40th European Rotorcraft Forum, 2–5 September 2014, Southampton, U.K. – Paper 008

ADJOINT METHODS FOR THE EFFICIENT COMPUTATION OF AERODYNAMIC DERIVATIVES WITH HIGH-FIDELITY CFD

Massimo Biava and George N. Barakos

CFD Laboratory, School of Engineering
University of Liverpool, L69 3GH, U.K.
http://www.liv.ac.uk/cfd
Email: *M.Biava@liverpool.ac.uk, G.Barakos@liverpool.ac.uk*

Abstract

This paper presents the development of a discrete adjoint method by means of automatic differentiation in the framework of the Helicopter Multiblock CFD solver. The method is suitable for applications in flight mechanics as well as shape optimisation and is demonstrated in this paper for cases reported in the literature. The application of automatic differentiation is first presented for a simple flight mechanics software with indicative results for ADS-33 maneouvres. Subsequently adjoint CFD computations were undertaken for aerofoil, wing and rotor blade cases. The obtained results were found to agree well with other published solutions or with data obtained using finite differences for computing the flow derivatives. The method has so far been demonstrated for inviscid flow cases and suggests that the current implementation is robust and efficient. The cost of the adjoint computations is relatively low due to the employed source code differentiation and most of the times it is no more than the cost for a steady-state flow solution.

1 INTRODUCTION

The design of new generation helicopters with increased performance and improved handling qualities requires a deeper understanding of the aerodynamics, not only in steady flight, but also during manoeuvres. Because of the nonlinearity and unsteadiness of the flow it is extremely challenging to determine its aerodynamic characteristics. To reduce the complexity of the problem, it is commonly assumed that for small deviations from a given steady flight condition the flight dynamics behaviour can be described by means of a linearised model, defined by a set of aerodynamic derivatives. These derivatives can be obtained via finite differences (FD) out of a CFD computation. Nevertheless, finite differencing becomes prohibitive in terms of computational cost, since two or more complete flow solutions are required to compute each derivative.

A more economic way to obtain the aerodynamic derivatives with CFD is via solving the *sensitivity equation*, casted in either tangent or adjoint form [1, 2]. The basic idea is to write any aerodynamic force and moment coefficient I as a function of the flow variables $\boldsymbol{W}$ and of the input flight dynamics variable of interest x (angle of attack, sideslip, Mach number, etc.), that is $I = I(\boldsymbol{W}(x), x)$. The flow variables are subject to satisfy the fluid dynamics governing equations, written in compact form as $\boldsymbol{R}(\boldsymbol{W}(x), x) = 0$. Formally taking the derivative of I with respect to x we obtain:

$$\frac{\mathcal{D}I}{\mathcal{D}x} = \frac{\partial I}{\partial x} + \frac{\partial I}{\partial \boldsymbol{W}}\frac{\partial \boldsymbol{W}}{\partial x}, \tag{1}$$

which represents the tangent form of the sensitivity equation. All the partial derivatives appearing in the right-hand side can be computed with limited effort, with exception of the term $\partial\boldsymbol{W}/\partial x$, that represents the variation of the flow variables with respect to the variation of the independent input parameter. This last term may be obtained by differentiating the governing equations to yield the following linear system for the unknown $\partial\boldsymbol{W}/\partial x$:

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{W}}\frac{\partial \boldsymbol{W}}{\partial x} = -\frac{\partial \boldsymbol{R}}{\partial x}. \tag{2}$$

Therefore, the computation of a flow sensitivity is reduced to the solution of the nonlinear governing flow equations plus the solution of the linear system (2). Note that one linear system for each flight dynamics variable must be solved to compute the sensitivities, since the right-hand side of (2) depends upon x. Thus, as with finite differencing, the overall computational cost scales with the number of inputs. However, the sensitivity equation approach requires the solution of only one nonlinear system of equations and does not suffer of the cancellation problem, yielding derivatives accurate up to machine precision.

The sensitivity problem (1)-(2) can be recast in dual form by introducing the adjoint vector variable $\boldsymbol{\lambda}$ as the solution of the following linear system:

$$\left(\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{W}}\right)^{\mathrm{T}}\boldsymbol{\lambda} = -\frac{\partial I}{\partial \boldsymbol{W}}. \tag{3}$$

Substituting this equation into the expression (1) and using the duality between a matrix and its transpose we obtain:

$$\frac{\mathcal{D}I}{\mathcal{D}x} = \frac{\partial I}{\partial x} + \boldsymbol{\lambda}^{\mathrm{T}}\frac{\partial \boldsymbol{R}}{\partial x}. \tag{4}$$

The computational cost of the dual sensitivity problem (3)-(4) scales with the number of outputs, since the right-hand side of (3) depends on I, but it is independent of the input parameters. The choice between the use of the direct or dual sensitivity problem depends consequently on the balance between the number of outputs and the number of inputs. The two methods, for instance, should perform equally well in computing the flight mechanics derivatives, since the number of output force and moment coefficients is similar to the number of input flight mechanics parameters.

The calculation of the partial derivatives appearing in the sensitivity equation can be done manually, deriving analytical expressions and writing the needed computer code. Nonetheless, this approach can be tedious if the flow equations involve complex terms, like, for instance, upwinding terms for the inviscid fluxes or source terms of turbulence models. Recent advances in *automatic differentiation* (AD) tools, however, enable to produce the computer code for the differentials of these complex terms directly from the source code of the CFD solver [3].

To assess the use of AD in complex computer codes, such as CFD solvers, we first applied the methodology to the Helicopter Flight Mechanics (HFM) code, developed at the University of Liverpool. HFM integrates the rigid body equations of dynamics for the helicopter and makes use of strip-theory model to compute the aerodynamic forces. The code has been differentiated by means of the source transformation tool TAPENADE [4, 5], so as to produce all the aerodynamic derivatives needed for a state space representation of the helicopter dynamics. As an application, the state space representation is used to build a real-time trajectory tracking autopilot based on a Linear Quadratic Regulator (LQR) and Proportional-Integral (PI) controller. To assess the autopilot, the ADS-33 [6] lateral reposition manoeuvre and the slalom manoeuvre have been simulated using HFM.

The experience gained with HFM was then used for the CFD solver Helicopter Multi Block (HMB) [7, 8] of Liverpool and AgustaWestland. Taking inspiration from the work of Jones *et al.* [9] and of Mader *et al.* [2], the individual functions of the CFD solver have been automatically differentiated and assembled afterward to build the neecessary terms in the sensitivity equation, both in tangent and adjoint form. The linear system associated to the sensitivity problem is solved using the fully implicit fixed-point iteration scheme of the base flow solver. The resulting code is able to compute the aerodynamic derivatives of fixed wing aircraft or of rotors in hover flight, at a fraction of the cost required by finite differencing. The method is demonstrated for the aerodynamic sensitivities of the NACA0012 airfoil, the ONERA M6 wing and the ONERA 7AD rotor in hover.

2 Background

The first application of the adjoint method to fluid dynamics is dated back to 1974 with the pioneering work of Pironneau [10], where adjoint methods and control theory were applied to drag minimisation. Starting from the late eighties the first applications to CFD problems begin to appear, thanks to the work of Jameson and other co-authors [11, 12, 13]. They exploited the adjoint method and control theory for aerodynamic shape optimisation in conjunction with CFD techniques, whose complexity increased over the years from the solution of the potential flow equations to that of the Navier–Stokes equations [14, 15, 16]. The derivation of the adjoint problem in these works is based on the *continuous approach* (CA), where the adjoint equations are analytically derived from the primary flow equations and discretised afterwards.

The alternative *discrete approach* (DA) to the adjoint problem consists in deriving the adjoint equations directly from the discretised formulation of the flow equations. This has been pursued in the works of Elliott and Peraire [17] and Anderson [18] in the context of aerodynamic shape optimisation with unstructured grids. A fairly complete overview of the development of continuous and discrete adjoint methods in the last two decades of the 20th century can be found in Newman *et al.* [19]. Both continuous and discrete approaches have advantages and disadvantages, as pointed out by Giles and Pierce [20]. They are summarised in table 1.

The implementation of the DA for flow equations involving complex terms (upwinding terms, terms depending on spectral radii, source terms appearing in turbulence models, etc.) is not generally straightforward. A technique to tackle the problem of deriving the discrete adjoint in such complex cases is automatic differentiation, in which the adjoint code to evaluate the gradients is obtained manipulating directly the original CFD code, as in the work of Mohammadi [21, 22].

AD may be obtained by means of source-traformation tools or via operator overloading in programming languages such as FORTRAN 90 and C++. Tools that use source code transformation add new statements to the original source code that compute the derivatives of the original statements. The operator overloading approach consists of a new user-defined type that is used instead of floating points. This new type includes not only the value of the original variable, but its derivative as well. The operator overloading approach results in fewer changes to the original code, but is usually less efficient [23]. AD tools are available for a variety of programming languages. ADIFOR [24], TAF [25] and TAMC [26] are some of the tools available for FORTRAN. TAPENADE [4, 5] supports both FORTRAN 90 and C. A complete list of AD tools available for each programming language may be found at [27].

There are two different modes of operation for automatic differentiation of a computer code: the *forward* (or *tangent*) mode and the *reverse* (or *adjoint*) mode. The forward mode uses the chain rule to propagate the required derivatives in the same direction of the original computer code. The cost of forward AD is proportional to the number of inputs of the computed function. In reverse mode the derivatives are propagated backward, from the last statement of the code to the first. The reverse mode is analogous to the adjoint method and the cost is proportional to the number of outputs of the computed function. However, the memory requirements of the reverse mode are considerably higher, since the storage of intermediate results of the function evaluation is required for the backward propagation of the derivatives.

It is to be noted that AD cannot be applied directly to the whole residual evaluation chain to produce the adjoint of the flow equations, because it would lead to an inefficient code

Discrete approach	Continuous approach
Provides the exact gradients, since the discrete adjoint operator is simply the transpose of the matrix arising from the discretisation of the primary flow equations	Gives an approximation to the continuous gradient based on some alternative discretization
The implementation requires less coding effort, especially if AD is employed	Requires hand coding of the discretization scheme applied to the continuous adjoint equations
Straight application of AD to the CFD code produces inefficient adjoint code, so that application of AD to individual nonlinear subroutines and partial re-coding is necessary	The continuous code is often considerably simpler than the discrete in terms of operation count and memory requirements, as well as easier to implement
The derivation of the adjoint equations and BCs is purely algebraic, and gives no insight in the physics of the problem	Gives a more clear interpretation of the physics behind the adjoint variables and of the associated BCs

Table 1: Advantages and disadvantages of the continuous and discrete approaches.

in terms of memory and CPU time. A more realistic goal for AD is in assisting the derivation of the discrete adjoint by hand-differentiation, by automatically differentiating and transposing individual routines. This was adopted in Mader *et al.* [3] and in Jones *et al.* [9].

