Available online at www.sciencedirect.com
ScienceDirect
Journal of Computational Physics 224 (2007) 267–287
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

Adjoint-based aerodynamic shape optimization on unstructured meshes

G. Carpentieri [a,*], B. Koren [a,b], M.J.L. van Tooren [a]

[a] Delft University of Technology, Faculty of Aerospace Engineering, Kluyverweg 1, 2629 HS Delft, The Netherlands
[b] Centre for Mathematics and Computer Science, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands

Received 15 September 2006; received in revised form 5 January 2007; accepted 9 February 2007
Available online 23 February 2007

This paper is dedicated to Professor Pieter Wesseling, aeronautical engineer and expert in computational fluid dynamics.

Abstract

In this paper, the exact discrete adjoint of an unstructured finite-volume formulation of the Euler equations in two dimensions is derived and implemented. The adjoint equations are solved with the same implicit scheme as used for the flow equations. The scheme is modified to efficiently account for multiple functionals simultaneously. An optimization framework, which couples an analytical shape parameterization to the flow/adjoint solver and to algorithms for constrained optimization, is tested on airfoil design cases involving transonic as well as supersonic flows. The effect of some approximations in the discrete adjoint, which aim at reducing the complexity of the implementation, is shown in terms of optimization results rather than only in terms of gradient accuracy. The shape-optimization method appears to be very efficient and robust.

Keywords: Shape optimization; Discrete adjoint; Unstructured solvers

1. 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 [16]. This method is very attractive in its discrete approach where the adjoint problem is directly formulated for the discretized flow equations [14]. The method is relatively easy to understand since only some linear algebra is involved. Nevertheless, the derivation of the discrete adjoint code may be challenging due to the complexity of differentiating the original discrete formulation. Hand-coding the adjoint may require a lot of human work but it has the potential of yielding an efficient code as well as a deep understanding of the process

* Corresponding author.
E-mail addresses: g.carpentieri@tudelft.nl (G. Carpentieri), barry.koren@cwi.nl (B. Koren), m.j.l.vantooren@tudelft.nl (M.J.L. van Tooren).

0021-9991/$ - see front matter
doi:10.1016/j.jcp.2007.02.011

[6,2,29,27,24,1]. Automatic Differentiation tools to generate the code may be successfully applied at a strongly reduced human effort. However, computational efficiency issues should not be underestimated when using the so-called reverse mode [25,13,26]. A broad review of sensitivity analysis and shape optimization can be found in [28].

In this work, the adjoint algorithm is hand-coded following a methodology outlined in previous works [3,5], in the context of implicit solver development. Only the geometric sensitivities are computed by using the forward mode of Automatic Differentiation. The discrete adjoint implemented here is exact since the residual Jacobian and all the geometric sensitivities are obtained from the exact differentiation of the code. The exactness of the hand-coded residual Jacobian is demonstrated through the quadratic convergence property of Newton iterations.

In practice, to simplify the derivation of the adjoint code, different approximations can be made in the differentiation. In turn, these approximations can have a detrimental effect on the gradient accuracy [6,2,29,18,27]. To judge whether some of these approximations are acceptable, their effects on the optimization behavior should be considered [12,9]. These effects will be investigated here in terms of optimization results rather than only in terms of gradient accuracy.

The solution process for the adjoint problem, despite the linearity of the equations, can be the same time marching as adopted in the flow solver [29,13,30]. 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 fact, in constrained shape optimization more than one functional is usually involved so that it is convenient to solve the adjoint problems simultaneously rather than sequentially.

The flow and the adjoint solver are part of an optimization framework, which includes the shape parameterization and optimization algorithms. The shape parameterization used here is based upon orthogonal Chebyshev polynomials [42]. This parameterization is relatively unexplored in the literature [34] in spite of its interesting features. It gives completeness in the design space and behaves smoothly during the optimization process even in case of large shape deformations.

Two optimization algorithms are used. The first one is a Sequential Quadratic Programming algorithm taken from an external library. The second one is a Sequential Linear Programming algorithm, known as the method of centers, which has been applied for the first time to shape optimization problems by the present authors [10]. Two test cases are presented that show the effectiveness of the algorithm.

In summary, the paper contributes to the field of adjoint-based shape optimization with the following: (i) an exact discrete adjoint assembly for unstructured finite-volume solvers, of which a detailed description is given and exactness is demonstrated, (ii) the investigation of several approximations in the exact adjoint code in terms of optimization results, (iii) the efficient simultaneous solution of multiple adjoint equations, and (iv) shape parameterization by orthogonal Chebyshev polynomials in the context of adjoint-based shape optimization.

The effectiveness and robustness of the method are demonstrated on several constrained design cases in transonic and supersonic flow.

2. Finite volume formulation

The Euler equations are written in integral form as

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

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

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

Both are defined as functions of the primitive variables $\mathbf{v} = [\rho, \mathbf{w}, p]^{\mathrm{T}}$, which are density, velocity ($\mathbf{w} = [w_x, w_y]^{\mathrm{T}}$ 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.

Eq. (1) is discretized in a finite-volume framework [35,43]. For each internal control volume it holds

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

where $\mathbf{\Phi}$ is the numerical flux evaluated at the interface ∂V_{ik} between two control volumes i and k. The numerical flux depends on the integrated normal $\mathbf{n}_{ik}$ ($\equiv \int_{\partial V_{ik}} \mathbf{n} \mathrm{d}\Gamma$) and on the left and right states $\hat{\mathbf{u}}_i$ and $\hat{\mathbf{u}}_k$. Summation of the numerical flux across all the control volume interfaces ∂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 following a MUSCL-type approach [4]. For each variable v_i a limited linear reconstruction is made across each control–volume interface ∂V_{ik}:

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

where σ_i is a slope limiter, ∇v_i the variable gradient and $\mathbf{x}_{ik}$ the mid-point location of ∂V_{ik}. The slope limiter is that of Venkatakrishnan [40] 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 $\Delta v_{ik} = \nabla v_i^{\mathrm{T}} (\mathbf{x}_{ik} - \mathbf{x}_i)$ being the unlimited differential, ε a threshold, $\alpha_i = v_{\max} - v_i$ for $\Delta v_{ik} \geqslant 0$ 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 $\mathcal{N}_i$, which is a set composed by i and all its distance-one neighbors ($k = 1, N_i$; see Fig. A.1 in the Appendix). The gradient ∇v_i is computed using a linear least-squares technique [4] that reconstructs linear polynomials exactly. A Green–Gauss gradient is also available. The numerical flux is evaluated using Roe's approximate Riemann solver [33], which for a generic edge ij reads

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

where $|\mathbf{A}| = |\mathrm{d}(\mathbf{F}(\mathbf{u}) \cdot \mathbf{n})/\mathrm{d}\mathbf{u}|$ is the absolute flux Jacobian evaluated with the Roe averages $\tilde{\mathbf{u}}_{ij}$. Eq. (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. Eq. (3) can be rewritten in the semi-discrete form

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

where $\mathbf{D}$ is a diagonal matrix containing the control volumes, $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N]^{\mathrm{T}}$ and $\mathbf{R} = [\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N]^{\mathrm{T}}$ are the conservative variables and the residual vector, respectively. The total number of control volumes is denoted by 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 a mesh. An edge-based data structure is used in the solver [4]. Due to the flexibility of the formulation, hybrid meshes of triangular and quadrilateral elements can be easily processed [35]. Time marching of Eq. (7) is addressed in Section 4.

Fig. 1 compares the pressure distribution of the present solver with that of a structured flow solver [19] that uses variable extrapolation together with an Osher flux and Koren's limiter [20]. The mesh used for the computation was a 128×80 O-mesh. The present solver uses an unstructured triangular mesh of 8000 nodes with almost the same number of nodes on the wall. Additional comparisons can be found in [7].

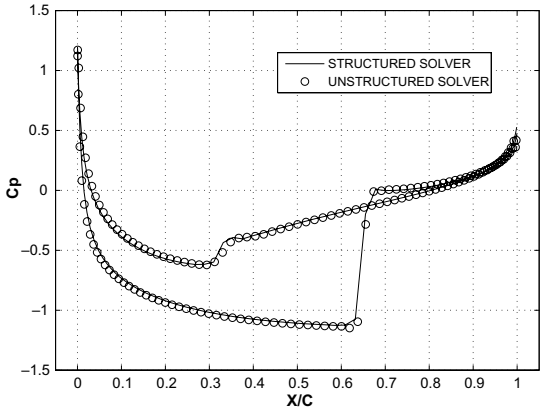


Fig. 1. Pressure distribution of the unstructured solver compared with that of a structured solver (NACA0012, $\alpha = 1.25°$ and $M_\infty = 0.8$).

3. Adjoint formulation: exact discrete adjoint for the MUSCL scheme

3.1. The adjoint method

Consider a functional J (e.g., lift or drag) for which the sensitivity with respect to a set of shape parameters (α_k is one of these parameters) must be computed. The functional is also dependent on the flow variables as well as on the shape of the domain according to the shape parameters: $J = J(\mathbf{U}, \alpha_k)$. An efficient way to accomplish the computation is via an augmented functional L. Using a terminology similar to that used in control theory [16] the conservative variables $\mathbf{U}$ can be identified as state variables and the set of shape parameters as decision variables. The state of the system is represented by the residual vector $\mathbf{R}(\mathbf{U}, \alpha_k)$, which depends on both state and decision variables. The functional L is obtained by augmenting J with the state of the system. A vector of multipliers $\mathbf{\Lambda}_J$ is introduced:

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

At the stationary point of the augmented functional L its sensitivity coincides with that of the original functional J, $\mathrm{d}J/\mathrm{d}\alpha_k \equiv \partial L/\partial \alpha_k$. Imposing the stationary condition to the augmented functional, $[\partial L/\partial \mathbf{U}, \partial L/\partial \alpha_k, \partial L/\partial \mathbf{\Lambda}_J] = 0$, gives a set of three equations. The first equation and the second equation are the adjoint equation and the equation for the sensitivity of the functional J, respectively:

$$\frac{\partial \mathbf{R}^{\mathrm{T}}}{\partial \mathbf{U}} \mathbf{\Lambda}_J = \frac{\partial J^{\mathrm{T}}}{\partial \mathbf{U}}, \quad \frac{\mathrm{d}J}{\mathrm{d}\alpha_k} = \frac{\partial J}{\partial \alpha_k} - \mathbf{\Lambda}_J^{\mathrm{T}} \frac{\partial \mathbf{R}}{\partial \alpha_k}. \tag{9}$$

The third equation is the state equation, $\mathbf{R} = 0$, which is solved using the finite-volume solver introduced before. The sensitivity is calculated using the multipliers or adjoint variables $\mathbf{\Lambda}_J$ 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. However, since each functional J has its own adjoint $\mathbf{\Lambda}_J$, the adjoint equation must be solved as many times as the number of functionals. The two equations in (9) can be referred to as the dual problem. In contrast, the primal problem evaluates the functional sensitivity by directly using the flow sensitivity $\mathrm{d}\mathbf{U}/\mathrm{d}\alpha_k$ computed from the linearized flow equations:

$$\frac{\partial \mathbf{R}}{\partial \mathbf{U}} \frac{\mathrm{d}\mathbf{U}}{\mathrm{d}\alpha_k} = -\frac{\partial \mathbf{R}}{\partial \alpha_k}, \quad \frac{\mathrm{d}J}{\mathrm{d}\alpha_k} = \frac{\partial J}{\partial \alpha_k} + \frac{\partial J}{\partial \mathbf{U}} \frac{\mathrm{d}\mathbf{U}}{\mathrm{d}\alpha_k}. \tag{10}$$

A drawback of this approach is that the linearized flow equation, the first equation in (10), must be solved as many times as the number of shape parameters. On the other hand, one solution of the linearized flow equation can be used to compute the sensitivity for all functionals. Hence, 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 the case, 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. By manipulation of Eqs. (9) and (10) it can be shown that the sensitivities obtained by the two approaches are identical.

