Structural and Multidisciplinary Optimization
https://doi.org/10.1007/s00158-020-02799-7

EDUCATIONAL PAPER

Discrete adjoint aerodynamic shape optimization using symbolic analysis with OpenFEMflow

Ali Elham[1] · Michel J. L. van Tooren[2]

Received: 10 June 2020 / Revised: 25 September 2020 / Accepted: 25 November 2020
© The Author(s) 2021

Abstract
The combination of gradient-based optimization with the adjoint method for sensitivity analysis is a very powerful and popular approach for aerodynamic shape optimization. However, differentiating CFD codes is a time consuming and sometimes a challenging task. Although there are a few open-source adjoint CFD codes available, due to the complexity of the code, they might not be very suitable to be used for educational purposes. An adjoint CFD code is developed to support students for learning adjoint aerodynamic shape optimization as well as developing differentiated CFD codes. To achieve this goal, we used symbolic analysis to develop a discrete adjoint CFD code. The least-squares finite element method is used to solve the compressible Euler equations around airfoils in the transonic regime. The symbolic analysis method is used for exact integration to generate the element stiffness and force matrices. The symbolic analysis is also used to compute the exact derivatives of the residuals with respect to both design variables (e.g., the airfoil geometry) and the state variables (e.g., the flow velocity). Besides, the symbolic analysis allows us to compute the exact Jacobian of the governing equations in a computationally efficient way, which is used for Newton iteration. The code includes a build-in gradient-based optimization algorithm and is released as open-source to be available freely for educational purposes.

Keywords OpenFEMflow · Discrete adjoint · Symbolic analysis · Aerodynamic shape optimization

1 Introduction

Design is an iterative trial and error method, which includes synthesis, analysis, and decision-making steps. These iterative steps try to drive an initial design to the best design by analyzing the initial behavior of the design. Then repeatedly change the design (by changing the values of the parameters of its parametrized description), re-analyze the design, and change the direction of the design based on some criterion that includes the change in analysis results with respect to the previous analysis step. In the modern design approach, such iteration is done using numerical optimization methods. A very popular example of such a design approach is found in aerodynamic shape optimization.

Aerodynamic shape optimization is performed by the combined use of computational fluid dynamics (CFD) and numerical optimization techniques. Such an optimization may include hundreds of design variables. Although applications of heuristic optimization algorithms for aerodynamic shape optimization can be found in literature (Antunes and Azevedo 2014; Elham and van Tooren 2014), gradient-based algorithms are still the most efficient methods for shape optimizations based on high fidelity CFD analysis including a large number of design variables. However, a sensitivity analysis is required for using gradient-based algorithms.

Adjoint methods for sensitivity analysis have been widely used in recent years for aerodynamic shape optimization (Kenway et al. 2019). Both continuous (Palacios et al. 2015; Baysal and Ghayour 2001; Brezillon and Gauger 2004) and discrete adjoint (Mader et al. 2008; Carpentieri et al. 2007) methods have been used to develop CFD codes for

Responsible Editor: Joaquim R. R. A. Martins

This paper has been modified from A. Elham and M.J.L. van Tooren "Aerodynamic Shape Optimization Using Symbolic Sensitivity Analysis," AIAA SciTech conference, January 2017, Grapevine, TX, USA.

✉ Ali Elham
a.elham@tu-braunschweig.de

[1] Institute of Aircraft Design and Lightweight Structures, Technische Universität Braunschweig, Braunschweig, Germany

[2] McNair Center for Aerospace Research and Innovation, University of South Carolina, Columbia, SC, USA

aerodynamic shape optimization. In the continuous adjoint method, the partial differential equations are differentiated first and then discretized to be solved numerically. However, in the discrete adjoint method, the governing equations are first discretized and then differentiated. The computational efficiency of the continuous adjoint method is typically higher than the discrete adjoint; however, it faces some difficulties. In the continuous adjoint first, the flow equations are differentiated and then discretized; therefore, there can be a mismatch between the gradient computed using the continuous adjoint and the gradient of the solution based on the discretized equations. The discrete adjoint method computes the derivatives of the discretized equations, i.e., the equations that are used to numerically solve the governing equations. So there is no mismatch between the gradients and the solution. However, differentiating the discrete equations and implementing it in code needs some considerations. One major issue is the need to compute all the partial derivatives required for discrete adjoint sensitivity analysis.

Kenway et al. (2019) reviewed available methods to compute the partial derivatives. It includes analytical methods, finite differencing, complex-step method, automatic differentiation, and symbolic analysis. Analytical differentiation is computationally the most efficient approach; however, it is not always possible due to the complexity of the equations. Automatic Differentiation (AD) is suggested to ease differentiating computer codes (Grienwank and Walther 2008; Hovland et al. 1998). AD can be used in direct and reverse modes (Grienwank and Walther 2008; Martins and Hwang 2013). The computational cost of the direct mode is the same as finite differencing, as the function has to be evaluated as many times as the number of design variables. However, in the reverse mode, the function is just needed to be evaluated one time independent of how many design variables are defined. In the reverse AD method, all the dependent variables in the code are differentiated with respect to the design variables, and the results are stored in the memory. In the end, the chain rule of differentiation is used to compute the derivative of the objective function with respect to the design variables. Although the application of reverse AD results in the need for higher memory requirement, by a smart application, i.e., limiting the use of AD to small pieces of the code, efficient CFD codes based on AD can be developed (Mader et al. 2008; Carpentieri et al. 2007).

With the growing interest in the application of aerodynamic shape optimization for aircraft as well as automotive design, educating students to learn adjoint-based optimization as well as developing and maintaining such codes is essential. Top-level universities around the world offering education on this topic in courses such as CFD, engineering optimization, or multidisciplinary design optimization. Despite the advantages of the adjoint method for sensitivity analysis, this method is far from intuitive, and in cases like this, learning by doing is often a best practice in engineering education. However, the math and programming complexity is such that implementation is difficult and error-prone; therefore, leading to frustration and distraction of the actual learning objectives, especially when the computational analysis by itself is complex and requires modification to implement the adjoint method to make the design sensitivity analysis efficient.

The authors have experience in developing educational tools for teaching different courses, such as aircraft design and MDO. The student version of the tools Q3D (Mariens et al. 2014) (a code for drag prediction of 3D wings using quasi-three-dimensional analysis), EMWET (Elham et al. 2013) (a code for structural weight estimation of lifting surfaces using the quasi-analytical method), and FEMWET (Elham and van Tooren 2016) (a coupled-adjoint code based on quasi-three-dimensional aerodynamic analysis and equivalent finite beam structure model for aerostructural analysis and optimization of lifting surfaces) have been used in different course taught by the authors in the past 10 years. The authors have a very successful experience in using Q3D and EMWET for teaching MDO as well as advanced aircraft design courses. However, teaching aerostructural optimization using coupled-adjoint sensitivity analysis is more complex. The code FEMWET is based on discrete adjoint augmented by automatic differentiation. But the authors' experience shows there is a need for a simpler and computationally more efficient tool to support the teaching of the adjoint sensitivity analysis method.

The recent development in software like Mathematica, Matlab, Maple, and Mathcad allows symbolic analysis. Using this capability, analysis such as integration or differentiation can be done symbolically. Symbolic analysis has been used for automatic finite element code generation, which has been claimed to result in reducing the complexity and time requirements for finite element code development, as well as reducing the possibilities for error and bug in the program (Korelc and Wriggers 2016; Farrell et al. 2012). The symbolic analysis can also be an alternative method for the analytical differentiation of computer codes. It can result in the same efficiency as analytical differentiation, without the need for tedious, error-prone, task of analytically differentiating complex equations.

The idea of combining symbolic analysis with the discrete adjoint method to develop an efficient, while easy to develop, understand, and maintain CFD code, motivated the authors to develop a new educational tool to support teaching adjoint-based optimization. This paper will show the reader how code development can be structured and simplified using a symbolic definition of variables and basic matrix equations and symbolic manipulation of these equations, including partial differentiation and integration to derive complex computational computer codes. Also, this paper and the associated open source Matlab code, named OpenFEMflow, are intended to guide the reader to all the steps required to parametrize a design, discretize 2D space and 2D flow variable fields, build a CFD solver suitable for the adjoint method, implementation of the adjoint method, and automate mesh adjustments (simple refinement as well as adaptation to flow variable values) to prevent manual convergence studies. For this purpose, the finite element approach will be used to discretize the 2D space as well as the flow field. The approach will be used to optimize single element airfoils for minimizing drag at high velocities (transonic, supersonic speeds). Several example design studies are discussed to illustrate the completeness and correctness of the approach.

2 Aerodynamic shape optimization

An optimization problem is mathematically defined as follows:

$$\begin{array}{lll} \min & f(X) & \\ & X_i & i = 1..n \\ \text{s.t.} & h_j(X) = 0 & j = 1..m \\ & g_k(X) \leq 0 & k = 1..l \\ & X_L \leq X \leq X_U & \end{array} \tag{1}$$

where f is the objective function, i.e., the design figure of merit which should be minimized, X is the vector of design variables, h is the vector of equality, and g is the vector of inequality constraints. The design variables can be bounded between a set of lower values of X_L and upper values X_U.

In an aerodynamic shape optimization, the objective function is usually drag, or a combination of minimizing drag and maximizing lift, e.g., minimizing drag over lift. The constraints are usually defined on some geometrical parameters, such as minimum thickness, and some aerodynamic characteristics such as pitching moment.