The introduction of automatic differentiation, the advances in techniques for solving the adjoint problem and the growing power of computing hardware allowed application of the adjoint method to more complex cases. Also, driven by the industry need of more realistic flight mechanics models, the related research widened its initial objective of aerodynamic shape optimisation to make space to novel applications such as aeromechanics. In the work of Limache and Cliff [1] and of Mader and Martins [2, 28], for instance, the aerodynamic derivatives of airfoil and wings are computed by solving the sensitivity problem. This concept is then extended to compute the sensitivities of time-periodic flow solutions, such as those generated by turbomachinery and helicopters, by applying the adjoint method to the time-spectral formulation of the flow equations, which reduces the time-dependent problem to a steady problem in the frequency domain. This is described in Choi *et al.* [29] and in Mader and Martins [28].

These past works proved the superiority of the sensitivity equation approach with respect to finite differencing for aeromechanics applications but, at the same time, showed the difficulties associated to the convergence of the sensitivity equation and the demanding memory requirements of adjoint solvers, which can represent a limiting factor for realistic large-scale applications. The objective of the present work is to partially overcome these drawbacks, while keeping the efficiency and accuracy of the sensitivity problem approach for the computation of aerodynamic derivatives.

3 Application to flight mechanics

3.1 Code description

The computer code HFM is capable of simulating a freely-moving rotorcraft as well as of computing the vehicle trim state. It is based on the classical equation of motion for rigid bodies and on BEM and the Peters–HaQuang [30] inflow model to compute the rotor forces.

In HFM the vector $\boldsymbol{z}$, describing the rotorcraft state, is defined as follows:

$$\begin{aligned}\boldsymbol{z} =&(u\,v\,w\,p\,q\,r\,x_\mathrm{E}\,y_\mathrm{E}\,z_\mathrm{E}\,\varPhi\,\varTheta\,\varPsi\\ &\lambda_0^i\,\lambda_{1s}^i\,\lambda_{1c}^i\,\dot\beta^{ij}\,\beta^{ij}\,\dot\zeta^{ij}\,\zeta^{ij}\,\dot\theta^{ij}\,\theta^{ij}\,Q^i\,\omega^i\,\psi^i),\\ &i=1,\dots,N_\mathrm{R},\ \ j=1,\dots,N_\mathrm{B}^i,\end{aligned} \tag{5}$$

where

- N_R is the number of rotors and N_B^i is the the number of blades of the i-th rotor;
- u, v, w are the vehicle velocity components in body axis;
- p, q, r are the vehicle angular velocity components in body axis;
- x_E, v_E, w_E are the vehicle position vector components in Earth reference frame;
- $\varPhi$, $\varTheta$, $\varPsi$ are the vehicle Euler angles in Earth reference frame;
- λ_0^i, λ_{1s}^i, λ_{1c}^i are the inflow model coefficients for the i-th rotor;
- β^{ij}, ζ^{ij}, θ^{ij} are the hinge angles for the j-th blade of the i-th rotor;
- Q_i and ω_i are the torque and rotational speed of the i-th rotor;
- ψ_i is the azimuth angle of the i-th rotor.

The control vector $\boldsymbol{u}$ contains the collective and cyclic pitch angles of the rotors:

$$\boldsymbol{u} = \left(\theta_0^i\,\theta_{1c}^i\,\theta_{1s}^i\right), \quad i=1,\dots,N_\mathrm{R}. \tag{6}$$

For a typical helicopter $N_\mathrm{R}=2$ (the main and the tail rotor), and for the tail rotor only the collective pitch is controlled; the control vector has therefore only four components.

With the above definitions, the flight mechanics equations can be written in compact form as a first-order system of ordinary differential equations:

$$\dot{\boldsymbol{z}} = D(\boldsymbol{z}, \boldsymbol{u})^{-1}\boldsymbol{r}(\boldsymbol{z}, \boldsymbol{u}), \tag{7}$$

where $D(\boldsymbol{z}, \boldsymbol{u})$ is a matrix and $\boldsymbol{r}(\boldsymbol{z}, \boldsymbol{u})$ is the nonlinear residual vector. Equations (7) are solved in HFM with either an explicit Euler method or with a fourth order explicit Runge–Kutta method.

3.2 Automatic differentiation

The core of the flight mechanics code is the function `HFM_compute`, which solves numerically equation (7). The flowchart of the function is shown in figure 1a. `HFM_compute` takes as input the current state of the helicopter and the controls, and then integrates the equations of motion in time for a given number of time steps. At each time step the operations performed are the following:

(1) compute the rotors(s) equations of motion;

(2) compute the contribution of the fuselage and tail surfaces to the equations of motion;

(3) sum all the force and moment contributions into the body equations of motion;

(4) compute the engine terms;

(5) assembly all the computed terms into the matrix $D(\boldsymbol{z}, \boldsymbol{u})$ and into the vector $\boldsymbol{r}(\boldsymbol{z}, \boldsymbol{u})$ appearing in equation (7);

(6) solve the linear system (7) by LU decomposition to compute the first derivative of the state vector $\dot{\boldsymbol{z}}$;

(7) integrate in time with either the explicit Euler or explicit fourth order Runge–Kutta[1] scheme to obtain the new state vector $\boldsymbol{z}$.

It is possible to differentiate the function `HFM_compute` (in tangent mode, for instance) with a single invocation of TAPENADE as follows:

```
tapenade -d -root HFM_compute \
      -vars "z u"
      -outvars "F M"
      src/HFM_compute.c \
      src/HFM_vehicle.c \
      src/HFM_rotors.c
```

The option "`-d`" tells the tool to operate differentiation in tangent mode. Options "`-vars`" and "`-outvars`" specify the name of the function input and output differentiable variables, respectively. The produced differentiated subroutine is automatically named `HFM_compute_d`.

The input variables are the state vector $\boldsymbol{z}$ and the control vector $\boldsymbol{u}$, while the output variables are the resulting force $\boldsymbol{F}$ and moment $\boldsymbol{M}$ on the helicopter. A single call to `HFM_compute_d` produces the partial derivative of $\boldsymbol{F}$ and $\boldsymbol{M}$ with respect to one of the inputs. Differentiating with respect to the state we obtain the *stability derivatives*. Differentiating with respect to the control we obtain the *control derivatives*.

The flowchart of the subroutine `HFM_compute_d` is shown in figure 1b. All calls to the original functions are replaced by calls to the corresponding differentiated subroutines. An exception is made for the call to the linear solver, which has been differentiated manually. Indeed, the mathematical operation of solving a linear system can be differentiated by hand and the resulting algebra can be directly implemented. In fact, if $A\boldsymbol{x} = \boldsymbol{b}$ is the original problem, hand differentiation gives $\delta A\boldsymbol{x} + A\delta\boldsymbol{x} = \delta\boldsymbol{b}$, and the sought differential can be thus computed as $\delta\boldsymbol{x} = A^{-1}(\delta\boldsymbol{b} - \delta A\boldsymbol{x})$. In our specific case, the differential of $\dot{\boldsymbol{z}}$ is computed solving the system:

$$\begin{aligned} D(\boldsymbol{z}, \boldsymbol{u})\delta\dot{\boldsymbol{z}} =& \delta\boldsymbol{r}(\boldsymbol{z}, \boldsymbol{u}, \delta\boldsymbol{z}, \delta\boldsymbol{u}) \\ &- \delta D(\boldsymbol{z}, \boldsymbol{u}, \delta\boldsymbol{z}, \delta\boldsymbol{u})\boldsymbol{r}(\boldsymbol{z}, \boldsymbol{u}). \end{aligned} \tag{8}$$

3.3 LQR based autopilot

AD provides an efficient method to compute the stability and control derivatives of a rotorcraft computer model. We now describe how to use this information to build an autopilot for trajectory tracking based on a LQR feedback controller [31, 32]. To this end, for a conventional helicopter with main and tail rotors, we consider the following state space and control vectors:

$$\boldsymbol{x} = (u\, v\, w\, p\, q\, r\, x_\mathrm{E}\, y_\mathrm{E}\, z_\mathrm{E}\, \Phi\, \Theta\, \Psi), \tag{9}$$

$$\boldsymbol{u} = \left(\theta_0^\mathrm{MR}\, \theta_{1c}^\mathrm{MR}\, \theta_{1s}^\mathrm{MR}\, \theta_0^\mathrm{TR}\right), \tag{10}$$

and build the linearised 6-DoF model of the rotorcraft around the trim state $(\boldsymbol{x}^*, \boldsymbol{u}^*)$ as

$$\delta\dot{\boldsymbol{x}} = A\delta\boldsymbol{x} + B\delta\boldsymbol{u}. \tag{11}$$

where

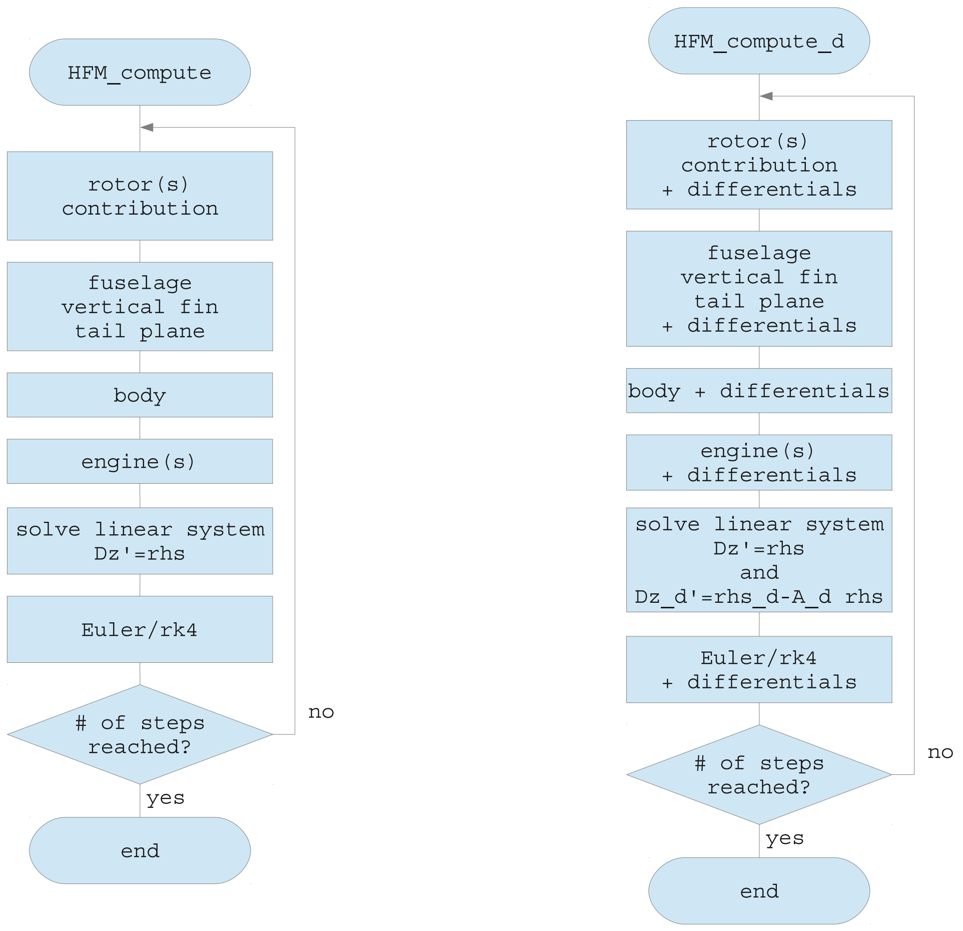
$$A = \frac{\partial \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{x}},\; B = \frac{\partial \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{u}} \quad \text{at} \quad \boldsymbol{x} = \boldsymbol{x}^*,\; \boldsymbol{u} = \boldsymbol{u}^*. \tag{12}$$

The nonlinear function $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ describes the evolution of the state space vector from the trim state $\boldsymbol{x}^*$ to the state $\boldsymbol{x}$ under the action of the input $\boldsymbol{u}$ (held fixed), and is computed by integrating equation (7) over some revolutions of the rotor in order to let the flapping motion transient be sufficiently damped.