3.2. The derivation of the exact discrete adjoint

In the present work a discrete adjoint [14] is used, which means that the dual problem is directly formulated from the discretized equations. The challenging part of this approach is the derivation of the transposed residual Jacobian $[\partial\mathbf{R}/\partial\mathbf{U}]^{\mathrm{T}}$ which, as mentioned, involves differentiation of the original finite-volume solver. To properly derive, implement and check the accuracy of the transposed residual Jacobian $[\partial\mathbf{R}/\partial\mathbf{U}]^{\mathrm{T}}$, it is necessary to have the residual Jacobian $\partial\mathbf{R}/\partial\mathbf{U}$ available.

In order to derive both Jacobians, following previous work [5], three new vectors of length E equal to the number of control volume interfaces are 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 $\mathbf{H} = [\widehat{\mathbf{\Phi}}_1, \widehat{\mathbf{\Phi}}_2, \ldots, \widehat{\mathbf{\Phi}}_E]^{\mathrm{T}}$ contains the second-order fluxes for each edge. The second and third vectors, $\mathbf{U}_{\mathrm{L}} = [\hat{\mathbf{u}}_{\mathrm{L}1}, \hat{\mathbf{u}}_{\mathrm{L}2}, \ldots, \hat{\mathbf{u}}_{\mathrm{L}E}]^{\mathrm{T}}$ and $\mathbf{U}_{\mathrm{R}} = [\hat{\mathbf{u}}_{\mathrm{R}1}, \hat{\mathbf{u}}_{\mathrm{R}2}, \ldots, \hat{\mathbf{u}}_{\mathrm{R}E}]^{\mathrm{T}}$, contain the reconstructed left and right states for each edge, respectively. The dependency of the residual Jacobian on the conservative variables can be expressed with these vectors as $\mathbf{R} = \mathbf{R}(\mathbf{H}(\mathbf{U}_{\mathrm{L}}(\mathbf{U}), \mathbf{U}_{\mathrm{R}}(\mathbf{U})))$. Therefore, by means of the chain rule, the residual Jacobian and the transposed residual Jacobian are obtained as

$$\frac{\partial\mathbf{R}}{\partial\mathbf{U}} = \frac{\partial\mathbf{R}}{\partial\mathbf{H}}\left(\frac{\partial\mathbf{H}}{\partial\mathbf{U}_{\mathrm{L}}}\frac{\partial\mathbf{U}_{\mathrm{L}}}{\partial\mathbf{U}} + \frac{\partial\mathbf{H}}{\partial\mathbf{U}_{\mathrm{R}}}\frac{\partial\mathbf{U}_{\mathrm{R}}}{\partial\mathbf{U}}\right), \tag{11}$$

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

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. (A.3)). $\partial\mathbf{H}/\partial\mathbf{U}_{\mathrm{L}}$ and $\partial\mathbf{H}/\partial\mathbf{U}_{\mathrm{R}}$ are square diagonal matrices 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 $\partial\mathbf{U}_{\mathrm{L}}/\partial\mathbf{U}$, 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 holds for the matrix $\partial\mathbf{U}_{\mathrm{R}}/\partial\mathbf{U}$ 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 4×4 for the 2D Euler equations.

In practice $[\partial\mathbf{R}/\partial\mathbf{U}]^{\mathrm{T}}$ or $\partial\mathbf{R}/\partial\mathbf{U}$ are not constructed at all. More likely their products with vectors are directly computed on-the-fly by looping over the edges of the mesh in much the same way as for the residual vector $\mathbf{R}$. In fact, $\mathbf{R}$ is assembled with a loop over the edges exploiting the conservation property, see Eq. (A.3). In the appendix, the derivation of the assembly of Eqs. (11) and (12) is given together with some details about the exact differentiation of the reconstruction operator and the numerical flux.

The exactness of the residual Jacobian $\partial\mathbf{R}/\partial\mathbf{U}$ has been tested using two methods. The first method is the direct comparison with matrix–vector products computed by different means. For instance, the Fréchet derivative (FD) of the residual vector, $[\partial\mathbf{R}/\partial\mathbf{U}]\mathbf{P} = (\mathbf{R}(\mathbf{U} + \epsilon\mathbf{P}) - \mathbf{R}(\mathbf{U}))/\epsilon$ with $\epsilon \to 0$, can be used. Unfortunately, this derivative is not very accurate due to truncation and cancellation errors. Therefore, the residual code has also been differentiated in forward mode using the automatic differentiation (AD) tool Tapenade [37]. A code that is capable of computing exact matrix–vector products has been generated. Fig. 2a shows the differences of the matrix–vector product implemented here compared with the Fréchet derivative ($\epsilon = 10^{-7}$) and with the AD code. As can be seen, the agreement with the AD code is up to machine zero.

The second method has never been used in the context of sensitivity analysis, it is a very robust way of verifying the exactness of the code. It checks the accuracy of the Jacobian indirectly, employing the latter in an implicit pseudo-time stepping procedure (see Section 4). This iterative procedure becomes Newton's method for infinitely large time steps. As can be seen from the residual history in Fig. 2b, the exact Jacobian attains a quadratic convergence and in only 5 iterations the residual is reduced by 10 orders of magnitude. Quadratic

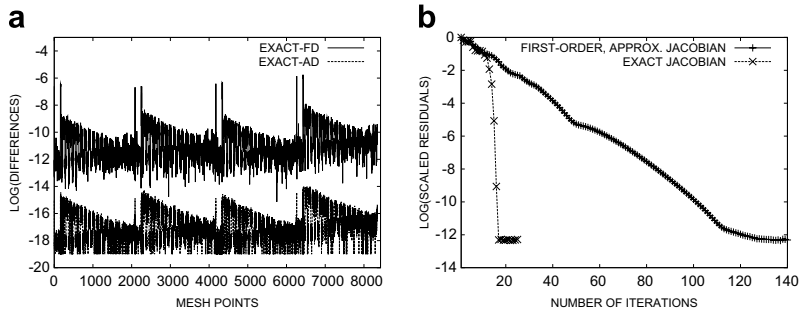


Fig. 2. Exactness of the Jacobian. Results are obtained from the computation of the RAE2822 airfoil at $M_\infty = 0.73$ and $\alpha = 2^\circ$ on an unstructured 2107 nodes mesh.

convergence is not obtained if the differentiation is not exact, for instance, if limiters are ignored (i.e., if they are considered to be constant in the differentiation, see Eq. (A.12)) or if a programming error is present. Finally, once the accuracy of $\partial\mathbf{R}/\partial\mathbf{U}$ has been verified, the accuracy of the transposed Jacobian is checked by means of the matrix identity $\mathbf{P}_1^{\mathrm{T}}[\partial\mathbf{R}/\partial\mathbf{U}]\mathbf{P}_2 = \mathbf{P}_2^{\mathrm{T}}[\partial\mathbf{R}/\partial\mathbf{U}]^{\mathrm{T}}\mathbf{P}_1$, which in this case is satisfied to machine accuracy ($\mathbf{P}_1$ and $\mathbf{P}_2$ are two generic vectors).

4. Time marching of flow and adjoint equations

Both the flow equations 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.

4.1. Implicit pseudo-time stepping

In order to derive an implicit pseudo-time stepping scheme for the semi-discrete system in Eq. (7), the time derivative can be discretized using a forward approximation, $\mathbf{D}(\mathrm{d}\mathbf{U}/\mathrm{d}t) \approx \mathbf{D}_t(\mathbf{U}^{n+1} - \mathbf{U}^n)$, whereas the residual term can be expanded linearly, $\mathbf{R}(\mathbf{U}^{n+1}) \approx \mathbf{R}(\mathbf{U}^n) + \partial\mathbf{R}/\partial\mathbf{U}^n(\mathbf{U}^{n+1} - \mathbf{U}^n)$. After rearrangement, the backward Euler scheme is obtained, which can be written as

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

where the diagonal matrix $\mathbf{D}_t$ contains the control volumes divided by their local time steps ($V_i/\Delta t_i$). For infinitely large time steps, the time derivative vanishes and Eq. (13) becomes Newton's method. Because of the second-order spatial discretization, the Jacobian $\partial\mathbf{R}/\partial\mathbf{U}$ is poorly diagonally dominant. As a consequence, the solution of the linear system of equations arising at each time step is hard to obtain. To get a more diagonally dominant Jacobian, better suited for iterative solutions, a common practice is to use an approximate Jacobian [4,41,23,29] in Eq. (13). This practice is equivalent to a defect correction approach [19] in pseudo-time,

$$\mathbf{D}_t(\mathbf{U}^{n+1} - \mathbf{U}^n) + \widetilde{\mathbf{R}}(\mathbf{U}^{n+1}) = \widetilde{\mathbf{R}}(\mathbf{U}^n) - \mathbf{R}(\mathbf{U}^n), \tag{14}$$

where $\widetilde{\mathbf{R}}$ is the residual of a lower-order discretization. In fact, taking a linear expansion of the residual $\widetilde{\mathbf{R}}(\mathbf{U}^{n+1})$, after some rearrangement one obtains

$$\left(\mathbf{D}_t + \frac{\partial\widetilde{\mathbf{R}}}{\partial\mathbf{U}}\right)^n (\mathbf{U}^{n+1} - \mathbf{U}^n) = -\mathbf{R}^n. \tag{15}$$

Compared to the Jacobian in Eq. (13), the Jacobian $\partial\widetilde{\mathbf{R}}/\partial\mathbf{U}$ is more diagonally dominant since it is obtained from a lower-order discretization. Consequently, the solution of the linear system is relatively easy and robust.

The price to pay for using a lower-order approximation of the Jacobian in Eq. (15) is the loss in quadratic convergence rate (see Fig. 2b). In the present work, the Jacobian $\partial\widetilde{\mathbf{R}}/\partial\mathbf{U}$ is first-order accurate, since the reconstruction contribution is neglected, and approximate, since the Roe matrix is frozen in the differentiation of the numerical flux in Eq. (6) (see next section).

In order to speed up the convergence, at each iteration the time step is increased according to a CFL-number 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 β a suitable parameter. Depending on the flow type the CFL-number is limited to a maximum value or it is left free to increase to infinity.

