JOURNAL OF AIRCRAFT
Vol. 46, No. 6, November–December 2009

Industrial Computational Fluid Dynamics Tools for the Evaluation of Aerodynamic Coefficients

Carlo Necci* and Nicola Ceresola†
Alenia Aeronautica, 10146 Turin, Italy
and
Giorgio Guglieri‡ and Fulvia Quagliotti§
Politecnico di Torino, 10129 Turin, Italy

DOI: 10.2514/1.42547

An automatic differentiation technique is applied here to an industrial computational fluid dynamics code with the goal of efficiently evaluating aerodynamic coefficients. Basic concepts about the numerical technique to automatically differentiate a code are first discussed, and then methodologies to apply the capabilities of the Institut National de Recherche en Informatique et en Automatique derivation tool to an industrial finite volume solver are discussed. Afterward, two examples of application to representative two-dimensional and three-dimensional test cases are shown and, for each one, comparisons are carried out between the results gathered using the automatic differentiation and the classical second-order finite difference technique for the evaluation of derivatives. Such comparison highlights the capability of automatic differentiation to produce reliable results even if applied to complex configurations and in very complex numerical structures. Finally, tradeoff considerations are made as a conclusion.

Nomenclature

C_D	=	drag coefficient of the complex configuration
$C_{D\alpha}$	=	drag coefficient derivative vs incidence of the complex configuration
C_L	=	lift coefficient of the complex configuration
$C_{L\alpha}$	=	lift coefficient derivative vs incidence of the complex configuration
C_{Mx}	=	roll moment coefficient of the complex configuration
$C_{Mx\alpha}$	=	roll moment derivative vs incidence of the complex configuration
C_{My}	=	pitch moment coefficient of the complex configuration
$C_{My\alpha}$	=	pitch moment derivative vs incidence of the complex configuration
C_{Mz}	=	yaw moment coefficient of the complex configuration
$C_{Mz\alpha}$	=	yaw moment derivative vs incidence of the complex configuration
C_d	=	drag coefficient of the airfoil
$C_{d\alpha}$	=	drag coefficient derivative vs incidence of the airfoil
C_l	=	lift coefficient of the airfoil
$C_{l\alpha}$	=	lift coefficient derivative vs incidence of the airfoil
C_{mx}	=	roll moment coefficient of the airfoil
$C_{mx\alpha}$	=	roll moment derivative vs incidence of the airfoil
C_{my}	=	pitch moment coefficient of the airfoil
$C_{my\alpha}$	=	pitch moment derivative vs incidence of the airfoil
C_{mz}	=	yaw moment coefficient of the airfoil
$C_{mz\alpha}$	=	yaw moment derivative vs incidence of the airfoil
α	=	angle of incidence
β	=	angle of sideslip

Received 4 December 2008; accepted for publication 3 August 2009.

*Ph.D. Student, Research Engineer, CFD Group, Numerical Simulation Laboratory, Corso Marche 41; cnecci@aeronautica.alenia.it.

†Principal Research Engineer, CFD Group, Numerical Simulation Laboratory, Corso Marche 41; nceresola@aeronautica.alenia.it.

‡Professor, Dipartimento di Ingegneria Aeronautica e Spaziale, Corso Duca degli Abruzzi 24; giorgio.guglieri@polito.it. Member AIAA.

§Professor, Dipartimento di Ingegneria Aeronautica e Spaziale, Corso Duca degli Abruzzi 24; fulvia.quagliotti@polito.it. Fellow AIAA.

Introduction

AERODYNAMIC computation widely involves derivatives of overall forces and moments with respect to flight parameters and design variables; these derivatives strongly affect the static and dynamic behavior of the airplane, thus making their importance fundamental from the design point of view, but their evaluation is not a trivial problem. In the past decades, the design and optimization activity, including the study of aerodynamic coefficients and related derivatives, was carried out using different conceptual approaches, which included the experience of the researcher. But scientifically based tools were not available, thus the sensitivity coming from the experience of the designer was fundamental in gathering good results and, despite all the limitations, results were good. To see this, one can think about supersonic cruisers such as the Concorde or relaxed stability airplanes such as the F-16, both studied and developed with an extensive use of empirical approaches. Nowadays, the human component is still the most important and powerful tool to optimize the design of a flying vehicle, but computational techniques have progressed and matured enough to provide a design team with increased numerical capabilities. These new features can be used both to reduce the human workload and to speed up the iterative process of optimization, also avoiding uncertainties that affected the work of the last-century designers. An important step in this evolution has been the complex variable differentiation (CVD) method. Conceptually, it used a series expansion of involved functions and a complex perturbation of terms. In this way, it split up the real and imaginary parts of the function expansion and allowed a separate analysis of terms for first and second derivatives. As described in [1], CVD approach showed to be reliable and to have several advantages such as implementation and maintenance simplicity and also reliability when managing turbulent flows, despite a couple of disadvantages related to memory usage and execution time. Another approach, automatic differentiation (AD), has been introduced, tested, and then applied by some aerospace companies and research centers, both in Europe and in the United States. Formally speaking, AD can be seen as a mathematical overstructure, based on solid and well-tested numerical bases, able to provide a new way to work on the software and autonomously solve the problem of creating new functions to calculate derivatives. As we will see, the term *automatic* is just valid for a part of the activity, surely not for all of it; the human factor in software reengineering is still necessary and, given the state of the art, it cannot be skipped, thus highlighting that man is still the key node of the development and optimization process. When a computational fluid dynamics source code $P \in \Re^m$ has been properly developed and optimized and each function $Y = F(X) \in \Re^n$ has been discretized both in space and in time, the use of an AD tool provides the new code lines for the derivation that, given X, compute the derivatives $F'(X)$ and do not require the user or the developer to go back to the discretization phase. This means that a new (augmented) source code will be generated by the tool, embodying both the original code lines and the new lines that calculate the functions and subfunctions necessary for the derivation. AD software works on a source code as a transformation tool does, which means in a way similar to a compiler and, as such, it has to be considered as a tool able to ease the work but not to completely avoid it. Green et al. [2] gathered good results using the ADIFOR version 2.0 and version 3.0 tool to, respectively, derive a FORTRAN77 code in forward and reverse modes. They applied their work to an F-16XL stability and control study and showed the possibilities of automatic differentiation to reduce design costs by shortening the design phase and reducing wind-tunnel testing by application of AD to calculate static and dynamic derivatives in a numerical way. Almosnino [3] showed a way to calculate aerodynamic characteristics, even if not using automatic differentiation, using a panel method on an F-16, thus highlighting the capability to shorten the design process using a different numerical approach. Our goal is to be able to calculate aerodynamic derivatives using industrial finite volume software, applicable to CATIA models, augmented by the automatic differentiation methodology. The AD tool we speak about in this paper (namely, Tapenade), has been developed by the Institut National de Recherche en Informatique et en Automatique (INRIA) and, in a single version, embodies the capability to derive in direct and reverse modes. Alenia Aeronautica Numerical Research Laboratory successfully applied Tapenade features to its finite volume unstructured (UNS3-D) solver code and carried out a testing phase to evaluate the new capabilities thus earned. This paper recaps the base concepts over which automatic differentiation is built, what Alenia people learned from the use of Tapenade, and what has been reapplied so far as a new technology. Conclusions will describe the activity still ongoing and the targets for the future.

AD Concept

Let $X \in \Re^n$ be a vector argument, and let $Y = F(X) \in \Re^m$ be a corresponding vector function. Equation $F_m = F_m(x_1, x_2, \ldots, x_n)$ expresses the concept to have m functions depending on n variables. Using a computer program P to evaluate Y, it has to be considered that P evaluates only simple functions; this means that the generic F_m has to be split up into several subfunctions f_k, each one being implemented by a corresponding instruction I_k in P. With F given by the composition $F = f_p \cdot f_{p-1} \cdot \ldots \cdot f_1$, corresponding source code has to be a sequence $P = \{I_1; I_2; \ldots; I_p\}$. As a mathematical limitation, code-implemented functions have to be differentiable, despite some exceptions (e.g., the square root); in general functions implemented by arithmetic, operations are indeed differentiable. AD tool works using the chain rule to evaluate derivatives according to the following expression:

$$F'(X) = \underbrace{f'_p(X_{p-1}) \times f'_{p-1}(X_{p-2}) \times \cdots \times f'_1(X_0)}_{J} \qquad (1)$$

where $X_k = f_k(X_{k-1})$ is the value of a generic subfunction, and $X_0 = X$ the first value. J is hence the Jacobian of F.

