Centrum voor Wiskunde en Informatica

**REPORT***RAPPORT*

*MAS*

Modelling, Analysis and Simulation

*Modelling, Analysis and Simulation*

# Improving the efficiency of aerodynamic shape optimization on unstructured meshes

G. Carpentieri, M.J.L. van Tooren, B. Koren

**REPORT MAS-E0604 JANUARY 2006**

Centrum voor Wiskunde en Informatica (CWI) is the national research institute for Mathematics and Computer Science. It is sponsored by the Netherlands Organisation for Scientific Research (NWO).
CWI is a founding member of ERCIM, the European Research Consortium for Informatics and Mathematics.

CWI's research has a theme-oriented structure and is grouped into four clusters. Listed below are the names of the clusters and in parentheses their acronyms.

Probability, Networks and Algorithms (PNA)

Software Engineering (SEN)

Modelling, Analysis and Simulation (MAS)

Information Systems (INS)


P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 1386-3703

# Improving the efficiency of aerodynamic shape optimization on unstructured meshes

ABSTRACT
In this paper the exact discrete adjoint of a finite volume formulation on unstructured meshes for the Euler equations in two dimensions is derived and implemented to support aerodynamic shape optimization. The accuracy of the discrete exact adjoint is demonstrated and compared with that of the approximate adjoint. A solution process for the adjoint equations, which is similar to that used for the flow equations, is modified to account for multiple functionals. An optimization framework, which couples an analytical shape parameterization to the flow/adjoint solver and to a Sequential Quadratic Programming optimization algorithm, is tested on constrained and unconstrained airfoil design cases. Preliminary results are also presented for a Sequential Linear Programming algorithm, which appears to be able to deal properly with constrained design in spite of its simplicity.

*2000 Mathematics Subject Classification:* 65Kxx, 65N30, 76H05
*Keywords and Phrases:* shape optimization; adjoint-equation method; Euler equations; finite-volume discretization; airfoil flows
*Note*: This research was supported by the Dutch Technology Foundation STW, applied science division of NWO and the technology program of the Dutch Ministry of Economic Affairs.

# Improving the Efficiency of Aerodynamic Shape Optimization on Unstructured Meshes

Giampietro Carpentieri* and Michel J.L. van Tooren†
*Delft University of Technology, Kluyverweg 1, 2629 HS Delft, The Netherlands*
Barry Koren‡
*Centre for Mathematics and Computer Science, Kruislaan 413, 1090 SJ Amsterdam, The Netherlands*

**In this paper the exact discrete adjoint of a finite volume formulation on unstructured meshes for the Euler equations in two dimensions is derived and implemented to support aerodynamic shape optimization. The accuracy of the discrete exact adjoint is demonstrated and compared with that of the approximate adjoint. A solution process for the adjoint equations, which is similar to that used for the flow equations, is modified to account for multiple functionals. An optimization framework, which couples an analytical shape parameterization to the flow/adjoint solver and to a Sequential Quadratic Programming optimization algorithm, is tested on constrained and unconstrained airfoil design cases. Preliminary results are also presented for a Sequential Linear Programming algorithm, which appears to be able to deal properly with constrained design in spite of its simplicity.**

## I. Introduction

Aerodynamic shape optimization can be efficiently performed by means of the adjoint method which enables functional gradients to be calculated at the price of roughly one additional flow computation.[1] This method is very attractive in its discrete approach where the adjoint problem is directly formulated on the discretized flow equations. The method is straightforward to understand; only some linear algebra is involved. However, the derivation of the discrete adjoint code can be challenging due to the complexity of differentiating the original discrete formulation. Hand–coding the adjoint may require a lot of human work.[2] Automatic Differentiation tools to derive the code[3–5] may be successfully applied.

In this work the adjoint algorithm is hand–coded following a methodology outlined in previous work,[6,7] in the context of implicit solver development. The discrete adjoint implemented here is exact since the residual Jacobian is obtained from the exact differentiation of the residual vector. It gives gradients that are consistent with the ones obtained by finite difference applied to the original solver. In practice, to simplify the derivation of the adjoint code, different approximations can be made in the differentiation. In turn, those approximations can have a detrimental effect on the accuracy.[2] Some of those approximations are discussed here.

The solution process for the adjoint problem, despite the linearity of the equations, can be the same time marching adopted in the flow solver.[2,4,8,9] This greatly simplifies the coding and gives a robust adjoint solver. Here an implicit time stepping is used which is modified to account for the solution of the adjoint of multiple functionals. In constrained shape optimization more than one functional is usually involved so that it is convenient to solve the adjoint problem simultaneously rather than sequentially.

The flow and the adjoint solver are part of an optimization framework, which includes the shape parameterization and an optimization algorithm. The latter is in charge of driving the optimization process. The shape parameterization used here is based upon orthogonal Chebyshev polynomials.[10] This parameterization gives completeness in the design space and behaves smoothly during the optimization process even in

*PhD Student, Faculty of Aerospace Engineering, G.Carpentieri@tudelft.nl, AIAA member.
†Full Professor, Faculty of Aerospace Engineering, M.J.L.vanTooren@tudelft.nl, MDO TC member.
‡Full Professor, Faculty of Aerospace Engineering, B.Koren@tudelft.nl, AIAA senior member.

case of large shape deformations. Shape optimization is performed by means of two algorithms. The first one is a Sequential Quadratic Programming algorithm from an external library, which is used for direct and inverse design cases. The second one is a Sequential Linear Programming algorithm implemented by means of an existing Linear Programming routine. The latter algorithm still needs to be refined but it appears to be already effective for constrained aerodynamic shape optimization. Preliminary results obtained with the algorithm will be presented.

The paper is structured as follows. In section II the Finite Volume solver implemented in this work is briefly presented. In section III the exact discrete adjoint is presented. Due to the lengthy derivation of the exact adjoint more details are presented in the appendix for the sake of completeness. In section IV the approach used to solve both the flow problem and the adjoint problem is presented. Section VI briefly addresses the shape parameterization whereas in section VII some applications in shape optimization are presented in order to demonstrate the optimization framework.

## II. Finite volume formulation

The Euler equations are a system of conservation laws which can be written in integral form as

$$\frac{d}{dt}\int_V \mathbf{u}\, d\Omega + \oint_{\partial V} \mathbf{F}\cdot\mathbf{n}\, d\Gamma = \mathbf{0}\,, \tag{1}$$

where $V$ is a volume contained in the domain $\Omega$. The vector $\mathbf{n}$ is the outward unit normal on the boundary $\partial V$ of $V$. The quantities $\mathbf{u}$ and $\mathbf{F}$ are the conservative variables vector and the flux vector respectively

$$\mathbf{u} = \begin{bmatrix} \rho \\ \rho\mathbf{w} \\ \rho e_t \end{bmatrix}, \qquad \mathbf{F}(\mathbf{u}) = \begin{bmatrix} \rho\mathbf{w}^T \\ \rho\mathbf{w}\mathbf{w}^T + p\mathbf{I} \\ \rho h_t \mathbf{w}^T \end{bmatrix}. \tag{2}$$

Both are defined as functions of primitive variables $\mathbf{v} = [\,\rho, \mathbf{w}, p\,]^T$ which are density, velocity ($\mathbf{w} = [w_x, w_y]^T$ in 2D) and pressure. Other quantities to be defined are the total specific energy $e_t = p/((\gamma-1)\rho) + \mathbf{w}\cdot\mathbf{w}/2$ and the total specific enthalpy $h_t = e_t + p/\rho$. The perfect gas equation $p = \rho RT$ is used to provide closure of the system.

Equation (1) is discretized in a Finite Volume framework.[11] For each internal control volume

$$V_i \frac{d\mathbf{u}_i}{dt} + \sum_{k=1,N_i} \mathbf{\Phi}(\hat{\mathbf{u}}_i, \hat{\mathbf{u}}_k, \mathbf{n}_{ik}) = \mathbf{0}\,, \tag{3}$$

where $\mathbf{\Phi}$ is the numerical flux evaluated at the interface $\partial V_{ik}$ between two control volumes $i$ and $k$. The numerical flux depends on the integrated normal $\mathbf{n}_{ik}$ ($\equiv \int_{\partial V_{ik}} \mathbf{n}\, d\Gamma$) and on the left and right state $\hat{\mathbf{u}}_i$ and $\hat{\mathbf{u}}_k$. Summation of the numerical flux across all the control volume interfaces $\partial V_{ik}$ gives the control volume residual $\mathbf{r}_i \equiv \sum_{k=1}^{N_i} \mathbf{\Phi}(\hat{\mathbf{u}}_i, \hat{\mathbf{u}}_k, \mathbf{n}_{ik})$. The hat on the states denotes extrapolation/reconstruction at the control volume interfaces in order to distinguish from the cell average values $\mathbf{u}_i$ and $\mathbf{u}_k$. The primitive variables are reconstructed according to a MUSCL–like extension:[12] for each variable $v_i$ a limited linear reconstruction is performed across each control volume interface $\partial V_{ik}$:

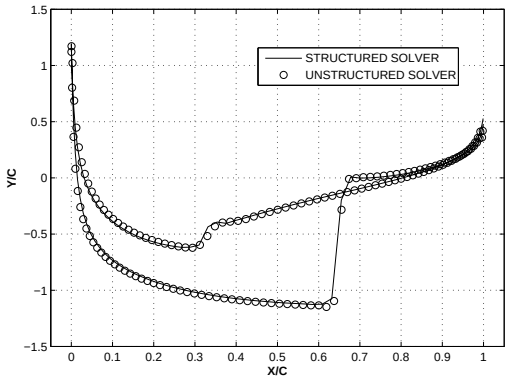



**Figure 1. Pressure distribution of the unstructured solver compared with that of a structured solver (NACA0012, $\alpha = 1.25$ deg and $\mathbf{M}_\infty = 0.8$).**

$$\hat{v}_i \;=\; v_i + \sigma_i \nabla v_i^T (\mathbf{x}_{ik} - \mathbf{x}_i)\,, \tag{4}$$

where $\sigma_i$ is a slope limiter, $\nabla v_i$ is the variable gradient and $\mathbf{x}_{ik}$ is the mid–point location of $\partial V_{ik}$ . The slope limiter is that of Venkatakrishnan[13] defined as:

$$\sigma_i = \min_{k=1,N_i} \left( \frac{\alpha_i^2 + 2\alpha_i \Delta v_{ik} + \varepsilon}{\alpha_i^2 + \alpha_i \Delta v_{ik} + 2\Delta v_{ik}^2 + \varepsilon} \right), \tag{5}$$

with $\Delta v_{ik} = \nabla v_i^T(\mathbf{x}_{ik} - \mathbf{x}_i)$ being the unlimited differential, $\varepsilon$ a threshold, $\alpha_i = v_{\max} - v_i$ for $\Delta v_{ik} \geq 0$ and $\alpha_i = v_{\min} - v_i$ for $\Delta v_{ik} < 0$. The values $v_{\max}$ and $v_{\min}$ are the extrema of $v_i$ on the stencil composed by $i$ and all its distance one neighbors ($k = 1, N_i$; see Fig. 13 in the Appendix). The gradient $\nabla v_i$ is computed using a linear Least–Squares technique[12] which reconstructs linear polynomials exactly. A Green-Gauss gradient is also available. The numerical flux is evaluated using Roe's approximate Riemann solver

$$\mathbf{\Phi}_{ij} = \mathbf{\Phi}(\mathbf{u}_i, \mathbf{u}_j, \mathbf{n}_{ij}) = \frac{1}{2}(\mathbf{F}(\mathbf{u}_i) + \mathbf{F}(\mathbf{u}_j)) \cdot \mathbf{n}_{ij} - \frac{1}{2}|\mathbf{A}(\tilde{\mathbf{u}}_{ij}, \mathbf{n}_{ij})|(\mathbf{u}_j - \mathbf{u}_i)\,, \tag{6}$$

where $|\mathbf{A}|$ is the absolute flux Jacobian evaluated with the Roe averages $\tilde{\mathbf{u}}_{ij}$. Equation (3) is completed by adding suitable terms when a control volume $i$ is lying on the domain boundary. More specifically, flux vector splitting is used for far–field boundaries and zero normal velocity is imposed on the wall flux.[14] Equation (3) can be rewritten in the semi–discrete form

$$\mathbf{D}\frac{d\mathbf{U}}{dt} + \mathbf{R} = \mathbf{0}\,, \tag{7}$$

where $\mathbf{D}$ is a diagonal matrix containing the control volumes, $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_N]^T$ and $\mathbf{R} = [\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_N]^T$ are the conservative variable and the residual vector respectively. The total number of control volumes is indicated with $N$. Since a median dual formulation is used, which stores the unknowns at the nodes of the mesh, the number of control volumes $N$ is the number of nodes in such mesh. An edge–based data structure is used in the solver.[12] Due to the flexibility of the formulation, hybrid meshes of triangular and quadrilateral elements can be easily processed.[11] Time marching of Eq. (7) is addressed in section IV.