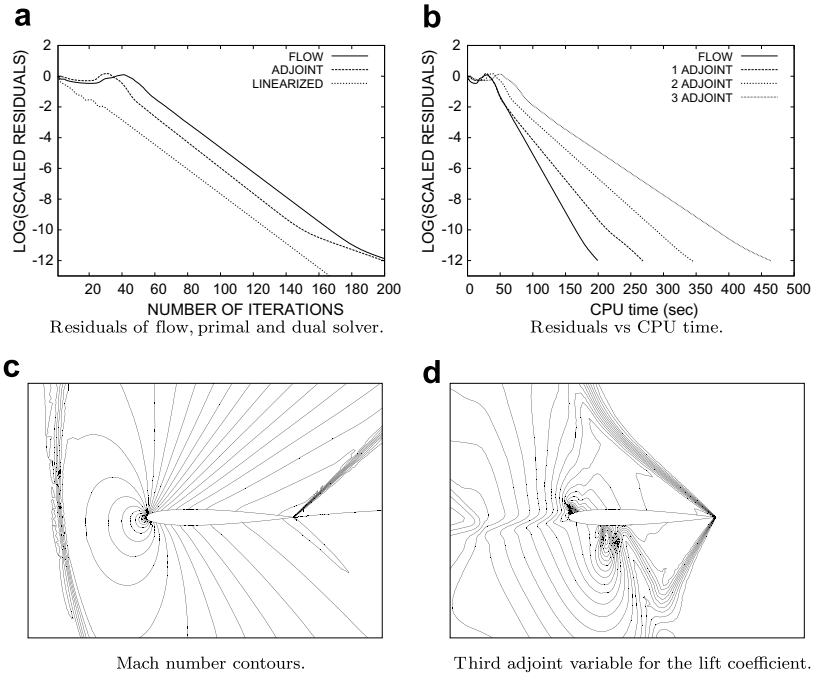
The adjoint equation appearing in the dual problem of Eq. (9) is a linear system for the adjoint variables $\mathbf{\Lambda}_J$. Due to the off-diagonal contribution arising from the reconstruction operator the system is poorly diagonally dominant. A time-like contribution can be added, which results in a more robust solver [30]. In practice, the time-stepping and settings used for the flow solver are also used in the adjoint solver:

$$\left(\mathbf{D}_t + \frac{\partial\widetilde{\mathbf{R}}}{\partial\mathbf{U}}^{\mathrm{T}}\right)^n (\mathbf{\Lambda}_J^{n+1} - \mathbf{\Lambda}_J^n) = -\left(\frac{\partial\mathbf{R}}{\partial\mathbf{U}}^{\mathrm{T}}\mathbf{\Lambda}_J^n - \frac{\partial J}{\partial\mathbf{U}}^{\mathrm{T}}\right). \tag{16}$$

Constrained shape optimization problems may require Eq. (16) to be solved as many times as the number of functionals. One has to compute the adjoint $\mathbf{\Lambda}_J$ of each functional J. Eq. (16) is a linear system of equations, which is solved iteratively at each time step n. As described in the next section, the solution method does not require the inversion of the matrix appearing at the left-hand side of the equation. Only matrix–vector products are required for both the left- and right-hand sides, products that are always computed on-the-fly.

The matrix terms are expensive to compute, but they are identical for all functionals. Hence, it makes sense to perform more matrix–vector products simultaneously. Thus, rather than solving Eq. (16) sequentially for each adjoint $\mathbf{\Lambda}_J$, one can simultaneously solve for all the adjoints at once. Numerical experiments show that simultaneous time stepping gives an appreciable time saving compared to sequential solution.

Results for a supersonic flow are shown in Fig. 3. An unstructured mesh of 27,684 triangles and 13,962 nodes has been used for the computation. Values of $c_l = 0.5202$ and $c_d = 0.1553$ are found, which are in agreement with the values $c_l = 0.5237$ and $c_d = 0.1551$ given in [19]. The residuals of the adjoint and the linearized



Residuals of flow, primal and dual solver.

Residuals vs CPU time.

Mach number contours.

Third adjoint variable for the lift coefficient.

Fig. 3. NACA0012 airfoil at $M_\infty = 1.2$ and $\alpha = 7°$.

solver, see Fig. 3a, converge at the same rate as the flow residual. The linearized solver for the primal problem is defined in Eq. (10), and similarly to the adjoint, it uses the same solution method as the flow solver. The convergence in terms of CPU time, which is useful to know the efficiency of the adjoint solver compared to that of the flow solver, is depicted in Fig. 3b. Considering the slope of the linear part for each line, the flow solver takes 13 s to lower the residual one order of magnitude whereas one adjoint solution (second line from the left) takes 19 s. Hence, solving three adjoint problems sequentially would take 57 s. As can be seen from Fig. 3b, with the solution method implemented here, three adjoints (fourth line from the left) can be solved simultaneously spending 35 s for each order of magnitude; a time saving of almost 40%.

The contours of the third adjoint variable, which represents the sensitivity of the lift coefficient to changes in vertical momentum, are depicted in Fig. 3d. It is nice to see that they have a reverse trend compared to the Mach contours depicted in Fig. 3c. 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 [31]. The quiet zone is simply the zone in which any perturbation in y-momentum does not affect the lift coefficient. The interested reader is referred to [14] for a further discussion of the mathematical and physical meaning of the adjoint variables.

4.2. Linear system of equations

At each time step, Eq. (15) implies the solution of a linear system. A simple iterative procedure is employed which, given the linear system $\mathbf{A}\mathbf{z} = \mathbf{b}$, computes corrections of the type:

$$\mathbf{z}^{k+1} = \mathbf{z}^k + \Delta\mathbf{z}, \quad \Delta\mathbf{z} = \mathbf{P}^{-1}\mathbf{r}_{\mathrm{L}}^k, \quad \mathbf{r}_{\mathrm{L}}^k = \mathbf{b} - \mathbf{A}\mathbf{z}^k, \tag{17}$$

where $\mathbf{r}_{\mathrm{L}}^k$ is the residual of the linear system and $\mathbf{P}$ the preconditioner computed from $\mathbf{A}$. 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. The Symmetric Successive Overrelaxation, SSOR, preconditioner has been implemented. It can be expressed as $\mathbf{P} = (\mathbf{D} + \mathbf{L})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U})$, where 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 4×4). The preconditioner can be inverted by means of a forward solve, $(\mathbf{D} + \mathbf{L})\Delta\mathbf{z}^* = \mathbf{r}_{\mathrm{L}}^k$, which is performed with one sweep on the nodes:

$$\Delta\mathbf{z}_i^* = \mathbf{D}_i^{-1}\left(\mathbf{r}_{\mathrm{L}_i}^k - \sum_{j\in\mathscr{L}_i}\mathbf{L}_{ij}\Delta\mathbf{z}_j^*\right), \quad (i = 1, N), \tag{18}$$

followed by a backward solve, $(\mathbf{I} + \mathbf{D}^{-1}\mathbf{U})\Delta\mathbf{z} = \Delta\mathbf{z}^*$, which is performed with another sweep on the nodes:

$$\Delta\mathbf{z}_i = \Delta\mathbf{z}_i^* - \mathbf{D}_i^{-1}\sum_{j\in\mathscr{U}_i}\mathbf{U}_{ij}\Delta\mathbf{z}_j, \quad (i = N, 1), \tag{19}$$

with $\mathscr{L}_i$ and $\mathscr{U}_i$ subsets of the stencil $\mathscr{N}_i$ ($\forall j \in \mathscr{L}_i : \ j < i$ and $\forall j \in \mathscr{U}_i : j > i$). For a first-order Jacobian, each edge produces four non-zero entries: two on the diagonal and two off-diagonal [23]. Therefore, the non-zero entries for each node (a Jacobian row) are only the nodes in the stencil. A pointer which links each node with its nearest neighbors, with suitably ordered elements, can be easily created in order to perform the sweeps in Eqs. (18) and (19). The elements of $\mathbf{D}$, $\mathbf{L}$ and $\mathbf{U}$ are

$$\mathbf{D}_i = \frac{V_i}{\Delta t_i}\mathbf{I} + \sum_{j=1}^{N_i}\frac{\partial\mathbf{\Phi}_{ij}}{\partial\mathbf{u}_i}, \quad \mathbf{L}_{ij} = -\frac{\partial\mathbf{\Phi}_{ij}}{\partial\mathbf{u}_i}, \quad \mathbf{U}_{ij} = \frac{\partial\mathbf{\Phi}_{ij}}{\partial\mathbf{u}_j}. \tag{20}$$

The diagonal elements, $\mathbf{D}_i$, are precomputed and stored prior to the execution of the two sweeps. The off-diagonal elements can also be precomputed and stored or, optionally, computed on-the-fly for each node during the two sweeps, making the method completely matrix-free. The numerical flux Jacobians in Eq. (20) are approximated (recall that the Jacobian in Eq. (15) was defined to be first-order and approximate) since the Roe matrix $|\mathbf{A}|$ of Eq. (6) is frozen in the differentiation:

$$\frac{\partial\mathbf{\Phi}_{ij}}{\partial\mathbf{u}_i} \approx \frac{1}{2}(\mathbf{A}_i + |\mathbf{A}|), \quad \frac{\partial\mathbf{\Phi}_{ij}}{\partial\mathbf{u}_j} \approx \frac{1}{2}(\mathbf{A}_j - |\mathbf{A}|), \tag{21}$$

where $\mathbf{A}_i = \mathbf{A}(\mathbf{u}_i, \mathbf{n}_{ij})$, $\mathbf{A}_j = \mathbf{A}(\mathbf{u}_j, \mathbf{n}_{ij})$ and $|\mathbf{A}| = |\mathbf{A}(\tilde{\mathbf{u}}_{ij}, \mathbf{n}_{ij})|$.

For the adjoint solution, for which the elements of the preconditioner are transposed, the matrices $\mathbf{D}_i^{\mathrm{T}}$, $\mathbf{L}_{ij}^{\mathrm{T}}$ and $\mathbf{U}_{ij}^{\mathrm{T}}$ are employed. Moreover, $\mathbf{U}_{ij}^{\mathrm{T}}$ should be used in Eq. (18) instead of Eq. (19) and vice versa. In the case of multiple right-hand sides, there are a number of linear residuals that must be preconditioned, one for each functional J. For instance, consider Eq. (18); there $\Delta\mathbf{z}_i^*$ is computed given $\mathbf{r}_{\mathrm{L}_i^k}$ for each functional, with $\mathbf{D}_i^{-\mathrm{T}}$ and $\mathbf{U}_{ij}^{\mathrm{T}}$ being equal for all functionals. This practice consists of simultaneously solving several linear systems that have the same matrix.