Equation (1) shows derivatives that can be translated back into a sequence of instructions I'_k, coded, and inserted back into a copy of the control program P; this new set of instructions will be added to P, yielding program P' (see [4]), which now has instructions for both primitive and derivative functions. Conceptually, AD works on some basic assumptions: First, P is considered simply as a runtime sequence of instructions and, actually, AD differentiates this sequence. Second, any sequence is a composition of vector functions, with each one assumed to be differentiable. This means that with each function differentiable, we have the theoretical base to derive according to the fundamental mathematical rule, deriving not only the sequence but also single functions. Furthermore, we now have to observe that the generic function $F_m = F_m(x_1, x_2, \ldots, x_n)$ depends on n terms, so derivation of a single function implies n derivatives. Applying this concept to a vector function, it comes evident that Eq. (1) is a Jacobian and, as such, it requires multiplying matrixes with matrixes; this opens another issue about how to manage computation. The concept of derivation in direct (tangent) or backward (reverse) mode stems from this.

Physically, we often just need to evaluate sensibility of a quantity over a design parameter; an example is the sensibility of lift coefficient with respect to incidence ($C_{L\alpha}$) or lateral force on sideslip ($C_{Y\beta}$). Mathematically, this means projecting a function (total derivative) onto a certain direction (design parameter), as shown in Fig. 1.

This implies that with a generic n components base over which derived function does exist, directional derivative is the projection of the function along $\dot{X}$ component of the base. Formally speaking, one has to compute

$$F'(X) \cdot \dot{X} = f'_p(X_{p-1}) \times f'_{p-1}(X_{p-2}) \times \cdots \times f'_1(X_0) \cdot \dot{X} \qquad (2)$$

where $F'(X)$ is an $\{m \times n\}$ matrix; the most efficient way to evaluate it is from right to left, which implies multiplying a matrix by a vector. The method [5] requires starting the derivation from the first instructions $f'_1(X_0)$ in the source code and then progressing on; this makes things easier from the algorithmic point of view, as it allows enriching the original source code using the lines of derived functions after each corresponding primitive instruction. Such a concept is the base of direct (or tangent) derivation.

Conversely, in optimization processes or inverse problems, it is required to minimize a generic cost function with respect to a number of design parameters (e.g., shape parameters); this means deriving a function with respect to a number of parameters (i.e., deriving a function over a vector), and this implies evaluating a gradient $\partial F/\partial X$. A gradient is applicable to scalar functions, and thus a linear combination is necessary to get a scalar. With $Y = F(X) \in \Re^m$ being generically a vector, one can weight each of vector terms with a corresponding weighting coefficient to get a scalar. Mathematically, this implies transforming the $Y = F(X)$ vector into a row (e.g., using the transposition) and then multiplying this row by a weight vector $\bar{F}(m)$, which clearly has to be an input. Formally, the product becomes $F^T(X) \cdot \bar{F}$ and its gradient is $F'^T(X) \cdot \bar{F}$; transposing F' means transposing the product $F' = f'_p \times f'_{p-1} \times \cdots \times f'_1$, and remembering that $(A \times B)^T = B^T \times A^T$, we get

$$F'^T = (f'_p \times f'_{p-1} \times \cdots \times f'_1)^T = f'^T_1 \times f'^T_2 \times \cdots \times f'^T_p$$

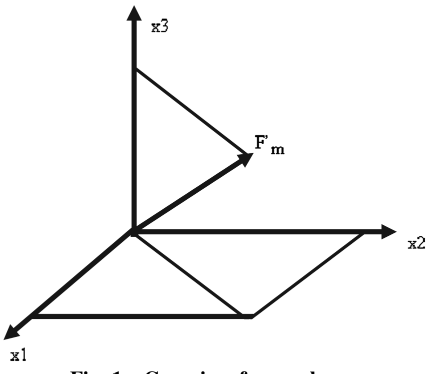


Fig. 1 Generic reference base.

from which

$$\frac{\partial F}{\partial X} = F'^T \times \bar{F} = \underbrace{f_1'^T(X_0) \times f_2'^T(X_1) \times \cdots \times f_p'^T(X_{p-1})}_{J^T} \times \bar{F} \quad (3)$$

where $F'^T\{n \times m\}$ is the transposed Jacobian. 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 the second case, computation starts deriving the last pth function and then going back to the first function; from this comes the definition of backward (or inverse) derivation.

A general rule to select the best derivation method comes from the following observation; as low as the number of rows in the right-hand side multiplying vectors ($\dot{X}$ for direct mode and $\bar{F}$ for backward) is as low as the number of operations will be. For direct mode, short $\dot{X}$ implies small n (i.e., a few parameters above which to derive); thus, direct mode can be computationally effective when deriving a number of functions with respect to a few parameters. Conversely, for backward mode, short $\bar{F}$ implies small m, which means a few functions to derive; this is attractive when deriving with respect to a number of parameters (i.e., in the case of optimization problems).

Numerical Approach

Numerical differentiation both in tangent and in reverse modes implies the numerical definition of derivatives; taking into account a generic solution function $\underline{U}(\underline{x}, \underline{\eta}_{ij}, \underline{\eta}_i^b) = 0$ depending on the position vector, normal vectors, and boundary conditions, sensitivity of the aerodynamic solution to each parameter can be expressed as (see [6])

$$\frac{\partial \underline{U}}{\partial \beta} = \frac{\partial \underline{U}}{\partial \underline{x}}\frac{\partial \underline{x}}{\partial \beta} + \frac{\partial \underline{U}}{\partial \underline{\eta}_{ij}}\frac{\partial \underline{\eta}_{ij}}{\partial \beta} + \frac{\partial \underline{U}}{\partial \underline{\eta}_i^b}\frac{\partial \underline{\eta}_i^b}{\partial \beta} \quad (4)$$

which shows how the AD tool will produce solution derivatives with respect to (virtually) all design parameters. This highlights two issues:

1) When needing just a subset of derivatives (i.e., when we just need sensibilities with respect to a parameter), relevant parameter β has to be set to 1 before entering derivation routines, and all other parameters have to be set to 0; this can be done directly into the code or using an external input file.

2) In order to reduce computational costs, derivation could be properly carried out just after a steady-state solution has been reached, thus avoiding the iteration of derivatives. In this way, applying derived functions to a converged solution will imply one only further calculation run.

Despite conceptual simplicity and elegance of such an optimized approach, real implementation is very challenging, especially for complex industrial tools. To explore the problem, we can see some different structures of the solution and derivation (shortly augmented) algorithm that can be used. In structure 1 (Fig. 2), the calculation of derivatives is carried out in the steady loop; for each iteration, both primitive and derived parameters are evaluated. A first phase of checks requires guaranteeing a complete adherence of managed data structure to the solving block we are going to use. After this, the parameter over which we want to derive has to be set to one; after this, the augmented algorithm can run. The benefit of this configuration is in structure simplicity, but it requires computing derivatives even in the steady iterative loop, thus computing more functions per iteration and increasing the runtime cost.

A possible different approach is shown in structure 2; the preliminary checking phase is the same as structure 1, but in structure 2 derivatives are computed in the steady loop just after the primitive (not derived) solution has converged, thus requiring just one more iteration. In this way, the numerical benefit coming from the derivation is paid with a very low increase of the computational cost, virtually just one more computation, thus carrying to a very high numerical efficiency. At this point, we have to remember that the AD tool provides the capability to produce derivatives with respect to one

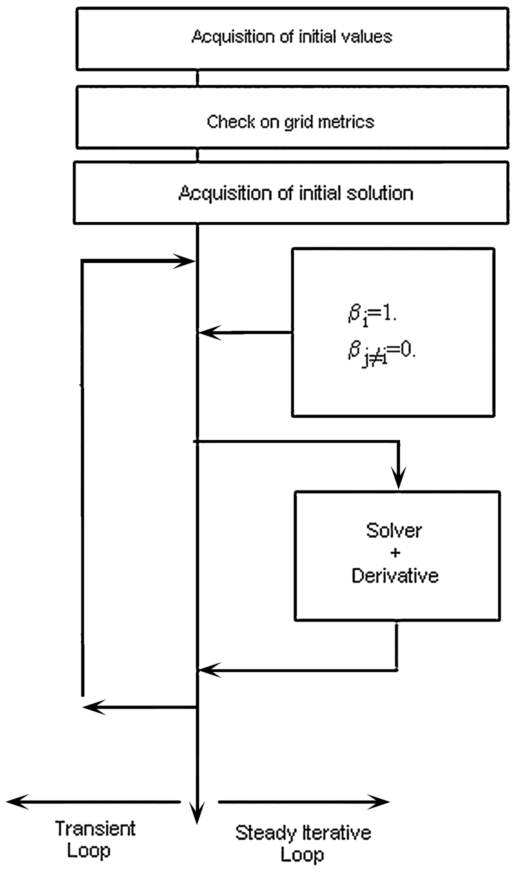


Fig. 2 Structure 1.

or more parameters. Surely, derivation with respect to several parameters involves a higher complexity of the augmented software due to the higher number of instructions to manage, but, conceptually speaking, structures 1 and 2 can be used interchangeably to derive one or more parameters.