Figure 1 shows the comparison of pressure distribution between the present solver and a structured flow solver[15] which uses variables extrapolation together with an Osher flux and Koren's limiter.[16] The mesh used for the computation was a $128 \times 80$ O–mesh. The present solver uses an unstructured triangular mesh of 8000 nodes with almost the same number of nodes on the wall.

## III. Adjoint formulation: exact discrete adjoint for the MUSCL scheme

First of all, the adjoint method is briefly outlined for the purpose of defining the problem and introducing the approach preferred for this work, which is the discrete approach. Consider a functional $J$ (e.g. Lift, Drag) for which the sensitivity with respect to a set of shape parameters must be computed. The functional is dependent on the flow as well as the shape of the domain. An efficient way to accomplish the computation is via an augmented functional $L$. Using a terminology similar to that used in control theory[1,17] the conservative variables $\mathbf{U}$ can be identified as state variables and the set of shape parameters as decision variables ($\alpha$ is one of those shape parameters). The state of the system is represented by the residual vector $\mathbf{R}$ which depends on both state and decision variables. The functional $L$ is obtained by augmenting $J$ with the state of the system. Introducing a vector of multipliers $\mathbf{\Lambda}$:

$$L(\mathbf{U}, \alpha, \mathbf{\Lambda}) = J(\mathbf{U}, \alpha) - \mathbf{\Lambda}^T\mathbf{R}(\mathbf{U}, \alpha)\,. \tag{8}$$

At the stationary point of the augmented functional $L$ its sensitivity coincides with that of the original functional $J$ ($dJ/d\alpha \equiv \partial L/\partial\alpha$). Imposing the stationary condition to the augmented functional ($[\partial L/\partial\mathbf{U}, \partial L/\partial\alpha, \partial L/\partial\mathbf{\Lambda}] = \mathbf{0}$) gives a set of three equations. The third equation is the state equation $\mathbf{R} = \mathbf{0}$ which is solved using the finite volume solver introduced before. The first and the second equations are the adjoint equation and the sensitivity of the functional $J$:

$$\frac{\partial\mathbf{R}}{\partial\mathbf{U}}^T\mathbf{\Lambda} = \frac{\partial J}{\partial\mathbf{U}}^T\,, \qquad \frac{dJ}{d\alpha} = \frac{\partial J}{\partial\alpha} - \mathbf{\Lambda}^T\frac{\partial\mathbf{R}}{\partial\alpha}\,. \tag{9}$$

The sensitivity is calculated using the multipliers or adjoint variables $\mathbf{\Lambda}$ obtained by solving the adjoint equation. Interesting feature of the method is that a single adjoint solution can be used to compute the sensitivity of one functional with respect to many shape parameters. The last two equations can be referred as the dual problem. The primal problem evaluates the functional sensitivity by directly using the flow sensitivity $d\mathbf{U}/d\alpha$ computed from the linearized flow equations:

$$\frac{\partial\mathbf{R}}{\partial\mathbf{U}}\frac{d\mathbf{U}}{d\alpha} = -\frac{\partial\mathbf{R}}{\partial\alpha}\,, \qquad \frac{dJ}{d\alpha} = \frac{\partial J}{\partial\alpha} + \frac{\partial J}{\partial\mathbf{U}}\frac{d\mathbf{U}}{d\alpha}\,. \tag{10}$$

A drawback of this approach is that the linearized flow equation (first in Eq. (10)) must be solved as many times as the number of shape parameters. By manipulation of Eq. (9) and Eq. (10) it can be shown that the sensitivities obtained by the two approaches are identical. The dual problem is convenient when the number of functionals to be solved is less than the number of shape parameters. If this is not true the primal problem becomes more efficient. Usually, in aerodynamic shape optimization there is a limited number of functionals and a large number of shape parameters so that the dual problem is advantageous.

In the present work a discrete adjoint is used which means that the dual problem of Eq. (9) is directly formulated from the discretized equations. The challenging part of this approach is the derivation of the transposed residual Jacobian $[\partial\mathbf{R}/\partial\mathbf{U}]^T$ which, as already said, involves differentiation of the original finite volume solver. Different levels of approximations may be employed to fulfill this task. However, these approximations may have a detrimental effect on the accuracy of the computed gradient.[2] For this reason, in the present work, an exact differentiation of the residual Jacobian has been performed.

To properly derive the transposed residual Jacobian $[\partial\mathbf{R}/\partial\mathbf{U}]^T$ it is best to start from the residual Jacobian $\partial\mathbf{R}/\partial\mathbf{U}$. Deriving first the residual Jacobian is essential also for the purpose of testing the accuracy of the adjoint. The residual Jacobian has been derived here following previous work.[7]

Three new vectors of length $E$ equal to the number of control volume interfaces have to be introduced. For a median–dual formulation, as the one used here, $E$ is equal to the number of edges in the mesh. The first vector $\mathbf{H} = [\hat{\mathbf{\Phi}}_1, \hat{\mathbf{\Phi}}_2, ..., \hat{\mathbf{\Phi}}_E]^T$ contains the second order fluxes for each edge. The second and third vectors, $\mathbf{U}_L = [\hat{\mathbf{u}}_{L1}, \hat{\mathbf{u}}_{L2}, ..., \hat{\mathbf{u}}_{LE}]^T$ and $\mathbf{U}_R = [\hat{\mathbf{u}}_{R1}, \hat{\mathbf{u}}_{R2}, ..., \hat{\mathbf{u}}_{RE}]^T$, contain the reconstructed left and right states for each edge. Using these vectors, by means of the chain rule, the residual Jacobian and the transposed residual Jacobian are obtained as:

$$\frac{\partial\mathbf{R}}{\partial\mathbf{U}} = \frac{\partial\mathbf{R}}{\partial\mathbf{H}}\left(\frac{\partial\mathbf{H}}{\partial\mathbf{U}_L}\frac{\partial\mathbf{U}_L}{\partial\mathbf{U}} + \frac{\partial\mathbf{H}}{\partial\mathbf{U}_R}\frac{\partial\mathbf{U}_R}{\partial\mathbf{U}}\right), \tag{11}$$

$$\frac{\partial\mathbf{R}}{\partial\mathbf{U}}^T = \left(\frac{\partial\mathbf{U}_L}{\partial\mathbf{U}}^T\frac{\partial\mathbf{H}}{\partial\mathbf{U}_L}^T + \frac{\partial\mathbf{U}_R}{\partial\mathbf{U}}^T\frac{\partial\mathbf{H}}{\partial\mathbf{U}_R}^T\right)\frac{\partial\mathbf{R}}{\partial\mathbf{H}}^T, \tag{12}$$

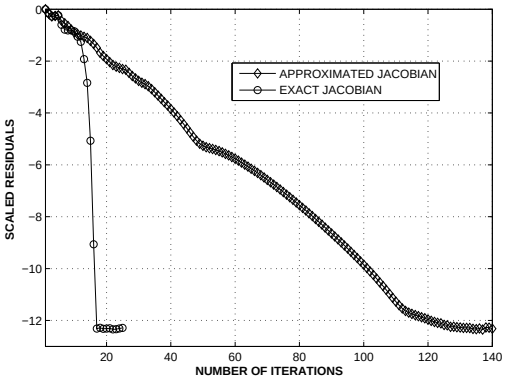



**Figure 2. Implicit pseudo time stepping for the flow equations. Approximate and exact Jacobian are employed. The exact Jacobian exhibits quadratic convergence.**

where $\partial\mathbf{R}/\partial\mathbf{H}$ is a rectangular dummy matrix of size $E \times N$. Each column numbered as an edge has only two non–zero entries on the rows corresponding to the left and the right nodes of the edge. Values for these two non–zero entries are $-1$ and $+1$ respectively (see Eq. (24)).

$\partial\mathbf{H}/\partial\mathbf{U}_L$ and $\partial\mathbf{H}/\partial\mathbf{U}_R$ are square diagonal matrix of size $E \times E$. They contain for each edge the numerical fluxes differentiated with respect to the left and right states respectively.

More complex is the matrix $\partial\mathbf{U}_L/\partial\mathbf{U}$, rectangular of dimensions $N \times E$, containing the differentiation of the reconstructed left states with respect to the cell averages in their stencil. Since the reconstruction is linear, on each row the non-zero entries will be positioned in the columns corresponding to the left state and its distance–one neighbors. The same is true for the matrix $\partial\mathbf{U}_R/\partial\mathbf{U}$ where in this case the right state must be considered. Each element of these matrices is a square matrix of size equal to the number of variables, for instance 4 for the 2D Euler equations.

In practice $[\partial\mathbf{R}/\partial\mathbf{U}]^T$ or $\partial\mathbf{R}/\partial\mathbf{U}$ are not formed at all. More likely their products with vectors are directly computed (matrix–free assembly). In the present work an edge–based assembly of such matrix–vector products has been derived and implemented. Using the edge–based assembly, both matrix–vector products can be performed in the same way as for the residual vector. In fact, $\mathbf{R}$ is assembled with a loop on the edges exploiting the conservation property. In the appendix the derivation of the edge-based assembly of Eq. (11) and Eq. (12) is given together with some details about the exact differentiation of the reconstruction operator as well as of the numerical flux.

The exactness of the transposed Jacobian $[\partial\mathbf{R}/\partial\mathbf{U}]^T$ has been tested indirectly employing the Jacobian $\partial\mathbf{R}/\partial\mathbf{U}$ in an implicit pseudo–time stepping procedure (see Eq. (13)). This iterative procedure becomes Newton's method for infinitely large time steps. This method is known to exhibit quadratic convergence only when the Jacobian employed is exact. Figure 2 shows the residual history for the time–stepping of

a transonic test case. The exact Jacobian is turned on after 10 iterations with the approximate Jacobian. As can be seen, the exact Jacobian attains a quadratic convergence and in only 5 iterations the residual is reduced by 10 orders of magnitude. Quadratic convergence is not obtained if the differentiation is not exact, for instance, if limiters are ignored (i.e. they are considered constants in the differentiation, see Eq. (33)) or if a programming error is present. Finally, the accuracy of the transposed Jacobian is checked by means of the matrix identity $\mathbf{v}^T[\partial\mathbf{R}/\partial\mathbf{U}]\mathbf{u} = \mathbf{u}^T[\partial\mathbf{R}/\partial\mathbf{U}]^T\mathbf{v}$ which in this case is satisfied to machine accuracy.