An important issue in aerodynamic shape optimization is the method used for parameterizing the shape. Various methods are suggested to mathematically model the outer geometry of 2D and 3D aerodynamic objects. The Free Form Deformation (FFD) (Sederberg and Parry 1986), and the Class Shape Transformation (CST) (Kulfan 2008) are examples of such parameterization methods. The CST method is used in OpenFEMflow for shape parameterization. The CST method is symbolically described as follows:

$$\zeta(\psi) = C_{N2}^{N1}(\psi) \cdot S(\psi) \tag{2}$$

where ψ and ζ are normalized x and y positions respectively. C_{N2}^{N1} is the class function and $S(\psi)$ presents the shape function. The class function expresses the basis class of shapes:

$$C_{N2}^{N1}(\psi) = (\psi)^{N1} (1-\psi)^{N2} \tag{3}$$

$N1$ and $N2$ are equal to 0.5 and 1 respectively for an airfoil with a round leading edge and sharp trailing edge. The shape function is a Bernstein polynomial representation which describes the permutation around this basic shape:

$$S(\psi) = \sum_{i=0}^{p} \bar{P}_i B_{i,p}(\psi) \tag{4}$$

where p is the order of the Bernstein polynomial, $\bar{P}_i$ is the vector of control points and $B_{i,p}(\psi)$ are the Bernstein polynomials of degree p. The values of $\bar{P}_i$ are defined as design variables and referred to as the CST modes.

A typical airfoil shape optimization can be formulated as follows:

$$\begin{array}{lll} \min & C_d(X) & \\ & X = [\text{CST}_i, \alpha] & i = 1..n \\ \text{s.t.} & C_l - C_{l_t} = 0 & \\ & C_{m_t} - C_m \leq 0 & \\ & t_{\max_{\text{init}}} - t_{\max} \leq 0 & \end{array} \tag{5}$$

In this case, it is assumed that the airfoil shape is parameterized using the CST method with n control points (for both upper and lower side of the airfoil). In addition to the CST modes, the angle of attack is also defined as a design variable. This is required to give freedom to the optimizer to change the angle of attack until the constraint on lift is satisfied. The lift coefficient is constrained to be equal to a target (desired) value. The pitching moment is also constrained to be larger (less negative) than a target (minimum) value. And eventually, the airfoil maximum thickness is constrained to be larger than a minimum value. In some cases instead of the maximum thickness, the thickness at some chordwise positions, for example, the locations of the spars, is constrained (both cases are implemented in OpenFEMflow). Such an optimization can be solved based on the Design Structure Matrix (DSM), shown in Fig. 1.

From the DSM, one can observe that a numerical optimization algorithm, a geometry generator, a mesh generator, and a CFD solver (both the primal and adjoint solvers) are the main components of such an optimization framework. The optimizer is sending the values of the

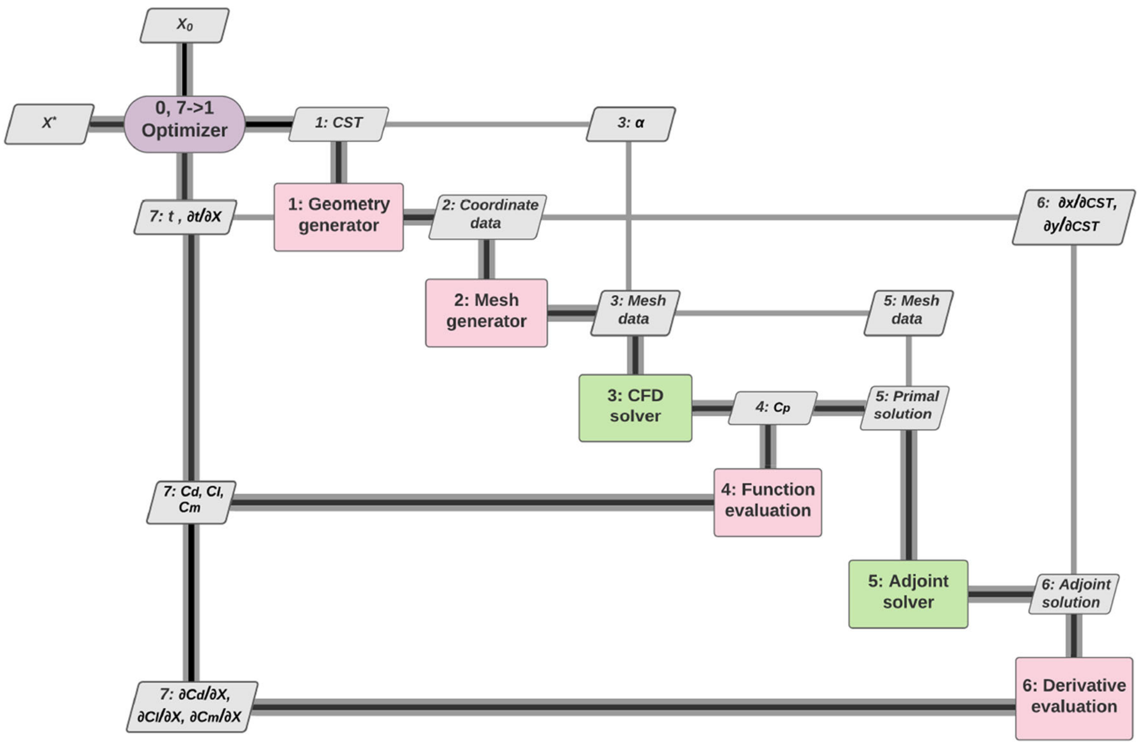

Fig. 1 Design structure matrix for aerodynamic shape optimization

design variables to the geometry generator to create the airfoil geometry and compute the geometrical properties required for the optimizer, e.g., the airfoil maximum thickness. This geometry is sent to a mesh generator to generate the mesh, which is used by the CFD solver to compute the aerodynamic characteristics of the airfoil. This general aerodynamic shape optimization can be used for optimizing any object (2D or 3D), with any optimization algorithm and CFD solver. However, for a specific choice of algorithm and CFD solver, the DSM can be modified for more efficient optimization. For example, instead of generating a new mesh for every new set of shape parameters, an original mesh can be generated once at the beginning and then is deformed every time based on new shape variables.

If a gradient-based algorithm is used for optimization, the derivatives of the objective function, and the constraints with respect to the design variables are required. The paper of Martins and Hwang (2013) reviews different methods available for computing derivatives. As explained earlier, the discrete adjoint method is used in OpenFEMflow for sensitivity analysis. In the following sections of this paper, first, the CFD method used in OpenFEMflow is described, and then the sensitivity analysis method is explained.

3 Flow modelling and numerical analysis method

Since OpenFEMflow is mainly developed for educational purposes, the Euler method is considered to model the flow, which is computationally more efficient than RANS. Two dimensional unsteady compressible Euler equations can be presented in the following matrix form:

$$\frac{dU}{dt} + A_1\frac{dU}{dx} + A_2\frac{dU}{dy} = 0 \tag{6}$$

where $U = [\rho, u, v, p]^T$ is the vector of the primitive variables and the A_1 and A_2 matrices are as follows:

$$A_1 = \begin{bmatrix} u & \rho & 0 & 0 \\ 0 & u & 0 & \rho^{-1} \\ 0 & 0 & u & 0 \\ 0 & \gamma p & 0 & u \end{bmatrix} \tag{7}$$

$$A_2 = \begin{bmatrix} v & 0 & \rho & 0 \\ 0 & v & 0 & 0 \\ 0 & 0 & v & \rho^{-1} \\ 0 & 0 & \gamma p & v \end{bmatrix} \tag{8}$$

where u and v are the velocity components, ρ is the density, p is the pressure, and γ is the specific heat ratio.

Equation (6) can be linearized by setting $A^n = A(U^n)$, i.e., to compute U for the next iteration matrices A_1 and A_2 are constructed using the vector U from the previous iteration. An implicit time differentiation results in the following equation:

$$R = U^{n+1} - U^n + \Delta t A_1^n \frac{\partial U^{n+1}}{\partial x} + \Delta t A_2^n \frac{\partial U^{n+1}}{\partial y} = 0 \quad (9)$$

Different methods can be used to numerically solve this equation. In OpenFEMflow, the least-squares finite element method (LSFEM) is used (Jiang and Carey 1990; Bochev and Gunzburger 2009) to numerically solve Euler equations. Finite element methods and in particular LSFEM in this case, have several advantages for such a problem. The main advantages of LSFEM for computational fluid dynamics are summarized in the book of Jiang (1998). In short, the following advantages are mentioned by Jiang for application of LSFEM in CFD:

1. **Universality**: LSFEM has a unified formulation for numerical solution of all types of differential equations, i.e., elliptic, parabolic, hyperbolic, or mixed.
2. **Efficiency**: LSFEM always leads to symmetric, positive-definite matrices for linear partial differential equations, which can be efficiently solved by matrix free methods.
3. **Robustness**: Due to the mathematical foundation of LSFEM, especial numerical treatments such as unwinding, artificial dissipation, staggered grid or non-equal order elements, artificial compressibility, operator splitting, and operator preconditioning etc. are unnecessary.

Although it has to be noted that, the LSFEM is not a magic solution to all the CFD problems, as it has it is own disadvantages. However, since the main purpose of developing OpenFEMflow is educating students on adjoint-based optimization, the choice of numerical method was not the main concern of the authors.

Using the LSFEM, the L^2-norm of the residual R is minimized, i.e.:

$$\min \quad \Phi = \int_\Omega R^T R \, dxdy \quad (10)$$

In a finite element approach, the primitive variables inside each element are approximated using the so-called shape functions as follows:

$$\tilde{U} = \sum_{i=1}^{nn} N_i U_i \quad (11)$$

where nn is the number of nodes for each element, N_i is the ith shape function and U_i is the value of the primitive variables at node i. Using the least-squares finite element method the following element stiffness and force matrices are obtained:

$$K_{ij}^e = \int_{\Omega_e} (LN_i)^T (LN_j) dxdy \quad (12)$$

$$F_{ij}^e = \int_{\Omega_e} (LN_i)^T U^n dxdy \quad (13)$$

where:

$$LN_i = N_i I + \Delta t \frac{\partial N_i}{\partial x} A_1^n + \Delta t \frac{\partial N_i}{\partial y} A_2^n \quad (14)$$

I is the identity matrix. In finite element analysis, the element matrices are usually computed using numerical integration techniques such as Gauss Quadrature (Krishnamoorthy 1994). This approach requires multiple time calculation of the integral function and also introduces numerical error. In this research, the symbolic analysis technique is used to derive the exact solution of the integrals shown in (12) and (13).