The solution method for the preconditioner presented above, the lower and upper sweeps of Eqs. (18) and (19), is similar to that used for the LU-SGS method [22]. However, the latter uses increments to compute matrix–vector products of the flux Jacobian with vectors, e.g., $\mathbf{U}_{ij}\Delta\mathbf{z}_j = (\mathbf{F}(\mathbf{u}_j + \Delta\mathbf{z}_j)\cdot\mathbf{n}_{ij} - \mathbf{F}(\mathbf{u}_j)\cdot\mathbf{n}_{ij} + \rho_{Aij}\Delta\mathbf{z}_j)/2$. Moreover, rather than the Roe matrix $|\mathbf{A}(\tilde{\mathbf{u}}_{ij}, \mathbf{n}_{ij})|$, its spectral radius $\rho_{Aij} = |\mathbf{w}_{ij}\cdot\mathbf{n}_{ij}|$ is used [17]. Note that for the adjoint equations this method is not suitable since increments cannot be employed because of the transposition.

5. Shape parameterization, mesh deformation and geometric sensitivities

5.1. Shape parameterization by means of Chebyshev polynomials

The parameterization of the geometry plays a crucial role in shape optimization. A limited parameterization would inhibit 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 parameterization can be defined as a series of basis functions [42,36],

$$
\begin{aligned}
&f_d(x) = \sum_{k=0}^{N_D} \alpha_k D_k(x), \quad D_k = T_k - T_{k+2},\\
&T_k(x) = \cos(k\gamma(x)), \quad \gamma(x) = \cos^{-1}(2\sqrt{x} - 1), \quad (k \geqslant 0, 0 \leqslant x \leqslant 1),
\end{aligned}
\tag{22}
$$

where N_D is the number of basis functions, and where D_k are the basis functions and α_k the shape parameters or basis function coefficients. Fig. 4 shows the basis functions D_k. It is interesting to see how the first basis, $k = 0$, resembles an airfoil shape.

Others use orthogonal shape functions [21,32,11] aiming at a complete representation of the design space, hence at a better convergence of the optimization process. These works derive the shape functions by means of a Gram–Schmidt orthonormalization process, given the initial shape. With the Chebyshev representation used here, the basis functions are available from Eq. (22). Only the coefficients α_k must be evaluated in order to approximate a given initial shape.

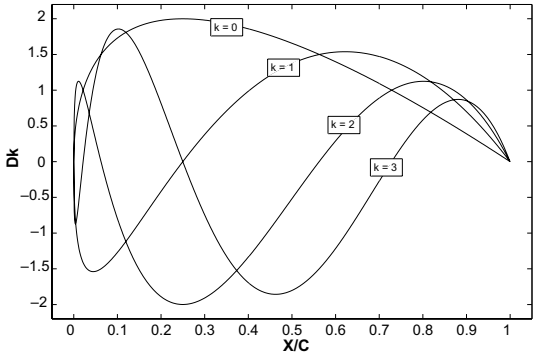


Fig. 4. Basis functions.

Eq. (22) 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 distributions are approximated independently. 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_p airfoil coordinates $([x_i, y_i]; i = 1, N_p)$ a local error $(\epsilon_i = |f_d(x_i) - y_i|)$, an average error $(\epsilon_m = 1/N_p \sum_{i=1}^{N_p} \epsilon_i)$ 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 too large discrepancies between the functionals computed on the original airfoil and on the approximated airfoil. An investigation in [38] shows that for most CFD solvers in use, a maximum error of $\epsilon_M = 8 \times 10^{-5}$ suffices. Using the Chebyshev polynomials different airfoils have been successfully approximated to the required maximum error [8]. $N_D = 11$ basis functions per curve was found to be satisfactory for the airfoils considered here. Thus, in the present work a total of 22 design basis functions is employed for each airfoil.

Since the shape parameterization is defined analytically, it is easy to calculate the slope and the radius of curvature at any point of the parameterized curve by taking derivatives of Eq. (22). They are obtained as $\arctan f_d$ and $(1 + f_d'^3)^{\frac{1}{2}}/f_d''$, respectively.

These quantities are used to compute the trailing edge angle, θ, and the radii of curvature at the nose, r_u and r_l, for the purpose of imposing geometric constraints. Two radii of curvature are computed at the leading edge due to a lack of continuity of the derivatives at that point. In fact, the derivatives of $f_d(x)$ in $x = 0$ and $x = 1$ can only be considered in the limit as $x \to 0$ and $x \to 1$.

5.2. Mesh deformation

When a shape parameter α_k changes, a displacement $\Delta \mathbf{X}_B$ in the boundary of the computational domain is imposed and therefore the mesh coordinates $\mathbf{X}$ must be updated accordingly. The latter are functions $\mathbf{X} = \mathbf{X}[\Delta \mathbf{X}_B(\alpha_k)]$ of the shape parameter.

A mesh deformation technique, known as spring analogy, is used to evaluate a displacement $\Delta \mathbf{X}$. The displacement of each internal node is initialized to zero whereas the displacement of the boundary points, contained in the set $\mathscr{B}$, is initialized to $\Delta \mathbf{X}_B$. The displacement of each point $\Delta \mathbf{X}_i$ is then iteratively solved with Jacobi iterations:

$$\Delta \mathbf{X}_i^{k+1} = \frac{\sum_{j=1}^{N_i} k_{ij} \Delta \mathbf{X}_j^k}{\sum_{j=1}^{N_i} k_{ij}}, \qquad (i = 1, N; i \notin \mathscr{B}), \tag{23}$$

where N_i is the number of nearest neighbors of the node i and k_{ij} is the stiffness associated with the edge ij. The latter has a numerical value equal to the inverse of the edge length. By means of Eq. (23), the boundary displacement $\Delta \mathbf{X}_B$ is iteratively propagated in the whole domain. When the iterations are stopped, the mesh coordinates are updated as $\mathbf{X}^{\text{new}} = \mathbf{X} + \Delta \mathbf{X}$.

5.3. Geometric sensitivities via Automatic Differentiation

Once the adjoint variables have been computed, the gradient of the functional $\mathrm{d}J/\mathrm{d}\alpha_k$ is obtained using the second formula in Eq. (9), which requires the computation of the geometric sensitivities $\partial J/\partial \alpha_k$ and $\partial \mathbf{R}/\partial \alpha_k$. For this purpose, the forward mode of Automatic Differentiation can be employed. Contrary to the reverse mode, the forward mode relies on basic linearization rules and it does not suffer from memory issues. The AD tool Tapenade [37] has been used. It allows to compute $\partial \mathbf{R}/\partial \alpha_k$ for the complete design vector $[\alpha_1, \alpha_2, \ldots, \alpha_N]$ in one shot. In abstract form, the residual vector has a dependency $\mathbf{R} = \mathbf{R}[\mathbf{X}, \mathbf{N}(\mathbf{X}), \nabla \mathbf{V}(\mathbf{X}), \mathbf{\Sigma}(\mathbf{X})]$ on the mesh points. $\mathbf{N}$ is the grid metrics, the integrated normal vectors in the grid and on the boundary, and $\nabla \mathbf{V}$ and $\mathbf{\Sigma}$ are the primitive variables gradient and the limiter vector, respectively. Recalling that $\mathbf{X} = \mathbf{X}[\Delta \mathbf{X}_B(\alpha_k)]$, after application of the chain rule the complete derivative reads:

$$\frac{\partial \mathbf{R}}{\partial \alpha_k} = \left(\frac{\partial \mathbf{R}}{\partial \mathbf{X}} + \frac{\partial \mathbf{R}}{\partial \mathbf{N}} \frac{\partial \mathbf{N}}{\partial \mathbf{X}} + \frac{\partial \mathbf{R}}{\partial \nabla \mathbf{V}} \frac{\partial \nabla \mathbf{V}}{\partial \mathbf{X}} + \frac{\partial \mathbf{R}}{\partial \mathbf{\Sigma}} \frac{\partial \mathbf{\Sigma}}{\partial \mathbf{X}} \right) \frac{\partial \mathbf{X}}{\partial \Delta \mathbf{X}_B} \frac{\partial \Delta \mathbf{X}_B}{\partial \alpha_k}, \tag{24}$$

where in practice, each term corresponds to a differentiated sub-routine. $\partial\Delta\mathbf{X}_B/\partial\alpha_k$ is the differentiation of the shape parameterization routine with respect to the shape parameter, $\partial\mathbf{X}/\partial\Delta\mathbf{X}_B$ is the differentiation of the spring analogy routine with respect to the boundary displacement. $\partial\nabla\mathbf{V}/\partial\mathbf{X}$ and $\partial\boldsymbol{\Sigma}/\partial\mathbf{X}$ are the limiter and gradient routines, which are differentiated with respect to the mesh points. $\partial\mathbf{R}/\partial\mathbf{X}$, $\partial\mathbf{R}/\partial\mathbf{N}$, $\partial\mathbf{R}/\partial\nabla\mathbf{V}$ and $\partial\mathbf{R}/\partial\boldsymbol{\Sigma}$ represent the differentiation of the residual routine with respect to mesh points, metrics, gradients and limiters, respectively.

6. Shape optimization

6.1. The optimization problem

The present work focuses on a shape optimization problem in which the drag coefficient c_d of the airfoil must be reduced and a certain number of constraints must be satisfied. The constraints are enforced on the lift coefficient c_l and on geometric quantities such as the relative maximum thickness $(t/c)_{\max}$, the trailing edge angle θ, and the radii of curvature at the nose, r_u and r_l (see Section 5). Only for one test case, a constraint is also imposed on the pitching moment coefficient c_m.

The lift constraint is considered as an equality, necessary in order not to reduce the drag coefficient at the expense of the lift. The relative maximum thickness constraint, considered as inequality, is necessary for avoiding the section to become too thin. The other constraints, on the nose radius and on the trailing edge angle, are included to ensure the feasibility of the shape. For instance, very sharp nose and very thin trailing edge are not desirable from a point of view of manufacturability as well as off-design conditions. Moreover, apart from real-life issues, unfeasible shapes can cause the break-down of the optimization process.

If bounds on the design variables are included, the problem is a bound-constrained optimization which, in general, can be formally stated as

$$\begin{aligned} \min \quad & f(\mathbf{x}), \\ & g_j(\mathbf{x}) \leqslant 0 \quad (j = 1, n_g), \\ & h_k(\mathbf{x}) = 0 \quad (k = 1, n_h), \\ & \mathbf{x}_\mathrm{L} \leqslant \mathbf{x} \leqslant \mathbf{x}_\mathrm{U}, \end{aligned} \tag{25}$$

where $\mathbf{x} = [\alpha_1, \alpha_2, \ldots, \alpha_N]$ is the design vector which contains the shape parameters. The objective function is the scaled drag coefficient, $f = c_\mathrm{d}/c_{\mathrm{d}R}$ with $c_{\mathrm{d}R}$ indicating a reference value (in the following the subscript R will always indicate reference values). The thickness constraint is defined as $g_1 = 1 - (t/c)_{\max}/(t/c)_{\max R}$, the upper and lower nose radius constraints are $g_2 = 1 - r_\mathrm{u}/r_{\mathrm{u}R}$ and $g_3 = 1 - r_\mathrm{l}/r_{\mathrm{l}R}$ whereas the trailing edge angle constraint is $g_4 = 1 - \theta/\theta_R$. The lift constraint is defined as $h = c_\mathrm{l}/c_{\mathrm{l}R} - 1$.