## IV. Time marching of flow and adjoint equations

Both the flow and the adjoint equations are advanced in time using implicit time stepping. The scheme is essentially the same for both solvers. At each time step a system of linear equations arises, which is solved iteratively to the required level of accuracy.

### A. Implicit pseudo-time stepping

The implicit pseudo–time stepping used to advance the semi–discrete system in Eq. (7) is a first order backward Euler scheme which can be written as

$$\left(\mathbf{D}_t + \widetilde{\frac{\partial\mathbf{R}}{\partial\mathbf{U}}}\right)^n (\mathbf{U}^{n+1} - \mathbf{U}^n) = -\mathbf{R}^n\,, \tag{13}$$

where the diagonal matrix $\mathbf{D}_t$ contains the control volumes divided by the local time step. An approximate Jacobian $\widetilde{\partial\mathbf{R}/\partial\mathbf{U}}$ is employed instead of the exact Jacobian of Eq. (11). Approximations are in terms of the reconstruction and the numerical flux. The reconstruction is ignored (i.e. first order accuracy) and the flux Jacobian $|\mathbf{A}(\tilde{\mathbf{u}}_{ij}, \mathbf{n}_{ij})|$ of Eq. (6) is frozen in the differentiation.[12] Advantages of making such approximations are the faster assembly and the increased diagonal dominance of this Jacobian. The latter results in a less problematic iterative solution of the linear system arising at each time step. However, due to the approximations the rate of convergence in Eq. (13) is linear instead of quadratic. A quadratic convergence rate can only be attained using an exact Jacobian (see Fig. 2). In order to speed up the convergence, at each iteration the time step is increased according to a CFL update of the type $\mathrm{CFL}^n = \beta\,\mathrm{CFL}^{n-1}L_2(\mathbf{R}^{n-2})/L_2(\mathbf{R}^{n-1})$ where $L_2(\mathbf{R})$ is a discrete norm of the residual vector and $\beta$ is a suitable parameter. Depending on the flow type the CFL is limited to a maximum value or it is left free to increase to infinity.

The adjoint equation appearing in the dual problem of Eq. (9) is a linear system for the adjoint variables $\mathbf{\Lambda}$. Due to the off–diagonal contribution arising from the reconstruction operator the system is poorly diagonally dominant and so it is stiff to solve. A time-like contribution can be added which results in a more robust solver.[8] In practice, the same time-stepping and settings used for the flow solver are also used in the adjoint solver:

$$\left(\mathbf{D}_t + \widetilde{\frac{\partial\mathbf{R}}{\partial\mathbf{U}}}^T\right)^n (\mathbf{\Lambda}^{n+1} - \mathbf{\Lambda}^n) = -\left(\frac{\partial\mathbf{R}}{\partial\mathbf{U}}^T\mathbf{\Lambda} - \frac{\partial J}{\partial\mathbf{U}}^T\right)^n . \tag{14}$$

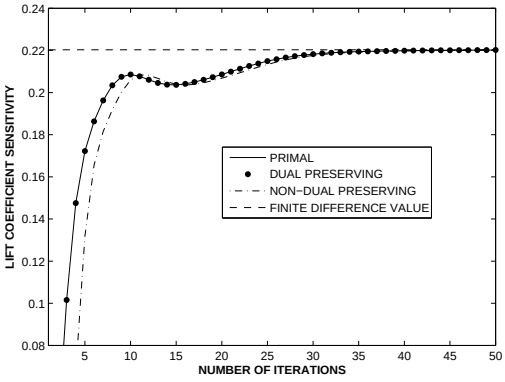



**Figure 3. Convergence history of the lift sensitivity with primal, dual and dual preserving iterations.**

A similar procedure can be devised for the primal problem of Eq. (10). As already said the sensitivity computed from the solution of the primal problem and the dual problem are identical. However, if an iterative procedure is involved, equality between the two values is only guaranteed at full convergence. During the iterations the two computed sensitivities can possibly have different values. In order to ensure exact equality of the two sensitivities also during the iterative procedure a duality preserving iterative scheme[4,8,9] can be used. Figure 3 shows that the primal sensitivity and the dual (non–duality preserving) sensitivity converge to the same value. Exact agreement is attained only at full convergence. As can be seen, if the iterations

are duality preserving, the dual sensitivity matches the primal sensitivity during the entire iteration process. The match is up to machine accuracy. In this work duality preserving iterations have only been used for the purpose of checking the consistency between the dual and the primal problem in the development of the code. In all applications presented in this work the adjoint sensitivity is usually converged to a level of accuracy for which consistency during the iterative procedure is not an issue.

Except in the case of inverse design where the optimization problem is unconstrained, shape optimization problems can have many constraints involving more than one functional. Simultaneous time stepping of more than one adjoint with Eq. (14) gives an appreciable time saving compared to sequential solutions. Multiple right–hand sides in Eq. (14) imply minor modifications. Only some more substantial coding is required to modify the linear system solver employed at each iteration. In this work up to 3 functionals (lift, drag and pitching moment coefficient) have been solved simultaneously and a time saving of 50% compared with a sequential solution is obtained. The time saving comes from the evaluation of the residual Jacobian terms. In a matrix–free procedure those terms are computed at each matrix–vector product. As a consequence it is convenient to perform more matrix–vector products at the same time for both the left and right–hand side of Eq. (14).

### B. Linear system of equations

The implicit time-stepping of the flow equations according to Eq. (13) implies the solution of a linear system at each time step. The same holds for the adjoint problem in Eq. (14).

Two options are followed in this work to solve the linear system. The first one is a simple iterative procedure which, given the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$, computes corrections of the type:

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \Delta\mathbf{x}\,, \qquad \Delta\mathbf{x} = \mathbf{P}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{x}^k)\,, \qquad (15)$$

where $\mathbf{P}$ is a preconditioner computed from $\mathbf{A}$. The second one is the more sophisticated GMRES[18] algorithm which also uses preconditioning. Libraries[19] of a re–started versions of this algorithm are linked to the present implementation.

The efficiency of both iterative procedures heavily relies on the effectiveness of the preconditioner. If the preconditioner is not effective the convergence can be poor for both procedures or in some cases it can stall. The preconditioner should be a good approximation of the original matrix ($\mathbf{P} \approx \mathbf{A}$) and moreover it should be relatively simple to invert. Two types of preconditioners are used in this work. The first one is the Incomplete Lower Upper factorization with zero fill–in ILU(0). It is a lower–upper factorization for which only elements positioned in the sparsity pattern of the original matrix $\mathbf{A}$ are kept. The second one is the Symmetric Successive Overrelaxation, SSOR, which can be expressed as $\mathbf{P} = (\mathbf{D} + \mathbf{L})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U})$. The matrices $\mathbf{D}$, $\mathbf{L}$ and $\mathbf{U}$ are the diagonal, strictly lower and strictly upper part of the matrix $\mathbf{A}$ (again, each matrix element is a matrix of size $4 \times 4$).

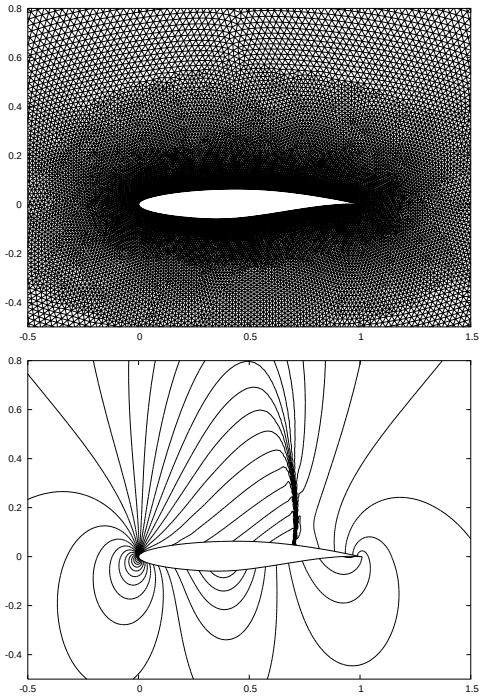
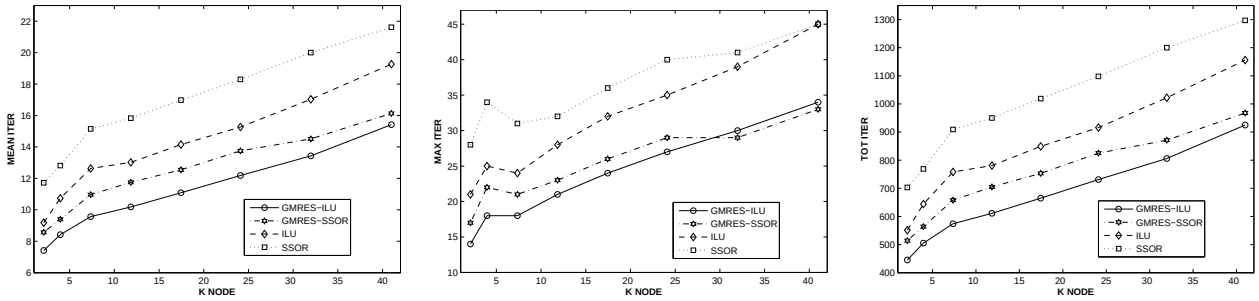



**Figure 4. Triangular mesh of 30 k nodes (top) and pressure contours (bottom) for the RAE2822 airfoil at $\alpha = 2$ deg and $M_\infty = 0.75$.**

In the following SSOR indicates the simple iterative procedure using the SSOR preconditioner whereas GMRES-SSOR indicates GMRES using SSOR as preconditioner. The same is for ILU and GMRES-ILU. The ILU(0) preconditioner needs to be computed and stored before being used. When the ILU(0) preconditioner is computed an additional data structure must be introduced according to the type of libraries employed (in this work the SLATEC[20] libraries are used). Usually the data structure of the libraries can be very different from that used in the solver implying conversion and rearrangement of the data.

As opposed to that, the SSOR preconditioner does not require any preparatory work. This preconditioner has been implemented here directly using the flow solver data structure. Only an additional pointer is created which links each node to its neighbors. Optionally, at least for the Euler equations, it is possible to apply the




(a) Average number of linear iterations.

(b) Maximum number of linear iterations.

(c) Total number of linear iterations.

**Figure 5. Comparison of the four sparse linear solvers for increasing mesh density.**

preconditioner in a matrix–free fashion without storing the elements. This preconditioner is applied by means of a forward solve $(\mathbf{D}+\mathbf{L})\,\mathbf{z}=\mathbf{b}$ followed by a backward solve $(\mathbf{I}+\mathbf{D}^{-1}\mathbf{U})\,\mathbf{x}=\mathbf{z}$. The non-zero entries for each node (a Jacobian row) are only the nodes in the stencil. In fact, neglecting the reconstruction contribution, each edge produces four entries: two on the diagonal and two off–diagonal.[21] With proper ordering of the nodes in the pointer it is possible to perform the backward and forward solve without storing $\mathbf{L}$ and $\mathbf{U}$ but by recomputing them at each step. In this case the storage requirements decrease considerably at the price of an increase in the execution time.