In this research, three nodes linear triangular elements are used, where the shape functions are defined based on the local coordinates L_1, L_2, and L_3 as shown in (15).

$$L_1 = \frac{A_1}{A}$$
$$L_2 = \frac{A_2}{A}$$
$$L_3 = \frac{A_3}{A} \quad (15)$$

where A is the area of the triangle, and A_1, A_2, and A_3 are the areas of the three sub-triangles as shown in Fig. 2. From this figure, one can observe that $A_1 + A_2 + A_3 = A$ and $L_1 + L_2 + L_3 = 1$. The shape functions for this type of element are defined as:

$$N_1 = L_1$$
$$N_2 = L_2$$
$$N_3 = 1 - L_2 - L_2 \quad (16)$$

To derive the equations of K^e and F^e using symbolic analysis, the x and y positions of the element nodes (for 2D elements) as well as the flow variables ρ, u, v, and p are defined as symbols. Matlab symbolic analysis toolbox is used to derive the solution of (12) and (13) symbolically. Lines 1 to 62 of the Matlab code shown in the Appendix are used to derive K^e and F^e.

The resulting symbolic equations for K and F were subsequently hard-coded in a function for computing the element matrices. This function receives the x and y

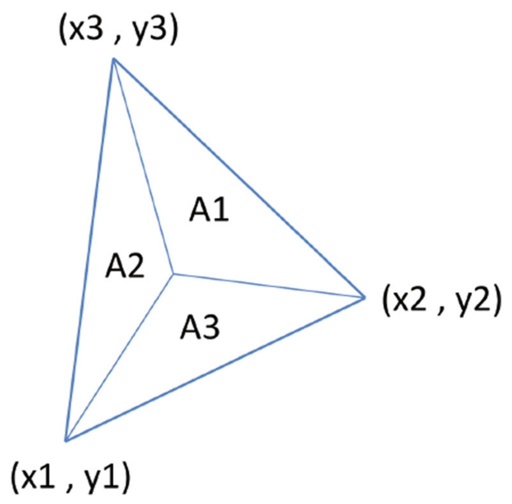

Fig. 2 Triangular element local coordinates

positions of the nodes (x_1, y_1 to x_3, y_3 in Appendix) and the values of the flow variables at each node (ρ_1, u_1, v_1, p_1 to ρ_3, u_3, v_3, p_3 in Appendix) as well as the time step (dt in Appendix) and returns the stiffness and the force matrices for each element. Using the proper assembly, the global stiffness and force matrices are generated.

Two types of boundary conditions are used for airfoil analysis; the far-field boundary condition, where the pressure, velocity, and density are given, and the wall boundary condition, where for inviscid flow the normal velocity to the wall is set to zero. Applying the first type of boundary condition is easy. The values of the variables of the nodes placed on this boundary are defined as fixed degrees of freedom. However, the second type of boundary condition requires some attention. To apply the wall condition, the local angle of the wall at each node, θ, is first determined from the airfoil geometry, see Section 5. A transformation matrix, T is defined based on θ of the nodes on the walls. To compute K and F for the elements with at least one node on the wall, the coordinate of the velocity variables of the node(s) on the wall is rotated in such a way to have the tangential and normal velocity to the wall instead of u and v in x and y directions respectively. This transformation is done using the T matrix. The normal velocity of the nodes on the wall is set to zero and is defined as a fixed degree of freedom.

4 Solving the nonlinear system of equations

Both Picard and Newton methods are used to solve the nonlinear system $R(U) = K(U)U - F(U) = 0$. The Newton method has a larger rate of convergence compared to the Picard method, but it has a smaller radius of convergence (Reddy and Gartling 2001). Therefore, the iteration is started using the Picard method and after a few iterations, the Newton method is used.

Using the Picard iteration method, in this case, is identical to using the Newton method to solve the linearized system $R^{n+1} = K(U^n)U^{n+1} - F(U^n) = 0$, where $\partial R/\partial U$ is equal to $K(U^n)$. In such an approach, first, the stiffness and force matrices are evaluated using the U vector from the previous iteration (or the initial guess at the first iteration). Then ΔU (the change in U) is computed as:

$$\Delta U = -\frac{\partial R}{\partial U}^{-1} R(U) \tag{17}$$

where as mentioned earlier $\partial R/\partial U$ is approximated to be equal to K, and R is evaluated as $R(U) = K(U^n)U^n - F(U^n)$.

The next step is to solve the nonlinear system $R^{n+1} = K(U^{n+1})U^{n+1} - F(U^{n+1}) = 0$ using the Newton method. In this approach, the exact derivative of R with respect to U is required, which includes the partial derivatives of K and F with respect to U. The symbolic analysis is used to derive $\partial R^e/\partial U^e$ as a function of the node position as well as the value of the primitive variables at each node, see lines 66 to 94 of the Appendix. The equations for R^e and $\partial R^e/\partial U^e$ are hard-coded and called to evaluate the residual and its derivatives with respect to the primitive variables for each element. Using this approach R^e is directly evaluated; therefore, the need for calculating and storing the stiffness and force matrices is eliminated. It improves the speed as well as the memory requirement of the code. Using a proper assembly, the values of R and $\partial R/\partial U$ of the whole system are computed.

5 Mesh deformation

The spring analogy is used to deform the mesh. Defining the position of the nodes on the wall using the CST method, the position of the other nodes is determined iteratively. Initially, the displacement of each node is set to zero, except those on the airfoil surface, which are displaced according to the new airfoil shape, and those on the external boundary which should stay fixed. The location of the internal nodes, then is determined iteratively by adding Δx to their current location at each iteration:

$$\Delta x_i^{m+1} = \frac{\sum_{j=1}^{N_i} k_i j \Delta x_j^m}{\sum_{j=1}^{N_i} k_i j} \tag{18}$$

where N_i is the number of the nodes connected to node i, k_{ij} is the inverse of the length of the edge ij, and m is the number of iteration. The iteration stops when the value of Δx becomes lower than a certain tolerance. Using (18), the sensitivity of the position of each node with respect to the CST modes is determined analytically.

6 Adjoint method for sensitivity analysis

Besides computing the exact Jacobian of the residuals, symbolic analysis is also used for sensitivity analysis required for optimization. The derivative of a function of interest I, for example, the drag coefficient, with respect to a design variable X (not to be confused with x position of the nodes) is as follows:

$$\frac{dI}{dX} = \frac{\partial I}{\partial X} + \frac{\partial I}{\partial U}\frac{dU}{dX} \tag{19}$$

In the discrete adjoint approach, the derivatives of the residuals with respect to the design variables are used to eliminate dU/dX from (19):

$$\frac{dR}{dX} = \frac{\partial R}{\partial X} + \frac{\partial R}{\partial U}\frac{dU}{dX} = 0 \tag{20}$$

$$\frac{dU}{dX} = -\frac{\partial R}{\partial U}^{-1}\frac{\partial R}{\partial X} \tag{21}$$

Substituting (21) in (19), dI/dX is calculated as follows:

$$\frac{dI}{dX} = \frac{\partial I}{\partial X} - \lambda^T\frac{\partial R}{\partial X} \tag{22}$$

where the adjoint vector, λ, is calculated from the following equation:

$$\frac{\partial R}{\partial U}^T\lambda = \frac{\partial I}{\partial U} \tag{23}$$

In an airfoil optimization, I can be the lift coefficient C_l, the drag coefficient C_d, and the pitching moment coefficient C_m. The design variables X can be the airfoil geometry as well as the angle of attack α and the free stream Mach number M.

Beside $\partial R/\partial U$, the partial derivatives of R and I with respect to the design variables X and the partial derivative of I with respect to U are required to compute the total derivatives dI/dX. The symbolic analysis is used to derive the partial derivatives of the element residuals with respect to the x and y positions of the element nodes, i.e. $\partial R^e/\partial x^e$ and $\partial R^e/\partial y^e$, see lines 96 to 181 of the code shown in Appendix. These equations were hard-coded in a function that receives the nodes positions as well as the value of the primitive variables at each node and returns $\partial R^e/\partial x^e$ and $\partial R^e/\partial y^e$. Using a proper assembly the partial derivatives of the global residuals with respect to the position of the nodes are computed.

The partial derivative $\partial R/\partial X$ is then computed:

$$\frac{\partial R}{\partial X} = \frac{\partial R}{\partial x}\frac{\partial x}{\partial X} + \frac{\partial R}{\partial y}\frac{\partial y}{\partial X} + \frac{\partial R}{\partial \theta}\frac{\partial \theta}{\partial X} \tag{24}$$

where x and y are the positions of the nodes, θ is the slope of the wall at the position of the node, and X is the vector of CST modes. The terms $\partial R/\partial\theta$ and $\partial\theta/\partial X$ are computed using the transformation matrix T and the airfoil geometry (the CST formulation) respectively.

