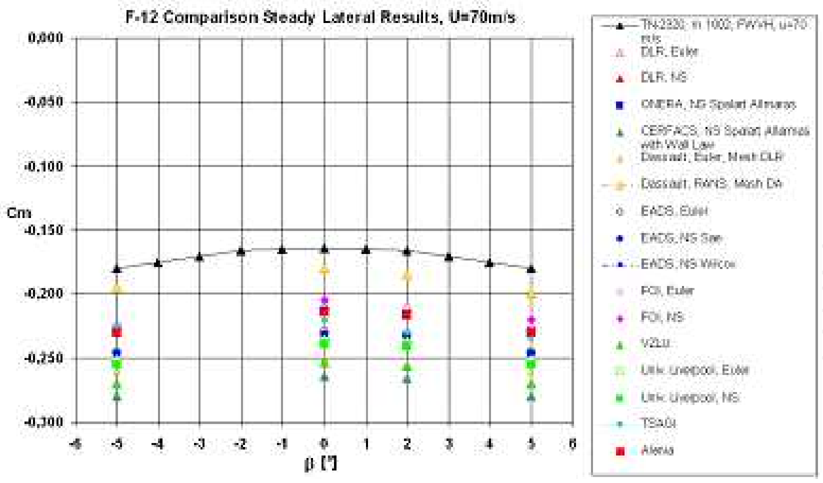

Fig. 26. Cm Comparison – Numerical vs. Experimental Results.

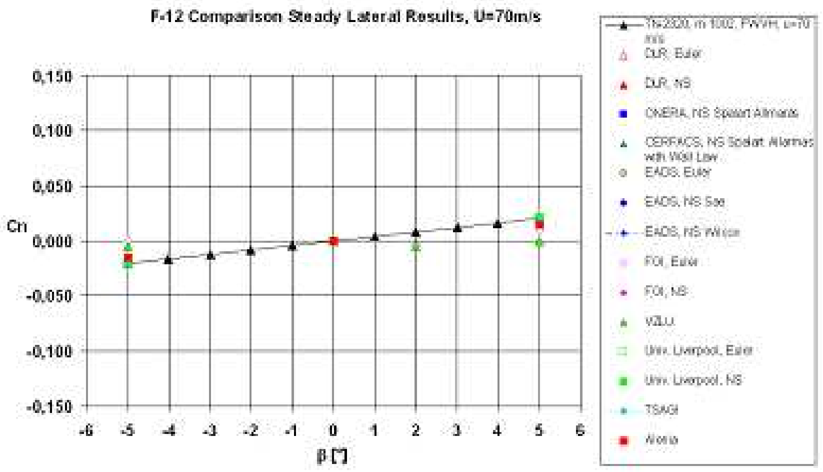

Fig. 27. Cn Comparison – Numerical vs. Experimental Results.

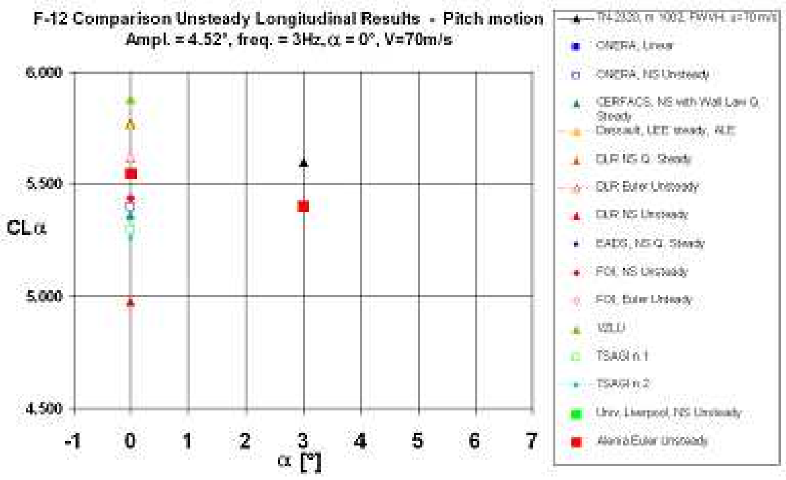

Fig. 28. CLα Comparison – Numerical vs. Experimental Results.