A comparison of the four linear solvers is made for a transonic test case with increasing mesh density. The test case is that of Fig. 4. The mesh density is increased only close to the wall and not uniformly in the domain. The initial mesh size has around 2000 nodes and the final mesh has 40000 nodes. For all the computations 10 directions are used for GMRES after which it is restarted. The linear system is solved at each iteration to an accuracy which is 2 orders of magnitude smaller than the discrete norm of the flow residual $L_2(\mathbf{R})$. Such a level of accuracy is found empirically. In general, for cases with less clustered nodes around the boundary than the one in Fig. 4, 1 order of magnitude suffices. Performances in terms of average, maximum and total number of iterations for 60 time steps are shown in Fig. 5. As already expected[22] the best performance in terms of required iterations is that of GMRES-ILU followed by GMRES-SSOR, ILU and SSOR respectively. At least for the Euler equations, the SSOR preconditioner seems to be effective for increasing mesh density as much as ILU. Moreover, in spite of the larger number of iterations to be performed at each time step, the SSOR preconditioner (storage allowed) is found to be faster to apply than the ILU preconditioner. For this reason in this work the SSOR preconditioner is used for all computations. When the linear system is solved to a lower level of accuracy, for instance 1 order of magnitude smaller than $L_2(\mathbf{R})$, the difference in performances between GMRES and the simple iterative procedure reduce considerably and no particular benefit is noticed from the application of GMRES.

In the case of the adjoint equation, for more than one functional, multiple right–hand sides must be taken into account when solving the linear system. In this work the iterative procedures that use SSOR and ILU have been implemented to efficiently solve multiple right–hand sides whereas in the case of GMRES this option is not considered due to the complexity of the implementation.

## V. Overall accuracy of the discrete adjoint

The accuracy of the transposed Jacobian has been discussed in Section III and it was proven to be exact. This guarantees that one has accurate adjoint variables for the evaluation of the sensitivity once the adjoint equations are solved. The terms $\partial\mathbf{R}/\partial\alpha$ and $\partial J/\partial\alpha$, which are also required to evaluate the sensitivity, are computed here by means of a finite difference step. With this method a truncation error must be accounted for, contrary to the complex variable method which can give the exact sensitivity.[2,4] Even though the finite difference step is not exact, it is able $(i)$ to show whether the sensitivity of the adjoint is consistent and $(ii)$ to quantify the error given by approximations in the adjoint.

In Fig. 7 results for the NACA0012 airfoil at $M_\infty = 0.85$ and $\alpha = 1\,\mathrm{deg}$ are shown. The flow, the adjoint and the linearized (primal) solver have been converged to machine accuracy (convergence is only shown for the first 200 iterations). Values of $c_l = 0.3533$ and $c_d = 0.0581$ are found which are in agreement with

$c_l = 0.3565$ and $c_d = 0.0582$ found in the references.[15] The adjoint variables are solved simultaneously for the lift, drag and pitching moment. In order to avoid the computation to stall, it was found necessary to limit the CFL number to 100. The Mach number contours reveal shocks on both the upper and the lower surface of the airfoil. Probably due to these conditions the convergence rate of the solver is poor, for instance compared to the rate of convergence obtained with the approximated Jacobian, as given in Fig. 2. The adjoint and the linear solver appear to have the same rate of convergence as the flow solver. The contours of the third adjoint variable for the lift coefficient are also shown. This variable represents the sensitivity of the lift coefficient to changes in vertical momentum.

In Fig. 8 the same mesh is used to compute the flow at supersonic flow conditions $M_\infty = 1.2$ and $\alpha = 7\,\text{deg}$. The convergence rate is three time faster than for the previous case. Also for this case, values of $c_l = 0.5202$ and $c_d = 0.1553$ are found which are in agreement with $c_l = 0.5237$ and $c_d = 0.1551$ found in the references.[15] Contours of the third adjoint variable for the lift and the second adjoint variable for the drag are shown. They have a reverse trend as compared to the Mach contours. This is due to the fact that the functional on the airfoil cannot be influenced by the downstream region due to the supersonic nature of the flow in which disturbances do not propagate upstream.

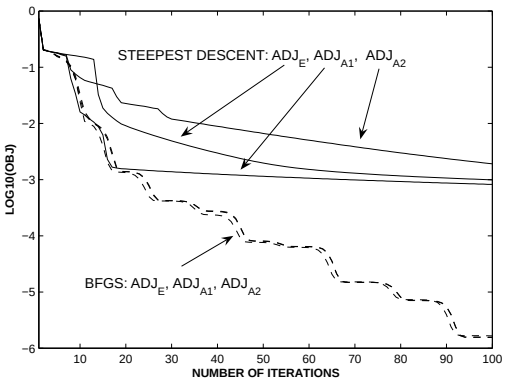



**Figure 6. Inverse design with a steepest descent and a BFGS algorithm for different adjoint approximations. The BFGS algorithm seems to be insensitive to the approximations in the adjoint.**

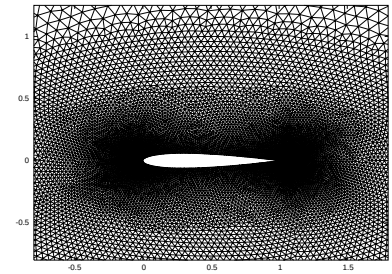
In table 1 sensitivities with respect to the angle of attack are shown for the two cases. The table includes the exact adjoint $ADJ_E$, two approximated adjoints and the finite difference values $FD$ which are calculated with a suitable step. In the approximated adjoint $ADJ_{A1}$ the limiter contribution is neglected whereas in $ADJ_{A2}$ also the numerical flux Jacobian is approximated as the one employed in the time stepping. For comparison purposes, $\Delta(\%)$ is used to indicate the average value of the percentage difference between an adjoint sensitivity and the finite difference sensitivity for the three coefficients.

The exact adjoint sensitivity in the supersonic case seems to more closely agree with the finite difference value than the transonic case does. This is probably due to the more critical flow conditions of the transonic case. The approximated adjoint $ADJ_{A1}$ compared to $ADJ_E$ gives a loss of accuracy of about two orders of magnitude for both cases. Finally, the approximated adjoint $ADJ_{A2}$ shows a further reduction of accuracy which is more evident in the transonic case. As can be seen, the accuracy loss caused by the use of approximated adjoints is evident in terms of gradient results. Nevertheless, this should not lead to the conclusion that approximated adjoints have bad performances in shape optimization. For instance, in Fig. 6 an objective function for an inverse design of a transonic airfoil (see also Fig. 11) is shown. The three adjoint approximations have been used with two different optimization algorithm.

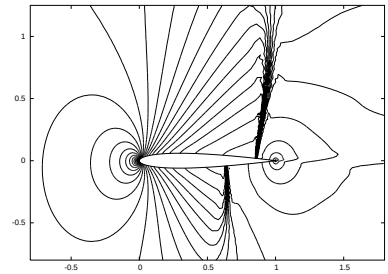
**Table 1. Comparison of exact and approximate discrete adjoint with finite differences for transonic and supersonic NACA0012 cases.**

| | | $FD$ | $ADJ_E$ | $ADJ_{A1}$ | $ADJ_{A2}$ |
|---|---|---|---|---|---|
| | $\partial c_l/\partial\alpha$ | 0.272780 | 0.272864 | 0.263342 | 0.253712 |
| $M_\infty = 0.85$, | $\partial c_d/\partial\alpha$ | 0.017794 | 0.017790 | 0.017263 | 0.016785 |
| $\alpha = 1\,\text{deg}$ | $\partial c_m/\partial\alpha$ | -0.086478 | -0.086517 | -0.082095 | -0.07757 |
| | $\Delta(\%)$ | // | 0.03 | 3.5 | 7.5 |
| | $\partial c_l/\partial\alpha$ | 0.073865 | 0.073866 | 0.073712 | 0.0740152 |
| $M_\infty = 1.2$, | $\partial c_d/\partial\alpha$ | 0.016737 | 0.016737 | 0.016685 | 0.016679 |
| $\alpha = 7\,\text{deg}$ | $\partial c_m/\partial\alpha$ | -0.016285 | -0.016285 | -0.016157 | -0.016152 |
| | $\Delta(\%)$ | // | 0.002 | 0.4 | 0.45 |

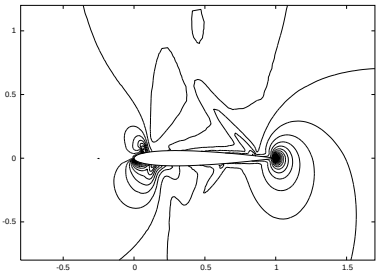
The first one is the steepest descent. With this algorithm a difference between the three adjoint approximations is noticed only in the initial phase. In fact, asymptotically they seem to reach the same poor



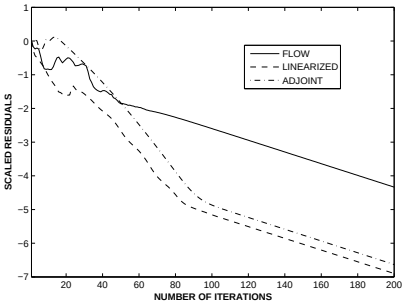
(a) 14 K nodes unstructured mesh.



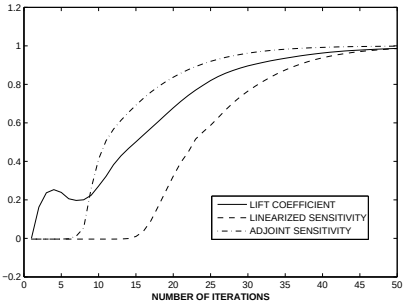
(b) Mach number contours. Shocks are visible on both sides of the airfoil.



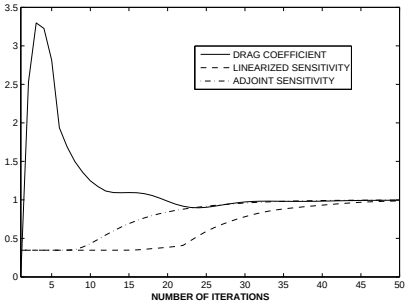
(c) Third adjoint variable for the lift coefficient.




(d) Residuals of flow, primal and dual solver.

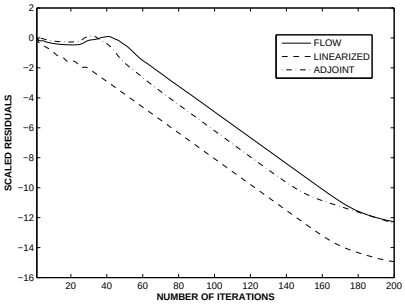



(e) Convergence history of the Lift coefficient and its sensitivity.

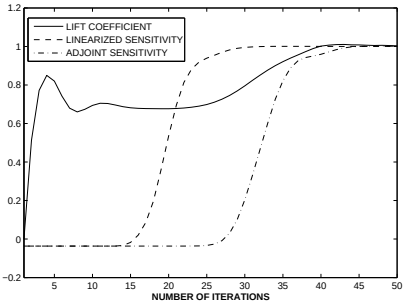



(f) Convergence history of the Drag coefficient and its sensitivity.