The aim of an autopilot is to control the position $(x_\mathrm{E}, y_\mathrm{E}, z_\mathrm{E})$ of the helicopter in Earth reference frame and its heading Ψ. We recast this trajectory tracking problem into the LQR setting as follows. At each time instance we consider the closest trimmed condition of the helicopter and compute the associated linearised model. Then, if $\delta\boldsymbol{x}$ is the deviation of the state vector from the desired state, the variation $\delta\boldsymbol{u}$ of the controls is determined as the LQR optimal feedback due to the deviation $\delta\boldsymbol{x}$. The LQR controller will in fact drive $\delta\boldsymbol{x}$ to zero by minimising the quadratic cost function

$$J = \int_0^\infty \left(\delta\boldsymbol{x}^\mathrm{T} Q\delta\boldsymbol{x} + \delta\boldsymbol{u}^\mathrm{T} R\delta\boldsymbol{u}\right) \mathrm{d}t, \tag{13}$$

[1]When using the Runge–Kutta scheme, steps 1–6 are repeated once for each of the four stages of the scheme.


HFM_compute
rotor(s) contribution
fuselage vertical fin tail plane
body
engine(s)
solve linear system Dz'=rhs
Euler/rk4
# of steps reached?
no
yes
end
HFM_compute_d
rotor(s) contribution + differentials
fuselage vertical fin tail plane + differentials
body + differentials
engine(s) + differentials
solve linear system Dz'=rhs and Dz_d'=rhs_d-A_d rhs
Euler/rk4 + differentials
# of steps reached?
no
yes
end

(a) `HFM_compute`

(b) `HFM_compute_d`

Figure 1: Flowcharts of the top-level functions `HFM_compute` and `HFM_compute_d`.

with Q and R being weighting matrices that define the "importance" of the the states and of the controls in the cost function. The solution to the minimisation problem is

$$\delta \boldsymbol{u}_{\mathrm{LQR}} = -K \delta \boldsymbol{x}, \tag{14}$$

where K is the optimal feedback matrix given by

$$K = R^{-1} B^{\mathrm{T}} P, \tag{15}$$

and P is the solution of the continuous algebraic Riccati equation:

$$A^{\mathrm{T}} P + PA - PBR^{-1}B^{\mathrm{T}}P + Q = 0. \tag{16}$$

As can be seen, the optimal LQR feedback matrix K does not depend on the solution and may therefore be precalculated prior to the simulation for the various representative trim states.

To achieve better tracking performance the LQR controller has been augmented with a simple PI controller:

$$\begin{aligned} \delta \boldsymbol{u}_{\mathrm{PI}} = & -\mathrm{diag}(K_1^{\mathrm{P}}, K_2^{\mathrm{P}}, K_3^{\mathrm{P}}, K_4^{\mathrm{P}}) \boldsymbol{e} \\ & - \mathrm{diag}(K_1^{\mathrm{I}}, K_2^{\mathrm{I}}, K_3^{\mathrm{I}}, K_4^{\mathrm{I}}) \int_{t-\Delta t}^{t} \boldsymbol{e} \, \mathrm{d}t, \end{aligned} \tag{17}$$

where $\boldsymbol{e}$ is the tracking error

$$\boldsymbol{e} = \left\{ \begin{array}{c} \boldsymbol{x}_{\mathrm{E}} - \hat{\boldsymbol{x}}_{\mathrm{E}} \\ \Psi - \hat{\Psi} \end{array} \right\} \tag{18}$$

and $\boldsymbol{x}_{\mathrm{E}}$ and $\hat{\boldsymbol{x}}_{\mathrm{E}}$ are the actual and desired trajectories in Earth reference frame, Ψ and $\hat{\Psi}$ the actual and desired headings. The coefficients K_i^{P} and K_i^{I} ($i = 1, \ldots, 4$) are, respectively, the proportional and integral gains.

The value of the control angles at each time instant is therefore given by their value in the reference trimmed condition plus the feedback given by the LQR and PI controllers:

$$\boldsymbol{u} = \boldsymbol{u}^* + \delta \boldsymbol{u}_{\mathrm{LQR}} + \delta \boldsymbol{u}_{\mathrm{PI}}. \tag{19}$$

3.4 *Numerical results*

To assess the performance of the developed autopilot we considered the HFM model of a generic MR/TR helicopter and simulated two manoeuvres from the ADS-33 aeronautical design standard [6], specifically the *lateral reposition* (ADS-33 3.11.8) and the *slalom* (ADS-33 3.11.9).

For both the manoeuvres only one trimmed condition has been considered to set up the autopilot, corresponding to the flight condition at the beginning of the manoeuvre. The weighting matrices appearing in the LQR cost function and the PI controller gains have been set as follows:

$$Q = \text{diag}\{1\,1\,1\;\; 1\,1\,1\;\; 0.2\,0.2\,0.4\;\; 0.01\,0.01\,2\}, \tag{20}$$
$$R = \text{diag}\{750\,1500\,1500\,400\}, \tag{21}$$
$$K^{\text{P}} = \text{diag}\{0\,0\,0\,2\}, \tag{22}$$
$$K^{\text{I}} = \text{diag}\{0\,0\,0\,0.1\}. \tag{23}$$

The weighting coefficient values were obtained by trial and error, driven by the following considerations:

- to obtain smooth changes in the pitch, roll and yaw angles, the angular velocities should be kept low, and therefore a relative high value should be put on the corresponding weights;
- a high weight is to be assigned to the velocities, because the autopilot is required to track as close as possible the given trajectory;
- similar considerations apply to the position variables, but the given weights are lower than those assigned to the velocities because the effort of the controller to track the position is higher and high weights would result in a too strong feedback response;
- the weight on the pitch and roll angles should be low, because the helicopter will often be in a condition where non-zero pitch and roll angles are used and thus they should not be forced by the autopilot;
- a null weight is given to the yaw angle, because numerical experiments revealed that it is better tracked by the PI controller;
- the values associated to the control angles should be high in order to keep the control signal within the limits of the actuators and to ensure a smooth flight;
- the PI gains for the main rotor control angles are set to zero in order to leave the control entirely to the LQR controller; a relatively high gain is assigned instead to the tail rotor control angle to ensure a close tracking of the prescribed heading.

Lateral reposition This manoeuvre starts with the helicopter in a stabilized hover at 35 ft (10.7 m) wheel height with the longitudinal axis of the rotorcraft (the x axis in the simulation) oriented 90 degrees to a reference line marked on the ground (the y axis). Then the helicopter must initiate a lateral acceleration to approximately 35 knots (18 m/s) groundspeed followed by a deceleration to laterally reposition the rotorcraft in a stabilized hover 400 ft (122 m) down the course within a specified time. For a utility helicopter and in good visual conditions, the time to complete the manoeuvre is 18 s.

Figure 2 and 3 show, respectively, the comparison between desired and actual helicopter lateral velocity and position. The velocity profile is tracked fairly well, the greatest deviation being a slight overshoot at the transition point between the acceleration and deceleration phases and at the end of the deceleration phase. As a result, there is only a small deviation in the imposed and actual lateral position of the vehicle, and the manoeuvre is correctly concluded after the 18 s prescribed by the ADS-33 normative.

The control inputs generated by the autopilot to track the trajectory is represented in figure 4. The variation is smooth and none of the four control angles is saturated during the manoeuvre.

Slalom This manoeuvre is initiated in level unaccelerated flight and lined up with the centerline of the test course (the x axis in the simulation). The rotorcraft must perform a series of smooth turns at 500 ft (152 m) intervals (at least twice to each side of the course). The turns shall be at least 50 ft (15.2 m) from the centerline, with a maximum lateral error of 50 ft (15.2 m). The manoeuvre ends on the centerline, in coordinated straight flight. The velocity during the manoeuvre should be no less than 60 knots (30.9 m/s) to comply with the "desired" performance specification, or no less than 40 knots (20.6 m/s) to comply with the "adequate" performance specification.

The autopilot is able to perform the slalom manoeuvre at the prescribed speed of 60 knots (30.9 m/s). The given and computed trajectories in the xy plane are displayed in figure 5, where the ability of the autopilot to follow the slalom path is clearly proven. Figure 6 shows a rendering of the computed slalom manoeuvre obtained with a software based on the Presagis VEGA Prime library [34].

The commands history is given in figure 7, where we can observe that the variation of the control angles is fairly smooth and that no saturation of any control angle occours. The computed commands history can be compared with that of figure 8, taken from [33], where the commands for the slalom manoeuvre are obtained via nonlinear optimal control theory for different helicopter models, both linear and nonlinear (named M1 to M4 in the figure). As can be noted, the commands generated by the LQR controller are characterized by a lesser pilot effort in terms of main rotor collective and cyclics controls, but also by a higher use of the tail rotor collective.

4 Application to the CFD solver HMB

4.1 *Overview of the HMB Flow Solver*

The following contains a brief outline of the approach used in the Helicopter Multi-Block solver version 2.0. The Navier–Stokes (NS) equations are discretised using a cell-centred finite volume approach. The computational domain is divided into a finite number of non-overlapping control-volumes, and the governing equations are applied to each cell in turn. Also, the Navier–Stokes equations are re-written in a curvilinear coordinate system which simplifies the formulation of the discretised terms since body-conforming grids are adopted here. The spatial discretisation of the NS equations leads to a set of ordinary differential equations in time:

$$\frac{d}{dt}\left(\boldsymbol{W}_{ijk}V_{ijk}\right) = -\boldsymbol{R}_{ijk}\left(\boldsymbol{W}\right). \tag{24}$$

where $\boldsymbol{W}$ and $\boldsymbol{R}$ are the vectors of cell conserved variables and residuals respectively. The convective terms are discretised using Osher's upwind scheme for its robustness, accuracy, and stability properties. MUSCL variable extrapolation is used to provide second-order accuracy with the Van Albada limiter to prevent spurious oscillations around shock waves.