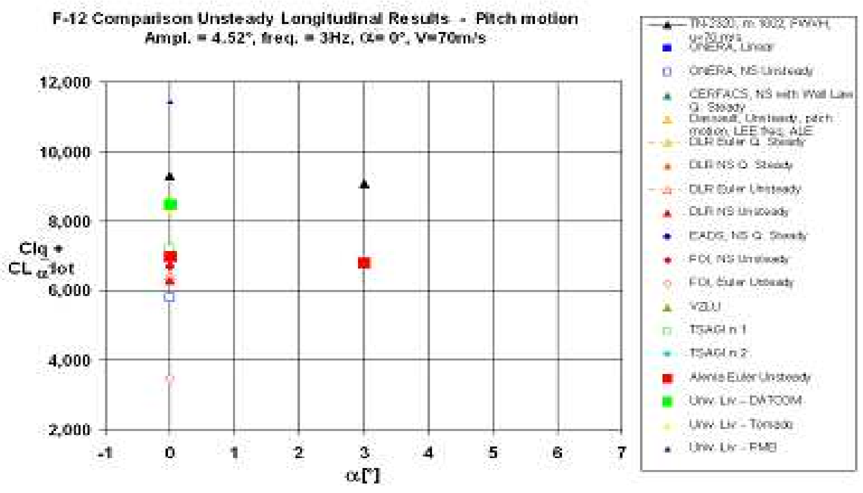

Fig. 29. CLq+CLαdot Comparison – Numerical vs. Experimental Results.

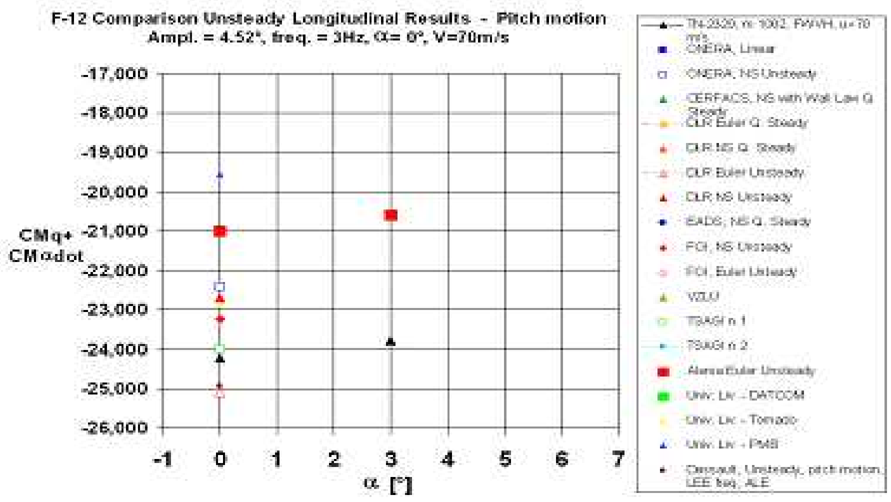

Fig. 30. CMq+CMαdot Comparison – Numerical vs. Experimental Results.

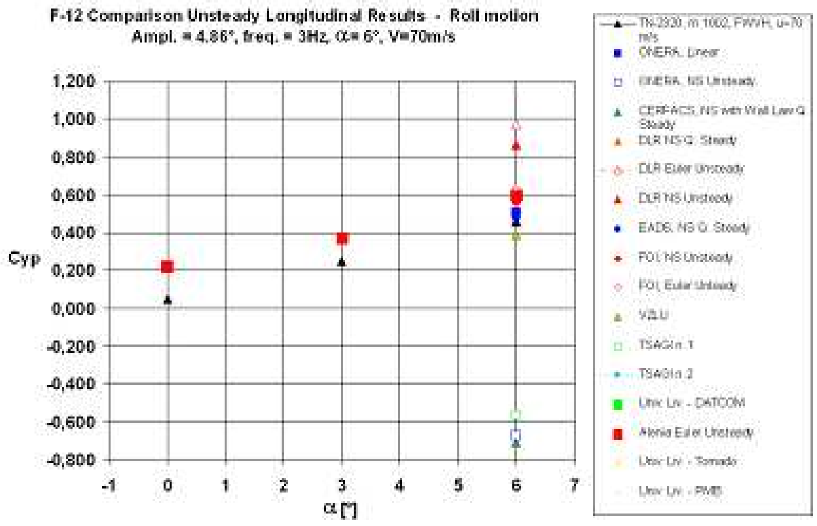

Fig. 31. Cyp Comparison – Numerical vs. Experimental Results.