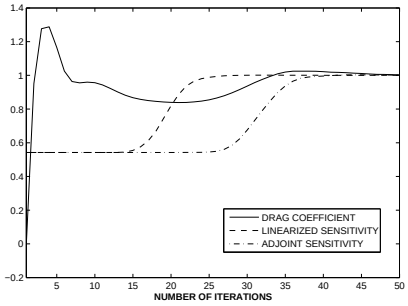
**Figure 7. NACA0012 airfoil at $M_\infty = 0.85$ and $\alpha = 1$ deg. This case exhibits a poor convergence rate of the residuals which drop only 4 orders of magnitude in the first 200 iterations. 600 iterations (not shown here) are required to drop the residuals of 12 orders of magnitude. Forces and sensitivities are scaled by their converged values.**



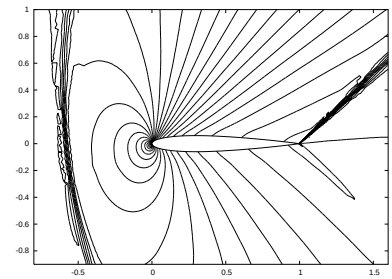

(a) Residuals of flow, primal and dual solver.



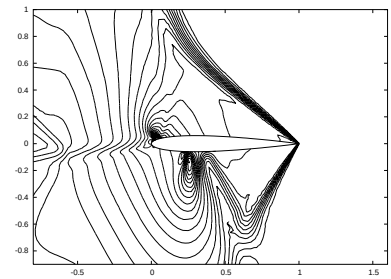

(b) Convergence history of the lift coefficient and its sensitivity.



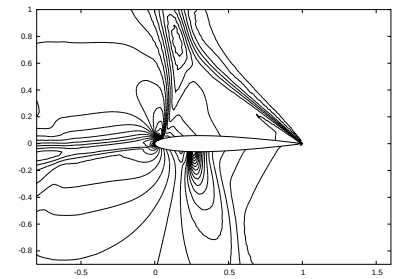

(c) Convergence history of the drag coefficient and its sensitivity.



(d) Mach number contours. Bow shock, tail shock and slip line are visible.



(e) Third adjoint variable for the Lift coefficient.



(f) Second adjoint variable for Drag coefficient.

**Figure 8. NACA0012 airfoil at $M_\infty = 1.2$ and $\alpha = 7$ deg. The mesh used is the 14 K nodes unstructured mesh used in the previous case, Fig. 7. Compared to that case the convergence rate is 3 time faster. For the residuals, 12 orders of magnitude reduction have been achieved in 200 iterations.**

convergence rate. The second one is the BFGS algorithm[23] which converges the objective much faster than the steepest descent. With this algorithm there is virtually no difference between the three approximations.

Such a forgiveness in terms of gradient accuracy is probably due to the fact that the search direction, contrary to the steepest descent, is not only determined as a function of the gradient but also using an approximated Hessian which is updated at each iteration.

The steepest descent method does not seem to find practical application because of its poor performances. On the contrary, BFGS or more in general variable metrics methods are widely used. Therefore, the fact that those methods are relatively insensitive to the gradient accuracy can have implications on the possibilities to use the approximate instead of the exact adjoint. Approximate adjoint requires much less efforts in the implementation than exact adjoint. As can be seen in the Appendix, neglecting different contributions in the adjoint as in $ADJ_{A1}$ and $ADJ_{A2}$ reduces the complexity of the implementation considerably.

## VI. Shape parameterization

The parameterization of the geometry plays a crucial role in shape optimization. The way the shape is parametrized is the way the design space is represented. A limited representation of such a space would limit the chance to approach an optimum design. In this work an orthogonal Chebyshev polynomial representation is used first to approximate the airfoil and then to deform its shape during the optimization process. The orthogonality property is important since it gives completeness of the design space. This approximation can be defined as a series of orthogonal basis functions[10,24]

$$f_t(x) = \sum_{k=0}^{N_T} c_k T_k(z(x)) \,, \tag{16}$$

where $N_T$ is the number of such functions, each of them defined as

$$T_k(x) = \cos(k\theta(x)) \,, \qquad \theta(x) = \cos^{-1}(2\sqrt{x} - 1) \,, \qquad (k \geq 0 \,, \quad 0 \leq x \leq 1) \,. \tag{17}$$

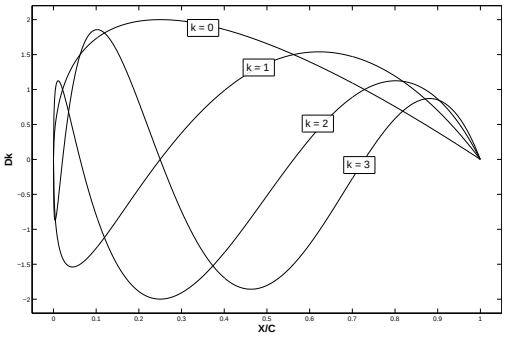



**Figure 9. Basis functions for the Design polynomial.**

It is assumed that the airfoil is of unit chord. A curve approximated by Eq. (16) is not closed at the leading and the trailing edge. In order to have closure at these points the "Design" polynomial must be defined :

$$f_d(x) = \sum_{k=0}^{N_T-2} d_k D_k(z(x)) \,, \qquad D_k = T_k - T_{k+2} \,. \tag{18}$$

Figure 9 shows the basis functions $D_k$ for the Design polynomial ($d_k$ coefficients have been set equal to one). It is interesting to see how the first Design basis ($k = 0$) resembles an airfoil shape.

In practice Eq. (18) can be used to approximate the airfoil following two approaches. One approach is to use the camber line method in which the camber and the thickness distribution are independently approximated. Another option, which is the one preferred in this work, is to directly approximate both the upper and the lower curves.

Given a set of $N$ airfoil coordinates ($[x_i, y_i]$; $i = 1, N$) a local error ($\epsilon_i = |f_d(x_i) - y_i|$), an average error ($\epsilon_m = 1/N \sum_{i=1}^{N} \epsilon_i$) and a maximum error ($\epsilon_M = \max \epsilon_i$) can be defined. In order to approximate the airfoil an unconstrained minimization algorithm can be used to reduce a weighted linear combination of maximum and average error. A certain maximum error in the representation must be achieved in order not to have discrepancies between functional computed on the original airfoil and on the approximated airfoil. An investigation[25] shows that for most CFD solvers in use a maximum error of the order of $\epsilon_M = 8 \times 10^{-5}$ suffices. Using the Chebyshev polynomials different airfoils have been successfully approximated to the required maximum error.[26]

# VII. Shape optimization

In shape optimization an objective function, a functional usually defined on the wall, must be minimized in order to improve performances or to achieve a desired design. Depending on the case, constraints have to be taken into account. Constraints can be of a geometric nature or they can be imposed on some other functional. In this work drag minimization problems have been addressed with equality constraint on the lift coefficient and inequality constraint on the maximum airfoil thickness. Such a problem can be stated as:

$$\begin{aligned} \min\ f(\mathbf{d}) &= \frac{c_d(\mathbf{d})}{c_{d\,R}} \\ h(\mathbf{d}) = \frac{c_l(\mathbf{d})}{c_{l\,R}} - 1 = 0\,, &\qquad g(\mathbf{d}) = 1 - \frac{t/c_{\max}(\mathbf{d})}{t/c_{\max R}} \leq 0\,, \end{aligned} \tag{19}$$

where $\mathbf{d}$ is a vector containing the design variables: shape parameters and angle of attack. For this vector upper and lower bounds ($\mathbf{d}_L \leq \mathbf{d} \leq \mathbf{d}_U$) can be prescribed. The symbol $t/c_{\max}$ indicates the maximum relative thickness of the airfoil and the subscript $R$ refer to reference (initial) values. The two constraints are necessary otherwise the drag will be reduced at the expense of the lift and the thickness. Usually the lift coefficient is given and the thickness cannot decrease under a certain value for structural and other design considerations. Usually the shape parameter vector $\mathbf{d}$ is also scaled in a way that at the beginning of the optimization each component of the vector is equal to 1. Additional constraints on the geometry as well as on the functional can be included. For instance a pitching moment constraint, nose radius constraint or a trailing edge angle constraint and so on.

Another kind of problem, referred to as inverse design is also addressed here. In such a problem the pressure distribution on the airfoil is assigned and the optimization algorithm is used to find the proper set of shape parameters which give that pressure. An inverse design problem is an unconstrained problem defined as

$$\min f(\mathbf{d}) = \int_{\text{wall}} (p\,(\mathbf{d}) - p_{\text{ref}})^2 dS(\mathbf{d})\,, \tag{20}$$

where $p_{\text{ref}}$ is the assigned pressure distribution on the wall.

## A. Using non–linear optimization algorithm

The flow and the adjoint solver can be coupled together with the shape parameterization and linked to an optimizer. In this work a routine from the optimization toolbox of Matlab[27] has been used. This algorithm follows a Sequential Quadratic Programming (SQP) approach with a BFGS Hessian update. This methodology is suitable for non–linear problems with many constraints. Tolerances for constraints are strictly satisfied using this kind of algorithm. The application only requires an interface to the flow solver and the adjoint solver plus all the required settings. Two airfoils have been optimized: the RAE2822 and the NACA64A410. For both airfoils a total of 22 design basis functions (11 on the upper and 11 on the lower side) are used. This is the number required to obtain the predefined maximum error in the geometric representation of the initial airfoil. The angle of attack is also included in the optimization as design variable.

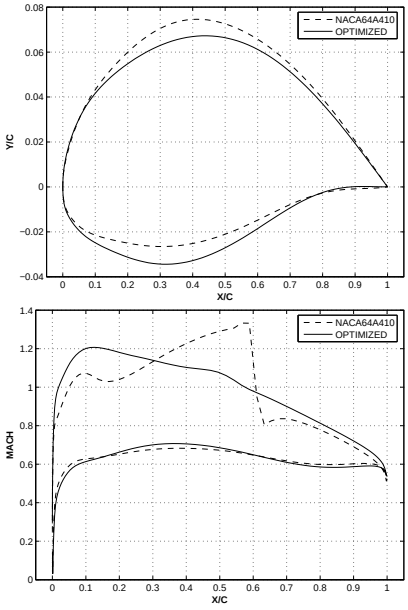



**Figure 10. Optimization of the NACA64A410 airfoil at $\mathbf{M_\infty = 0.7}$ and $\alpha = 2$ deg. Comparison of initial and optimized airfoil (top) and Mach number (bottom).**

Figure 10 shows the optimization performed on the NACA64A410 airfoil at transonic flow conditions. The optimization statement is that of Eq. (19). It means the lift coefficient cannot change and the maximum thickness of the airfoil cannot decrease. Upper and lower bounds on the variables have been chosen ($\mathbf{d}_L = 0.1\,\mathbf{d}_0$ and $\mathbf{d}_U = 2.5\,\mathbf{d}_0$) empirically. After 11 gradient and 26 function evaluations the drag coefficient is reduced of 52% and the maximum constraint value is equal to $10^{-5}$. The inequality thickness constraint is critical. The resulting airfoil is shock free and the shape has changed smoothly. The nose radius is not sharpened and the trailing edge angle is only slightly reduced. This is positive from the point of view of off–design performances.