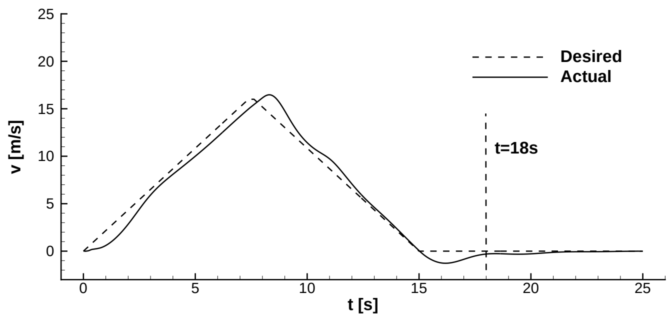


Figure 2: Lateral velocity of the helicopter during the lateral positioning manoeuvre (ADS-33 3.11.8).

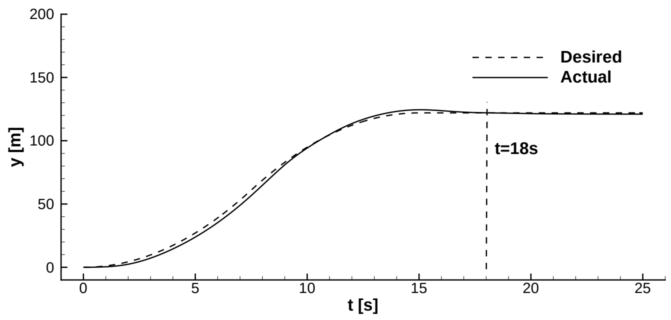


Figure 3: Lateral position of the helicopter during the lateral positioning manoeuvre (ADS-33 3.11.8).

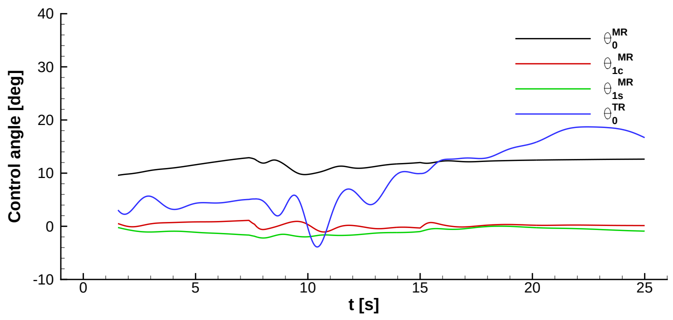


Figure 4: Autopilot commands history for the lateral positioning manoeuvre (ADS-33 3.11.8).

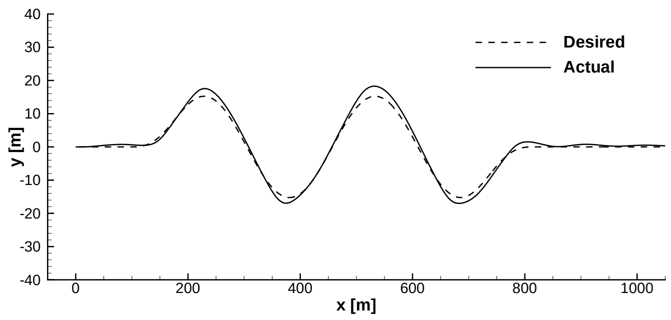


Figure 5: Position in the xy plane of the helicopter during the slalom manoeuvre (ADS-33 3.11.9).

Boundary conditions are set by using ghost cells on the exterior of the computational domain. In the far-field ghost cells are set at the free-stream conditions. At solid boundaries the no-slip condition is set for viscous flows, or ghost values are extrapolated from the interior (ensuring the normal component of the velocity on the solid wall is zero) for Euler flow.

The integration in time of equation 24 to a steady-state solution is performed using a fully implicit time-marching scheme by:

$$\frac{\boldsymbol{W}_{ijk}^{n+1} - \boldsymbol{W}_{ijk}^{n}}{\Delta t} = -\frac{1}{V_{ijk}}\boldsymbol{R}_{ijk}\left(\boldsymbol{W}_{ijk}^{n+1}\right), \qquad (25)$$

where $n+1$ denotes the time $(n+1)*\Delta t$. Equation 25 represents a system of non-linear algebraic equations and to simplify the solution procedure, the flux residual $\boldsymbol{R}_{ijk}\left(\boldsymbol{W}_{ijk}^{n+1}\right)$

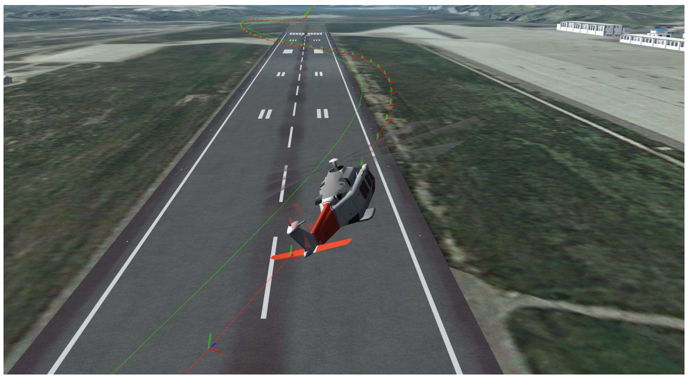


Figure 6: Visualisation of the helicopter during the slalom manoeuvre with a software based on the Presagis VEGA Prime library.

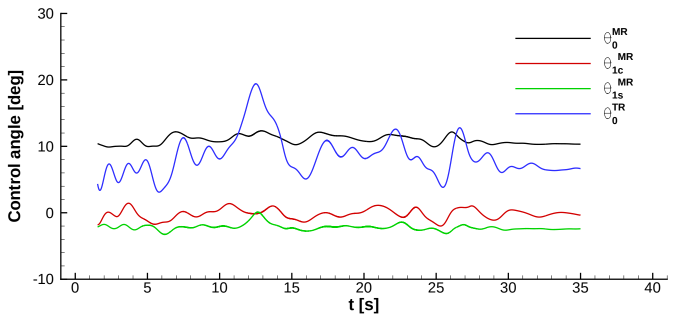


Figure 7: Autopilot commands history for the slalom manoeuvre (ADS-33 3.11.9).

is linearised in time as follows:

$$\boldsymbol{R}_{ijk}\left(\boldsymbol{W}^{n+1}\right) \approx \boldsymbol{R}_{ijk}^{n}\left(\boldsymbol{W}^{n}\right) + \frac{\partial \boldsymbol{R}_{ijk}}{\partial \boldsymbol{W}_{ijk}} \Delta \boldsymbol{W}_{ijk}, \qquad (26)$$

where $\Delta \boldsymbol{W}_{ijk} = \boldsymbol{W}_{ijk}^{n+1} - \boldsymbol{W}_{ijk}^{n}$. Equation 25 now becomes the following linear system:

$$\left[\frac{V_{ijk}}{\Delta t}\mathbf{I} + \frac{\partial \boldsymbol{R}_{ijk}}{\partial \boldsymbol{W}_{ijk}}\right] \Delta \boldsymbol{W}_{ijk} = -\boldsymbol{R}_{ijk}^{n}\left(\boldsymbol{W}^{n}\right). \qquad (27)$$

The left hand side of (27) is then rewritten in terms of primitive variables $\boldsymbol{P}$:

$$\left[\left(\frac{V_{ijk}}{\Delta t}\right)\frac{\partial \boldsymbol{W}_{ijk}}{\partial \boldsymbol{P}_{ijk}} + \frac{\partial \boldsymbol{R}_{ijk}}{\partial \boldsymbol{P}_{ijk}}\right] \Delta \boldsymbol{P}_{ijk} = -\boldsymbol{R}_{ijk}^{n}\left(\boldsymbol{W}^{n}\right), \qquad (28)$$

and the resulting linear system is solved with a GCG (Generalised Conjugate Gradient) iterative solver. Since at steady state the left hand side of (28) must go to zero, the Jacobian $\partial \boldsymbol{R}/\partial \boldsymbol{P}$ can be approximated by evaluating the derivatives of the residuals with a first-order scheme. The first-order Jacobian requires less storage and, being more dissipative, ensures a better convergence rate to the GCG iterations.

The steady state solver for the turbulent case is formulated and solved in an identical manner to that described above for the mean flow. The eddy-viscosity is calculated from the latest values of k and ω (for example) and is used to advance both the mean flow solution and the turbulence solution. An approximate Jacobian is used for the source term by only taking into account the contribution of the dissipation terms $\hat{\boldsymbol{D}}_k$ and $\hat{\boldsymbol{D}}_\omega$, i.e. no account of the production terms is taken on the left hand side of the system.

4.2 *Fully implicit tangent and adjoint solvers*

To compute aerodynamic sensitivities we need to solve either the linear system (2) for the tangent formulation or the discrete adjoint equations (3), and then use the sensitivity equation (1) or (4), respectively. Despite the tangent mode formulation is slightly more efficient for aeromechanics applications, due to the limited number of input parameters, also the adjoint formulation has been implemented, in view of future applications of the sensitivity equation approach in shape optimisation problems.