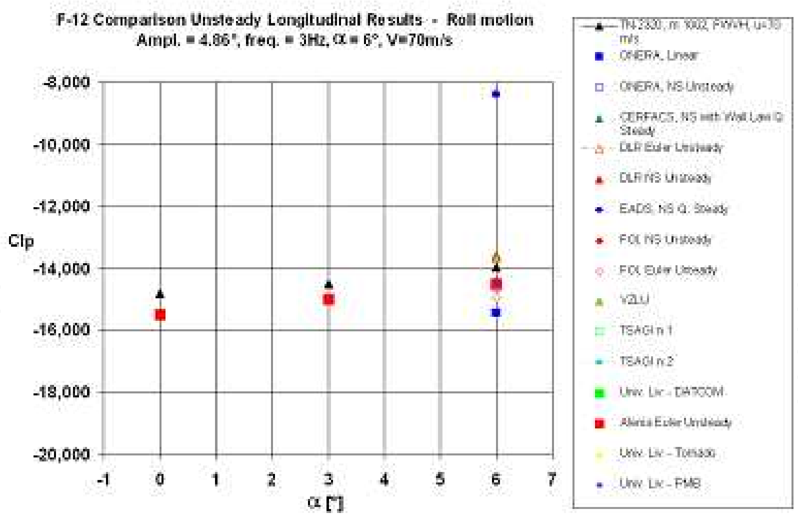

Fig. 32. Clp Comparison – Numerical vs. Experimental Results.

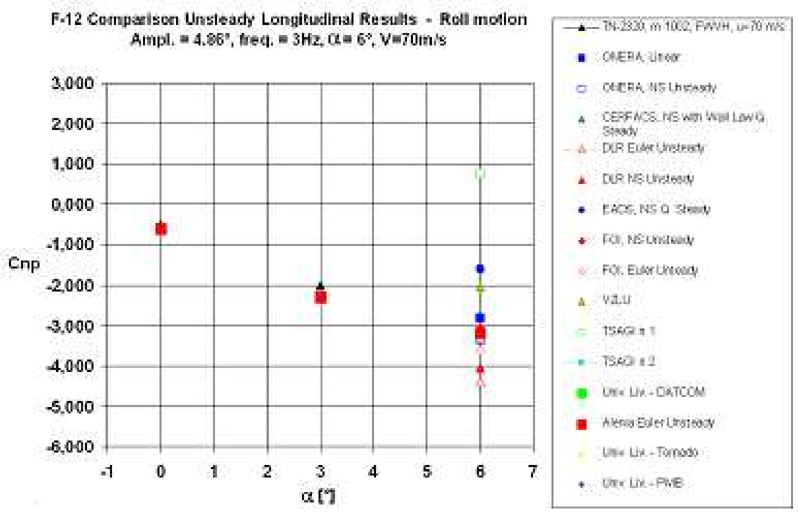

Fig. 33. Cnp Comparison – Numerical vs. Experimental Results.

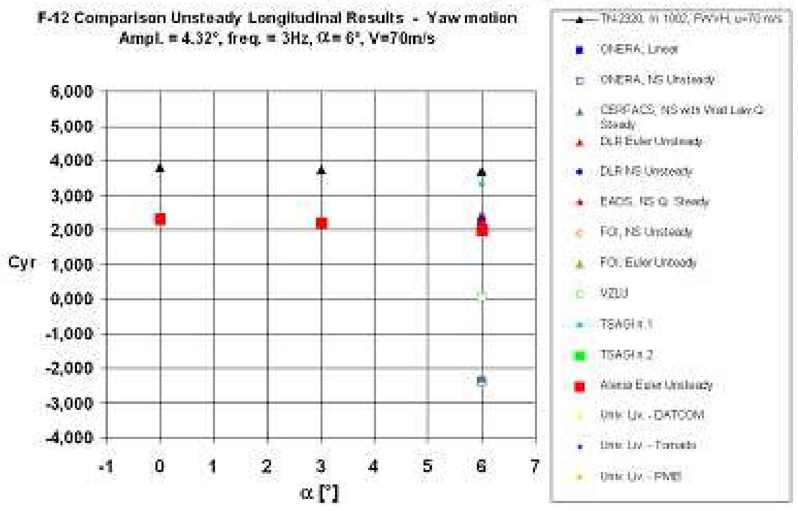

Fig. 34. Cyr Comparison – Numerical vs. Experimental Results.