Figure 11 shows the RAE2822 airfoil case, also optimized at transonic flow conditions. For this case the upper and lower bounds are reduced ($\mathbf{d}_L = 0.8\,\mathbf{d}_0$ and $\mathbf{d}_U = 1.2\,\mathbf{d}_0$). In fact with the same bounds used for the previous case the resulting optimized airfoil shows a sharp nose and a very thin profile close to the trailing edge. This kind of problems arise because of the lack in geometric constraints. The resulting airfoil using the reduced domain exhibits a smooth pressure distribution and a smooth airfoil shape. 8 gradient and 17 function evaluations are required. The maximum constraint value is $3 \times 10^{-6}$. Again, the thickness constraint is critical. For this airfoil also an inverse design is performed. Starting from the optimized airfoil the purpose is to step back to the original airfoil, the RAE2822. The inverse design problem is defined in Eq. (20). Contrary to the direct design cases, a tight tolerance on the objective function is specified. The algorithm finished after 189 iterations when the objective function showed a reduction of 10 orders of magnitude. A low maximum error of the order of $6.5 \times 10^{-7}$ is obtained for the airfoil representation which is also smaller than the accuracy to which the original RAE2822 has been approximated. Figure 11 shows the resulting Mach number distribution perfectly lying on the top of the one of the target RAE2822 airfoil.

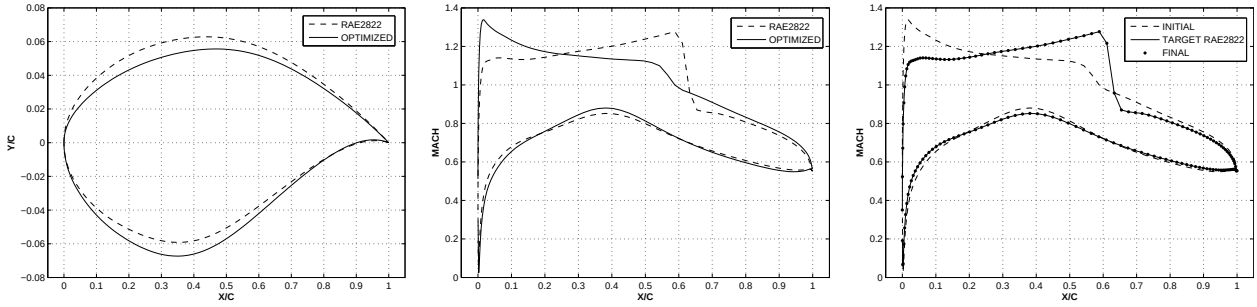



**Figure 11. RAE2822 airfoil direct and inverse design at $\mathbf{M_\infty = 0.73}$ and $\alpha = 2\,$deg. Initial and optimized airfoil (left) and Mach number (middle). At the end of the optimization the RAE2822 airfoil is re–obtained starting from the optimized airfoil by means of inverse design (right). After the inverse design, the maximum error on the airfoil coordinates is $\mathbf{6.5 \times 10^{-7}}$ and on the pressure distribution it is $\mathbf{5.7 \times 10^{-5}}$.**

For both cases the upper and lower bound on the design variables have been defined empirically in order to stay away from unfeasible design (e.g. very sharp nose or extremely thin trailing edge). Clearly additional geometric constraints (at the nose and trailing edge) should be imposed to remove this need for setting those bounds empirically. In such a way the design will not be driven in the unfeasible region.

## B. Sequential Linear Programming algorithm: preliminary result

Non–linear optimization algorithms are designed to face highly non–linear problems with many equality and inequality constraints. Usually the user makes no assumptions on the nature of the function or the constraints under consideration. In the test cases presented in the previous section the optimization algorithm is employed as a black–box, meaning that given the required settings the optimizer drives the flow and the adjoint solver without the user being involved in the internal computational process. In this sense the user plays a passive role not being able to bring his judgment into the design process, at least until the optimizer has finished. It is probably worthwhile to investigate simpler algorithms with the capability of making the design process more interactive and transparent to the user. Nevertheless, those algorithms have to be effective in improving the design and taking into account equality and inequality constraints in a satisfactory way.

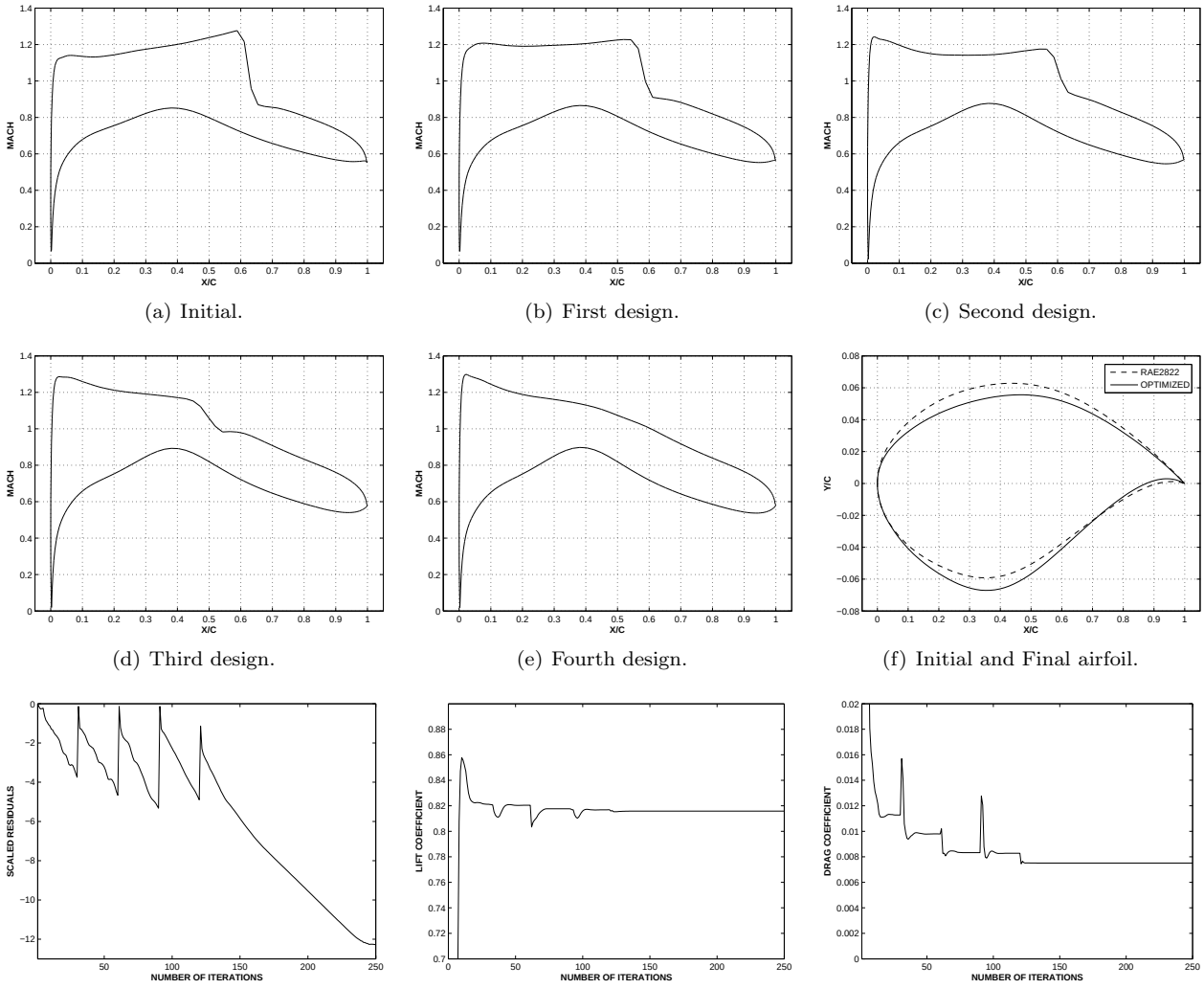
In this work a Sequential Linear Programming algorithm has been implemented and a preliminary result has been obtained which seems to be promising. The algorithm linearizes the objective and the constraints around a design point.[23] In the case of the optimization statement given in Eq. (19), linearization gives:

$$\begin{aligned} \min\ (f(\mathbf{d}_0) \quad &+ \quad \nabla {f_0}^T \Delta\mathbf{d}) \\ h(\mathbf{d}_0) + \nabla {h_0}^T \Delta\mathbf{d} = 0\,, \qquad & g(\mathbf{d}_0) + \nabla {g_0}^T \Delta\mathbf{d} \leq 0\,. \end{aligned} \tag{21}$$

with the upper and the lower bounds becoming $\mathbf{d}_L - \mathbf{d}_0 \leq \Delta\mathbf{d} \leq \mathbf{d}_U - \mathbf{d}_0$. This problem can be solved by means of a Linear Programming algorithm such as the Simplex method.[23] Implementations of those algorithms are available in many libraries.[20] After the solution of the linear problem is found, a new design point is available. The non–linear problem is linearized again around this new point and the Linear

Programming solution is repeated. This process can be iterated until a local minimum is found within a specified tolerance.

A drawback of this algorithm is that a reduced domain $\Delta\mathbf{d}^* = b\Delta\mathbf{d}$ to which the search is limited must be usually defined. In practice the domain resulting from the linearization cannot always be used in its entirety for the Linear Programming solution. This is because if the problem is underconstrained, the solution tends to be unbounded. Therefore it will lie on the bounds if those are available. As a consequence limited bounds are necessary, which should be reduced after each iteration, in order to reach the true non–linear minimum. The definition of the reduced domain is not easy since it should be a compromise between two concurrent necessities. As a matter of fact, one should avoid large steps that would bring the design outside of the region where the linearity assumption is valid. On the contrary small steps would make convergence to an optimum design very slow making the process inefficient. At the moment an empiric criteria for a compromise between the two necessities is used.




(a) Initial. (b) First design. (c) Second design.

(d) Third design. (e) Fourth design. (f) Initial and Final airfoil.

(g) Residual history. (h) Lift coefficient history. (i) Drag coefficient history.

**Figure 12. Preliminary result obtained from the Sequential Linear Programming algorithm. The algorithm is terminated after four design when a smooth pressure distribution is found. Lift and maximum thickness constraints are satisfied.**

An optimization test using the Sequential Linear Programming algorithm is performed on the RAE2822 airfoil at the same conditions as the previous section except for the bounds which in this case are larger ($\mathbf{d}_L = 0.6\,\mathbf{d}_0$ and $\mathbf{d}_U = 1.4\,\mathbf{d}_0$). By means of a numerical test, a reducing factor $b = 0.2$ is found imposing the error in Lift not to be greater than 5%. Four linearized problems have been solved for which partially converged flow and adjoint solutions have been used. As can be seen from Fig. 12 at the end of the fourth

step the Mach number distribution obtained is very smooth so that the design is terminated and the flow is converged to machine accuracy. A drag reduction of around 30% is noticed. The lift coefficient seems to satisfy the equality constraint: at the end the loss in lift is less than 1%. The same is true for the thickness constraint. Clearly the tolerance is not that strict as with the non–linear algorithm. This is because for a convex design space this method approaches the minimum from the unfeasible region. To overcome this problem another Linear Programming methodology known as the method of center can be used, which guarantees to approach the minimum from the feasible region.[23]

As can be seen, the Mach number distribution obtained shows to have a slightly less pronounced spike on the nose compared to the results of the non–linear algorithm. Also, the final airfoil shows a thinner trailing edge. Those differences are probably due to the size of the domain which in this case was double the size of the one used in the non–linear case.