The reference values for the constraints define the feasible design space. For the objective function, the reference value is only used for the purpose of scaling. In the following, the reference values are considered as initial values multiplied by a constant (the initial values are indicated with the subscript 0).

6.2. Optimization algorithms

An easy way to handle the constraints is to include them in the objective function as penalty terms with the clear advantage of being able to use unconstrained optimization techniques. However, the accuracy of this method is known to be poor and, moreover, it can lead to ill-conditioning of the optimization problem [39].

Optimization algorithms that are capable of dealing directly with the constraints are more appropriate. Two algorithms of this kind are employed here. One is the Sequential Quadratic Programming (SQP) algorithm, which uses second-order information that is obtained by updating the Hessian matrix of the Lagrangian with a BFGS formula. The other one is a Sequential Linear Programming (SLP) algorithm, known as the method of centers, which has been investigated recently by the authors for shape optimization purposes [10]. The algorithm uses only first-order information and is based upon linearization of the non-linear optimization problem defined in Eq. (25). Linear programming algorithms, such as the Simplex method, are then used to solve the linearized problem.

6.3. Approximation in the discrete adjoint

Approximations can be introduced in the differentiation in order to simplify the implementation of the discrete adjoint. These approximations imply that the adjoint equation, the first equation in (9), is solved with an approximate Jacobian rather than the exact Jacobian. Three approximations are considered in this work: (i) The first approximation is obtained by neglecting the differentiation of the limiter in the reconstruction operator, which is described in detail in Eq. (A.12). In practice, it means that approximations are introduced in the matrices $\partial \mathbf{U}_L/\partial \mathbf{U}$ and $\partial \mathbf{U}_R/\partial \mathbf{U}$ of Eq. (12). The simplification is appreciable since the limiter implemented here requires a construction phase which is quite involved compared to that of mono-dimensional limiters [4]. (ii) The second approximation is obtained by neglecting the differentiation of the Jacobian matrix in the Roe flux, i.e., it uses Eq. (21) instead of Eq. (A.14). In this case approximations are also introduced in the matrices $\partial \mathbf{H}/\partial \mathbf{U}_L$ and $\partial \mathbf{H}/\partial \mathbf{U}_R$ of Eq. (12). This approximation saves a lot of human work [3]. (iii) The third approximation is that obtained by ignoring the complete reconstruction operator, which makes the implementation of the adjoint trivial. In fact, the Jacobian is identical to the first-order approximate Jacobian already used for the implicit time stepping. In practice, $[\partial \widetilde{\mathbf{R}}/\partial \mathbf{U}]^{\mathrm{T}} \mathbf{\Lambda}_J = [\partial J/\partial \mathbf{U}]^{\mathrm{T}}$ is solved instead of the adjoint equation defined in Eq. (9).

The price to pay for these simplifications is a detrimental effect on the accuracy of the computed gradient. This aspect has been treated extensively in the literature [2,29,18,27] and, therefore, detailed results in terms of gradient accuracy for the numerical cases presented here are not shown. It is only mentioned that when the first and the second approximations are used, the gradient shows an error of 0.1–2.5% compared to that of the exact adjoint code. The error increases to a percentage of 10–30% when the third approximation is used.

To establish whether an approximation in the adjoint is acceptable, the error in the gradient is probably not the best indication. It seems more appropriate to consider the effect which the gradient has on the behavior of the optimization process. This aspect has received some attention only recently [12,9]. In the next section, optimization results are presented, obtained by using the exact and the approximate adjoint codes. There, it is possible to validate the effectiveness of these codes by directly looking at the solutions of the optimization problem.

7. Numerical results

The computations presented below have been performed on an unstructured mesh of triangles, see Fig. 5a, with 12,161 nodes. In order to use the mesh for all test cases, the spring analogy of Section 5.2 has been applied to the original mesh. For the transonic cases, to make sure that the mesh was capable of capturing weak shocks, a finer mesh of 30,092 nodes has been used to verify the optimization results, i.e., to check that the shock-free pressure distributions obtained on the first mesh after optimization are also obtained on the finer mesh. It is not shown, but for all cases the verification was successful.

In terms of settings, the convergence of the flow/adjoint solver is stopped when the residual norm has been reduced 6 orders of magnitude. For the optimization algorithms, a tolerance of 10^{-5} is used for the objective, the constraints as well as for the changes in design variables.

The reference values for the lift and the drag coefficient are always their initial values, $c_{lR} = c_{l0}$ and $c_{dR} = c_{d0}$. In fact, the lift coefficient is kept constant and the objective function, the drag coefficient, is scaled to unity at the beginning of the optimization. All other reference values can be extrapolated from Table 1. The table shows the factors with which the initial values have to be multiplied in order to obtain the reference values. For instance, if for θ_R the factor is 0.9, it means that $\theta_R = 0.9\theta_0$. The consequence of such a choice is that θ can only decrease 10% of its initial value.

7.1. NACA64A410 at $M_\infty = 0.75$ and $\alpha = 0°$

This airfoil is optimized with the SQP algorithm. The initial lift and drag coefficient are $c_{l0} = 0.6419$ and $c_{d0} = 0.0157$. The pressure contours for the original airfoil are shown in Fig. 5b. The airfoils optimized using the exact and the first two approximations are shock-free, see Figs. 5c–e. They achieve almost the

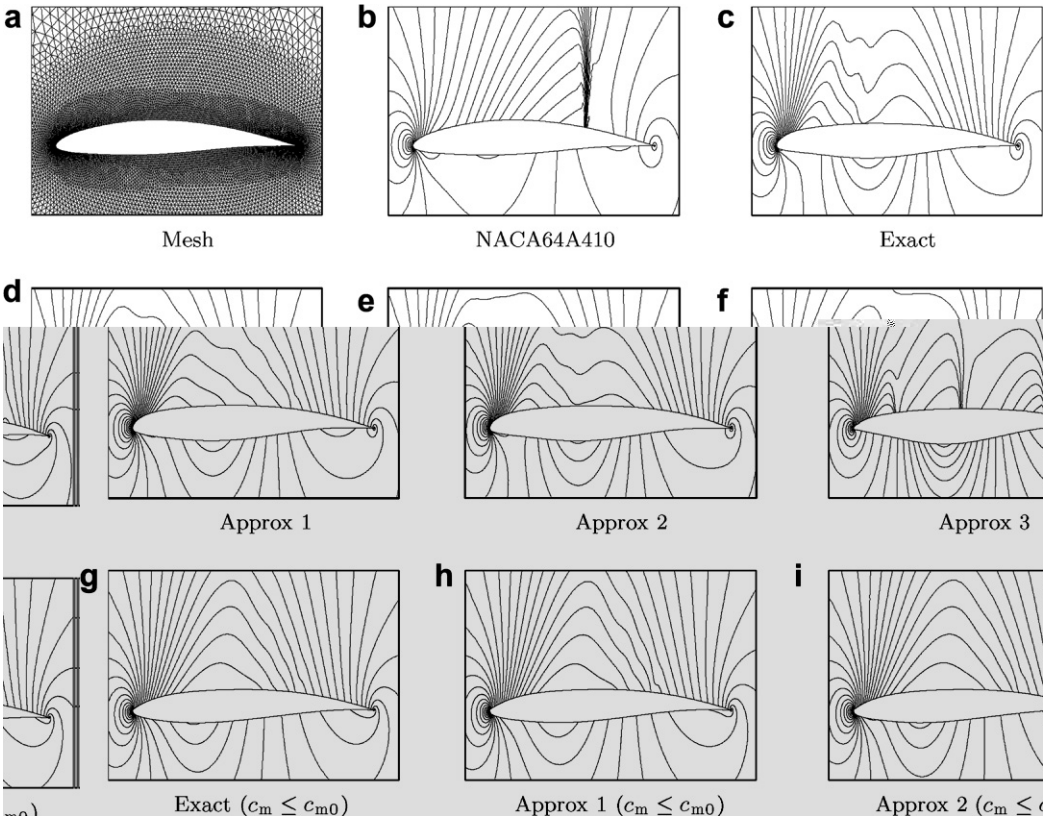


Fig. 5. Optimization of the NACA64A410 airfoil at $M_\infty = 0.75$ and $\alpha = 0°$.

Table 1
Factors with which initial values have to be multiplied to obtain reference values

	$(t/c)_{\max R}$	r_{uR}	r_{lR}	θ_R
NACA64A410	1.0	0.9	0.9	0.9
NACA64A410 ($c_m \leqslant c_{m0}$)	1.0	0.5	0.5	0.5
RAE2822	1.0	0.7	0.7	0.9
NACA0012 ($M_\infty = 0.75$)	1.0	0.9	0.9	0.9
NACA0012 ($M_\infty = 1.5$)	0.5	0.1	0.1	0.1

same reduction of 89.5% in the objective function (differences are less than 1%) for which they need 19, 17 and 34 adjoint and 49, 44 and 86 flow solutions, respectively. As can be seen from the contour plots, the three airfoils are different. A better view of these differences is given in Fig. 6a where only the geometries of the airfoil are shown. Compared to the airfoil obtained from the exact adjoint, the other two airfoils show a maximum difference in y-coordinates of 5.6×10^{-3} and 2×10^{-3}, respectively. As can be seen from Table 2 they all satisfy the design problem accurately. The airfoil obtained using the third approximation, see Fig. 5f, shows three weak shocks on the upper side; its optimization has stalled. The objective function has achieved a reduction of 87.6%. As can be seen from Table 2, the constraints g_2 and g_4 for this test case are slightly violated.

7.2. NACA64A410 at $M_\infty = 0.75$ and $\alpha = 0°$ with c_m constraint

The previous airfoil has experienced a reduction of almost 17% of the pitching moment coefficient with respect to its initial value, $c_{m0} = -0.1697$. In order to avoid the pitching moment to decrease, an additional

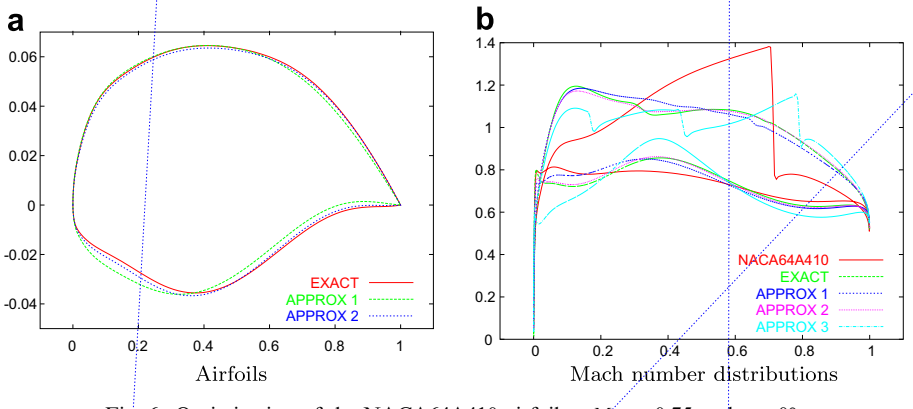


Fig. 6. Optimization of the NACA64A410 airfoil at $M_\infty = 0.75$ and $\alpha = 0°$.