The linear system (2) of the tangent formulation and the linear system (3) of the adjoint formulation tend to become very stiff as the dimension of the flow problem increases, and therefore a suitable preconditioner is required to stabilize the solution algorithm. Another way to tackle the stiffness problem is to reformulate the linear system as a fixed-point iteration problem [35], where an approximation of the linear system matrix, with better convergence properties, is introduced as a preconditioner to advance the solution at each iteration. Written in terms of primitive variables, the fixed-point iterative schemes reads:

$$\hat{J}\Delta \boldsymbol{P}_x^{n+1} = -\frac{\partial \boldsymbol{R}}{\partial x} - J\boldsymbol{P}_x^{n}, \qquad \text{(tangent form)} \qquad (29)$$

$$\hat{J}^{\mathrm{T}}\Delta \boldsymbol{\lambda}^{n+1} = -\frac{\partial I}{\partial \boldsymbol{P}} - J^{\mathrm{T}}\boldsymbol{\lambda}^{n}, \qquad \text{(adjoint form)} \qquad (30)$$

where we have set

$$J = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{P}}, \quad \hat{J} = \left(\frac{V}{\Delta t}\right)\frac{\partial \boldsymbol{W}}{\partial \boldsymbol{P}} + \left[\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{P}}\right]^{\text{1st}},$$

$$\boldsymbol{P}_x = \frac{\partial \boldsymbol{P}}{\partial x}, \quad \Delta \boldsymbol{P}_x^{n+1} = \boldsymbol{P}_x^{n+1} - \boldsymbol{P}_x^{n},$$

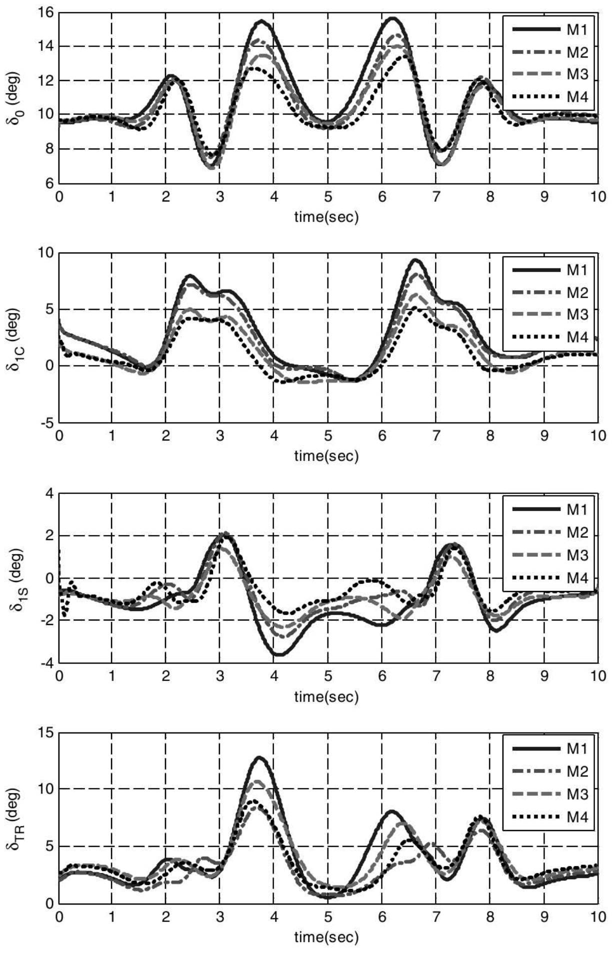


Figure 8: Commands history for the slalom manoeuvre from Ref. [33].

$$\varDelta\boldsymbol{\lambda}^{n+1} = \boldsymbol{\lambda}^{n+1} - \boldsymbol{\lambda}^{n}.$$

The matrix J represents the exact flow residual Jacobian. The natural choice for the preconditioner $\hat{J}$ is the matrix used for the base flow iterative scheme (28), sum of a stabilizing time derivative term and of the first-order residual Jacobian, which approximates J and is more diagonally dominant. The fixed point iteration (29) is solved using the same GCG iterative scheme used by HMB for the base flow implicit update. In the adjoint integration (30) the system matrix is the transpose of the preconditioner J, and the linear system can be solved with a slightly modified version of the GCG solver, that implicitly performs the matrix transposition.

Note also that the iteration schemes (29) and (30) do not require the full exact Jacobian J, but only the matrix-vector product $J\boldsymbol{v}$ or $J^{\mathrm{T}}\boldsymbol{v}$. As explained later in this section, the computer code to perform the former product can be obtained by automatic differentiation in tangent mode of the flow steady residual subroutine, while the code for the latter product can be obtained with automatic differentiation of the same subroutine in adjoint mode. This allows to avoid the storage of J, and hence the computation of sensitivities adds only a small memory overhead to the base solver.

Computation of the product $J\boldsymbol{v}$

To produce the matrix-vector product of the residual Jacobian J and a generic vector $\boldsymbol{v}$ we have isolated the CFD solver code that computes the steady residuals. In particular, the steps involved in the computation of the residuals have been grouped in the single subroutine `steady_residual`, described by the pseudo-code in figure 9 (the calls to the turbulence model subroutines have been omitted for simplicity). The inputs for the subroutine are the vector $\dot{\boldsymbol{X}}$ of mesh velocities, the vector $\boldsymbol{N}$ of surface normals (the mesh metrics), the solution vector $\boldsymbol{P}$ in primitive variables and the free-stream Mach number M_∞. It produces as output the steady residual vector $\boldsymbol{R}$.

The differentiated version of `steady_residual` in tangent mode, named `steady_residual_d`, has been hand-coded and it simply calls the differentiated version of the inner subroutines present in the original statements. These inner subroutines have been differentiated individually by means of the source transformation tool TAPENADE, operated in tangent mode. As a convention, the subroutines differentiated in tangent mode are identified by the postfix "`_d`" appended to the base name. The pseudo-code for `steady_residual_d` is shown in figure 10.

The differentiated residual subroutine has the additional arguments $\delta\dot{\boldsymbol{X}}$, $\delta\boldsymbol{N}$, $\delta\boldsymbol{P}$, δM_∞ and $\delta\boldsymbol{R}$ (`Xdot_d`, `N_d`, `P_d`, `M_d` and `R_d` in the pseudo-code, respectively) that represent the differentials of the quantities involved in the residuals computation. For any value of the input differentials, the action of `steady_residual_d` is to compute the consequent

```
subroutine steady_residual(Xdot, N, P, M, R)
{
    // Set the boundary and halo cells
    call set_boundary(Xdot, N, P, M);

    // Exchange data at block/inter-processor boundaries
    call exchange_halo_cells(P);

    // Calculate residual looping over the blocks
    do for each mesh block
    {
        // Compute inviscid terms with Osher's scheme
        call inviscid_osher(Xdot, N, P, M, R);

        // Compute viscous terms
        call viscous(N, P, M, R);
    }
}
```

Figure 9: Pseudo-code for the computation of the steady residual vector.

variation of the residual vector, that is,

$$\begin{aligned}\delta \boldsymbol{R} \;=\; & \frac{\partial \boldsymbol{R}}{\partial \dot{\boldsymbol{X}}}\delta \dot{\boldsymbol{X}} + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{N}}\delta \boldsymbol{N} + \\ & \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{P}}\delta \boldsymbol{P} + \frac{\partial \boldsymbol{R}}{\partial M_\infty}\delta M_\infty. \end{aligned} \tag{31}$$

The third term in the right hand side is the product between the exact residuals Jacobian matrix with an arbitrary vector of solution variations. Thus, invocation of `steady_residual_d` with $\delta\dot{\boldsymbol{X}} = 0$, $\delta\boldsymbol{N} = 0$, $\delta\boldsymbol{P} = \boldsymbol{P}_x^n$, $\delta M_\infty = 0$ produces the matrix-vector product necessary to compute the right hand side of the fixed-point iteration (29).

Note that the additional memory required to solve equation (2) via the fixed-point iterations (29) is given by the necessity of storing the differentials $\delta\dot{\boldsymbol{X}}$, $\delta\boldsymbol{N}$, $\delta\boldsymbol{P}$ and $\delta\boldsymbol{R}$, which represents only 10-15% of the memory used by the implicit solver for the base flow.

Computation of product $J^{\mathrm{T}}\boldsymbol{v}$

The computation of the matrix-vector product $J^{\mathrm{T}}\boldsymbol{v}$ requires the differentiation in adjoint mode of the steady residual subroutine. As for the tangent mode case, the adjoint code for the main subroutine `steady_residual_b` has been manually coded, while the inner subroutines have been differentiated using TAPENADE in reverse mode. The subroutines differentiated in adjoint mode are labeled by the addition of the postfix "`_b`" to the base name. The pseudo-code for `steady_residual_b` is shown in figure 11.

Note that the adjoint code is more complex with respect to the tangent mode code. There are calls to the non-differentiated subroutines at the beginning, in what is called the *forward sweep*, where all the quantities needed during the subsequent back-propagation of the derivatives are calculated. The back-propagation of derivatives is done in the *reverse sweep*, where the differentiated version of the subroutines called in the original statements are executed in reverse order. Observe also that not every call to the non-differentiated subroutines is present in the adjoint code forward sweep, like the call to `inviscid_osher` for instance, since the values computed by this subroutines are not needed during the reverse sweep.

For any value of the input residuals differentials $\delta\boldsymbol{R}$ (`R_b` in the pseudo-code), the action of the adjoint code in `steady_residual_b` is to compute the vectors $\delta\dot{\boldsymbol{X}}$, $\delta\boldsymbol{N}$, $\delta\boldsymbol{P}$ and δM_∞ (`Xdot_b`, `N_b`, `P_b` and `M_b` in the pseudo-code, respectively) of weighted partial derivatives of the residuals:

$$\delta\dot{\boldsymbol{X}} = \left(\frac{\partial \boldsymbol{R}}{\partial \dot{\boldsymbol{X}}}\right)^{\mathrm{T}} \delta\boldsymbol{R}, \tag{32}$$

$$\delta\boldsymbol{N} = \left(\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{N}}\right)^{\mathrm{T}} \delta\boldsymbol{R}, \tag{33}$$

$$\delta\boldsymbol{P} = \left(\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{P}}\right)^{\mathrm{T}} \delta\boldsymbol{R}, \tag{34}$$

$$\delta M_\infty = \left(\frac{\partial \boldsymbol{R}}{\partial M_\infty}\right)^{\mathrm{T}} \delta\boldsymbol{R}. \tag{35}$$

It is interesting to observe that the role of the dual variables $\delta\dot{\boldsymbol{X}}$, $\delta\boldsymbol{N}$, $\delta\boldsymbol{P}$, δM_∞ and $\delta\boldsymbol{R}$ in `steady_residual_d` is reversed in `steady_residual_b`: input quantities in the former are output in the latter, and *vice versa*.

The vector $\delta\boldsymbol{P}$ is the product between the transpose of the exact residuals Jacobian matrix and an arbitrary vector of residual variations. It follows that a call of `steady_residual_b` with $\delta\boldsymbol{R} = \boldsymbol{\lambda}^n$ produces the matrix-vector product appearing in the right hand side of the fixed-point iteration (30).