The angle of attack and the free stream Mach number only affect the values of u and v for the nodes defined as inlet boundary condition directly. Therefore, $\partial U/\partial\alpha$ and $\partial U/\partial M$ are nonzero only for the nodes on the inlet boundary. For these nodes, $\partial U/\partial\alpha$ and $\partial U/\partial M$ are as follows:

$$\frac{\partial U^e}{\partial \alpha} = [0 \quad -Msin(\alpha) \quad Mcos(\alpha) \quad 0]^T \tag{25}$$

$$\frac{\partial U^e}{\partial M} = [0 \quad cos(\alpha) \quad sin(\alpha) \quad 0]^T \tag{26}$$

The partial derivatives of R^e with respect to α and M are derived as follows:

$$\frac{\partial R^e}{\partial \alpha} = \frac{\partial R^e}{\partial U^e}\frac{\partial U^e}{\partial \alpha} \tag{27}$$

$$\frac{\partial R^e}{\partial M} = \frac{\partial R^e}{\partial U^e}\frac{\partial U^e}{\partial M} \tag{28}$$

By a proper assembly, $\partial R/\partial\alpha$ and $\partial R/\partial M$ are computed.

The airfoil aerodynamic coefficients C_l, C_d, and C_m are computed by a proper integration of the pressure over the airfoil surface. The pressure coefficient on the airfoil surface is a function of U of the nodes on the airfoil surface (defined as wall boundary condition) and the free-stream Mach number. The equations of the partial derivatives $\partial C_l/\partial U$, $\partial C_d/\partial U$, and $\partial C_m/\partial U$ for the nodes on the airfoil surface are derived symbolically (manually not by Matlab symbolic analysis toolbox) and hard-coded. These derivatives for the nodes other than those on the airfoil surface are zero. Similar equations for derivatives of the airfoil coefficients with respect to the free stream Mach number are derived. In the same way, the symbolic equations for the partial derivatives of the airfoil aerodynamic coefficients with respect to the x and y positions of the nodes on the surface of the airfoil are derived. Eventually the partial derivatives of I with respect to X are computed based on $\partial I/\partial x$, $\partial I/\partial y$, $\partial x/\partial X$, and $\partial y/\partial X$.

The adjoint equation, (23), is solved using MATLAB mldivide function. This function uses various algorithms, based on the type of the matrix $\partial R/\partial U$. Details about this function are available on Mathworks website.[1]

7 Validation

To validate the code, the results of OpenFEMflow for four different test cases were compared to experimental data from AGARD report (Viviand 1985) and shown in Table 1. Figure 3 shows the pressure contours for the four test cases shown in Table 1. All of these analyses were performed

[1]https://de.mathworks.com/help/matlab/ref/mldivide.html#d122e836649

Table 1 Validation results of OpenFEMflow

			OpenFEMflow		AGARD	
Airfoil	M	α	C_l	C_d	C_l	C_d
RAE2822	0.75	3	1.0103	0.0462	1.1058 ± 0.0322	0.0467 ± 0.0056
NACA0012	0.95	0		0.1070		0.1082 ± 0.0008
NACA0012	1.2	0		0.0938		0.0953 ± 0.0014
NACA0012	1.2	7	0.5091	0.1548	0.5211 ± 0.0142	0.1538 ± 0.0012

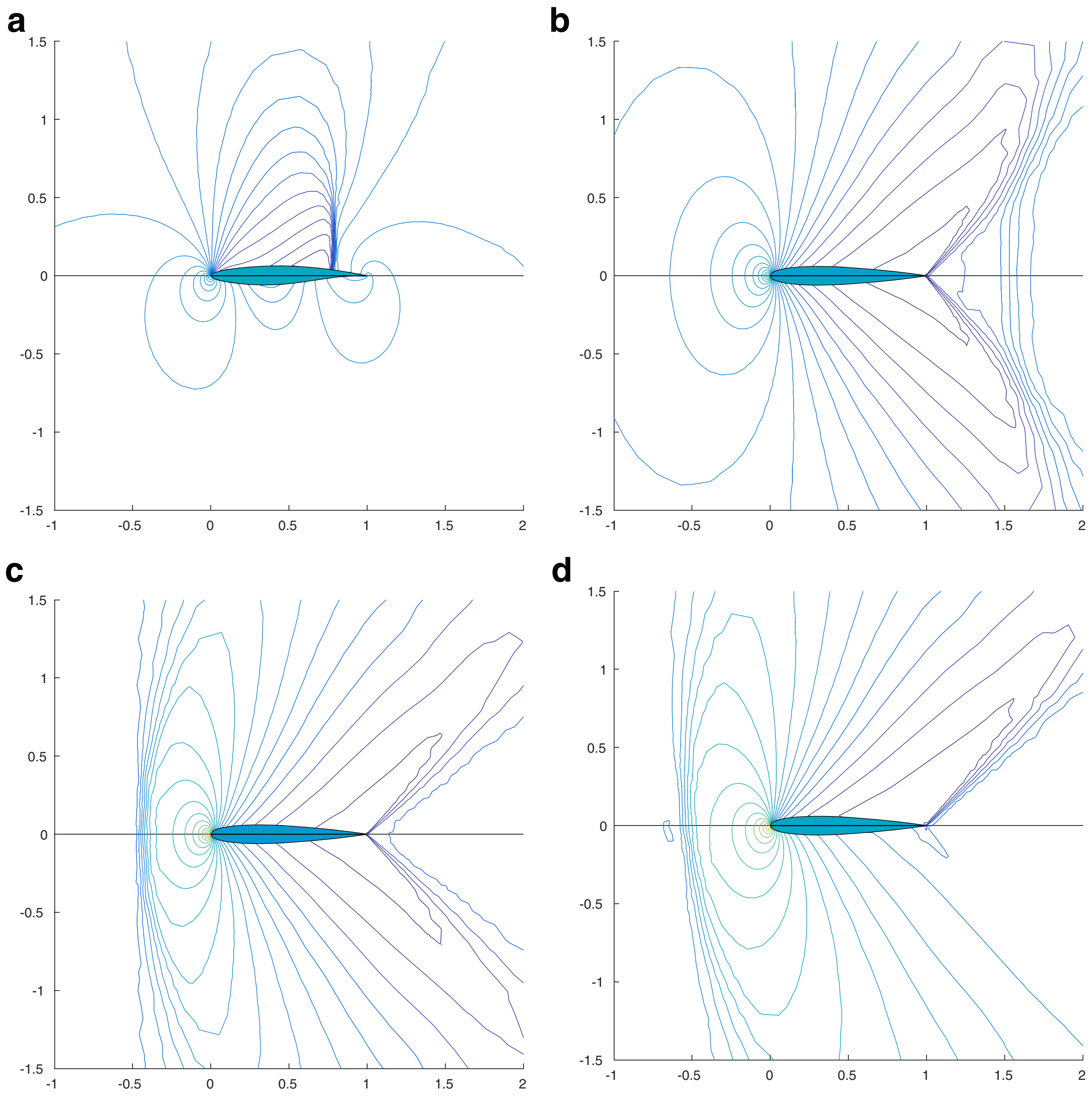

Fig. 3 **a**–**d** Pressure contours for the OpenFEMflow validation test cases

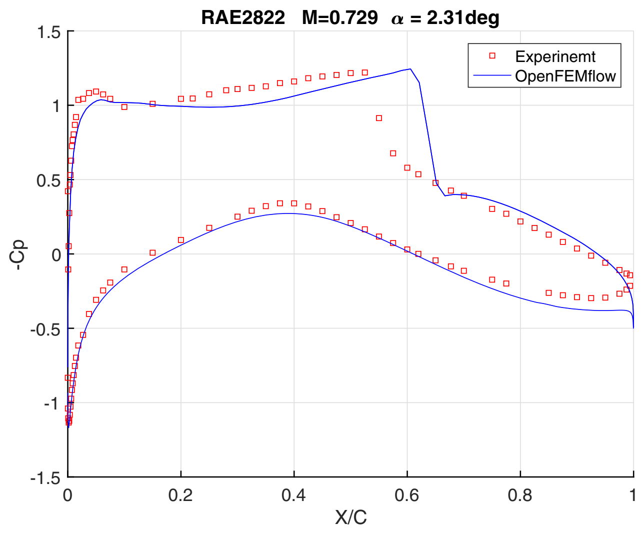

Fig. 4 Comparision of pressure distribution on RE2822 airfoil with experiment

using an unstructured mesh with 10216 elements and 5233 nodes, see Fig. 10a.

To verify the pressure distribution on airfoil computed by OpenFEMflow, the C_p distribution on RAE2822 airfoil on Mach 0.729 and $\alpha = 2.31$ is computed by the code and compared with experimental results (Viviand 1985) in Fig. 4. As one can observe from this figure, there is a good match between the OpenFEMflow result and the experiment, except the location of the chock wave, which is not completely inline with the experiment. However, it is a typical tendency of the Euler equations to overestimate the strength of shockwaves.

To verify the sensitivity analysis method the NACA0012 airfoil is parametrized using 20 CST modes (10 for the upper and 10 for the lower surfaces). Figure 5 shows the comparison between the derivatives of C_l, C_d, and C_m with respect to the CST modes computed by the adjoint method and the finite differencing.