One possible alternative is shown in structure 3. In the final case, a number of derivation blocks can be cascaded, each block modeling the derivation with respect to one only parameter; then, iteratively, all the blocks can be run in sequence. It is clearly necessary to loop the cycle in such a way to reset to one the proper β_i parameter before calling each block, but the benefit comes from the capability to evaluate all the derivatives with respect to β_i with just one iteration each, as in structure 2.

However, compared to structure 2 (Fig. 3), in structure 3 (Fig. 4), we have a pro and a con. The pro is that structure 2 manages a larger module, involving several derivatives; using only one block makes it easier to link it with the rest of the solver. The con is that, on the other side, if we need to again derive one derivative with respect to another parameter, structure 2 makes the work very hard. Conversely, structure 3 follows a different modularization approach; this implies more work to integrate each derivation module in the overall structure, but guarantees an higher flexibility in case of further reworking. A practical difficulty is evident in the structure 3 approach: although in the first two structures, the software has to be optimized for just one set of augmented variables, in structure 3, optimization is surely longer and more complex, due to the high number of augmented parameters managed separately. In this case, the modularity of primitive code and software engineering skills of the user make the difference. A possibility is that the parameters to be set to 1 (i.e., those over which we want to derive) are read from an input file and 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 highlighting that whatever the user wants to do, he needs a complete control of the software. In order to explain in more detail the process applied by Alenia Aeronautica to collect results discussed in this paper, we can describe the practical way the augmented code has been achieved. A first block of the primitive code (i.e., the one containing memory allocations) has been split from the resolution block to reduce the number of lines to be manipulated. After this, the solution block has been automatically derived. Internal structure of the block was modular and different subroutines responsible for calculation of convective terms, viscous stress, residual averaging, and turbulence model underwent the derivation. For each routine, AD tool provided the corresponding derived routine. The following phase dealt with the rearrangement of routine calls; formally speaking, it would be possible to compute not only inviscid derived parameters but also the corresponding viscid terms and turbulence characteristics. But the experience gathered during the testing phase showed that such an overcomplexity of instructions can carry to unpredictable behaviors of the software, easily implying numerical instability. Therefore, the call to derived routines has been limited to those that do not affect stability, leaving, for example, the turbulence calculation out of the derivation loop. In addition to this, as already said, the original software was structured using dynamic memory allocation, and AD tool provided a list of all the derived parameters that had to be dimensioned, with functions of dynamic primitives parameters. Because of this, a `DIFFSIZES.inc` file was suggested by the tool to store all the new functions, but the final decision to use this way or a different one is left to the user. We preferred using internal allocations. Furthermore, with the original software optimized for parallel computation, in phase of compilation the relevant routines had to be linked together to the augmented software and to the first (memory allocations) block. Finally, from the operative point of view, we have to remark that after the AD tool application, the new augmented source code still needs reengineering. AD provides the augmented source code with all the declarations required by the new derived functions and places the correct structure and syntax of new lines into the original listing. But, again, experience showed that each dynamically allocated variable has to be explicitly dimensioned in each subroutine in which it is used and/or recalled. This means that if variable dimensioning is implemented using an asterisk (*) after the first size declaration, AD tool will highlight the logical mismatch everywhere the variable is recalled and a list of hints will be provided. When this approach has been applied to parallelized calculation, no real difference with respect to the normal procedure has been met, but the call to message-passing-interface routines has been required and no further conceptual impact has been met.

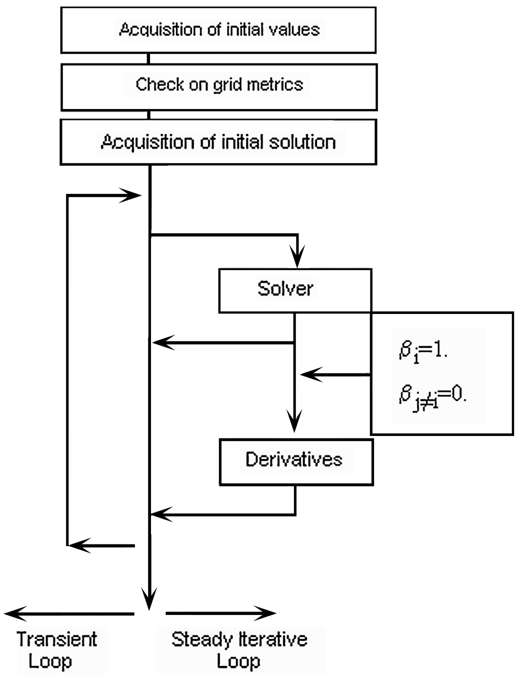


Fig. 3 Structure 2.

Numerical Applications

The following results have been gathered using the tangent mode derivation. Setting

$$H_d = \frac{\partial \underline{H}}{\partial \underline{B}_i} \tag{5}$$

$\underline{H} = \{C_L, C_D, C_{Mx}, C_{My}, C_{Mz}\}^T$ has been used as the aerodynamic coefficients vector and $\underline{B} = \{\alpha\}$ as has been used as the design variables vector. The derivatives are as follows:

$$\frac{\partial H}{\partial B_1} = \frac{\partial C_L}{\partial \alpha}, \frac{\partial C_D}{\partial \alpha}, \frac{\partial C_{Mx}}{\partial \alpha}, \frac{\partial C_{My}}{\partial \alpha}, \frac{\partial C_{Mz}}{\partial \alpha} \tag{6}$$

Two tests cases have been investigated: a RAE2822 airfoil, tested in viscid condition, and the Alenia small regional jet (SMJ) configuration studied in inviscid runs.

RAE2822 Airfoil

Working conditions explored are $M = 0.5$ and $Re = 6.2\text{e} + 06$, keeping the incidence variable. The grid used has been a C-type, with 18,604 nodes and 9540 elements (8740 hexahedra and 800 prisms), which has been generated by an Alenia proprietary mesh generator. Dimensions of the grid are as follows:

$$\Delta x = \pm 0.3\text{e} + 02 \qquad \Delta y = \pm 0.1\text{e} + 01 \qquad \Delta z = \pm 0.3\text{e} + 02$$

with the airfoil model in the center and having a unity chord length. The boundary-layer region has been meshed using 12 layers, each one having 255 nodes: 161 around the airfoil and 94 in the wake. The conceptual approach followed to use the 3-D solver on a 2-D mesh is simply the redefinition of the mesh; the bidimensional mesh is doubled onto a second layer with the same y axis and placed at a unitary distance in y direction; nodes and connectivity are the same. Then the two layers are joined together, thus generating a 3-D effective mesh usable by a 3-D solver. As already said, the overall number of nodes on the skin is 161, for 160 wall faces; furthermore,

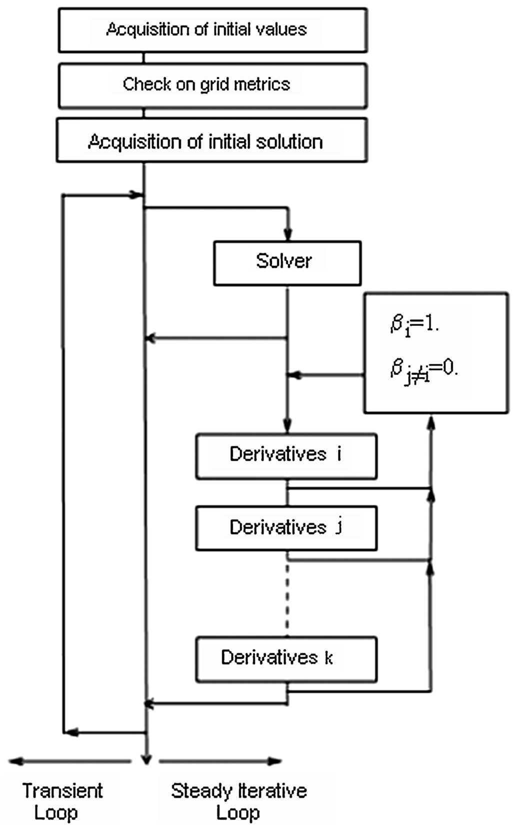


Fig. 4 Structure 3.

in order to better evaluate the wake evolution of the flow, 47 nodes have been added after the trailing edge, above and below the chord line for a total of 94 further nodes, thus carrying the total amount to 255 nodes on the first layer. Figure 5 shows a detail of the mesh focused on the airfoil, and Figs. 6 and 7 show details of, respectively, leading and trailing edges.