Table 2
Constraint values for the NACA64A410 case of Section 7.1

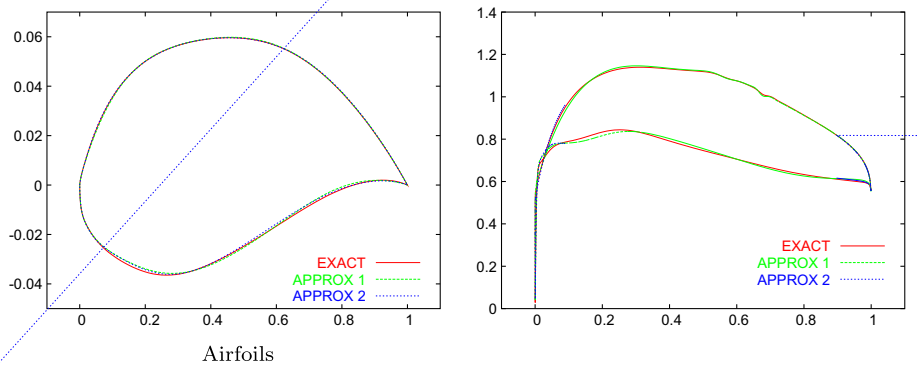
	h	g_1	g_2	g_3	g_4
EXACT	-9.9×10^{-6}	0	-7.3×10^{-1}	-1.3×10^{-1}	-3.5×10^{-1}
APPROX 1	4.6×10^{-7}	0	-7.8×10^{-1}	-1.7×10^{-1}	-3.2×10^{-2}
APPROX 2	7.3×10^{-6}	0	-7.0×10^{-1}	-4.1×10^{-7}	-3.0×10^{-1}
APPROX 3	1.2×10^{-4}	0	8.5×10^{-5}	-3.2×10^{-5}	6.1×10^{-6}

constraint, $g_5 = c_m/c_{m0} - 1$, is imposed (only for this case) so that $c_m \leqslant c_{m0}$. As can be seen from Table 1, the constraints on the nose radii and on the trailing edge angle are chosen to be less strict than for the previous case.

The airfoils optimized using the exact and the first two approximations are shock-free, see Figs. 5g, h and i, respectively. They need 24, 22 and 19 adjoint and 51, 63 and 71 flow solutions, respectively. Compared to the case with no pitching moment constraint, the differences in y-coordinates between the two approximations and the exact code are smaller, 1.3×10^{-3} and 1×10^{-3}, respectively, see Fig. 7a. The effect of the pitching moment constraint on the Mach number distribution is evident by comparing Fig. 7b with the previous case, Fig. 6b.

7.3. *RAE2822 at $M_\infty = 0.73$ and $\alpha = 2°$*

This airfoil is optimized with the SQP algorithm. As can be seen in Fig. 8b, a shock is present on the upper surface of the airfoil. The initial lift and drag coefficient are $c_{l0} = 0.8386$ and $c_{dR} = c_{d0} = 0.008$, respectively.



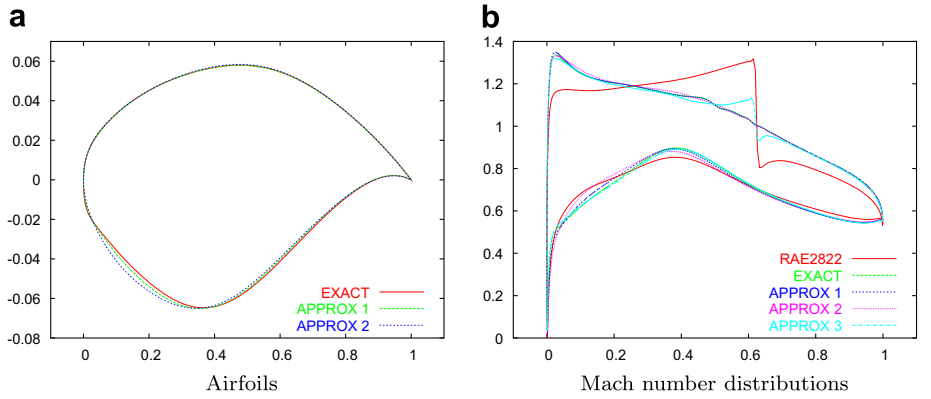


Fig. 8. Optimization of the RAE2822 airfoil at $M_\infty = 0.73$ and $\alpha = 2°$.

As before, the airfoils obtained from the exact and the first two approximate adjoint codes are shock-free, see Fig. 8b. They achieve almost the same reduction in the objective function, 69%, for which they need 15, 20 and 10 adjoint and 39, 62 and 31 flow solutions, respectively. The geometry of the three airfoils is different, see Fig. 8a, especially for the first 40% of the chord on the lower side. Maximum differences in y-coordinates are of the order of 10^{-3}. The optimization stalled when the third approximation was used and, see Fig. 8b, the airfoil still exhibits a weak shock. The objective function reached 67% reduction.

7.4. NACA0012 at $M_\infty = 0.75$ and $\alpha = 2°$

The airfoil is optimized with the SLP algorithm. The initial lift and drag coefficient are $c_{l0} = 0.4225$ and $c_{d0} = 0.0125$. The NACA0012 airfoil exhibits a strong shock on the upper side, see Fig. 9b, almost at half of the chord.

The SLP algorithm seems to be insensitive to the approximations since almost identical shock-free airfoils are obtained when using the exact and the first two approximations. The airfoils can be considered to be almost identical, see Fig. 9a, since the maximum difference between the y-coordinates is of the order of 10^{-4}, one order of magnitude less than in the previous case. A reduction of 92.8% is achieved in the objective function which takes 40 flow/adjoint solutions for the exact code and 42 for the two approximate codes. When the third approximation is used, also the SLP algorithm stalls and the resulting airfoil exhibits a shock on the upper side, see Fig. 9b. The objective function reduced with 90.1%.

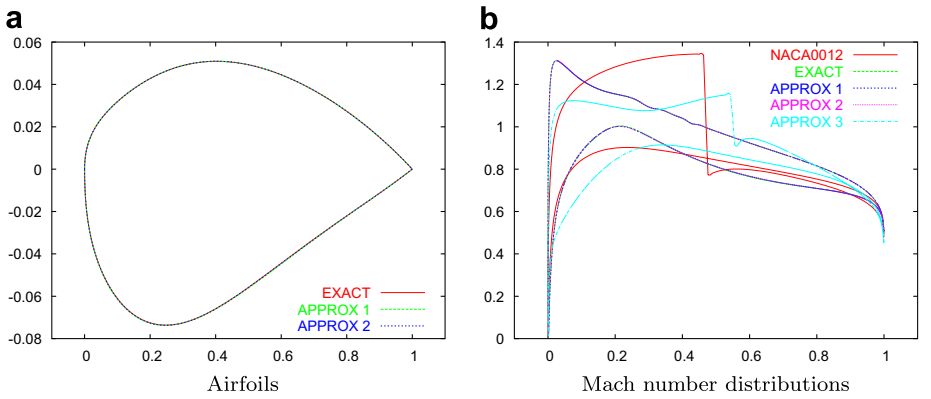


Fig. 9. Optimization of the NACA0012 airfoil at $M_\infty = 0.75$ and $\alpha = 2°$.

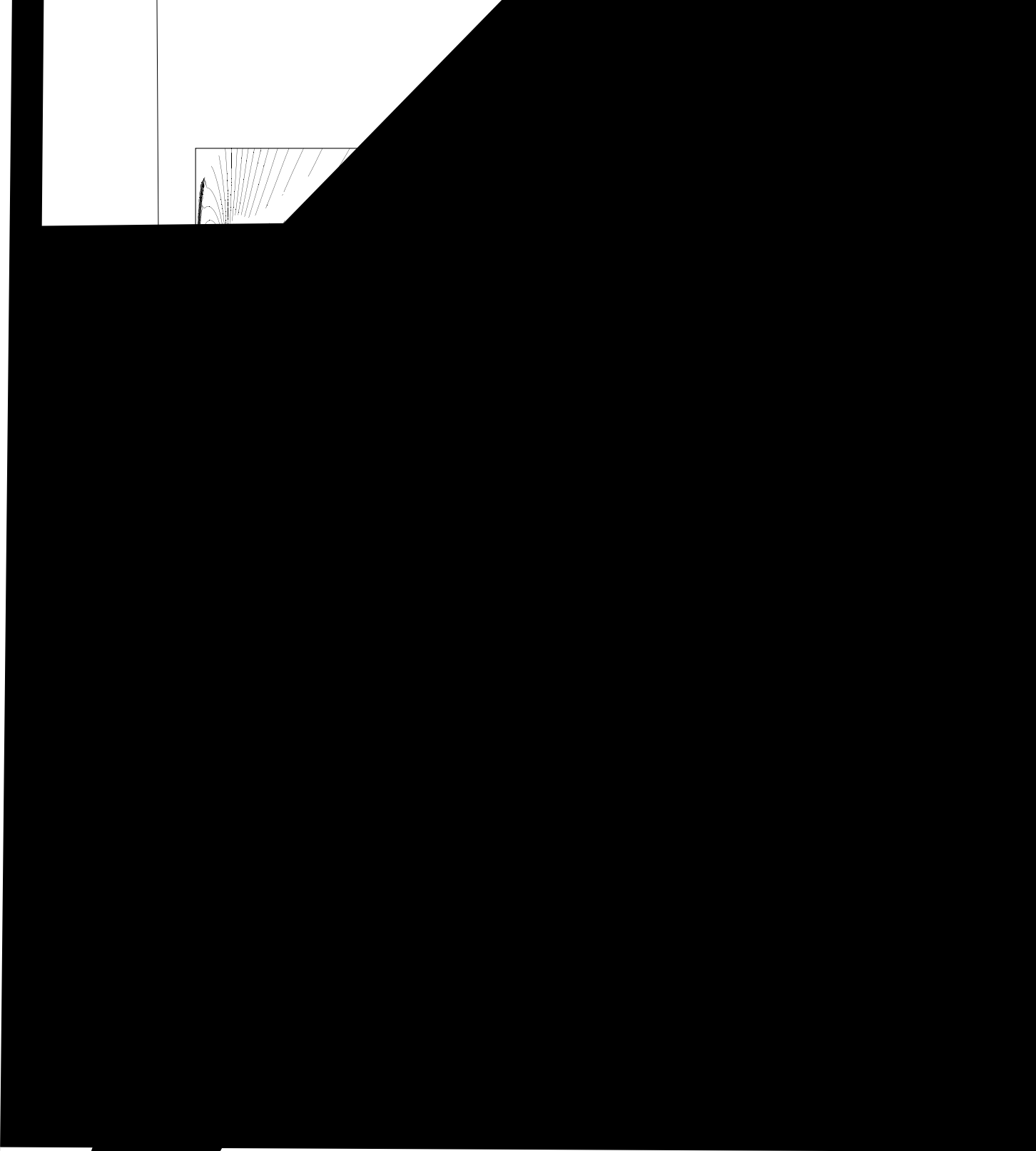


Fig. 11. Optimization

The results depicted in Figs. 10 and 11 have been obtained by the exact adjoint code. Although not shown, results of the approximate adjoint codes follow the same trend as observed in the previous case with the SLP algorithm, i.e., the y-coordinates of the airfoils obtained with approximate codes have very small differences, of the order of 10^{-4} compared to the one in Fig. 11b.