Besides, computing the exact value of dR/dU allows us to use the full benefit of the quadratic convergence rate of the Newton method. Figure 6 shows the convergence history of analysis of the NACA0012 airfoil in Mach number of 0.7 and the angle of attack of 2 degrees. First, the Jacobian of

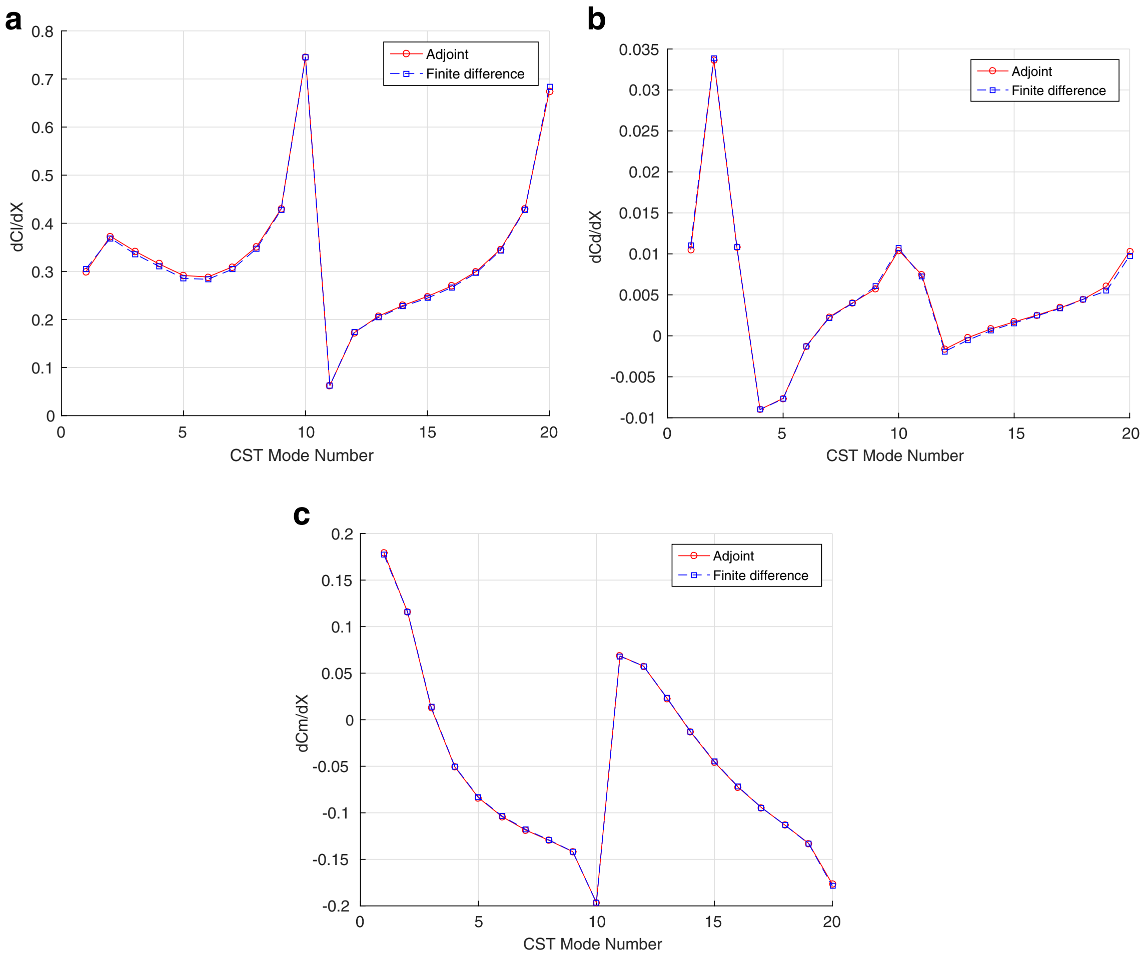

Fig. 5 **a–c** Verification of the sensitivity analysis. Results are for NACA0012 airfoil at $M = 0.7$ and $\alpha = 2°$

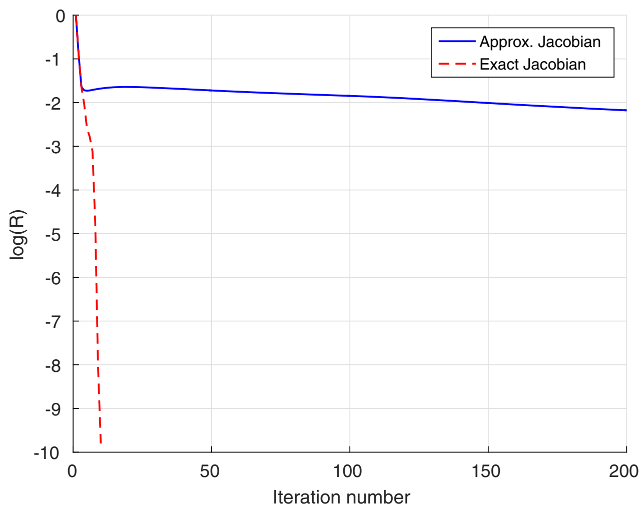

Fig. 6 The effect of exact Jacobina on the convergence history of NACA0012 airfoil in $M = 0.7$ and $\alpha = 2°$

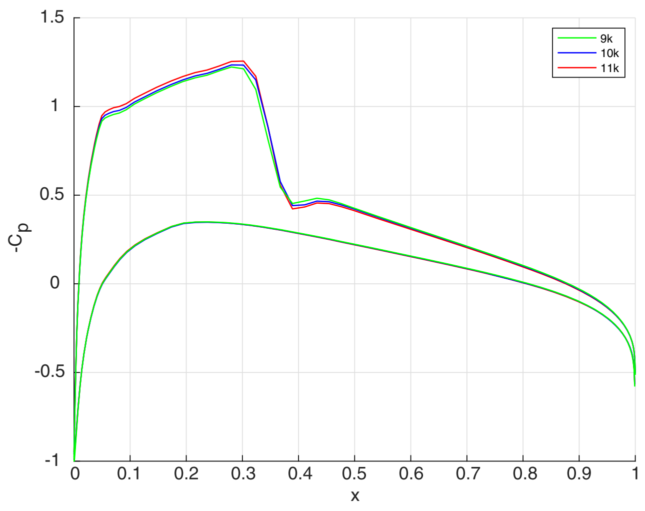

Fig. 8 Pressure distribution on NACA0012 for three different mesh sizes

the residuals is approximated by assuming a linear equation $R = KU - F$, where $dR/dU = K$. Then the exact Jacobian of the nonlinear system derived using symbolic analysis is used. From Fig. 6, one can observe that using the exact Jacobian, the Newton method reduced the norm of residuals by 9 order of magnitude in 10 iterations.

The Newton method is more prone to diverge than the Picard method. Therefore, in OpenFEMflow, the solution is started with the Picard method and then switched to the Newton method. The switch point is determined either based on the value of the residuals (e.g., if the residuals are reduced to half) or based on the simulation time. Figure 7a shows an example of convergence history and the switch point from the Picard iteration to the Newton method. From this figure, one can observe the rate of convergence is dramatically changed after switching to the Newton method. However, as mentioned earlier, the Newton method is more prone to diverge. To avoid this problem, OpenFEMflow can switch between the Picard and Newton multiple times. If after switching to the Newton method, the solution diverges, the solver switches back to the Picard method and continues the solution from the last Picard iteration. This switch back and forth can happen multiple times until a converged solution is achieved. An example of such a convergence history is shown in Fig. 7b, where multiple time switch between the Picard and the Newton method is shown. It needs to be mentioned that, such multiple times switching back and forth between the Picard and Newton method can happen only for high values of Mach numbers and high angle of attack. For moderate cases, usually, the first time switch to the Newton method results in a converged solution, as shown in Fig. 7a. For high values of Mach number and angle of attack, it is recommended to start the simulation from the converged solution of the same case with lower values of Mach and angle of attack.

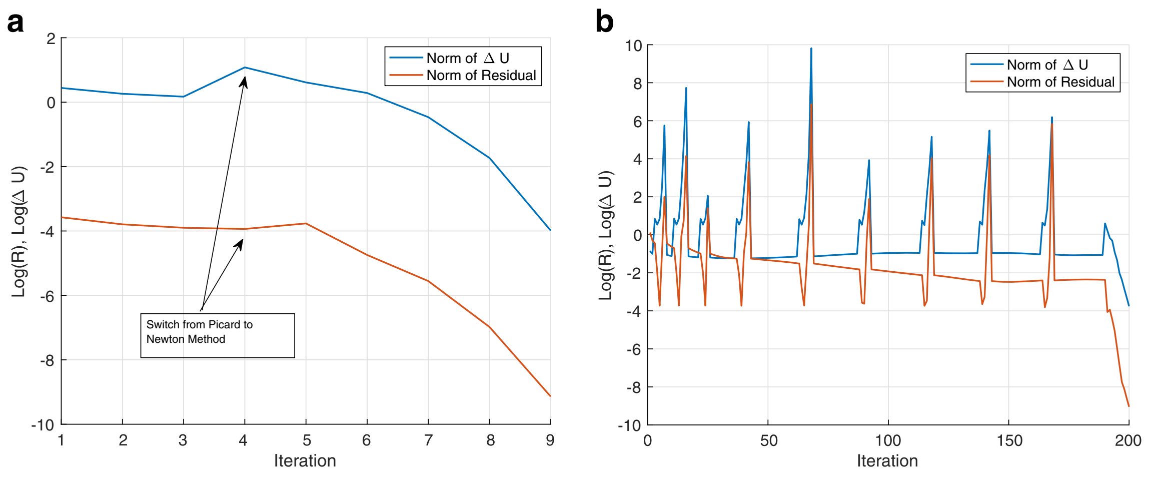

Fig. 7 **a**, **b** Examples of convergence history of OpenFEMflow

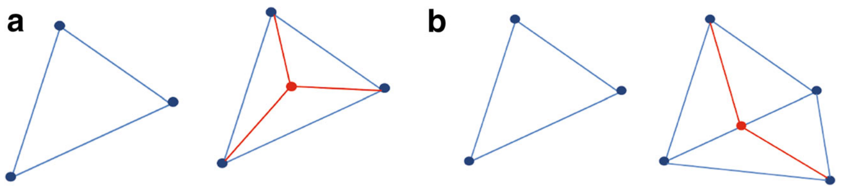

Fig. 9 **a**, **b** Dividing elements into smaller ones for mesh refinement

Last but bit least, a grid convergence study has been performed to investigate the influence of mesh on results of OpenFEMflow for an airfoil analysis. Three different airfoil meshes with the same topology but different number of nodes, 9K, 10K, and 11K nodes, are created for this study. The pressure distribution on NACA0012 airfoil analysis in Mach 0.72 and angle of attack of 2 degrees for these three different meshes are shown in Fig. 8. From this figure, one can observe that with a proper mesh topology, the influence of mesh density the pressure distribution is quite negligible.