A first run of the solver has been carried out to evaluate the distance at which the freestream flow was completely deployed; this run used a mesh with 15,289 nodes and only five layers in the boundary-layer region. The boundary layer has been identified by using the distance from the skin at which the flow speed reaches 99% of freestream speed. Freestream velocity has been evaluated, the 99% point identified, and then boundary-layer thickness defined. Then the boundary-layer region has been enriched by eight more layers, thus carrying the total amount of layers to 13 and the final number of nodes to 18,604. Figures 6 and 7 provide two detailed views of the leading edge and of the trailing edge. Leading-edge meshing has been carried out as follows: initial spacing layer has been fixed to $0.1\mathrm{e}-02$ chord units, and from the first layer, a geometrical progression has been applied using a factor of 1.208 in the normal direction. Naturally, given the nature of the mesh, this coefficient has been applied in a normal direction along the upper and lower surfaces, until the trailing edge. As mentioned, a peculiarity for the trailing edge is that after the edge itself, further enrichment of the grid has been applied; above and under the wake line, 47 more nodes have been placed in order to better investigate the behavior of the flow in the wake area. The length of the wake region has been quantified in 8.444 chord units. In order to increase the number of points in a smart way, which means avoiding too many nodes and a rough application of them, the additional wake nodes have been distributed in a stretched order, with a numerical progression given by a 1.09 scale factor.

Numerical testing has been carried out by running 5000 iterations.The laminar Prandtl number has been set to 0.72 and the turbulent one to 0.90; both are considered suitable for normal numerical runs. The Courant–Friedrichs–Lewy (CFL) number has been set to 2.5, and no residual averaging has been necessary, given the simplicity of the test case. Total computation time has been 156.7 s on a parallel Quadrics machine, and 32 processors have been used to run the calculations. In Fig. 8, a density plot of the solution around the airfoil is sketched (for $a = 2°$).

To properly evaluate reliability of the automatic differentiation, a first step to do is verifying solution convergence, as any consideration and comparison has to be applied to steady-state solutions. Figure 9 shows convergence histories for C_l and C_d, and for completeness, Fig. 10 shows the same for corresponding derivatives. It is clear that 5000 iterations are enough to reach a steady-state solution to use as a basis for data comparison.

Relative error applied as stop criterion has been $1.0\mathrm{e}-06$ for both coefficients; the solver can be set to reach whatever residual, but in terms of efficiency, it is meaningless to shrink the desired residual too much (to less than $1.0\mathrm{e}-06$), as the number of iteration increases

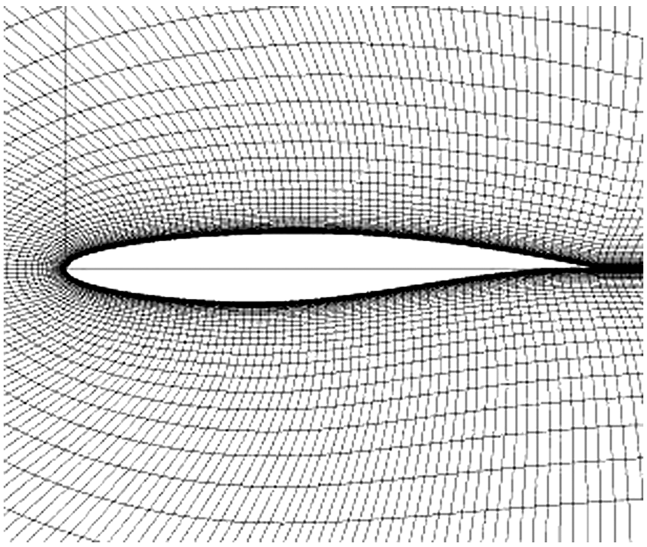


Fig. 5 Overall picture of the used RAE2822 mesh.

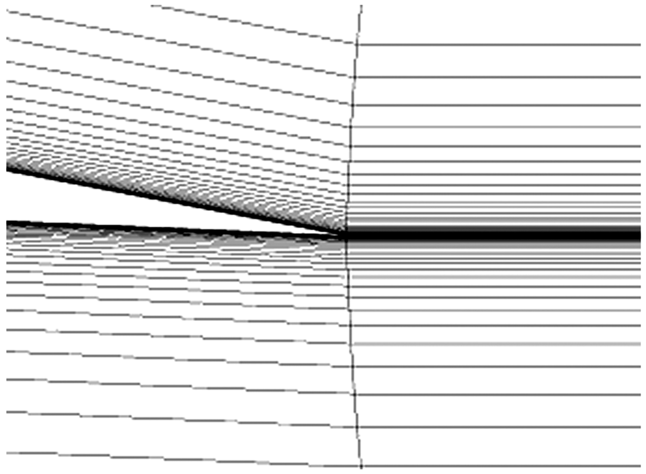


Fig. 7 Detail of the trailing edge.

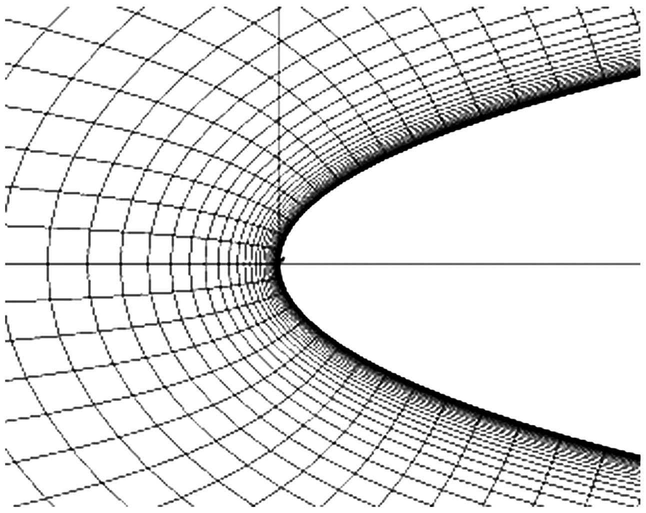


Fig. 6 Detail of the leading edge.

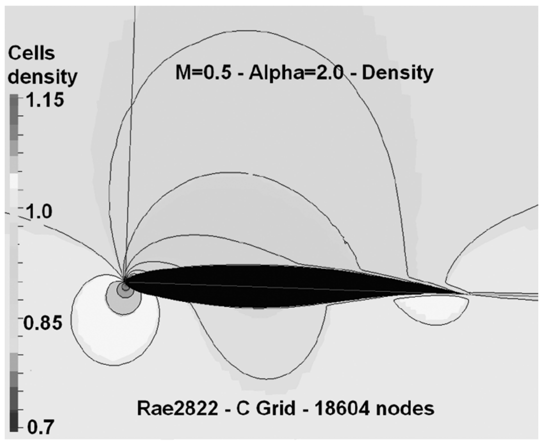
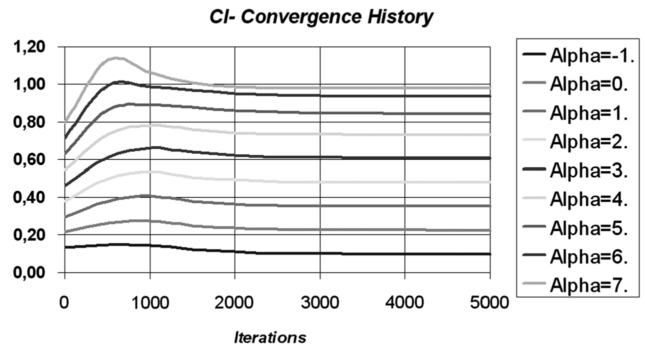


Fig. 8 Density plot of the RAE2822 working at $M = 0.5$, $\alpha = 2.0°$, and $Re = 6.2\mathrm{e}+06$.



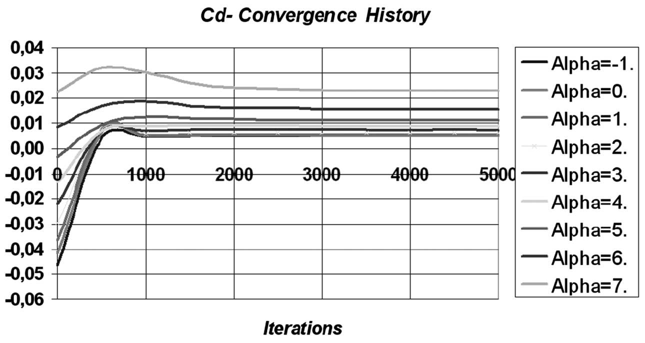


Fig. 9 Convergence history of C_l and C_d for nine different values of the incidence.

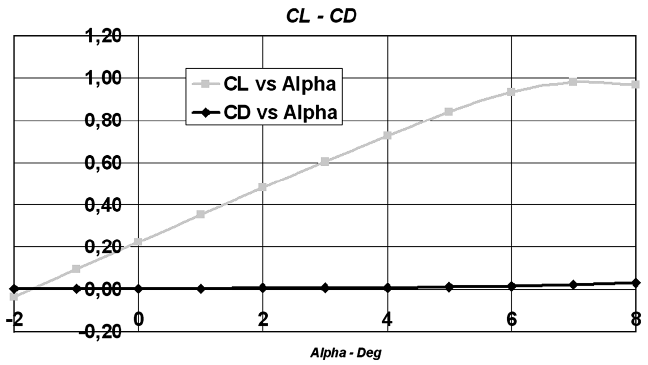


Fig. 11 Progress of C_l and C_d vs α.