The needed additional memory for calling `steady_residual_b` is given by the storage for the variables $\delta\dot{\boldsymbol{X}}$, $\delta\boldsymbol{N}$, $\delta\boldsymbol{P}$ and $\delta\boldsymbol{R}$. To this needs to be added the memory allocated temporarily by the inner subroutines differentiated in reverse mode by TAPENADE. Reverse mode differentiated code requires in fact to save some of the quantities computed during the forward sweep, when they are necessary to back-propagate derivatives at some stages of the reverse sweep. The temporary storage allocated by the reverse differentiated routines called by `steady_residual_b` is however very limited, and the associated memory overhead is almost negligible.

```
subroutine steady_residual_d(Xdot, Xdot_d, N, N_d, P, P_d, M, M_d, R, R_d)
{
    // Set the boundary and halo cells
    call set_boundary_d(Xdot, Xdot_d, N, N_d, P, P_d, M, M_d);

    // Exchange data at block/inter-processor boundaries
    call exchange_halo_cells_d(P, P_d);

    // Calculate residual differentials looping over the blocks
    do for each mesh block
    {
        // Compute inviscid terms differentials
        call inviscid_osher_d(Xdot, Xdot_d, N, N_d, P, P_d, M, M_d, R, R_d);

        // Compute viscous terms differentials
        call viscous_d(N, N_d, P, P_d, M, M_d, R, R_d);
    }
}
```

Figure 10: Pseudo-code for the steady residual subroutine differentiated in tangent mode.

```
subroutine steady_residual_b(Xdot, Xdot_b, N, N_b, P, P_b, M, M_b, R, R_b)
{
    // Set the boundary and halo cells
    call set_boundary(Xdot, N, P, M);

    // Exchange data at block/inter-processor boundaries
    call exchange_halo_cells(P);

    // Calculate residual differentials looping over the blocks
    do for each mesh block
    {
        // Compute viscous terms differentials
        call viscous_b(N, N_b, P, P_b, M, M_b, R, R_b);

        // Compute inviscid terms differentials
        call inviscid_osher_b(Xdot, Xdot_b, N, N_b, P, P_b, M, M_b, R, R_b);
    }

    // Exchange differentials at block/inter-processor boundaries
    call exchange_halo_cells_b(P_b);

    // Set the differentials at boundary and halo cells
    call set_boundary_b(Xdot, Xdot_b, N, N_b, P, P_b, M, M_b);
}
```

Figure 11: Pseudo-code for the steady residual subroutine differentiated in adjoint mode.

4.3 *Aerodynamic sensitivities for fixed wing aircrafts*

In the previous section we described how to solve the linear system yielding the derivative $\partial \boldsymbol{P}/\partial x$ for the tangent method or the adjoint variables vector $\boldsymbol{\lambda}$ for the adjoint method. Here we briefly describe how to compute the other terms of the sensitivity equations (1) and (4), namely $\partial \boldsymbol{R}/\partial x$, $\partial I/\partial \boldsymbol{P}$ and $\partial I/\partial x$. For fixed wing aircrafts, the outputs I of interest shall be any of the force and moment coefficients C_L, C_D, C_Y, C_l, C_m and C_n, while the independent parameters x shall be either the incidence α, the sideslip β, the free-stream Mach number M_∞ or any of the three rotational rates p, q and r around the wind axes.

The terms $\partial \boldsymbol{R}/\partial\alpha$, $\partial \boldsymbol{R}/\partial\beta$. These partial derivatives represent the variation of the residual vector $\boldsymbol{R}$ due to a variation of angle of attack or sideslip. For the computation of any of this two terms it is convenient to express the derivative as follows:

$$\frac{\partial \boldsymbol{R}}{\partial x} = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{N}}\frac{\partial \boldsymbol{N}}{\partial \boldsymbol{X}}\frac{\partial \boldsymbol{X}}{\partial x}, \quad x \in \{\alpha, \beta\}, \tag{36}$$

where the first derivative in the right hand side is the variation of the residual due to a change in the mesh metrics, the second is the variation of the metrics due to a change in mesh coordinates (denoted by the vector $\boldsymbol{X}$), and the third is the variation of the mesh coordinates given by a change in the input parameter.

Recalling that the action of the steady residual subroutine differentiated in tangent mode is given by (31), the desired term $\partial \boldsymbol{R}/\partial x$ can be computed by a single call to `steady_residual_d` with $\delta\dot{\boldsymbol{X}} = 0$, $\delta \boldsymbol{N} = (\partial \boldsymbol{N}/\partial \boldsymbol{X}) \cdot (\partial \boldsymbol{X}/\partial x)$, $\delta \boldsymbol{P} = 0$, $\delta M_\infty = 0$. The term $\partial \boldsymbol{N}/\partial \boldsymbol{X}$ can be obtained by tangent differentiation of the subroutine computing the mesh metrics, while the term $\partial \boldsymbol{X}/\partial x$, with $x \in \{\alpha, \beta\}$, can be computed directly, since it represents the variation of the mesh coordinates due to a variation of angle of attack or sideslip, respectively.

The terms $\partial \boldsymbol{R}/\partial p$**,** $\partial \boldsymbol{R}/\partial q$**,** $\partial \boldsymbol{R}/\partial r$. This partial derivatives represent the variation of the residual vector $\boldsymbol{R}$ due to a variation of the rotational speeds around the three wind axes. The computation of this terms requires to express the derivative as:

$$\frac{\partial \boldsymbol{R}}{\partial x} = \frac{\partial \boldsymbol{R}}{\partial \dot{\boldsymbol{X}}} \frac{\partial \dot{\boldsymbol{X}}}{\partial x}, \quad x \in \{p, q, r\}, \tag{37}$$

where the first derivative in the right hand side is the variation of the residual due to a change in the mesh velocities, and the second derivative is the variation of the mesh velocities given by a change in the input parameter. The desired term $\partial \boldsymbol{R}/\partial x$ can be computed by a single call to `steady_residual_d` with $\delta \dot{\boldsymbol{X}} = \partial \dot{\boldsymbol{X}}/\partial x$, $\delta \boldsymbol{N} = 0$, $\delta \boldsymbol{P} = 0$, $\delta M_\infty = 0$. The term $\partial \dot{\boldsymbol{X}}/\partial x$, with $x \in \{p, q, r\}$, is relatively easy to compute, as it represents the variation of the mesh velocities due to a change in the rotational speed around the three wind axis.

The term $\partial \boldsymbol{R}/\partial M_\infty$. This partial derivative represents the dependence of the residual vector upon the free-stream Mach number. Since the Mach number appears explicitly in the HMB formulation, and hence in the steady residual code, this term can be computed by calling `steady_residual_d` with $\delta \dot{\boldsymbol{X}} = 0$, $\delta \boldsymbol{N} = 0$, $\delta \boldsymbol{P} = 0$, $\delta M_\infty = 1$.

The terms $\partial I/\partial \alpha$**,** $\partial I/\partial \beta$. These partial derivatives represent the direct dependence of force and moment coefficients upon the variation of angle of attack or sideslip. They can be computed in a similar way to the residual terms above, by first expressing the derivative as

$$\frac{\partial I}{\partial x} = \frac{\partial I}{\partial \boldsymbol{N}} \frac{\partial \boldsymbol{N}}{\partial \boldsymbol{X}} \frac{\partial \boldsymbol{X}}{\partial x}, \quad x \in \{\alpha, \beta\}. \tag{38}$$

The term $\partial I/\partial \boldsymbol{N}$ expresses the dependence of force and moments upon the mesh metrics and is obtained by differentiating the subroutine for computing the integrated loads in tangent mode with respect to mesh metrics. The other terms have been already discussed above.

The terms $\partial I/\partial p$**,** $\partial I/\partial q$**,** $\partial I/\partial r$. These partial derivatives are zero, because there is no direct dependence of force and moment coefficients upon the rotational speeds around the wind axes.

The term $\partial I/\partial M_\infty$. This partial derivative represents the dependence of the force coefficient upon the free-stream Mach number. It is obtained by differentiating the subroutine for computing the integrated loads in tangent mode with respect to the Mach number.

The term $\partial I/\partial \boldsymbol{P}$. These partial derivatives represent the variation of force and moment coefficients due to a variation of the flow variables. They can be efficiently computed by differentiating the subroutine for computing the integrated loads with respect to the flow variables in adjoint mode. The use of the adjoint mode for this computation is justified by the fact that the input is represented by all the flow variables, which clearly outnumber the outputs, represented by the six force and moment coefficients.

4.4 *Aerodynamic sensitivities for rotors in hover*

Hover computations are performed in HMB formulating the equations in the rotating reference frame of the rotor, so that the solution is steady. Periodic boundary conditions are also used to take advantage of the symmetry of the problem, allowing for the discretization of a single rotor blade.

With respect to the sensitivities of rotorcraft in hover, the outputs I of interest are the thrust coefficient $C_T = T/(\rho A \Omega^2 R^2)$ and the torque coefficient $C_Q = T/(\rho A \Omega^2 R^3)$, while the independent parameters x are the collective pitch θ, the flap angle β and the vertical velocity w. Here follows a brief description of the sensitivity equation terms required to compute the aerodynamic derivatives for rotors in hovering flight.

The terms $\partial \boldsymbol{R}/\partial \theta$**,** $\partial \boldsymbol{R}/\partial \beta$. These partial derivatives represent the variation of the residual vector $\boldsymbol{R}$ due to a change of the pitch or flap angle. For the computation of any of this two terms it is convenient to express the derivative as follows:

$$\frac{\partial \boldsymbol{R}}{\partial x} = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{N}} \frac{\partial \boldsymbol{N}}{\partial \boldsymbol{X}} \frac{\partial \boldsymbol{X}}{\partial x}, \quad x \in \{\theta, \beta\}. \tag{39}$$

The first two derivatives in the right hand side can be computed as already explained above for the fixed wing aircraft case. The third derivative, $\partial \boldsymbol{X}/\partial x$, with $x \in \{\theta, \beta\}$, is computed by differentiation in tangent mode of a grid deformation routine. This routine evaluate the mesh differentials due to a rigid rotation around the pitch or flap axis of the blade surface, and of the consequent volume grid deformation, which is set up so as to keep the other mesh boundaries (periodic planes and far-field) fixed.

The term $\partial \boldsymbol{R}/\partial w$. This partial derivatives represent the variation of the residual vector $\boldsymbol{R}$ due to a variation of the velocity along the rotor axis of rotation. The computation of this terms requires to express the derivative as:

$$\frac{\partial \boldsymbol{R}}{\partial w} = \frac{\partial \boldsymbol{R}}{\partial \dot{\boldsymbol{X}}} \frac{\partial \dot{\boldsymbol{X}}}{\partial w}. \tag{40}$$

The first derivative has been already discussed. The second derivative is easy to compute, as it represents a uniform variation of the mesh velocities in the direction of the rotor rotation axis.

The terms $\partial I/\partial \theta$**,** $\partial I/\partial \beta$. These partial derivatives represents the direct dependence of force and moment coefficients upon the pitch and flap angle. They can be computed in a similar way to the $\partial I/\partial \alpha$ and $\partial I/\partial \beta$ terms for the fixed wing aircraft, by expressing the derivative as

$$\frac{\partial I}{\partial x} = \frac{\partial I}{\partial \boldsymbol{N}} \frac{\partial \boldsymbol{N}}{\partial \boldsymbol{X}} \frac{\partial \boldsymbol{X}}{\partial x}, \quad x \in \{\alpha, \beta\}, \tag{41}$$

and computing $\partial \boldsymbol{X}/\partial x$ as described here above for the derivatives of the residual vector with respect to the same input variables.