8. Conclusions

The exact discrete adjoint of a finite-volume formulation for the Euler equations in two dimensions has been implemented and tested. The implicit time stepping originally employed in the flow solver has been adapted for the adjoint solver. For a single adjoint solution it requires one and half times the effort of the flow solution. 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 appreciable time saving.

An optimization framework coupling the flow and the adjoint solver to the shape parameterization and to optimization algorithms, suitable for constrained optimization, has been implemented. The effectiveness and accuracy of this framework has been demonstrated on transonic and supersonic test cases involving several constraints. The shape parameterization provides a good representation of the design space and the capability to accommodate large changes in shape.

Different approximations in the discrete adjoint, which have the potential of providing appreciable simplifications in the implementation, have been considered. The effect of these approximations has been investigated directly on the optimization test cases. It appears that first-order accurate approximations are not effective since for all the test cases considered with these, the optimization stalled. When the limiter is neglected, or when the numerical flux is approximated, it appears that the codes are as effective as the exact adjoint code, i.e., the optimization produces optimal airfoils that satisfy the design problem. However, especially when the SQP algorithm is used, it also appears that these optimal airfoils show appreciable differences in shape. From an engineering point of view, the latter aspect is not an issue since all these designs are satisfactory. In future work, from a more theoretical point of view more light could be shed on this.

Acknowledgements

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.

Appendix A. Derivation of the exact discrete adjoint

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

$$\mathbf{Z} = \frac{\partial \mathbf{R}}{\partial \mathbf{U}}\mathbf{P}, \quad \overline{\mathbf{Z}} = \frac{\partial \mathbf{R}^{\mathrm{T}}}{\partial \mathbf{U}}\mathbf{P}. \tag{A.1}$$

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 ∂V_{ik},

$$\mathbf{r}_i = \sum_{k=1}^{N_i} \mathbf{\Phi}(\hat{\mathbf{u}}_i, \hat{\mathbf{u}}_k, \mathbf{n}_{ik}) = \sum_{k=1}^{N_i} \widehat{\mathbf{\Phi}}_{ik}. \tag{A.2}$$

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

For a linear reconstruction a distance-one stencil is used. As can be seen from Fig. A.1, the stencil $\mathcal{N}_i$ of a node i includes the distance-one neighbors of such a node as well as the node itself. Therefore, the reconstructed variable at node i has a dependence $\hat{\mathbf{u}}_i = \hat{\mathbf{u}}_i(\mathbf{u}_k; k \in \mathcal{N}_i)$.

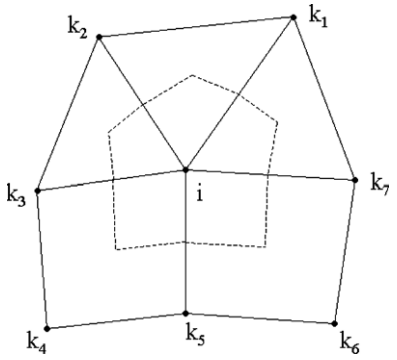


Fig. A.1. Median dual around node i. The number of distance-one neighbors is $N_i = 5$ and the stencil is $\mathscr{N}_i = [i, k_1, k_2, k_3, k_5, k_7]$.

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 added in the left node i and subtracted in the right node j:

$$\begin{aligned} \mathbf{r}_i &= \mathbf{r}_i + \widehat{\mathbf{\Phi}}_{ij}, \\ \mathbf{r}_j &= \mathbf{r}_j - \widehat{\mathbf{\Phi}}_{ij}, \quad (ij = 1, E). \end{aligned} \tag{A.3}$$

This loop can be modified to perform the assembly of the two matrix–vector products given in Eq. (A.1). In fact, each component i of the first matrix–vector product in Eq. (A.1) is given by:

$$\mathbf{z}_i = \sum_{k=1}^{N} \frac{\partial \mathbf{r}_i}{\partial \mathbf{u}_k} \mathbf{p}_k. \tag{A.4}$$

Differentiating Eq. (A.3) with respect to $\mathbf{u}_k$, multiplying by $\mathbf{p}_k$ and adding an additional internal loop on the nodes gives:

$$\begin{aligned} \frac{\partial \mathbf{r}_i}{\partial \mathbf{u}_k} \mathbf{p}_k &= \frac{\partial \mathbf{r}_i}{\partial \mathbf{u}_k} \mathbf{p}_k + \frac{\partial \widehat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k} \mathbf{p}_k, \\ \frac{\partial \mathbf{r}_j}{\partial \mathbf{u}_k} \mathbf{p}_k &= \frac{\partial \mathbf{r}_j}{\partial \mathbf{u}_k} \mathbf{p}_k - \frac{\partial \widehat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k} \mathbf{p}_k, \quad (ij = 1, E;\ k = 1, N), \end{aligned} \tag{A.5}$$

where according to Eq. (A.4) 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. (A.5) can be rewritten as

$$\begin{aligned} \mathbf{z}_i &= \mathbf{z}_i + \frac{\partial \widehat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k} \mathbf{p}_k, \\ \mathbf{z}_j &= \mathbf{z}_j - \frac{\partial \widehat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k} \mathbf{p}_k, \quad (ij = 1, E;\ k = 1, N). \end{aligned} \tag{A.6}$$

As indicated in Eq. (A.2) 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 stencils of the two nodes since $\hat{\mathbf{u}}_i = \hat{\mathbf{u}}_i(\mathbf{u}_k; k \in \mathscr{N}_i)$ and $\hat{\mathbf{u}}_j = \hat{\mathbf{u}}_j(\mathbf{u}_k; k \in \mathscr{N}_j)$.

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 ($\partial \widehat{\mathbf{\Phi}}_{ij} / \partial \mathbf{u}_k \neq \mathbf{0}; k \in \mathscr{N}_i \cup \mathscr{N}_j$). As a consequence, in Eq. (A.6) the inner loop on the nodes can be limited to a summation on both stencil $\mathscr{N}_i$ and $\mathscr{N}_j$ to give:

$$\begin{aligned}\mathbf{z}_i &= \mathbf{z}_i + \sum_{k\in\mathscr{N}_i} \frac{\partial \widehat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k}\mathbf{p}_k + \sum_{k\in\mathscr{N}_j} \frac{\partial \widehat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k}\mathbf{p}_k,\\ \mathbf{z}_j &= \mathbf{z}_j - \sum_{k\in\mathscr{N}_i} \frac{\partial \widehat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k}\mathbf{p}_k - \sum_{k\in\mathscr{N}_j} \frac{\partial \widehat{\mathbf{\Phi}}_{ij}}{\partial \mathbf{u}_k}\mathbf{p}_k, \quad (ij = 1, E).\end{aligned} \tag{A.7}$$

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:

$$\begin{aligned}\mathbf{z}_i &= \mathbf{z}_i + \frac{\partial \widehat{\mathbf{\Phi}}_{ij}}{\partial \hat{\mathbf{u}}_i} \sum_{k\in\mathscr{N}_i} \left.\frac{\partial \hat{\mathbf{u}}_i}{\partial \mathbf{u}_k}\right|_{ij} \mathbf{p}_k + \frac{\partial \widehat{\mathbf{\Phi}}_{ij}}{\partial \hat{\mathbf{u}}_j} \sum_{k\in\mathscr{N}_j} \left.\frac{\partial \hat{\mathbf{u}}_j}{\partial \mathbf{u}_k}\right|_{ij} \mathbf{p}_k,\\ \mathbf{z}_j &= \mathbf{z}_j - \frac{\partial \widehat{\mathbf{\Phi}}_{ij}}{\partial \hat{\mathbf{u}}_i} \sum_{k\in\mathscr{N}_i} \left.\frac{\partial \hat{\mathbf{u}}_i}{\partial \mathbf{u}_k}\right|_{ij} \mathbf{p}_k - \frac{\partial \widehat{\mathbf{\Phi}}_{ij}}{\partial \hat{\mathbf{u}}_j} \sum_{k\in\mathscr{N}_j} \left.\frac{\partial \hat{\mathbf{u}}_j}{\partial \mathbf{u}_k}\right|_{ij} \mathbf{p}_k, \qquad (ij = 1, E).\end{aligned} \tag{A.8}$$

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. (A.8) shows that it is equivalent to Eq. (11). The edge-based assembly of the transposed Jacobian-vector product $\mathbf{Z} = [\partial\mathbf{R}/\partial\mathbf{U}]^{\mathrm{T}}\mathbf{P}$ can be derived from Eq. (A.8) keeping in mind also the matrix form given in Eqs. (11) and (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 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 becomes a scattering on the stencil nodes:

$$\begin{aligned}\bar{\mathbf{z}}_p &= \bar{\mathbf{z}}_p + \left.\frac{\partial \hat{\mathbf{u}}_i}{\partial \mathbf{u}_p}\right|_{ij}^{\mathrm{T}} \frac{\partial \widehat{\mathbf{\Phi}}_{ij}}{\partial \hat{\mathbf{u}}_i}^{\mathrm{T}} (\mathbf{p}_i - \mathbf{p}_j), \quad p \in \mathscr{N}_i,\\ \bar{\mathbf{z}}_q &= \bar{\mathbf{z}}_q + \left.\frac{\partial \hat{\mathbf{u}}_j}{\partial \mathbf{u}_q}\right|_{ij}^{\mathrm{T}} \frac{\partial \widehat{\mathbf{\Phi}}_{ij}}{\partial \hat{\mathbf{u}}_j}^{\mathrm{T}} (\mathbf{p}_i - \mathbf{p}_j), \quad q \in \mathscr{N}_j, \quad (ij = 1, E).\end{aligned} \tag{A.9}$$

This assembly is equivalent to the matrix form given in Eq. (12). The assembly of both loops in Eqs. (A.9) and (A.8) involves, for each edge, the distance-one neighbors of the two nodes that share the edge. Thus, in order to perform the assembly in one pass, a pointer linking each node with its distance-one neighbors must be made available. In the present work, such a pointer is available from the solution of the linear system, see Section 4.2. Alternatively, a two-pass assembly of both loops may be performed using only the edge-based data structure of the flow solver, i.e., the edge pointer to the left and right nodes. The latter approach would be more efficient since the number of operations are the minimum necessary.