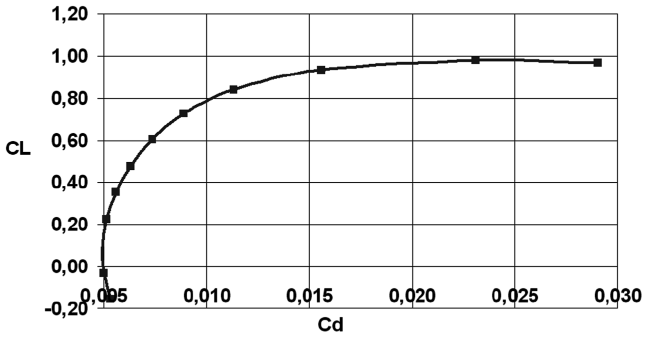


Fig. 12 C_l vs C_d.

very much without any real benefit in terms of accuracy. It has also to be highlighted that in industrial applications, acceptable errors range from 1e − 03 to 1e − 04, according to geometry complexity, aerodynamic setting, and grid refinement. Conversely, when research works are carried out (our case is an example), the error evaluation is more severe and acceptable relative error drops down to 1e − 06.

Figures 11 and 12 sketch, respectively, the C_l and C_d curves vs incidence and the polar of the airfoil. Gathered results show the classical behavior expected from the airfoil, highlighting a linear behavior until 5° of incidence; at higher angles, the linearity is lost. It is clear that, as an overall rule, 5000 iterations are enough for both primitive and derived parameter convergence; in some cases, even less is enough. In particular, the polar draw shows minimum drag area close to $\alpha = -1°$. It is now necessary to point out that running the same computation by using the primitive solver naturally gives the same results for C_l and C_d, both in terms of convergence history and in terms of achieved steady-state result (so the primitive solver convergence history is like in Fig. 9), with code lines computing primitive functions exactly the same in the primitive and augmented versions. A difference has been noted in computation time: the augmented solver is about 30% more time-expensive, because of augmented lines and numerical cross checks imposed for any iteration.

After this, we can be reasonably confident that augmented software generates good results, at least for primitive coefficients. The following step is the comparison between derivatives computed using AD and applying the finite differences (FD) approach. As a first issue, we have to note that an industrial solver, used daily for aerodynamic calculations, normally has an accuracy of the second order. Alenia software (namely, UNS3-D) applies the finite volume technique for a cell-centered method, and the final achieved accuracy is of the second order, both in time and in space. The same accuracy has been achieved for the derived solver, and this implies that the applied validation technique has to be at least second-order-accurate. A rough but effective way to validate the derived solver and relevant solution is to use the finite differences approach, able to provide a second-order accuracy. So we have to define how to reach a second-order accuracy in the FD approach, which is a very easy task; from the Taylor expansion series, we get [7]

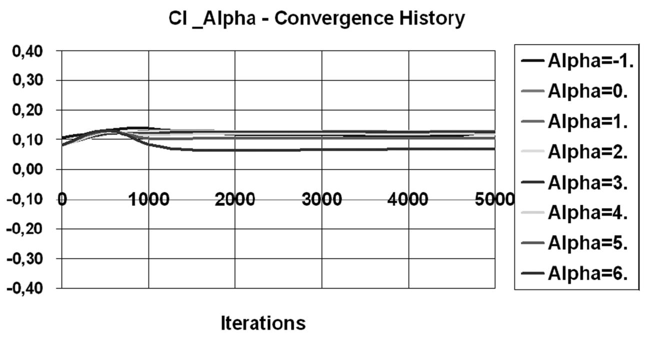
$$u(x_0 + \Delta x, y_0) = u(x_0, y_0) + \left.\frac{\partial u}{\partial x}\right|_0 \Delta x + \left.\frac{\partial^2 u}{\partial x^2}\right|_0 \frac{\Delta x^2}{2!} + o(\Delta x)^3$$

and

$$u(x_0 - \Delta x, y_0) = u(x_0, y_0) - \left.\frac{\partial u}{\partial x}\right|_0 \Delta x + \left.\frac{\partial^2 u}{\partial x^2}\right|_0 \frac{\Delta x^2}{2!} + o(\Delta x)^3$$

This carries to the classic central-difference scheme,

$$\left.\frac{\partial u}{\partial x}\right|_{i,j} = \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} + o(\Delta x)^2 \tag{7}$$



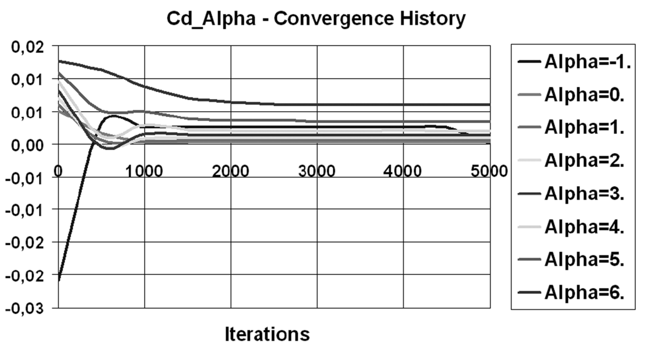
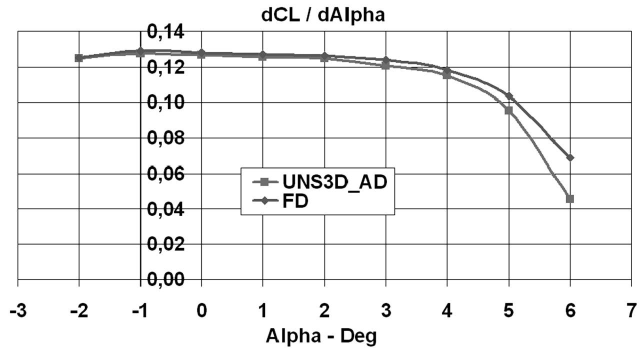


Fig. 10 Convergence history of $C_{l\alpha}$ and $C_{d\alpha}$ for the eight different values of the incidence seen above.



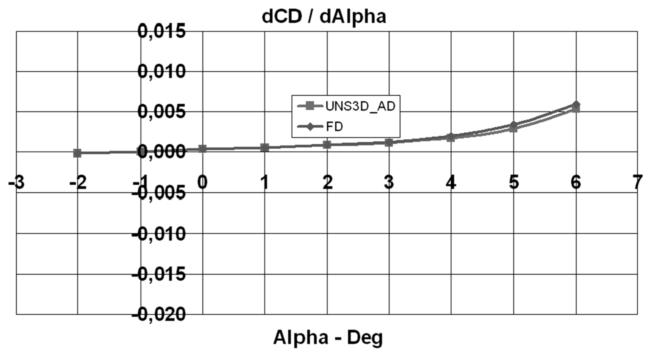


Fig. 13 $C_{l\alpha}$ and $C_{d\alpha}$ vs α comparison between centered FD scheme and AD augmented solver.

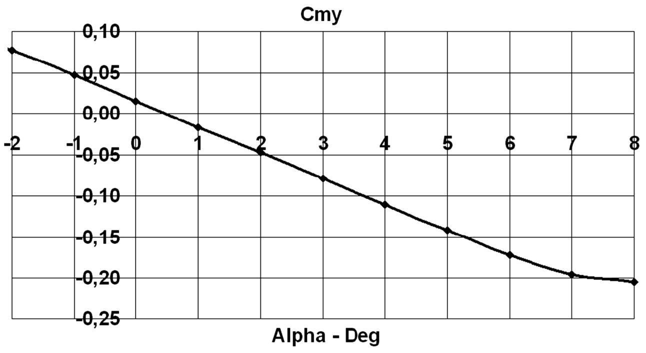


Fig. 14 C_{my} vs α evolution.

with a second-order accuracy. Testing has been carried out by running the augmented software UNS3D_AD for some incidences $\alpha = i°$ ($i \in [-2; +5]$) and comparing the results with the primitive block of results for $(i+1)°$ and $(i-1)°$. Then Eq. (7) has been applied and results of the comparison have been drawn. Figure 13 shows the output for C_{l_α} and C_{d_α}.

The numerical agreement between the two pairs of curves is evident; in the range of $-2° \leq \alpha \leq +4°$, the percentage error for C_{l_α} varies from 1 to 3%, thus showing a good agreement with FD evaluation in the linear part of the lift curve. The behavior is similar for C_{d_α}, even if the percentage error slightly changes. It is also clear that the hardest differences occur at higher incidence, when the linearity is lost. To clarify this, tables of results gathered with the two techniques, both for C_l and for C_d, are hereinafter included.