4.5 *Numerical results*

4.5.1 NACA0012 airfoil

The first considered test case is relative to the inviscid flow around a NACA0012 airfoil. Sensitivity computations for this

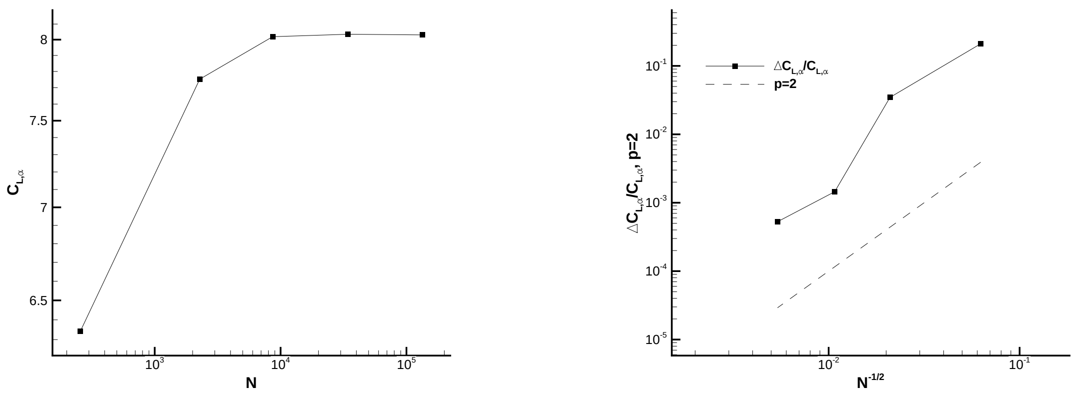


Figure 12: HMB grid convergence study for the NACA0012, $\alpha = 0°$, $M_\infty = 0.5$.

Grid size (# cells)	$\partial C_L/\partial\alpha$	$\partial C_M/\partial\alpha$	$\partial C_L/\partial q$	$\partial C_M/\partial q$
256	6.3410110158	-1.6086460408	8.65716016820	-2.7304784305
2278	7.7519604640	-2.0313981213	11.2142666118	-3.7744321358
8646	8.0187471791	-2.1086825882	11.6491348404	-3.9194602271
34320	8.0346114695	-2.1129930362	11.6730662785	-3.9270310121
134160	8.0305361599	-2.1115952062	11.6669030887	-3.9250582182

Table 2: Grid convergence study for the NACA0012 airfoil.

	$\partial C_L/\partial\alpha$	$\partial C_M/\partial\alpha$	$\partial C_L/\partial q$	$\partial C_M/\partial q$
Mader *et al.* [2]	7.9617567582	-2.0686236849	11.9210000000	-4.0000000000
HMB tangent	8.0305361599	-2.1115952062	11.6669030887	-3.9250582182
Difference [%]	0.86	2.07	2.13	1.87
HMB adjoint	8.0305361602	-2.1115952062	11.6669031274	-3.9250582304
Difference [%]	0.86	2.07	2.13	1.87

Table 3: Comparison of HMB sensitivity results on the fine grid with reference results for the NACA0012 airfoil.

airfoil have been performed by Limache and Cliff [1] and by Mader *et al.* [2].

A grid convergence study has been first done in order to select a reference grid for the subsequent computations. Table 2 shows the values of the flight mechanics derivatives for the airfoil at zero incidence and $M_\infty = 0.5$, obtained solving the sensitivity equation in tangent mode with grids of increasing size. The pseudo-time CFL for all the computations was set to 40.

Figure 12 show, for instance, the convergence of the derivative $\partial C_L/\partial\alpha$. As expect, the convergence is second order in space, and the fine grid with 134160 cells has been selected for the validation against reference results. The comparison between HMB in tangent mode, HMB in adjoint mode and results taken from Mader *et al.* [2] is given in table 3. As can be seen, the difference between the HMB sensitivities computed in tangent and adjoint mode is negligible, being the results equal up to the eighth significative digit, and thus proving the consistency of the method. The difference between the HMB and the reference results is below 2.2% which, considering the different grids and discretization methods, may be considered a good agreement.

4.5.2 M6 wing

The second test case considers the inviscid flow around an ONERA M6 wing in transonic flight, with free-stream Mach number $M_\infty = 0.8395$ and angle of attack $\alpha = 3.06°$. Again, the HMB results have been compared with the reference results of Mader *et al.* [2]. The HMB mesh is composed by 442200 cells (see figure 13), while the reference mesh has 14.7M cells. The pseudo-time CFL for both the base flow solver and for the tangent/adjoint solver was set to 20.

Figure 14 shows the contours of the pressure sensitivity with respect to the angle of attack, extracted from the solution to equation (2). As one could expect, the highest values of the pressure derivative are located in the shock region, while the lower surface presents instead a more regular positive pressure variation.

The comparison between HMB and reference flight mechanics derivatives is reported in table 4. The HMB results are in good agreement with the reference results, exception made for the derivatives involving integration in direction parallel to the wing planform and the derivatives with respect to the free-stream Mach number. The disagreement of the former derivatives is not surprising, since the forces parallel to wing planform are small and the results are therefore significantly

	C_L	C_D	C_Y	C_l	C_m	C_n
α	5.5800E+00	4.6593E-01	2.3675E-10	2.2080E-10	-4.1428E+00	-1.5021E-10
β	1.4600E-09	1.1901E-10	-7.8594E-03	-1.2684E-01	-1.1942E-09	1.7950E-02
M_∞	8.1377E-01	1.6121E-01	-7.6705E-10	-6.9969E-09	-9.3351E-01	8.7624E-10
p	6.8034E-13	1.0360E-10	2.2844E-01	-1.5445E+00	-1.1599E-10	-2.0465E-01
q	1.2976E+01	5.8162E-01	5.0051E-10	2.3937E-10	-1.0957E+01	-2.4555E-10
r	-4.3888E-09	-6.8873E-10	-4.7777E-02	4.5109E-01	4.8487E-09	1.8383E-02

HMB tangent mode

	C_L	C_D	C_Y	C_l	C_m	C_n
α	5.5772E+00	4.5422E-01	0.0000E+00	0.0000E+00	-4.0932E+00	0.0000E+00
β	-1.5168E-05	4.0961E-06	-6.8243E-03	-1.2667E-01	1.4722E-05	1.4088E-02
M_∞	7.7872E-01	1.3225E-01	0.0000E+00	0.0000E+00	-8.3126E-01	0.0000E+00
p	-2.2748E-06	3.3527E-07	2.3829E-01	-1.4971E+00	2.0630E-06	-2.1088E-01
q	1.3474E+01	6.0271E-01	0.0000E+00	0.0000E+00	-1.1437E+01	0.0000E+00
r	1.5217E-06	-4.4730E-07	-4.6747E-02	4.4085E-01	-1.4838E-06	2.3991E-02

Mader *et al.* [2]

	C_L	C_D	C_Y	C_l	C_m	C_n
α	0.05	2.58			1.21	
β			15.17	0.14		27.41
M_∞	4.50	21.90			12.30	
p			4.13	3.16		2.96
q	3.70	3.50			4.20	
r			2.20	2.32		23.37

Difference [%]

Table 4: Comparison of HMB sensitivity results with reference results for the M6 wing.

	$\partial C_L/\partial M_\infty$	$\partial C_D/\partial M_\infty$	$\partial C_Y/\partial M_\infty$	$\partial C_l/\partial M_\infty$	$\partial C_m/\partial M_\infty$	$\partial C_n/\partial M_\infty$
HMB FD	8.1360E-01	1.6127E-01	≈ 0	≈ 0	-9.3344E-01	≈ 0
HMB AD	8.1376E-01	1.6121E-01	≈ 0	≈ 0	-9.3351E-01	≈ 0
Difference [%]	0.02	0.04			0.01	

Table 5: Comparison of Mach sensitivity computed with AD and FD for the M6 wing.

grid dependent.

It is instead more difficult to justify the disagreement of the free-stream Mach number derivatives, and therefore we verified the obtained results against finite differences. The comparison is given in table 5. As can be seen, the finite difference result agrees with very good approximation to the result obtained solving the sensitivity equation.

Regarding the performance of the method, the time for computing the base flow with a relative convergence of 10^{-9} was 23 minutes on a 4-core 3.3GHz Intel Xeon. For the tangent method, the computational time for a single sensitivity computation with a relative convergence of 10^{-9} was between 45 and 80% the time needed by the base flow solver, depending on the input variable. For the adjoint method, the computational time was between 50 and 90% of the base flow solver time, depending on the output quantity. The overall time for the aerodynamic sensitivities computation is 100 minutes for the tangent mode solver and 118 minutes for the adjoint solver.

Note that for computing all the aerodynamic derivatives with finite differences, 12 CFD solutions are needed, for an overall computational time of 270 minutes.

4.5.3 ONERA 7AD rotor

To verify the computation of aerodynamic sensitivities for rotors we analyze the inviscid flow of an ONERA 7AD rotor in hover flight. The tip Mach number is set to $M_{\text{tip}} = 0.6612$ and the collective pitch is $\theta_{0.7} = 7.5°$. The mesh is a multi-block grid with 880000 cells for a single blade, where the effect of the other blades is accounted for using periodic boundary conditions (see figure 15). At the far-field boundaries Froude conditions are imposed to better represent the flow induced by the rotor and to avoid the formation of artificial recirculation regions. The CFL for the base flow solver and for the tangent solver was set to 8.

The derivatives of the thrust coefficient C_T and of the torque coefficient C_Q computed with HMB solving the sensitivity equation in tangent mode are shown in table 6, where θ denotes the collective angle, β the flap angle and w the vertical velocity of the rotor. Since no sensitivity result is available in the literature for this rotor, we compared the sensitivities computed with the automatically differentiated code

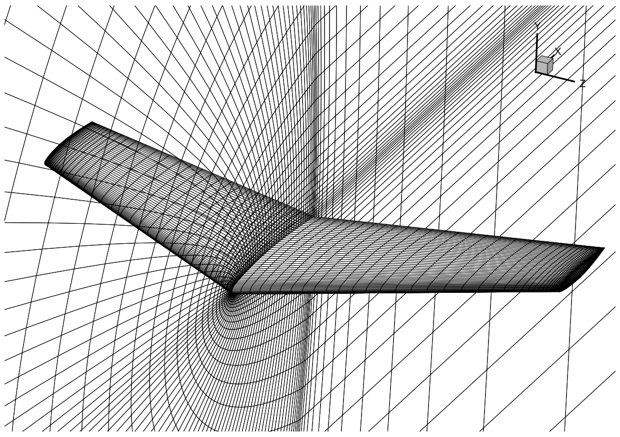


Figure 13: Computational grid for the ONERA M6 wing.