8 Mesh adaptation

The least-squares residuals of the Euler equations can also be used as an error indicator for mesh adaptation (Lefebvre et al. 1993). Using such an approach, the element error is defined based on the least-squares residuals of the steady Euler equations:

$$E = \frac{1}{\Omega_e} U^e \int_{\Omega_e} \left[\frac{\partial N_i}{\partial x} A_1 + \frac{\partial N_i}{\partial y} A_2\right]^T \left[\frac{\partial N_j}{\partial x} A_1 + \frac{\partial N_j}{\partial y} A_2\right] dxdy\ U^e \tag{29}$$

Using this error the elements that need to be refined are indicated. An automatic mesh adaptation code is included in OpenFEMflow. Two options are used for mesh refinement. In the first option, the elements marked for refinement are divided into three elements. This is done by connecting each of the three-element nodes to the point at the center of the element as shown in Fig. 9a. In the second option, the elements marked for refinement are divided into two elements. The longest edge of each element is divided into two parts from the middle point of the edge. This point is then connected to the third node of the element. In order to make the mesh conformal, the neighbor element is also divided into two elements as shown in Fig. 9b.

Figure 10 shows an example of mesh refinement in OpenFEMflow. The RAE2822 airfoil in Mach 0.73 and $\alpha = 2°$ is considered in this case. Figure 10 shows the mesh before and after refinement using the type two mesh refinement method. The initial mesh has 10216 elements and 5233 nodes, while the adapted mesh has 16262 elements and 8282 nodes. The C_p distributions over the airfoil for two different meshes are plotted in Fig. 11.

9 Airfoil optimization

In the first test case, optimization of the NACA0012 airfoil at $M = 0.75$ is considered. The optimization is formulated as follows:

$$\begin{aligned} \min \quad & C_d(X) \\ & X = [\mathrm{CST}_i, \alpha] \qquad i = 1..10 \\ \text{s.t.} \quad & C_l - C_{l_t} = 0 \\ & C_{m_t} - C_m \le 0 \\ & t_{\max_{\text{init}}} - t_{\max} \le 0 \end{aligned} \tag{30}$$

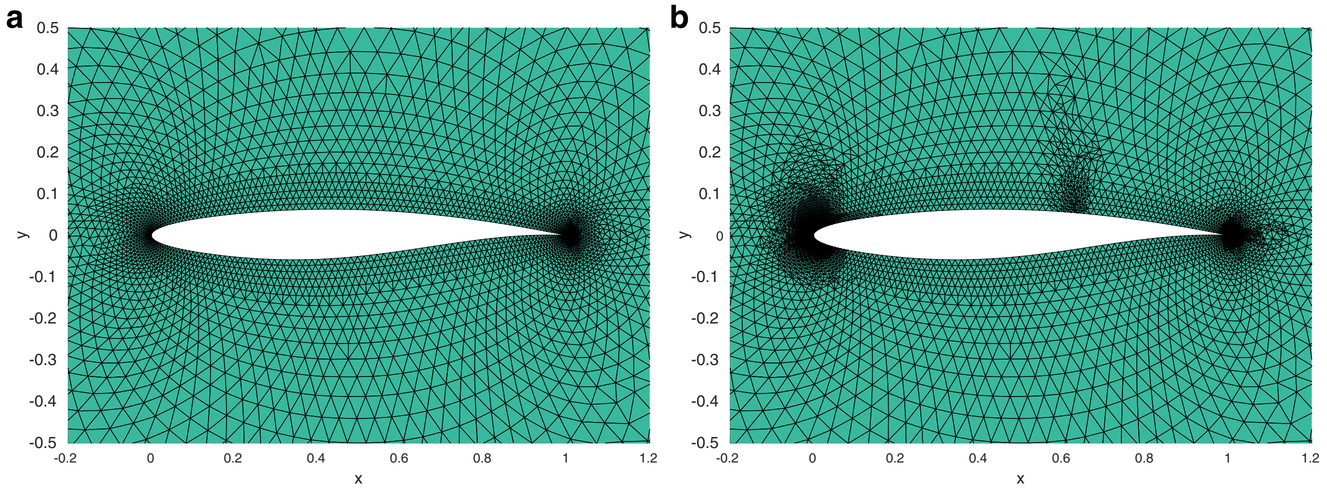

Fig. 10 **a**, **b** Unstructured mesh around RAE2822 airfoil before and after refinement

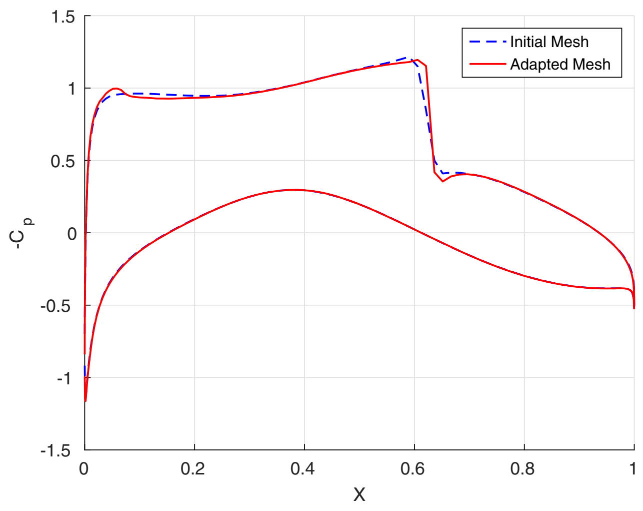

Fig. 11 Effect of mesh adaptation on airfoil pressure distribution

The airfoil geometry is parametrized using 10 CST modes, 5 for each surface. The airfoil lift coefficient is constrained to be equal to the lift coefficient of the initial airfoil at 2 degrees angle of attack. The airfoil pitching moment is constrained to be equal to or larger (less negative value) than the initial airfoil pitching moment. The airfoil maximum thickness to chord ratio is also constrained to be equal or higher than the thickness to chord ratio of the initial airfoil. The optimization is executed using a mesh with 5233 nodes and 10216 elements, see Fig. 12. The mesh adaptation has not been used during the optimization. The SNOPT optimization algorithm (Gill et al. 2005) is used as the optimizer. It should be noted that all the optimizations have been performed using the OpenFEMflow with symbolic analysis.

Table 2 summarizes the results of the optimization. Figure 13 shows the convergence history of the objective function. The geometry and the pressure distribution of the initial and the optimized airfoils are compared in Fig. 14. Pressure contours for the initial and the optimized airfoils are shown in Fig. 15.

From Fig. 14b, one can observe that the optimizer managed to eliminate the shock wave from the upper surface of the airfoil. This resulted in more than 65% reduction in the airfoil inviscid drag. The airfoil maximum thickness, the lift coefficient, and the pitching moment coefficient remained the same as the initial airfoil.

As the second test case, optimization of the NACA64A410 airfoil at Mach number of 0.75 is considered. The same constraints on airfoil lift (at zero degree angle of attack), pitching moment, and maximum thickness as the previous case are applied. The same mesh with 10216 elements is used for this optimization as well. The results of this optimization are shown in Figs. 16 and 17. Similar to the previous case, the optimizer managed to eliminate the shock wave from the upper side of the airfoil, while keeping the airfoil lift, pitching moment, and maximum thickness the same as the initial airfoil. The inviscid drag of the airfoil was reduced by more than 68%. Table 3 summarizes the results of this optimization. Pressure contours for the initial and the optimized airfoils are shown in Fig. 18.

10 Build-in optimization algorithm

Although SNOPT is a very powerful gradient-based optimization algorithm, it is not available as an open

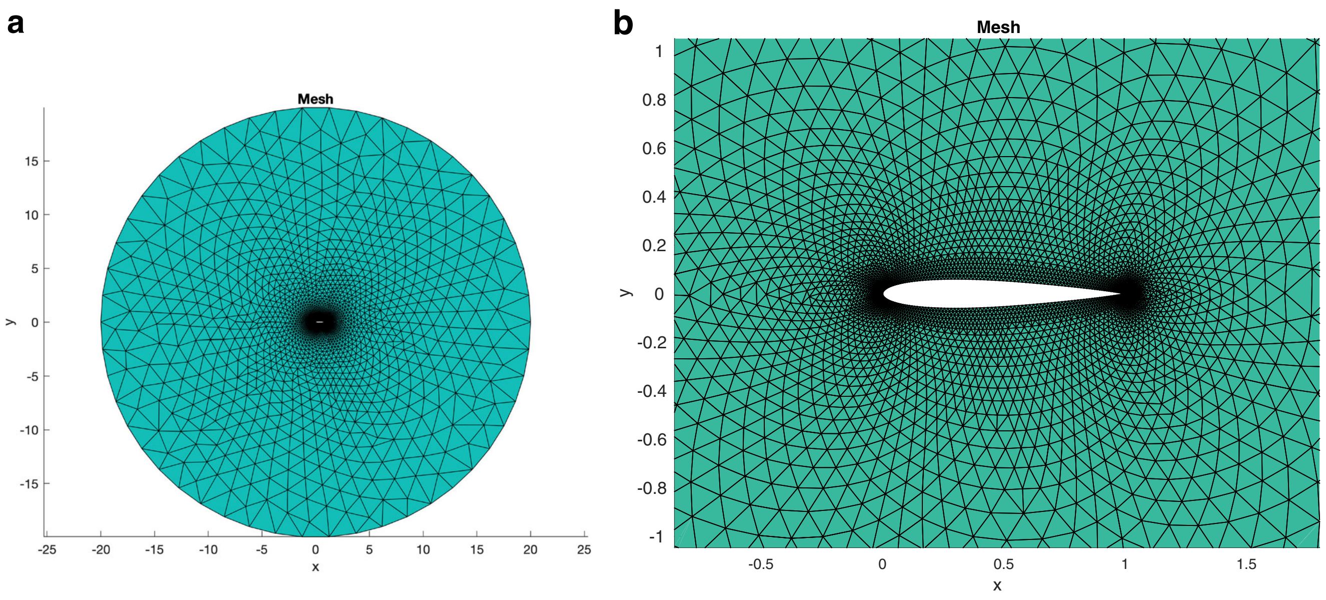

Fig. 12 **a**, **b** Unstructured mesh around NACA0012 airfoil with 10216 elements