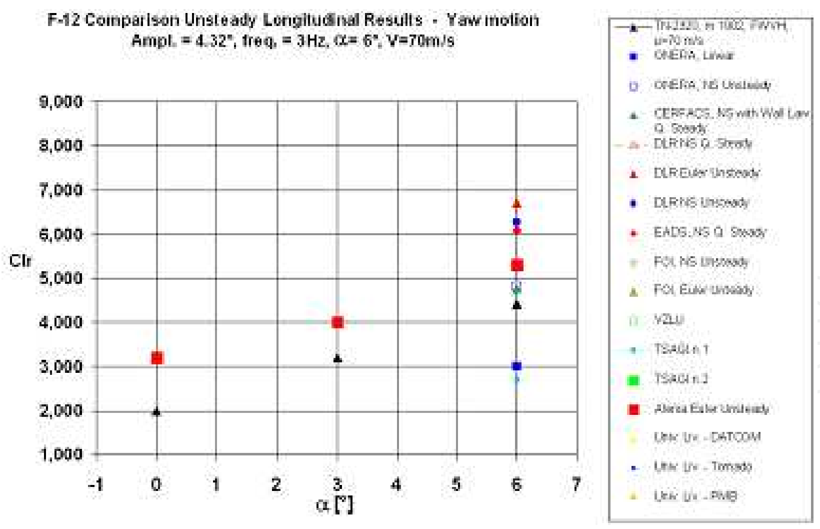

Fig. 35. Clr Comparison – Numerical vs. Experimental Results.

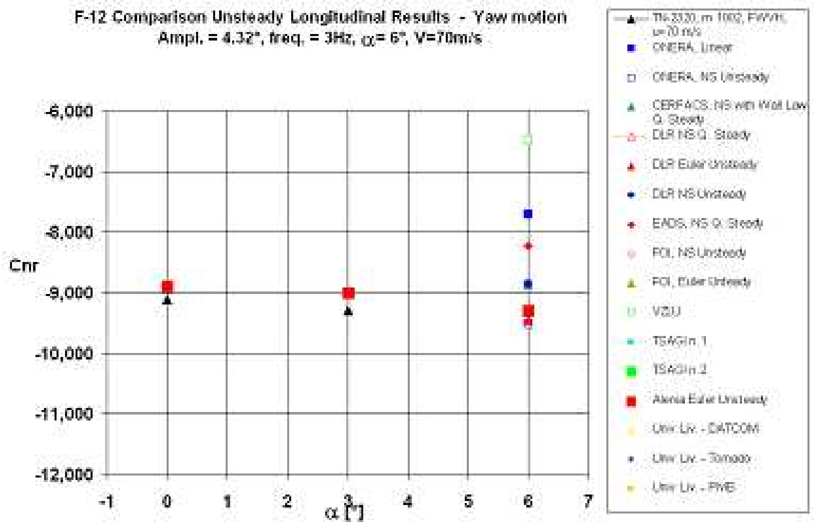

Fig. 36. Cnr Comparison – Numerical vs. Experimental Results.

5. Machine in use

All the runs have been carried out on a Quadrics multi-processor platform. The CPU is the AuthenticAMD working at 2594MHz. The system is made up of 7 computational nodes +1 node dedicated to the file system access. Each node has 4 processors. The physical memory of each node is 3943 MB and the virtual one is 8189 MB.

6. Conclusion

Automatic differentiation in static and dynamic condition has shown to be reliable for industrial application. Even if a comparison with the FD technique may be enough to qualify the AD approach, the final and definitive confirmation comes from the comparison with experimental data. According to what has been achieved, one can say that:

- AD is reliable;
- Alenia Aeronautica approach is reliable;
- AD provides a good alternative to the methods used by other Companies or Research Centres.