It is interesting to see that in spite of the simplicity of this method a design has been performed with very few resources. The cost is that of a single solution plus four partially converged solutions. Probably, the problems related to the definition of the reduced bound are the limiting factors in achieving accurate convergence to the minimum. Also, because of the lack in second order information, the algorithm is not expected to perform as the non–linear algorithm. However, the method seems to be very effective in finding a near optimal solution which usually is what the user is looking for in many design applications. Increasing the complexity of the optimization problem, for instance involving more constraints, should not have a negative effect in terms of performances of this method. In fact, it is expected to work better since the resulting linearized problem should have a reduced tendency to be unbounded.

## VIII. Conclusions and future work

The exact discrete adjoint of a finite volume formulation for the Euler equations in two dimensions has been implemented and tested. Comparisons have been made with approximated adjoints obtained by neglecting some contributions in the differentiation. In terms of gradient results an appreciable loss of accuracy is caused by those approximations. Nevertheless, no appreciable differences were noticed between the various adjoints in the convergence rate of an inverse design test case driven by a widely used optimization algorithm. Such a result can motivate the development of approximated adjoint codes which require much less effort than the exact adjoint. This aspect requires further investigations.

The implicit time stepping originally employed in the flow solver has been adapted for the adjoint solver. In order to address constrained shape optimization the solver has been modified to efficiently take into account multiple functionals. Simultaneous rather than sequential adjoint solutions were found to give considerable time saving. For both the flow and the adjoint solver the ILU and SSOR preconditioners have been employed by themselves or in combination with GMRES for the purpose of solving the linear system arising at each time step. The SSOR preconditioner, optionally matrix–free, has been implemented using the same data structure of the flow solver. When storage is allowed, SSOR seems to be faster than ILU in spite of the greater number of iterations required for a certain level of convergence.

An optimization framework coupling the flow and the adjoint solver to the shape parameterization and to an external optimizer has been implemented. Such a framework has been shown to be effective in addressing direct and inverse design test cases in transonic flow conditions. In order to simplify the optimization process and make it more accessible for the user, a simple Sequential Linear Programming algorithm has also been employed. Preliminary results show that the algorithm can deal properly with constraints and give appreciable improvements in the design. Further investigations are required to establish criteria to reduce the search domain for situations in which the design has the tendency to be unbounded. An extension of the complete optimization framework presented here is planned in order to address turbulent flows and three dimensional cases.

# Appendix

This appendix describes more in detail the edge–based assembly of the two matrix–vector products

$$\mathbf{Z} = \frac{\partial \mathbf{R}}{\partial \mathbf{U}} \mathbf{P}\,, \qquad \mathbf{Z} = \frac{\partial \mathbf{R}}{\partial \mathbf{U}}^{\mathbf{T}} \mathbf{P}\,. \tag{22}$$

$N$ is the number of nodes and $E$ the number of edges. The residual at node $i$ is defined as the sum of the numerical fluxes across the control volume interfaces $\partial V_{ik}$

$$\mathbf{r}_i = \sum_{k=1}^{N_i} \mathbf{\Phi}(\hat{\mathbf{u}}_i, \hat{\mathbf{u}}_k, \mathbf{n}_{ik}) = \sum_{k=1}^{N_i} \hat{\mathbf{\Phi}}_{ik}\,. \tag{23}$$

Those interfaces are lying at the mid–point of the edges $ik$ surrounding the node i. The hat on the conservative variables means that those variables have been extrapolated at the edge mid–point. The hat is also used on the numerical flux to indicate that it is evaluated using extrapolated variables.

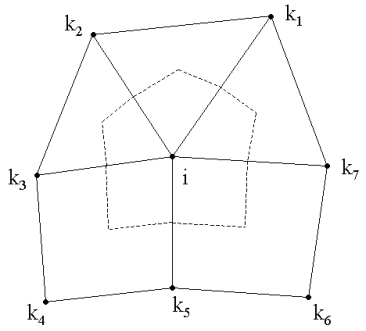



**Figure 13. Median dual around node $i$. The number of distance on neighbors is $N_i = 5$ and the stencil is $\mathcal{N}_i = [i, k_1, k_2, k_3, k_5, k_7]$.**

For a linear reconstruction a distance–one stencil is used. As can be seen from Fig. 13 the stencil $\mathcal{N}_i$ of a node $i$ include the distance–one neighbors of such node as well as the node itself. Therefore, the reconstructed variable at node $i$ has a dependence $\hat{\mathbf{u}}_i = \hat{\mathbf{u}}_i(\mathbf{u}_k; k \in \mathcal{N}_i)$ on the cell averages in such stencil.

The residual vector is collected on each edge exploiting the conservation property. A pointer is associated with each edge, it points to the left and the right node sharing the edge (edge–based data structure). Using this pointer, a loop on the edges is performed and the computed flux is accumulated on the left $i$ and (negated) on the right node $j$:

$$\begin{aligned} \mathbf{r}_i &= \mathbf{r}_i + \hat{\mathbf{\Phi}}_{ij}\,, \\ \mathbf{r}_j &= \mathbf{r}_j - \hat{\mathbf{\Phi}}_{ij}\,, \qquad (ij = 1, E)\,. \end{aligned} \tag{24}$$

This loop can be modified to perform the assembly of the two matrix–vector products given in Eq. (22). In fact, each component $i$ of the first matrix–vector product in Eq. (22) is given by:

$$\mathbf{z}_i = \sum_{k=1}^{N} \frac{\partial \mathbf{r}_i}{\partial \mathbf{u}_k} \mathbf{p}_k\,. \tag{25}$$

Differentiating Eq. (24) with respect to $\mathbf{u}_k$, multiplying by $\mathbf{p}_k$ and adding an additional internal loop gives:

$$\begin{aligned} \frac{\partial \mathbf{r}_i}{\partial \mathbf{u}_k}\mathbf{p}_k &= \frac{\partial \mathbf{r}_i}{\partial \mathbf{u}_k}\mathbf{p}_k + \frac{\partial \hat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k}\mathbf{p}_k\,, \\ \frac{\partial \mathbf{r}_j}{\partial \mathbf{u}_k}\mathbf{p}_k &= \frac{\partial \mathbf{r}_j}{\partial \mathbf{u}_k}\mathbf{p}_k - \frac{\partial \hat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k}\mathbf{p}_k\,, \qquad (ij = 1, E;\ k = 1, N)\,. \end{aligned} \tag{26}$$

where according to Eq. (25) the quantities accumulated on the nodes $i$ and $j$ are the components $\mathbf{z}_i$ and $\mathbf{z}_j$ of $\mathbf{Z}$. This means that the nested loop given in Eq. (26) can be rewritten as:

$$\begin{aligned} \mathbf{z}_i &= \mathbf{z}_i + \frac{\partial \hat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k}\mathbf{p}_k\,, \\ \mathbf{z}_j &= \mathbf{z}_j - \frac{\partial \hat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k}\mathbf{p}_k\,, \qquad (ij = 1, E;\ k = 1, N)\,. \end{aligned} \tag{27}$$

As indicated in Eq. (23) the numerical flux is dependent on the left and right states $i$ and $j$. Because of the reconstruction, such a dependency must be extended to the stencil since $\hat{\mathbf{u}}_i = \hat{\mathbf{u}}_i(\mathbf{u}_k; k \in \mathcal{N}_i)$.

The latter means that the numerical flux Jacobian is non–zero only for the elements contained in the stencil of node $i$ and node $j$ ($\partial\hat{\mathbf{\Phi}}_{ij}/\partial\mathbf{u}_k \neq \mathbf{0}$; $k \in \mathcal{N}_i \cup \mathcal{N}_j$). As a consequence, in Eq. (27) the inner loop on the nodes can be limited to a summation on both stencil $\mathcal{N}_i$ and $\mathcal{N}_j$ to give:

$$\begin{aligned} \mathbf{z}_i &= \mathbf{z}_i + \sum_{k\in\mathcal{N}_i} \frac{\partial\hat{\mathbf{\Phi}}_{ij}}{\partial\mathbf{u}_k}\mathbf{p}_k + \sum_{k\in\mathcal{N}_j} \frac{\partial\hat{\mathbf{\Phi}}_{ij}}{\partial\mathbf{u}_k}\mathbf{p}_k \,, \\ \mathbf{z}_j &= \mathbf{z}_j - \sum_{k\in\mathcal{N}_i} \frac{\partial\hat{\mathbf{\Phi}}_{ij}}{\partial\mathbf{u}_k}\mathbf{p}_k - \sum_{k\in\mathcal{N}_j} \frac{\partial\hat{\mathbf{\Phi}}_{ij}}{\partial\mathbf{u}_k}\mathbf{p}_k \,, \qquad (ij = 1, E)\,. \end{aligned} \tag{28}$$

The numerical flux derivative is with respect to the cell average $\mathbf{u}_k$. However, the numerical flux is actually evaluated with the reconstructed variables so that the chain rule can be applied to isolate the flux derivative from the reconstruction derivatives. In doing this the numerical flux derivative can be taken out of the stencil summation:

$$\begin{aligned} \mathbf{z}_i &= \mathbf{z}_i + \frac{\partial\hat{\mathbf{\Phi}}_{ij}}{\partial\hat{\mathbf{u}}_i} \sum_{k\in\mathcal{N}_i} \left.\frac{\partial\hat{\mathbf{u}}_i}{\partial\mathbf{u}_k}\right|_{ij} \mathbf{p}_k + \frac{\partial\hat{\mathbf{\Phi}}_{ij}}{\partial\hat{\mathbf{u}}_j} \sum_{k\in\mathcal{N}_j} \left.\frac{\partial\hat{\mathbf{u}}_j}{\partial\mathbf{u}_k}\right|_{ij} \mathbf{p}_k \,, \\ \mathbf{z}_j &= \mathbf{z}_j - \frac{\partial\hat{\mathbf{\Phi}}_{ij}}{\partial\hat{\mathbf{u}}_i} \sum_{k\in\mathcal{N}_i} \left.\frac{\partial\hat{\mathbf{u}}_i}{\partial\mathbf{u}_k}\right|_{ij} \mathbf{p}_k - \frac{\partial\hat{\mathbf{\Phi}}_{ij}}{\partial\hat{\mathbf{u}}_j} \sum_{k\in\mathcal{N}_j} \left.\frac{\partial\hat{\mathbf{u}}_j}{\partial\mathbf{u}_k}\right|_{ij} \mathbf{p}_k \,, \qquad (ij = 1, E)\,. \end{aligned} \tag{29}$$

The subscript $ij$ is necessary to remind that $\partial\hat{\mathbf{u}}_i/\partial\mathbf{u}_k$ and $\partial\hat{\mathbf{u}}_j/\partial\mathbf{u}_k$ are evaluated on the $ij$ edge. A careful examination of Eq. (29) shows that it is equivalent to Eq. (11). The edge–based assembly of the transposed Jacobian–vector product $\mathbf{Z} = [\partial\mathbf{R}/\partial\mathbf{U}]^T\mathbf{P}$ can be derived by Eq. (29) keeping in mind also the matrix–form given in Eq. (11) and Eq. (12). The numerical flux jacobians are easy to transpose since they lie on the diagonal of the matrix. The summation on the stencil for the reconstruction operator is relatively more complicated since it involves off-diagonal terms. Such a summation is on the row elements. Since by transposition they have to turn into column elements, the summation become a scattering on the stencil nodes:

$$\begin{aligned} \mathbf{z}_p &= \mathbf{z}_p + \left.\frac{\partial\hat{\mathbf{u}}_i}{\partial\mathbf{u}_p}\right|_{ij}^T \frac{\partial\hat{\mathbf{\Phi}}_{ij}}{\partial\hat{\mathbf{u}}_i}^T (\mathbf{p}_i - \mathbf{p}_j) \qquad p \in \mathcal{N}_i \,, \\ \mathbf{z}_q &= \mathbf{z}_q + \left.\frac{\partial\hat{\mathbf{u}}_j}{\partial\mathbf{u}_q}\right|_{ij}^T \frac{\partial\hat{\mathbf{\Phi}}_{ij}}{\partial\hat{\mathbf{u}}_j}^T (\mathbf{p}_i - \mathbf{p}_j) \qquad q \in \mathcal{N}_j \,, \qquad (ij = 1, E)\,. \end{aligned} \tag{30}$$