Table 2 Results of NACA0012 airfoil optimization

	α [deg]	C_l	C_d	C_m	$\left(\frac{t}{c}\right)_{\max}$
Initial	2	0.4202	0.0121	−0.0125	0.12
Optimized	1.73	0.4204	0.0042	−0.0125	0.12

source. To support education, we wrote a build-in gradient-based optimization algorithm for airfoil shape optimization in OpenFEMflow. The algorithm solves the following problem:

$$\begin{aligned} \min \quad & C_d(X) \\ & X = [\mathrm{CST}_i, \alpha, s] \\ \text{s.t.} \quad & C_l - C_{l_t} = 0 \\ & C_{m_t} - C_m + s_1^2 = 0 \\ & t_{\max_{\text{init}}} - t_{\max} + s_2^2 = 0 \end{aligned} \tag{31}$$

s_1 and s_2 are two sack variables used to change the inequality constraints to equality constraints (more slack variables can be used if for example more than one constraint on thickness is defined). The algorithm is based on solving the (31), where the Newton method is applied to find the root of the Lagrangian of the optimization problem, as described in Nocedal and Wright (2006). It results in the following equation in iteration k:

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L}_k & -A_k^T \\ A_k & 0 \end{bmatrix} \begin{bmatrix} S_k \\ \lambda_k \end{bmatrix} = - \begin{bmatrix} \nabla f_k \\ c \end{bmatrix} \tag{32}$$

where $\mathcal{L}$ is the Lagrangian of the problem, S is the search direction, and λ is the Lagrange multiplier. The Lagrangian is defined as:

$$\mathcal{L}(x, \lambda) = f(x) - \sum(\lambda c) \tag{33}$$

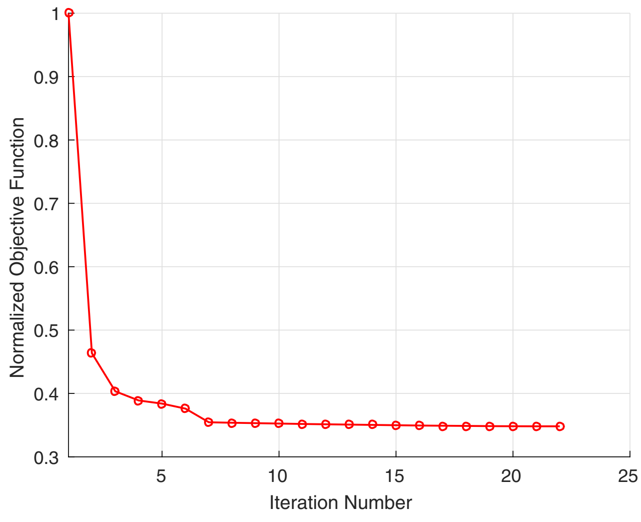

Fig. 13 Convergence history of NACA0012 airfoil optimization

The Hessian of the Lagrangian ($\nabla^2_{xx}\mathcal{L}$) is calculated using the BFGS method (Nocedal and Wright 2006). The backtracking algorithm is used to find the step length based on the search direction S.

To verify the build-in algorithm of OpenFEMflow, an airfoil optimization has been performed using this algorithm and the outcome is compared to the outcomes of the same optimization using Matlab FMINCON and SNOPT. The optimization is performed on RAE2822 airfoil at Mach number 0.75 and Angle of attack 1 degree. The shape of the initial and optimized airfoils and the pressure distribution on them are shown in Fig. 19a and b respectively. From these figures, it can be observed that the shape and pressure distribution of the airfoils optimized by all the algorithms are quite similar. Table 4 summarizes the results. From this table, one can observe that all the algorithms converged to the same values of drag. The pitching moment coefficients are the same in all cases; however, there is a slight difference in the lift coefficient. Interestingly the optimization time of the build-in algorithm of OpenFEMflow is lower than the other two algorithms (it should be noted that only the lower function call does not mean the lower computational time, since the convergence time of each airfoil analysis is different, depending on the shape of the airfoil).

From Table 4, one can observe that the total runtime of a full airfoil optimization varies between 3 and 7 min. These results were obtained using a 4.2 GHz Quad-Core Intel Core i7 processor. Generally each airfoil simulation may take about 1 min on a single processor if starting from a no initial solution. During the optimization, the code can be executed using the solution of the previous iteration, which reduces the computational time largely. There is also an option of parallel computing in OpenFEMflow. However, for a single airfoil simulation with number of nodes about 10000, parallel computing is not necessary and may not even result in large improvement in computational cost. But anyhow using this option further reduction in computational cost can be achieved depending on the number of available processors and the number of nodes in the mesh.

11 Executing the OpenFEMflow code

The OpenFEMflow code can be downloaded from https://github.com/mdotubs/OpenFEMflow. The Matlab file, Start.m should be used to start the CFD simulation and

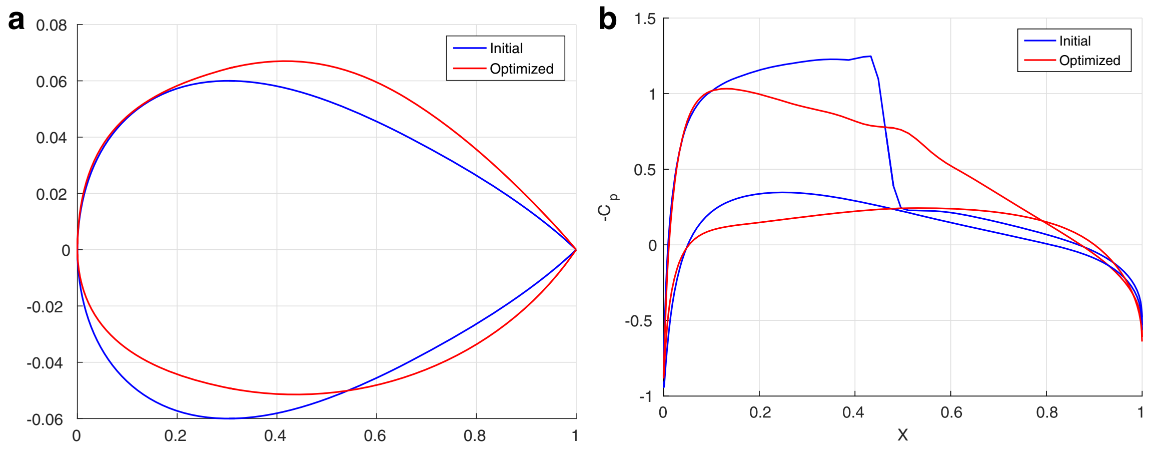

Fig. 14 **a**, **b** Comparison of the initial and optimized airfoils for NACA0012 airfoil optimization

optimization. At the beginning of this file, the inputs for analysis are defined. Inputs include, but are not limited to, the freestream Mach number and angle of attack, the CFL number, maximum number of iteration, and convergence criteria. A few settings variables are defined to control (on-off) the adjoint solution, mesh deformation, and mesh adaptation as well as plotting the results.

In the second part of the Start.m, the CST control points for the airfoil shape are defined, in case that an airfoil is used for simulation and mesh deformation is required. The CST coefficients are defined in a vector with an arbitrary dimension. However, an equal number of control points for the upper and the lower surfaces of the airfoil should be used. The first half of the vector includes the CST coefficients for the upper side and the second half includes the CST coefficients of the lower side of the airfoil. It should be noted that OpenFEMflow is not only developed for airfoil simulation but also other cases such as flow in a channel with a bump, or flow over a cylinder, etc. can be simulated with OpenFEMflow. In such cases, the mesh deformation option should be switched off, as it only works for airfoil shape parameterized using the CST method.

The third part of the Start.m includes a few line codes for running the OpenFEMflow CFD solver as well as calculating the airfoil thickness. Then some pieces of codes are provided to run airfoil shape optimization using SNOPT, MATLAB FMINCON as well as the build-in SQP algorithm of OpenFEMflow.