Following another work of the Authors (Ref. [12]) it is confirmed that a software reengineering activity is necessary after having generated the augmented code. This implies a cost in terms of time and deployed effort, for optimizing the augmented code lines, memory allocations and splitting the code in several components. Time saving achieved with AD is indeed remarkable if compared to other classical means to evaluate derivatives, and it provides an evaluation of the exact derivatives avoiding problems related to mesh refinement. Static and dynamic differencing procedures are clear enough to allow a daily use of AD features in the daily industrial activities. An extensive testing is now ongoing at Alenia premises in Turin in order to investigate:

- a faster procedure to obtain dynamic derivatives, avoiding three different computing phases;
- the application of AD for computing derivatives of higher order;
- the extension of AD to turbulence.

Further experiments related to the shape optimization, mesh adaptation and consequent resizing are currently in progress and are very promising.

7. Acknowledgments

The present work has been performed through a close cooperation between Politecnico di Torino, SimSAC project under funding from the Sixth Framework program of the European Union and Alenia Aeronautica S.p.A and through the use of Confidential Information and Data, property of Alenia Aeronautica S.p.A. which remains the sole owner of all such relevant IP rights.
The results of the present work shall be, therefore, property of Alenia Aeronautica S.p.A.

8. References

Almosnino, D. Aerodynamic Calculations of the Standard Dynamic Model in Pitch and Roll Oscillations, 32nd Aerospace Sciences Meeting & Exhibit, AIAA Paper 94-0287, January 1994.

Altun, M. & Iyigun, I. Dynamic Stability Derivatives of a Manoeuvring Combat Aircraft Model, Journal of Aeronautics and Space Technology, January 2004, Vol. I, No. 3, pp. 19-27.

Anderson, W. K., Newmann, J. C., Whitfield, D. L. & Nielsen, E. J. Sensitivity Analysis for the Navier-Stokes Equations on Unstructured Meshes Using Complex Variables, AIAA Paper 99-3294, June 1999.

Anderson, D. A., Tannehill, J. C. & Pletcher R. H., Computational Fluid Mechanics and Heat Transfer, 1st edition, McGraw-Hill, New York, 1984, pp. 39-46.

Guglieri, G. & Quagliotti, F. B. Dynamic Stability Derivatives Evaluation in a Low Speed Wind Tunnel, Journal of Aircraft, Vol. 30, No. 3, pp. 421-423, 1993.

Guglieri, G., Quagliotti & F. B., Scarabelli, P. L. Static and Oscillatory Experiments on the SDM at Politecnico di Torino, Nota Scientifica e Tecnica n. 74/93, 1993, Forced

Oscillation Technique-Reference Documentation, Vol. 3, Politecnico di Torino, DIASP.

Murphy, P. C. & Klein, V. Estimation of Aircraft Unsteady Aerodynamic Parameters from Dynamic Wind Tunnel Testing, AIAA Paper 2001-4016, 2001.

Newmann, J. C., Anderson, W. K. & Whitfield, D. L. Multidisciplinary Sensitivity Derivatives Using Complex Variables, MSSU-COE-ERC-98-09, Engineering Research Centre Report, Mississippi State University, July 1998.

Green, L., Spence, A. & Murphy, P. Computational Methods for Dynamics Stability and Control Derivatives, AIAA Paper 2004-0015, January 2004.

Hascoet, L. Analyses Statiques et Transformation de Programmes: de la parallélisation à la différentiation, Inria Database [online database], 2005, pp. 163-169, http://www-sop.inria.fr/members/Laurent.Hascoet/papers/ hdrHascoet5.html

Hascoet, L. & Pascual, L. Tapenade 2.1 User Guide, Hal-Inria Database [online database], September 2004, pp. 8-13, http://hal.inria.fr/docs/00/06/98/80/PDF/RT-0300.pdf [retrieved 19 May 2006].

Necci, C., Ceresola, N., Guglieri & G., Quagliotti, F. Industrial Computational Fluid Dynamics Tools for the Evaluation of Aerodynamic Coefficients, Journal of Aircraft Vol. 46, November-December 2009, pp. 1973-1983.

Selmin, V. Application of Automatic Differentiation to Aerodynamic Shape Optimization, ECCOMAS 2004, [online database], http://www.imamod.ru/~serge/arc/conf/ECCOMAS-2004/ECCOMAS_V2/ proceedings/pdf/611.pdf, [retrieved 28 July 2004]