This assembly is equivalent to the matrix form given in Eq. (12). A two–pass assembly of both loops in Eq. (30) and Eq. (29) can be performed in order to optimize the construction process in terms storage requirements. To do so it is possible to use the original data structure of the flow solver (the edge pointer to the the left and right nodes). A brute–force approach is to introduce another pointer which links each node with its distance–one neighbors and use it to perform the summation in Eq. (29) or the scattering in Eq. (30) in only one pass. Clearly the two–pass approach is more suitable for large scale applications. However, when the adjoint is exact and also the limiter is included it become a very tedious and error prone operation to perform this approach. In fact the reconstruction contribution is as follows:

$$\frac{\partial\hat{\mathbf{u}}_i}{\partial\mathbf{u}_k} = \frac{\partial\hat{\mathbf{u}}_i}{\partial\hat{\mathbf{v}}_i}\frac{\partial\hat{\mathbf{v}}_i}{\partial\mathbf{v}_k}\frac{\partial\mathbf{v}_k}{\partial\mathbf{u}_k} \,, \qquad \frac{\partial\hat{\mathbf{v}}_i}{\partial\mathbf{v}_k} = \begin{bmatrix} \partial\hat{\rho}_i/\partial\rho_k & 0 & 0 & 0 \\ 0 & \partial\hat{u}_i/\partial u_k & 0 & 0 \\ 0 & 0 & \partial\hat{v}_i/\partial v_k & 0 \\ 0 & 0 & 0 & \partial\hat{p}_i/\partial p_k \end{bmatrix} , \tag{31}$$

where the first and third matrix are transformation matrix between conservative and primitive variables. They are necessary if the reconstruction is on the primitive variables. For instance, considering for the $y$–velocity a reconstruction of the type

$$\hat{v}_i = v_i + \frac{\sigma_i}{2}\nabla v_i^T(\mathbf{x}_j - \mathbf{x}_i)\,, \tag{32}$$

differentiating with respect to $v_k$ gives:

$$\frac{\partial \hat{v}_i}{\partial v_k} = \delta_{ik} + \frac{1}{2}\frac{\partial \sigma_i}{\partial v_k}\nabla v_i^T(\mathbf{x}_j - \mathbf{x}_i) + \frac{\sigma_i}{2}\frac{\partial \nabla v_i}{\partial v_k}^T(\mathbf{x}_j - \mathbf{x}_i)\,, \tag{33}$$

where $\delta_{ik}$ is the Kronecker delta. If the limiter contribution $\partial\sigma_i/\partial v_k$ is neglected only the gradient contribution must be computed. Such a contribution is straightforward since only metrics quantities are involved. In the case of the Green–Gauss gradient and similar for the weighed least squares gradient:

$$\nabla v_i = \frac{1}{2A_s}\sum_{k=1}^{N_i}(v_i + v_k)\,\mathbf{n}_{ik}\,, \qquad \frac{\partial \nabla v_i}{\partial v_k} = \frac{\mathbf{n}_{ik}}{2A_s} \quad i \neq k\,, \qquad \frac{\partial \nabla v_i}{\partial v_k} = \frac{1}{2A_s}\sum_{k=1}^{N_i}\mathbf{n}_{ik} \quad i = k\,. \tag{34}$$

Looking at the definition of the limiter used in this work (Eq. (5)) , the differentiation is not as easy as for the gradient. The limiter has a dependency on all the nodes in the stencil and the differentiation does not involve metric quantities only. Exact adjoint construction has been performed using a brute–force approach: for each node the limiter contribution $\partial\sigma_i/\partial v_k$ for $(k = 1, N_i)$ is computed from the exactly differentiated limiter routine, stored and finally assembled on the edge.

For the contribution of the numerical flux, the exact differentiation of Roe's approximate Riemann solver is available from previous work:[6]

$$\begin{aligned}
\frac{\partial \mathbf{\Phi}_{ij}}{\partial \mathbf{u}_i} &= \frac{1}{2}(\mathbf{A}(\mathbf{u}_i, \mathbf{n}_{ij}) + |\mathbf{A}(\tilde{\mathbf{u}}_{ij}, \mathbf{n}_{ij})|) + (\mathbf{M}_1\mathbf{M}_2 + \mathbf{M}_3)\,\mathbf{M}_4\,,\\
\frac{\partial \mathbf{\Phi}_{ij}}{\partial \mathbf{u}_j} &= \frac{1}{2}(\mathbf{A}(\mathbf{u}_j, \mathbf{n}_{ij}) - |\mathbf{A}(\tilde{\mathbf{u}}_{ij}, \mathbf{n}_{ij})|) + (\mathbf{M}_1\mathbf{M}_2 + \mathbf{M}_3)\,\mathbf{M}_5\,,
\end{aligned} \tag{35}$$

where the five matrices $\mathbf{M}_1$, $\mathbf{M}_2$, $\mathbf{M}_3$, $\mathbf{M}_4$ and $\mathbf{M}_5$ have been derived here following an approach similar to that used in the original reference. For the most complex of those matrices, $\mathbf{M}_3$, symbolic differentiation was used. The differentiation given in Eq. (35) can also be used for the linearization of the far–field boundary fluxes computed with flux–vector splitting.

## Acknowledgments

This research is supported by the Dutch Technology Foundation STW, applied science division of NWO and the technology program of the Dutch Ministry of Economic Affairs.

## References

[1]Jameson, A., "Aerodynamic Design via Control Theory," *Journal of Scientific Computing*, Vol. 3, 1988, pp. 233–260.

[2]Nielsen, E. J., *Aerodynamic Design Sensitivities on an Unstructured Mesh Using the Navier–Stokes Equations and a Discrete Adjoint Formulation*, Ph.D. thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, Dec. 1998.

[3]Mohammadi, B., "A New Optimal Shape Design Procedure for Inviscid and Viscous Turbulent Flows," *International Journal for Numerical Methods in Fluids*, Vol. 25, 1997, pp. 183–203.

[4]Giles, M. B., Duta, M. C., Müller, J.-D., and Pierce, N. A., "Algorithm Developments for Discrete Adjoint Methods," *AIAA Journal*, Vol. 41, No. 2, 2003, pp. 198–205.

[5]Müller, J.-D. and Cusdin, P., "On the Performance of Discrete Adjoint CFD Codes using Automatic Differentiation," *Int. J. Numer. Meth Fluids*, Vol. 47, 2003, pp. 939–945.

[6]Barth, T. J., "Analysis of Implicit Local Linearization Techniques for Upwind and TVD Algorithms," AIAA Paper 87–0595, Reno, NV, Jan. 1987.

[7]Barth, T. J. and Linton, S. W., "An Unstructured Mesh Newton Solver for Compressible Fluid Flow and its Parallel Implementation," AIAA Paper 95–0221, Reno, NV, Jan. 1995.

[8]Nielsen, E. J., Lu, J., Park, M. A., and Darmofal, D. L., "An Implicit, Exact Dual Adjoint Solution Method for Turbulent Flows on Unstructured Grids," *Computers & Fluids*, Vol. 33, 2004, pp. 1131–1155.

[9]Mavriplis, D. J., "Formulation and Multigrid Solution of the Discrete Adjoint for Optimization Problems on Unstructured Meshes," AIAA Paper 05–0319, Reno, NV, Jan. 2005.

[10]Verhoff, A., Stooksberry, D., and Cain, A. B., "An Efficient Approach to Optimal Aerodynamic Design Part 1: Analytic Geometry and Aerodynamic Sensitivities," AIAA Paper 93–0099, Reno, NV, Jan. 1993.

[11]Selmin, V. and Formaggia, L., "Unified Construction of Finite Element and Finite Volume Discretizations for Compressible Flows," *International Journal for Numerical Methods in Engineering*, Vol. 39, 1996, pp. 1–32.

[12]Barth, T. J., "Aspects of Unstructured Grids and Finite–Volume Solvers for the Euler and Navier–Stokes Equations," Lecture Series 1991–06, Von Karman Institute for Fluid Dynamics, 1991.

[13]Venkatakrishnan, V., "On the Accuracy of Limiters and Convergence to Steady State Solutions," AIAA Paper 93–0880, Reno, NV, Jan. 1993.

[14]Mohammadi, B., "CFD with NSC2KE: a user guide," INRIA Tech. Rep. 164, 1994.

[15]Koren, B., "Defect Correction and Multigrid for an Efficient and Accurate Computation of Airfoil Flows," *Journal of Computational Physics*, Vol. 77, 1988, pp. 183–206.

[16]Koren, B., *A Robust Upwind Discretization Method for Advection, Diffusion and Source Terms*, Vol. 45, Notes on Numerical Fluid Mechanics, Vieweg, Braunschweig, 1993.

[17]Jameson, A., "Optimum Transonic Wing Design Using Control Theory," Symposium transsonicum iv, International Union of Theoretical and Applied Mechanics, DLR Gottingen, Germany, Sept. 2002.

[18]Saad, Y. and Schultz, M., "GMRES: A Generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.*, Vol. 7, 1986, pp. 856–869.

[19]Frayssé, V., Giraud, L., Gratton, S., and Langou, J., "A Set of GMRES Routines for Real and Complex Arithmetics on High Performance Computers," CERFACS Tech. Rep. TR/PA/03/3, 2003, public domain software available on www.cerfacs/algor/Softs.

[20]"SLATEC Common Mathematical Library, Version 4.1," 1993, public domain software available on www.netlib.org/slatec.

[21]Mavriplis, D. J., "On Convergence Acceleration Techniques for Unstructured Meshes," AIAA Paper 98–2966, Albuquerque, NM, June 1998.

[22]Venkatakrishnan, V. and Mavriplis, D. J., "Implicit Solvers for Unstructured Meshes," *Journal of Computational Physics*, Vol. 105, 1993, pp. 83–91.

[23]Vanderplaats, G. N., *Numerical Optimization Techniques for Engineering Design*, Vanderplaats Reasearch & Development, Inc, 3rd ed., 2001.

[24]Shim, J., Lee, K. D., and Verhoff, A., "An Efficient Aerodynamic Design Method Using Asymptotic Solution of Euler Equations," AIAA Paper 02–3141, St. Louis, Missouri, June 2002.

[25]Trépanier, J. I., Lépine, J., and Pépin, F., "An Optimized Geometric Representation for Wing Profile using NURBS," *CASI Journal*, Vol. 46, No. 1, 2000, pp. 12–19.

[26]Carpentieri, G., van Tooren, M. J. L., Kelly, M., and Cooper, R., "Airfoil Optimization Using an Analytical Shape Parameterization," CEIAT Paper 05–0073, Belfast, UK, Aug. 2005.

[27]"Matlab Help Documentation," Matlab 7, Optimization Toolbox.