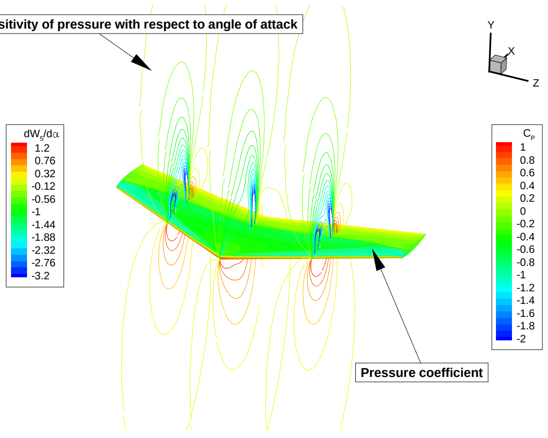


Figure 14: Pressure sensitivity with respect to the angle of attack of the ONERA M6 wing, $\alpha = 3.06°$, $M_\infty = 0.8395$.

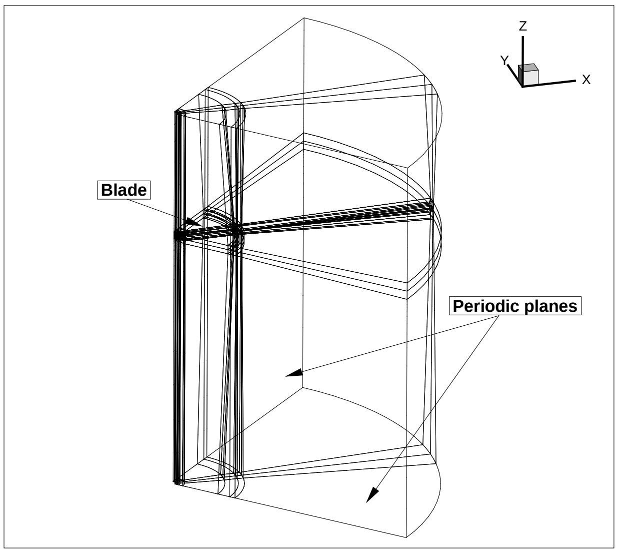


Figure 15: Computational grid for the ONERA 7AD rotor.

with those obtained via finite differencing. The comparison relative to the collective angle sensitivity is shown in table 7, where a good agreement between the two methods can be observed.

In figure 16 we also compared the distribution of the pressure sensitivity with respect to the collective angle $\partial P/\partial\theta$ computed with the AD code, solving equation (2), and with finite differences. The isolines of $\partial P/\partial\theta$ are shown for a section of the flow field at 75% of the blade span. There is a good agreement between the AD and FD results, but the isolines obtained with FD exhibit a more noisy behaviour, due to cancellation errors.

The computational time for the base flow of the rotor case was 6.3 hours on two 4-core 3.3GHz Intel Xeon. The time

	C_T	C_Q
θ	1.2296E-01	1.2599E-02
β	7.2029E-03	6.7284E-04
w	1.1053E-01	-9.5130E-04

Table 6: HMB sensitivity results for the ONERA 7AD rotor.

	$\partial C_T/\partial\theta$	$\partial C_Q/\partial\theta$
HMB FD	1.20826221E-01	1.25691808E-02
HMB AD	1.22959168E-01	1.25986274E-02
Difference [%]	1.73	0.23

Table 7: Comparison of collective angle sensitivity computed with AD and FD for the ONERA 7AD rotor.

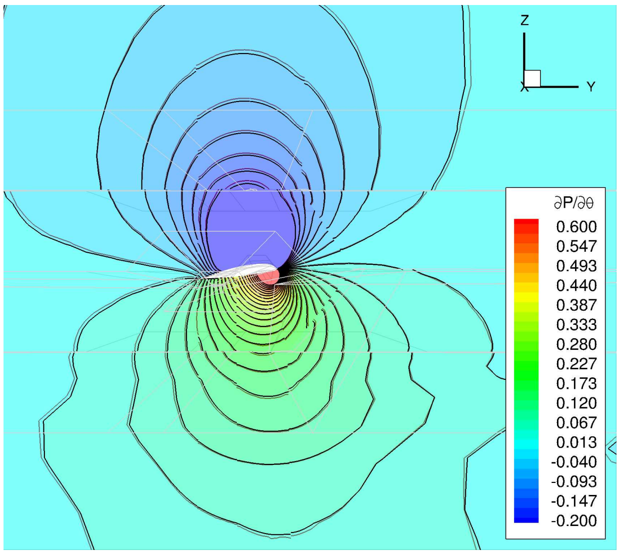


Figure 16: Pressure sensitivity with respect to the collective angle for the ONERA 7AD rotor. Colors and black isolines were computed solving equation (2), red isolines were computed via finite differences.

spent for each one of the three sensitivity solutions in tangent mode was 5 hours (80% of the base flow time).

5 Conclusions

This paper presented the implementation and assessment of automatic differentiation and discrete adjoint methods within the framework of implicit CFD solvers. In itself this is not a common approach since explicit solvers are easier to differentiate and modify for the adjoint method to be implemented. After implementation, validation of the method has been attempted using established cases found in the literature for airfoils and wings. The results show that the current method achieves results in agreement with theory and with published solutions. At a second step, the aerodynamics derivatives for rotors in hover flight were attempted and results are found to agree with finite difference computations.

Acknowledgement The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement n° 264672.

COPYRIGHT STATEMENT

References

[1] A. C. Limache and E. M. Cliff. “Aerodynamic sensitivity theory for rotary stability derivatives”. In: *Journal of Aircraft* 37.4 (2000), pp. 676–683.

[2] C. A. Mader and J. R. R. A. Martins. “Computation of aircraft stability derivatives using an automatic differentiation adjoint approach”. In: *AIAA Journal* 49.12 (2011), pp. 2737–2750.

[3] C. A. Mader et al. “ADjoint: An approach for the rapid development of discrete adjoint solvers”. In: *AIAA Journal* 46.4 (2008), pp. 863–873.

[4] L. Hascoët and V. Pascual. *TAPENADE 2.1 User’s Guide*. Tech. rep. 300. Sophia Antipolis: INRIA, 2004.

[5] *TROPICS internet website*. `http://www-sop.inria.fr/tropics/tapenade.html`.

[6] *Handling Qualities Requirements for Military Rotorcraft*. Tech. rep. ADS-33E-PRF. US Army Aviation and Missile Command, Mar. 2000.

[7] R. Steijl, G. Barakos, and K. Badcock. “A framework for CFD analysis of helicopter rotors in hover and forward flight”. In: *International journal for numerical methods in fluids* 51.8 (2006), pp. 819–847.

[8] G. Barakos et al. “Development of CFD capability for full helicopter engineering analysis”. In: *31st European Rotorcraft Forum*. 91. 2005.

[9] D. Jones, J.-D. Müller, and F. Christakopoulos. “Preparation and assembly of discrete adjoint CFD codes”. In: *Computers and Fluids* 46.1 (2011), pp. 282–286.

[10] O. Pironneau. “On Optimum Design in Fluid Mechanics”. In: *Journal of Fluid Mechanics* 64.Part 1 (1974), pp. 97–110.

[11] A. Jameson. “Aerodynamic design via control theory”. In: *Journal of Scientific Computing* 3.3 (1988), pp. 233–260.

[12] A. Jameson, L. Martinelli, and N. A. Pierce. “Optimum aerodynamic design using the Navier-Stokes equations”. In: *Theoretical and Computational Fluid Dynamics* 10.1-4 (1998), pp. 213–237.

[13] A. Jameson. “Control theory for optimum design of aerodynamic shapes”. In: *Proceedings of the IEEE Conference on Decision and Control*. Vol. 1. 1990, pp. 176–179.

[14] A. Jameson, N. A. Pierce, and L. Martinelli. “Optimum aerodynamic design using the Navier-Stokes equations”. In: *AIAA Paper 97-0101* 97.101 (1997).

[15] J. Reuther and A. Jameson. “Control based airfoil design using the Euler equations”. In: *AIAA Paper* 94.4272 CP (1994).

[16] J. Reuther et al. “Aerodynamic shape optimization of complex aircraft configurations via an adjoint formulation”. In: *AIAA Paper* 96.94 (1996).

[17] J. Elliott and J. Peraire. “Practical three-dimensional aerodynamic design and optimization using unstructured meshes”. In: *AIAA Journal* 35.9 (1997), pp. 1479–1485.

[18] W. K. Anderson and D. L. Bonhaus. “Airfoil Design on Unstructured Grids for Turbulent Flows”. In: *AIAA Journal* 37.2 (1999), pp. 185–191.

[19] J. C. Newman III et al. “Overview of sensitivity analysis and shape optimization for complex aerodynamic configurations”. In: *Journal of Aircraft* 36.1 (1999), pp. 87–96.

[20] M. B. Giles and N. A. Pierce. “An introduction to the adjoint approach to design”. In: *Flow, Turbulence and Combustion* 65.3-4 (2000), pp. 393–415.

[21] B. Mohammadi. “Optimal shape design, reverse mode of automatic differentiation and turbulence”. In: *AIAA Paper 97-0099* 97.99 (1997).

[22] B. Mohammadi. “Practical application to fluid flows of automatic differentiation for design problems”. In: *Inverse Design and Optimization Methods, Lecture Series 1997-05*. Rhode Saint Genese, Belgium: von Karman Institute for Fluid Dynamics, 1997.

[23] A. Griewank. *Evaluating Derivatives*. SIAM, 2000.

[24] A. Carle and M. Fagan. “ADIFOR 3.0 Overview”. In: *Rice University, TR CAAM-TR-00-02* (2000).

[25] R. Giering, T. Kaminski, and T. Slawig. “Generating efficient derivative code with TAF: Adjoint and tangent linear Euler flow around an airfoil”. In: *Future Generation Computer Systems* 21.8 (2005), pp. 1345–1355.

[26] M. S. Gockenbach. *Understanding code generated by TAMC*. IAAA Paper TR00-29. Texas, USA: Department of Computational and Applied Mathematics, Rice University, 2000.

[27] *AUTODIFF internet website*. `http://www.autodiff.org`.

[28] C. A. Mader and J. R. R. A. Martins. “Derivatives for time-spectral computational fluid dynamics using an automatic differentiation adjoint”. In: *AIAA Journal* 50.12 (2012), pp. 2809–2819.

[29] S. Choi et al. “Helicopter rotor design using a time-spectral and adjoint-based method”. In: *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, MAO*. 2008.

[30] D. A. Peters and N HaQuang. “Dynamic inflow for practical applications”. In: *J. of the American Helicopter Society* 33.4 (1988), pp. 64–68.

[31] H. Kwakernaak. *Linear optimal control systems*. New York: Wiley Interscience, 1972.

[32] J. P. How. *Principles of Optimal Control*. Lecture notes. 2008.

[33] C.-J. Kim et al. “Selection of Rotorcraft Models for Application to Optimal Control Problems”. In: *JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS* 31.5 (2008).

[34] *PRESAGIS website*. `http://www.presagis.com`.

[35] M. B. Giles. “On the Iterative Solution of Adjoint Equations”. In: *Automatic Differentiation of Algorithms*. Ed. by George Corliss et al. Springer New York, 2002, pp. 145–151. ISBN: 978-1-4612-6543-6.