C_{l_α} and C_{d_α} values are slightly variable as long as we move along the linear part of the lift curve; entering the nonlinear area, the slopes change as expected and the AD results qualitatively follow the FDs. As we can see, for C_{l_α}, final steady value provided by UNS3D_AD at $\alpha = -2°$ is

$$\left.\frac{dC_L}{d\alpha}\right|_{\alpha=-2°}^{\text{UNS3D}_{\text{AD}}} = 1.2500\text{e}-01$$

and the corresponding value with finite differences is

$$\left.\frac{dC_L}{d\alpha}\right|_{\alpha=-2°}^{\text{FD}} = 1.2570\text{e}-01$$

thus showing a relative error of order 0.56%. By repeating the check for $\alpha = 4°$, we get, respectively,

$$\left.\frac{dC_L}{d\alpha}\right|_{\alpha=4°}^{\text{UNS3D}_{\text{AD}}} = 1.1500\text{e}-01$$

for augmented software and

$$\left.\frac{dC_L}{d\alpha}\right|_{\alpha=4°}^{\text{FD}} = 1.18500\text{e}-01$$

for FD, thus having a relative error equal to 2.95%. Similar comparisons can be carried out for C_{my}; Fig. 14 shows the evolution of C_{my} with incidence, with the airfoil centered at $x = 0.25c$ and having $c = 1.0$. Tables 1–3 provide numerical values of all derivatives.

Numerical comparison between the two different approaches shows that for $\alpha = -2°$, we get

$$\left.\frac{dC_{my}}{d\alpha}\right|_{\alpha=-2°}^{\text{UNS3D}_{\text{AD}}} = -0.2800\text{e}-01$$

and the corresponding value with finite differences is

$$\left.\frac{dC_{my}}{d\alpha}\right|_{\alpha=-2°}^{\text{FD}} = -0.2785\text{e}-01$$

Table 1 $C_{l\alpha}$ comparisons between AD and FD approaches

Alpha	$C_{l\alpha}$ UNS3D_AD	$C_{l\alpha}$ FD	$C_{l\alpha}$ % error
−2	0.1250	0.1257	0.56%
−1	0.1280	0.1293	1.01%
0	0.1270	0.1283	1.01%
1	0.1260	0.1275	1.18%
2	0.1250	0.1265	1.19%
3	0.1210	0.1240	2.42%
4	0.1150	0.1185	2.95%
5	0.0957	0.1035	7.54%

Table 2 $C_{d\alpha}$ comparisons between AD and FD approaches

Alpha	$C_{d\alpha}$ UNS3D_AD	$C_{d\alpha}$ FD	$C_{d\alpha}$ % Error
−2	−0.000184	−0.000180	−2.22%
−1	0.000066	0.000065	−0, 77%
0	0.000314	0.000315	0.32%
1	0.000555	0.000585	5.13%
2	0.000847	0.000890	4.83%
3	0.001210	0.001290	6.20%
4	0.001780	0.001975	9.87%
5	0.002940	0.003360	12.50%

Table 3 $C_{my\alpha}$ comparisons between AD and FD approaches

Alpha	$C_{my\alpha}$ UNS3D_AD	$C_{my\alpha}$ FD	$C_{my\alpha}$ % Error
−2	−0.0280	−0.02785	−0.54%
−1	−0.0310	−0.03105	0.16%
0	−0.0311	−0.03110	0.00%
1	−0.0312	−0.03120	0.00%
2	−0.0314	−0.03145	0.16%
3	−0.0316	−0.03190	0.94%
4	−0.0315	−0.03160	0.32%
5	−0.0305	−0.03050	0.00%

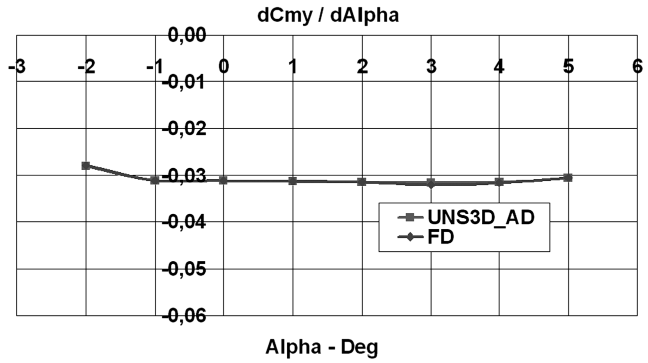


Fig. 15 $C_{my\alpha}$ **vs** α **evolution.**

thus giving a relative error equal to −0.54%. Repeating the comparison for $\alpha = 4°$, we have

$$\left.\frac{dC_{my}}{d\alpha}\right|_{\alpha=4°}^{\mathrm{UNS3D_{AD}}} = -0.3150\mathrm{e}-01$$

and

$$\left.\frac{dC_{my}}{d\alpha}\right|_{\alpha=4°}^{\mathrm{FD}} = -0.3160\mathrm{e}-01$$

thus having $\varepsilon = 0.32\%$. In particular, for $\alpha = 0, 1,$ and $5°$, derivatives have the same value, and error is null. The values are detailed in Table 3, and the corresponding derivative draw follows in Fig. 15.

The agreement of the two approaches is graphically evident. It is worth noting that numerical mismatch between direct derivatives and finite differences reduces as $\Delta\alpha$ does, and this especially happens in the nonlinear region. This can be directly shown by repeating the same computations, setting $\Delta\alpha = 0.5°$ (e.g., for an incidence $\alpha = 4°$),

$$\left.\frac{dC_L}{d\alpha}\right|_{\alpha=4°}^{\mathrm{UNS3D_{AD}}} = 1.2\mathrm{e}-01$$

and

$$\left.\frac{dC_L}{d\alpha}\right|_{\alpha=4°}^{\mathrm{FD}} = 1.17\mathrm{e}-01$$

with relative error changing from 2.95 to 2.73%. The same comparison for $\alpha = 5°$ gives

$$\left.\frac{dC_L}{d\alpha}\right|_{\alpha=4°}^{\mathrm{UNS3D_{AD}}} = 1.15\mathrm{e}-01$$

and

$$\left.\frac{dC_L}{d\alpha}\right|_{\alpha=4°}^{\mathrm{FD}} = 1.09\mathrm{e}-01$$

with relative error changing from 7.54 to 6.21%. Table 4 summarizes results for three derivatives in nonlinear domain (not all decimal values have been tabled).

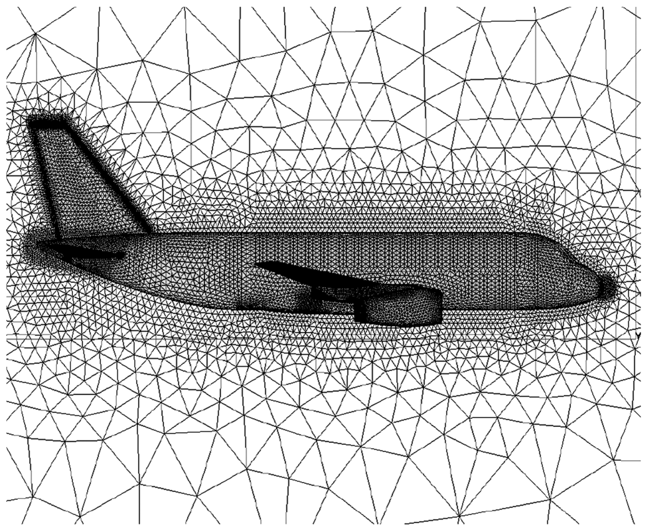


Fig. 16 Overall view of the SMJ mesh.

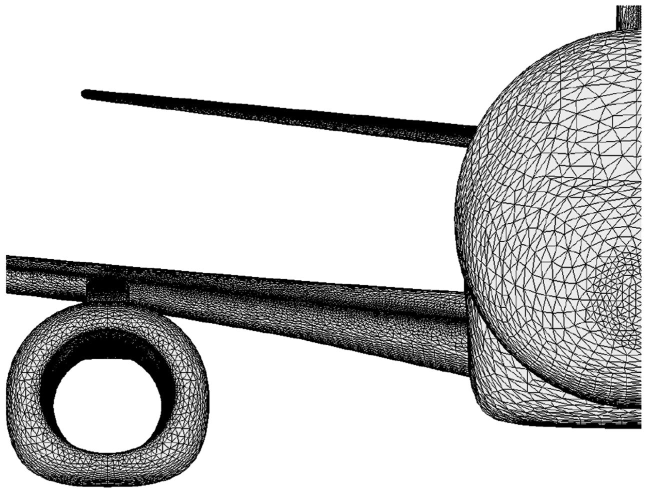


Fig. 17 Detail of the SMJ nose mesh.

Two issues can be observed: First, a reduction in the applied incidence step gives a benefit to numerical agreement between different methods, as relative error reduces. Second, changes are more important in direct AD computation than in FD. By repeating the same numerical checks with drag and pitch moment, the same behavior can be observed. The reason is that both AD and FD results are impacted, but FD method is based on difference, and thus the same (or quite similar) Δ coefficient almost erases when difference is computed. On the contrary, such a difference remains when computed by augmented code. Improvement of performance is evident when using finer increments.