The reconstruction contribution is as follows:

$$\frac{\partial \hat{\mathbf{u}}_i}{\partial \mathbf{u}_k} = \frac{\partial \hat{\mathbf{u}}_i}{\partial \hat{\mathbf{v}}_i}\frac{\partial \hat{\mathbf{v}}_i}{\partial \mathbf{v}_k}\frac{\partial \mathbf{v}_k}{\partial \mathbf{u}_k}, \quad \frac{\partial \hat{\mathbf{v}}_i}{\partial \mathbf{v}_k} = \begin{bmatrix} \partial\hat{\rho}_i/\partial\rho_k & 0 & 0 & 0\\ 0 & \partial\hat{u}_i/\partial u_k & 0 & 0\\ 0 & 0 & \partial\hat{v}_i/\partial v_k & 0\\ 0 & 0 & 0 & \partial\hat{p}_i/\partial p_k \end{bmatrix}, \tag{A.10}$$

where the first and third matrices are transformation matrices 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

$$\hat{v}_i = v_i + \frac{\sigma_i}{2}\nabla v_i^{\mathrm{T}}(\mathbf{x}_j - \mathbf{x}_i), \tag{A.11}$$

differentiation with respect to v_k gives

$$\frac{\partial \hat{v}_i}{\partial v_k} = \delta_{ik} + \frac{1}{2}\frac{\partial \sigma_i}{\partial v_k}\nabla v_i^{\mathrm{T}}(\mathbf{x}_j - \mathbf{x}_i) + \frac{\sigma_i}{2}\frac{\partial \nabla v_i}{\partial v_k}^{\mathrm{T}}(\mathbf{x}_j - \mathbf{x}_i), \tag{A.12}$$

where δ_{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 weighted least-squares gradient:

$$\nabla v_i = \frac{1}{2V_i}\sum_{k=1}^{N_i}(v_i + v_k)\mathbf{n}_{ik}, \quad \frac{\partial \nabla v_i}{\partial v_k} = \begin{cases} \frac{1}{2V_i}\mathbf{n}_{ik}, & i \neq k, \\ \frac{1}{2V_i}\sum_{k=1}^{N_i}\mathbf{n}_{ik}, & i = k, \end{cases} \tag{A.13}$$

where V_i is the volume of the cell around node i. 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. For space reasons, this differentiation is not given here.

For the contribution of the numerical flux, the exact differentiation of Roe's approximate Riemann solver is available from previous work [3]:

$$\begin{aligned} \frac{\partial \mathbf{\Phi}_{ij}}{\partial \mathbf{u}_i} &= \frac{1}{2}(\mathbf{A}(\mathbf{u}_i, \mathbf{n}_{ij}) + |\mathbf{A}(\tilde{\mathbf{u}}_{ij}, \mathbf{n}_{ij})|) + (\mathbf{M}_1\mathbf{M}_2 + \mathbf{M}_3)\mathbf{M}_4, \\ \frac{\partial \mathbf{\Phi}_{ij}}{\partial \mathbf{u}_j} &= \frac{1}{2}(\mathbf{A}(\mathbf{u}_j, \mathbf{n}_{ij}) - |\mathbf{A}(\tilde{\mathbf{u}}_{ij}, \mathbf{n}_{ij})|) + (\mathbf{M}_1\mathbf{M}_2 + \mathbf{M}_3)\mathbf{M}_5. \end{aligned} \tag{A.14}$$

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 these matrices, $\mathbf{M}_3$, symbolic differentiation is used. This differentiation can also account for the presence of a parabolic entropy fix [15] in the Roe flux. The latter is available in the present work and, apart from ensuring that the numerical dissipation is never zero, it also removes the lack of differentiability caused by the absolute value in the Roe flux. Nevertheless, when the entropy fix is not used, in both the numerical flux and its Jacobian, there are no appreciable differences according to the numerical experiments. It appears that in practice the lack of continuous differentiability is not an issue. For instance, the quadratic convergence shown in Fig. 2b has been obtained without any entropy fix. The differentiation given in Eq. (A.14) can also be used for the linearization of the far-field boundary fluxes computed with flux-vector splitting.

References

[1] O. Amoignon, M. Berggren, Adjoint of a median-dual finite-volume scheme: application to transonic aerodynamic shape optimization, Technical Report 2006-13, Uppsala University, 2006.
[2] W.K. Anderson, D.L. Bonhaus, Airfoil design on unstructured grids for turbulent flows, AIAA J. 37 (1999) 185–191.
[3] T.J. Barth, Analysis of implicit local linearization techniques for upwind and TVD algorithms, AIAA Paper No. 87–0595, 1987.
[4] T.J. Barth, Aspects of unstructured grids and finite-volume solvers for the Euler and Navier–Stokes equations, Lecture Series, vol. 06, Von Karman Institute for Fluid Dynamics, 1991.
[5] T.J. Barth, S.W. Linton, An unstructured mesh Newton solver for compressible fluid flow and its parallel implementation, AIAA Paper No. 95–0221, 1995.
[6] F. Beux, A. Dervieux, Exact-gradient shape optimization of a 2-D Euler flow, Finite Elem. Anal. Des. 12 (1992) 281–302.
[7] G. Carpentieri, A finite volume solver for unstructured meshes, Technical Report, Delft University of Technology, in press.
[8] G. Carpentieri, M.J.L. van Tooren, M. Kelly, R. Cooper, Airfoil optimization using an analytical shape parameterization, CEIAT Paper No. 05–0073, 2005.
[9] G. Carpentieri, M.J.L. van Tooren, B. Koren, Comparison of exact and approximate discrete adjoint for aerodynamic shape optimization, ICCFD 4, 2006.
[10] G. Carpentieri, M.J.L. van Tooren, B. Koren, Aerodynamic shape optimization by means of sequential linear programming techniques, ECCOMAS CFD, 2006.
[11] L.A. Catalano, A. Dadone, V.S.E. Daloiso, G. Mele, Progressive optimization using orthogonal shape functions and efficient finite-difference sensitivities, AIAA Paper No. 2003-3961, 2003.
[12] R.P. Dwight, J. Brezillon, Effect of various approximations of the discrete adjoint on gradient-based optimization, AIAA Paper No. 2006-690, 2006.
[13] M.B. Giles, M.C. Duta, J.-D. Müller, N.A. Pierce, Algorithm developments for discrete adjoint methods, AIAA J. 41 (2003) 198–205.
[14] M.B. Giles, N.A. Pierce, An introduction to the adjoint approach to design, Flow Turbul. Combust. 65 (2000) 393–415.
[15] A. Harten, High resolution schemes for hyperbolic conservation laws, J. Comput. Phys. 49 (1983) 357–393.
[16] A. Jameson, Aerodynamic design via control theory, J. Sci. Comput. 3 (1988) 233–260.

[17] A. Jameson, S. Yoon, Lower-upper implicit schemes with multiple grids for the Euler equations, AIAA J. 25 (1987) 929–935.
[18] C.S. Kim, C. Kim, O.H. Rho, Sensitivity analysis for the Navier–Stokes equations with two-equation turbulence models, AIAA J. 39 (2001) 838–845.
[19] B. Koren, Defect correction and multigrid for an efficient and accurate computation of airfoil flows, J. Comput. Phys. 77 (1988) 183–206.
[20] B. Koren, A robust upwind discretization method for advection, diffusion and source terms, Numerical Methods for Advection-Diffusion Problems, Notes on Numerical Fluid Mechanics, vol. 45, Vieweg, 1993, pp. 117–138.
[21] G. Kuruvila, S. Ta'asan, M.D. Salas, Airfoil design and optimization by the one-shot method, AIAA Paper No. 95–0478, 1995.
[22] H. Luo, J.D. Baum, R. Löhner, A fast matrix-free implicit method for compressible flows on unstructured grids, J. Comput. Phys. 146 (1998) 664–690.
[23] D.J. Mavriplis, On convergence acceleration techniques for unstructured meshes, AIAA Paper No. 98–2966, 1998.
[24] D.J. Mavriplis, Formulation and multigrid solution of the discrete adjoint for optimization problems on unstructured meshes, AIAA Paper No. 05–0319, 2005.
[25] B. Mohammadi, A new optimal shape design procedure for inviscid and viscous turbulent flows, Int. J. Numer. Meth. Fluids 25 (1997) 183–203.
[26] J.-D. Müller, P. Cusdin, On the performance of discrete adjoint CFD codes using Automatic Differentiation, Int. J. Numer. Meth. Fluids 47 (2003) 939–945.
[27] N. Nemec, D.W. Zingg, Newton–Krylov algorithm for aerodynamic design using the Navier–Stokes equations, AIAA J. 40 (2002) 1146–1154.
[28] J.C. Newman III, A.C. Taylor III, R.W. Barnwell, P.A. Newman, G.J.-W. Hou, Overview of sensitivity analysis and shape optimization for complex aerodynamic configurations, J. Aircraft 36 (1999) 87–96.
[29] E.J. Nielsen, W.K. Anderson, Aerodynamic design optimization on unstructured meshes using the Navier–Stokes equations, AIAA J. 37 (1999) 1411–1419.
[30] E.J. Nielsen, J. Lu, M.A. Park, D.L. Darmofal, An implicit, exact dual adjoint solution method for turbulent flows on unstructured grids, Comput. Fluids 33 (2004) 1131–1155.
[31] J. Reuther, Aerodynamic shape optimization for supersonic aircraft, AIAA Paper No. 2002-2838, 2002.
[32] G.M. Robinson, A.J. Keane, Concise orthogonal representation of supercritical airfoils, J. Aircraft 38 (2001) 580–583.
[33] P.L. Roe, Approximate Riemann solvers, parameter vectors, and difference schemes, J. Comput. Phys. 43 (1981) 357–372.
[34] J.A. Samareh, Survey of shape parameterization techniques for high-fidelity multidisciplinary shape optimization, AIAA J. 39 (2001) 877–884.
[35] V. Selmin, L. Formaggia, Unified construction of finite element and finite volume discretizations for compressible flows, Int. J. Numer. Meth. Eng. 39 (1996) 1–32.
[36] J. Shim, K.D. Lee, A. Verhoff, An efficient aerodynamic design method using asymptotic solution of Euler equations, AIAA Paper No. 02–3141, 2002.
[37] <http://tapenade.inria.fr:8080/tapenade/index.jsp> Software Tapenade© INRIA 2002, version 2.0.
[38] J.I. Trépanier, J. Lépine, F. Pépin, An optimized geometric representation for wing profile using NURBS, CASI J. 46 (2000) 12–19.
[39] G.N. Vanderplaats, Numerical Optimization Techniques for Engineering Design, third ed., Vanderplaats Reasearch & Development, Inc., 2001.
[40] V. Venkatakrishnan, Convergence to steady state solutions of the Euler equations on unstructured grids with limiters, J. Comput. Phys. 118 (1995) 120–130.
[41] V. Venkatakrishnan, D.J. Mavriplis, Implicit solvers for unstructured meshes, J. Comput. Phys. 105 (1993) 83–91.
[42] A. Verhoff, D. Stooksberry, A.B. Cain, An efficient approach to optimal aerodynamic design Part 1: analytic geometry and aerodynamic sensitivities, AIAA Paper No. 93–0099, 1993.
[43] P. Wesseling, Principles of Computational Fluid Dynamics, Springer, 2001.