SMJ

The second test has been carried out on the SMJ configuration, working in inviscid condition at $M = 0.2$. The grid in use had

Table 4 RAE2822 accuracy analysis in nonlinear domain

Alpha	$C_{l\alpha}$ AD	$C_{l\alpha}$	$C_{l\alpha}$ % error	$C_{d\alpha}$ AD	$C_{d\alpha}$ FD	$C_{d\alpha}$ % error	C_{my} AD	C_{my} FD	C_{my} % error
					$\Delta\alpha = 1°$				
4	0.1150	0.1185	2.95%	0.001780	0.001975	9.87%	−0.0315	−0.03160	0.32%
5	0.0957	0.1035	7.54%	0.002940	0.003360	12.50%	−0.0305	−0.03050	0.00%
					$\Delta\alpha = 0.5°$				
4	0.12	0.117	2.73%	0.00213	0.00203	7.85%	−0.0329	−0.0321	0.28%
5	0.115	0.109	6.21%	0.00377	0.00342	10.50%	−0.0308	−0.0309	1.25%

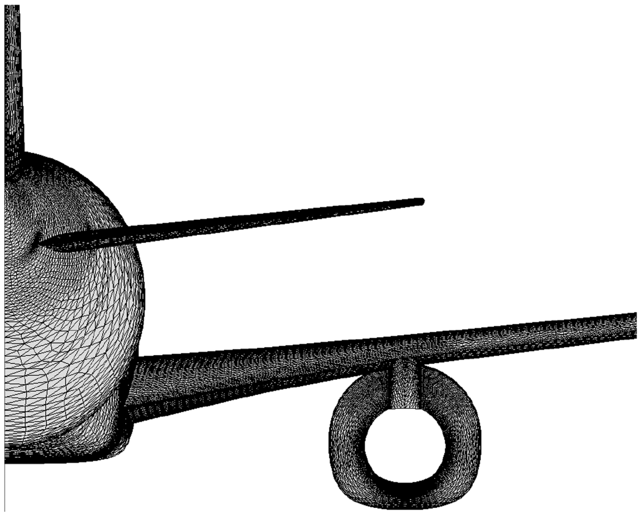


Fig. 18 Detail of the SMJ rear mesh.

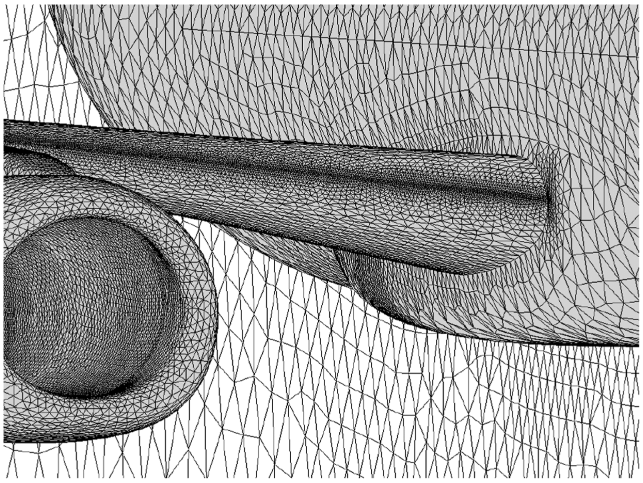


Fig. 19 Detail of the SMJ wing root mesh.

258,884 nodes, 1,356,978 elements, all tetrahedra (Fig. 14), and has been generated by the Alenia proprietary mesh generator. Dimensions of the grid are as follows:

$$-0.54661e+05 < \Delta x < 0.88614e+05$$

$$-0.25610e-07 < \Delta y < 0.32075e+05$$

$$-0.32075e+05 < \Delta z < 0.32075e+05$$

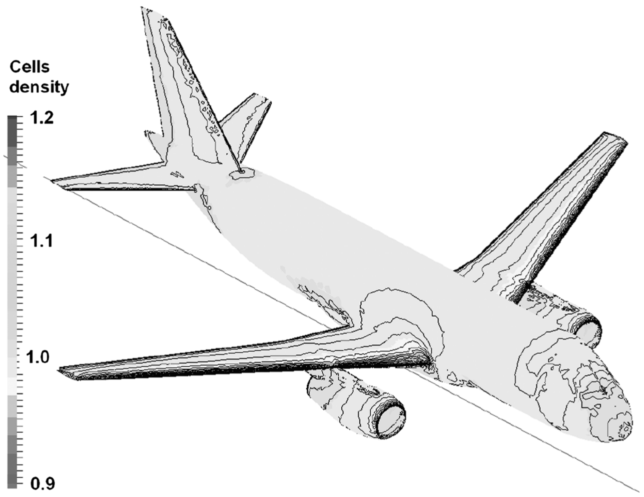


Fig. 20 Density plot of the SMJ working at $M = 0.2$ and $\alpha = 6.16°$.

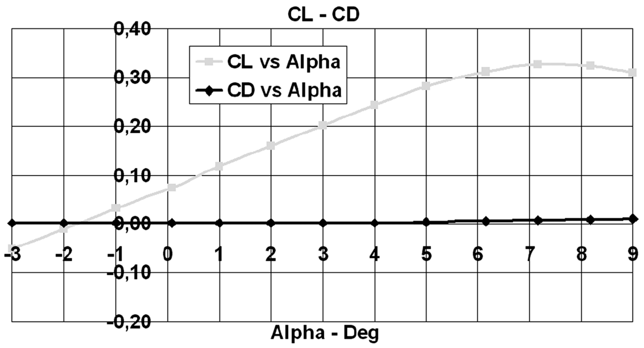


Fig. 21 Progress of C_L and C_D vs α.

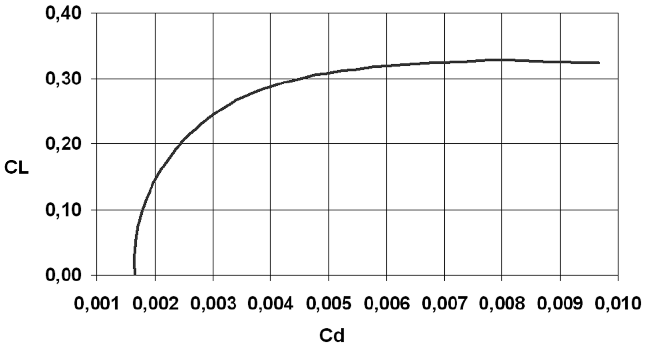


Fig. 22 SMJ aerodynamic efficiency curve.

Table 5 Primitive aerodynamic coefficients for the complete configuration

Alpha	$C_{my\alpha}$ UNS3D_AD	$C_{my\alpha}$ FD
−2	−0.011530	0.001670
−1	0.031800	0.001650
0	0.074670	0.001710
1	0.117330	0.001860
2	0.159670	0.002100
3	0.201670	0.002450
4	0.242330	0.002960
5	0.280670	0.003770
6	0.311330	0.005200

Table 6 $C_{L\alpha}$ comparisons between AD and FD approaches and relative error

Alpha	$C_{my\alpha}$ UNS3D_AD	$C_{my\alpha}$ FD	$C_{my\alpha}$ % error
−2	0.0425	0.04190	−1.43%
−1	0.0427	0.04310	0.93%
0	0.0424	0.042765	0.85%
1	0.0419	0.04250	1.41%
2	0.0415	0.04217	1.59%
3	0.0405	0.04133	2.01%
4	0.0380	0.0395	3.80%
5	0.0331	0.0345	4.06%
6	0.0210	0.02300	8.70%
7	0.0039	0.006055	35.59%

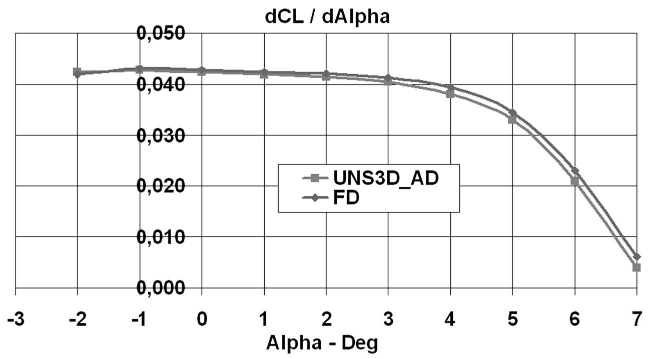


Fig. 23 Graphical comparison between $C_{L\alpha}$ with AD and FD.

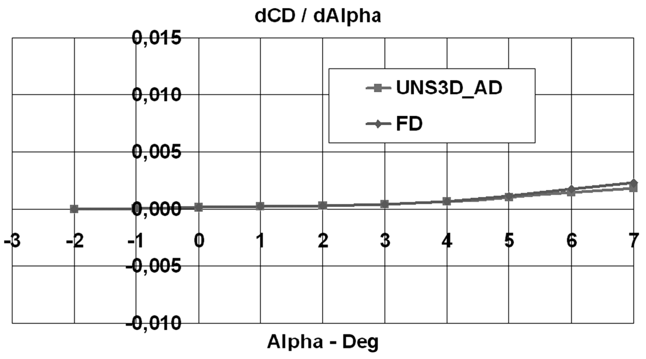


Fig. 24 Graphical comparison between $C_{D\alpha}$ with AD and FD.

Figures 16–18 provide detailed views of the side, nose, and rear grid. In the figures, it is possible to note the refinement in areas with strong pressure and velocity gradients, which means, for example, the leading and trailing edges of the wing. A detail of the wing root is provided in Fig. 19.

Numerical testing has been carried out running 5000 iterations and setting 1.0e − 4 as a residual value to stop. CFL has been set to 2.5 as in the previous case.

The test activity has been run considering normal test conditions; in particular, the solver was set for flying at $P = 1$ atm, $T = 25°\text{C}$, and $u = 70$ m/s. Table 5 recaps the numerical results that have been obtained for the primitive coefficients. As for RAE2822. it was mandatory to verify the reliability of UNS3D_AD for the primitive coefficients. In this case, the total calculation time has been 5164.32 s on a parallel Quadrics machine with 32 processors. Figure 20 shows a density plot of the solution.

Lift, drag, and efficiency curves are shown in Figs. 21 and 22; with respect to the evolution of primitive coefficients, the same considerations as for the airfoil can be applied, which means that convergence histories and steady-state results do not change, apart from an increase in computation time for augmented code.

A numerical comparison between augmented software and finite difference derivative of lift is given in Table 5.

Table 6 numerically highlights the strong adherence between the two methods, thus showing a good performance of the AD. In the nominal range of incidence (namely, $-1 \leq \alpha \leq 5$), the error is lower than 4%; error increases when incidence is out of the linear range; Fig. 23 shows this graphically.

It has to be highlighted that the two curves follow the same behavior even at the higher incidences; coefficients change almost in parallel, but percentage error increases as the denominator reduces, due to the lift reduction after the characteristic incidence. The same behavior is sketched for the drag; Table 7 and Fig. 24 detail this.

With respect to step amplitude and derivative accuracy, the same consideration made for the airfoil can be applied; even in this case, one has to observe that finer steps improve accuracy; Table 8 shows results in more detail. The lift coefficient derivative had its maximum error for $\alpha = 7°$ (35.59%) when a 1° incidence step was used, but halving the step, relative error drops down to 29.19% for the same incidence. Similar behavior is met for the drag derivative: a relative error close to 20% at $\alpha = 7°$ reduces to 17.32% when the step is set to 0.5°. Two aspects have to be considered; first, the incidence step has been halved, but formally speaking, the derivative has the mathematical meaning to locally evaluate the slope of a function. This implies that increment reduction goes toward a mathematical coherence with the concept of derivative, thus explaining why the error between two different ways to evaluate derivatives reduces when $\Delta\alpha$ reduces. Second, we are studying a nonlinear domain, and experience shows that numerical behavior in that region is not smooth. Nevertheless, smaller increments clearly have a positive effect and reduce the error. Hereinafter the complete list is provided.

Machine in Use and Elaboration Time

All of the runs have been carried out on a Quadrics multiprocessor platform. The CPU is the Authentic AMD working at 2594 MHz. The system is made up of seven computational nodes and one node dedicated to the file system access. Each node has four processors. The physical memory of each node is 3943 MB and the virtual one is 8189 MB. In the case of the RAE2822 airfoil, the machine has been in use 156.7 s to provide both the primitive and differentiated results. For the SMJ runs, which are far more complex, 5000 iterations have been run in 5164.32 s.

Possible Improvement of the AD Tool

A possibility to improve the AD system lies in its capability to automatically include the new size of derived vectors in the declarations, not requiring any intervention of the user, thus avoiding relatively large time consumption. Another possibility, less impactive from the software point of view, would be the automatic generation of the `include` file, collecting all the new declarations together; the user should just compile together the derived software and the `include` file, thus reducing the workload and transferring it to the machine.

Also, the automatic set to 1 of the initial value of the derived parameter should be taken into account; when the AD tool takes the derivation rule from the command line, it could automatically set to 0

Table 7 $C_{D\alpha}$ comparisons between AD and FD approaches and relative error

Alpha	$C_{my\alpha}$ UNS3D_AD	$C_{my\alpha}$ FD	$C_{my\alpha}$ % error
−2	−0.000058	−0.000058	0.57%
−1	0.000020	0.000020	2.50%
0	0.000101	0.000105	3.81%
1	0.000192	0.000195	1.54%
2	0.000289	0.000295	2.03%
3	0.000420	0.000430	2.33%
4	0.000640	0.000660	3.03%
5	0.001030	0.001120	10.71%
6	0.001470	0.001715	14.29%
7	0.001800	0.002235	19.46%

Table 8 SMJ accuracy analysis in nonlinear domain

Alpha	$C_{l\alpha}$ AD	$C_{l\alpha}$ FD	$C_{l\alpha}$ % error	$C_{d\alpha}$ AD	$C_{d\alpha}$ FD	$C_{d\alpha}$
			$\Delta\alpha = 1°$			
4	0.0380	0.0395	3.80%	0.000640	0.000660	3.03%
5	0.0331	0.0345	4.06%	0.001030	0.001120	10.71%
6	0.0210	0.02300	8.70%	0.001470	0.001715	14.29%
7	0.0039	0.00605	35.59%	0.001800	0.002235	19.46%
			$\Delta\alpha = 0.5°$			
4	0.0396	0.0384	3.24%	0.000641	0.000623	2.96%
5	0.0407	0.0392	3.87%	0.001158	0.001053	9.98%
6	0.0241	0.0227	5.98%	0.001768	0.001572	12.5%
7	0.0065	0.0051	29.19%	0.002514	0.002143	17.32%

all the derived parameters not impacting the relevant calculation and set to 1 the derived initial value, thus simplifying the work. However, as far as the authors know, INRIA is currently working to improve Tapenade.

Conclusions

The AD tool provides a useful capability to automatically derive required functions. With software reworking, however, it is very often necessary to rearrange different components of the code that may have been differentiated, even if not directly involved in the calculation of the variables of interest. This opens the possibility to optimize the parts of code really important for one's need and, on the other side, suggests splitting the code in several components, thus making more flexible any eventual change and rearrangement. A possibility to obtain a faster convergence of the augmented code is by applying the AD to a converged solution, for which the residual tends to zero; that is,

$$R_i(U, \eta, \eta^b) = 0$$

It has to be noted that despite the need to optimize the derived software and the consequent reworking, the time savings achieved with the AD is remarkable. With respect to other classical means to evaluate the derivatives (e.g., the finite differences), the AD provides an evaluation of the exact gradient and may be used in all processes in which a sensitivity analysis is requested. But a useful application is obtained if the software is under complete control of the user. Testing is still ongoing at Alenia premises in Turin to investigate the possibility of application of AD to do the following:

1) Obtain dynamic derivatives by differentiating the unsteady UNS3-D solver in such a way to be applied straightforward both to 2-D, 2.5-D, and 3-D problems.

2) Apply AD to higher-order terms in the solver.

Because of the complexity of the solver structure, the work is challenging but possible. Experience thus far gathered by Alenia Aeronautica has shown the following:

1) The assumptions made by INRIA and those at the base of Tapenade development were applicable.

2) The application to complex configuration is reliable and provides results in good agreement with those gathered with the former tools: namely, finite differences.

Further activity is planned in order to explore a different configuration in the solver structure to reduce the run time and optimize the execution. Given the good results achieved so far, further applications can be implemented in the augmented source code (e.g., those related to the shape optimization, which involves gradient calculation), mesh adaptation, and consequent resizing, thus providing a higher level of benefit in the calculation capability and optimization in multidisciplinary design.

References

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

[2] Green, L., Spence, A., and Murphy, P., "Computational Methods for Dynamics Stability and Control Derivatives," AIAA Paper 2004, Jan. 2004.

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

[4] Hascoët, L., and Pascual, V., "TAPENADE 2.1 User's Guide," Sept. 2004, pp. 8–13, http://hal.inria.fr/docs/00/06/98/80/PDF/RT-0300.pdf [retrieved 19 May 2006].

[5] Hascoet, L., "Analyses Statiques et Transformations de Programmes: de la Parallélisation à la Différentiation," National de Recherche en Informatique et en Automatique, Sophia Antipolis, France, 2005, pp. 163–169, http://www-sop.inria.fr/members/Laurent.Hascoet/papers/hdrHascoet05.pdf [retrieved Jan. 2005].

[6] Selmin, V., "Application of Automatic Differentiation to Aerodynamic Shape Optimisation," ECCOMAS 2004, 2004, http://www.imamod.ru/~serge/arc/conf/ECCOMAS_2004/ECCOMAS_V2/proceedings/pdf/611.pdf [retrieved 28 July 2004].

[7] Anderson, D. A., Tannehill, J. C., and Pletcher, R. H., "Basic Finite-Difference Methods," *Computational Fluid Mechanics and Heat Transfer*, 1st ed., McGraw–Hill, New York, 1984, pp. 40–42.