It should be noted that since OpenFEMflow is based on the Euler method, it does not make sense to use it for subsonic

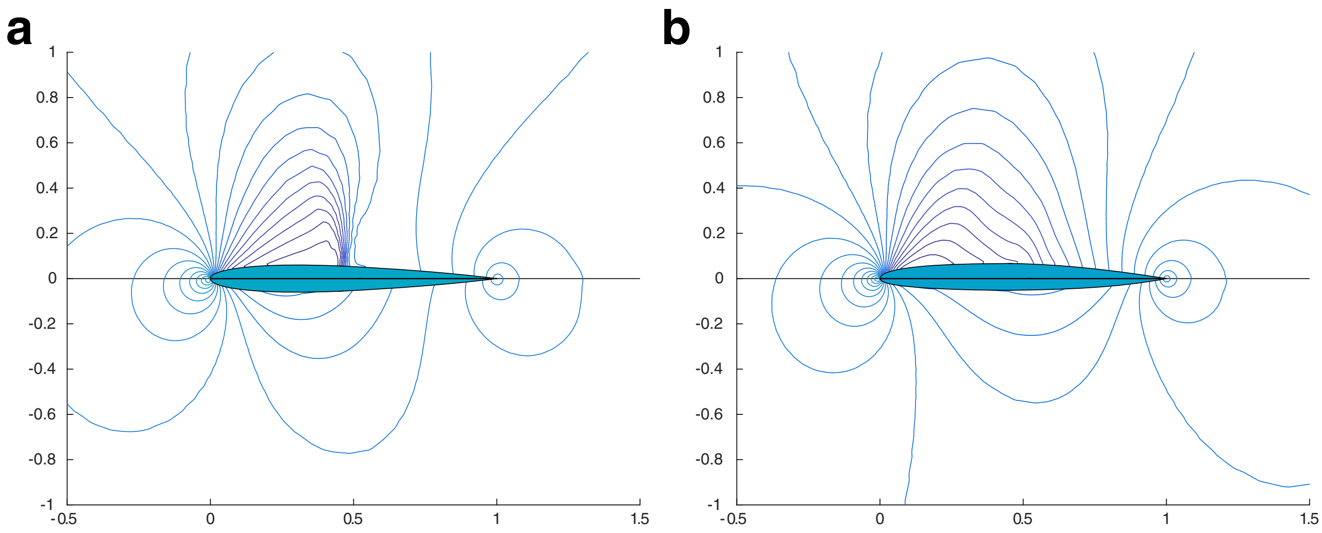

Fig. 15 **a**, **b** Pressure contours of the initial and optimized airfoils for NACA0012 airfoil optimization

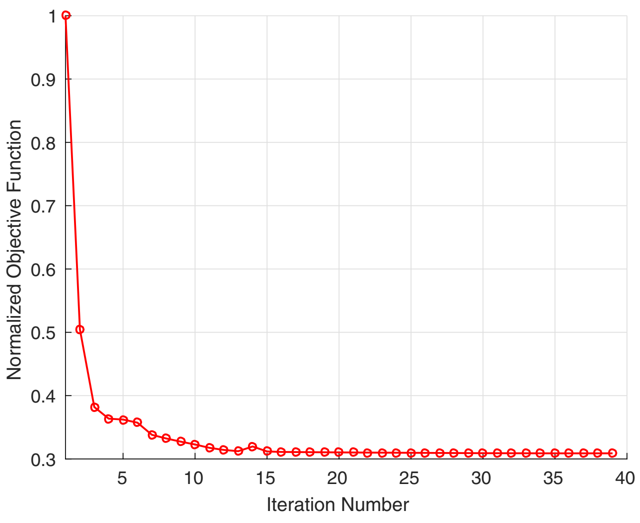

Fig. 16 Convergence history of NACA64A410 airfoil optimization

airfoil optimization. As the Euler method is only for an inviscid flow, it cannot predict the friction drag of an airfoil, which is the main drag component of an airfoil in the subsonic regime. An airfoil optimization in the subsonic regime usually tries to maximize the region of laminar flow on the airfoil to minimize the friction drag. Since this ability does not exist in the Euler method, it does not make sense to use it for subsonic airfoil optimization. The best ability of the Euler method is to minimize the wave drag of transonic airfoils, by eliminating or weakening the shock wave(s) on the airfoil.

12 Conclusions

An open-source adjoint CFD code is presented to support teaching CFD as well as aerodynamic shape optimization. The code is based on symbolic analysis to drive the sensitivity equations. The use of symbolic analysis resulted in a couple of advantages:

1- Increasing the accuracy of the computations by computing exact values of element matrices compared to numerical approximations, which can also result in better convergence.
2- Automation of code development which can reduce the possibilities of making errors and bugs in the code. This also reduces the time and human effort required to develop a new code.
3- Elimination of the need for reverse AD for computing the derivatives in discrete adjoint method, which can result in higher computational efficiency.

It should be mentioned that although running a CFD code developed based on symbolic analysis could be more efficient than a similar code based on reverse AD in computations time and memory requirements, the derivation of the symbolic equations can be a time-consuming task with a need for powerful computers. However, this should be done only once to develop the code and later the code can be executed with less computational power requirements. In addition to that, for complex elements, e.g., high-order nonlinear elements, using the available symbolic analysis toolboxes may result in prohibitive growth of expressions. Automatic differentiation can still be a good complementary option to symbolic analysis for more complex formulation. However, improving the symbolic analysis toolboxes may lead to the possibility to symbolically solve much more complex equations. Code transforming AD is another approach similar to Symbolic analysis, which creates new codes with the derivatives, which is more efficient than reverse AD mode in memory and computational cost. It is an alternative to symbolic analysis; however, only for

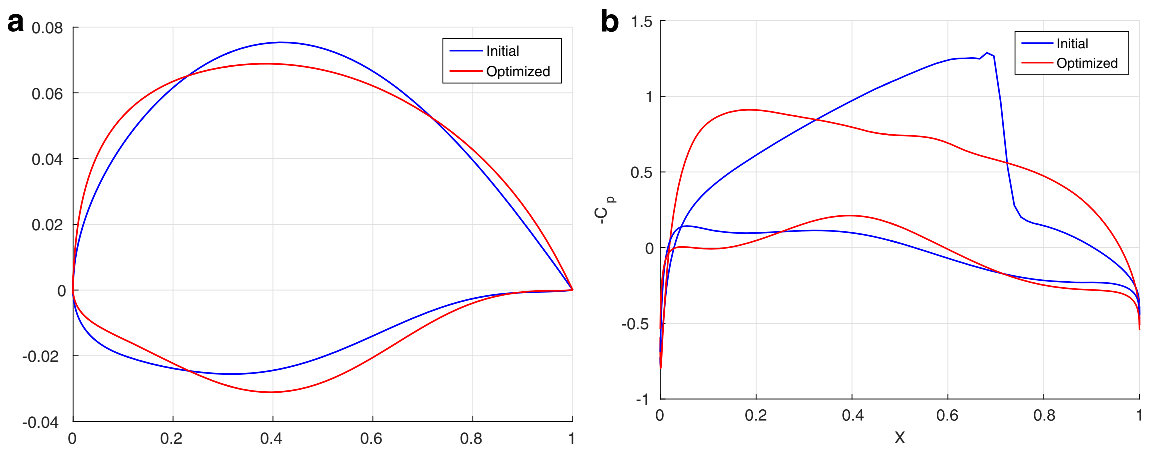

Fig. 17 **a**, **b** Comparison of the initial and optimized airfoils for NACA64A410 airfoil optimization

Table 3 Results of NACA64A410 airfoil optimization

	α [deg]	C_l	C_d	C_m	$\left(\frac{t}{c}\right)_{\max}$
Initial	0	0.6381	0.0160	−0.1709	0.10
Optimized	−0.3	0.6381	0.0050	−0.1484	0.10

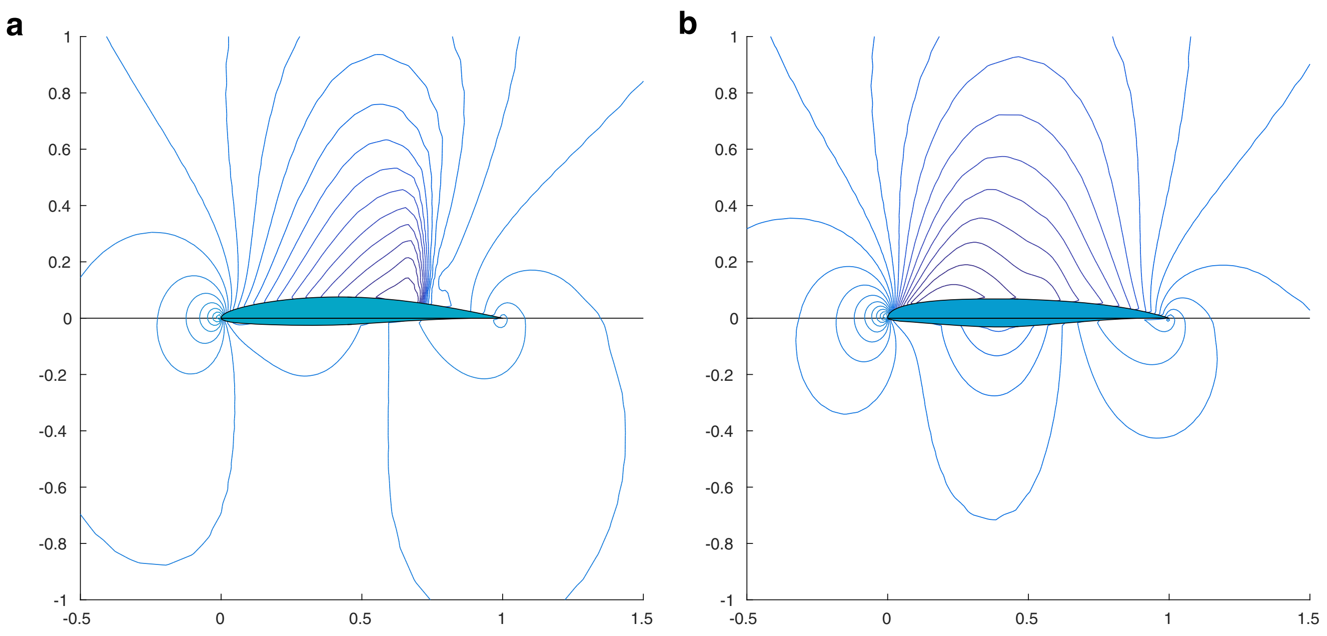

Fig. 18 **a**, **b** Pressure contours of the initial and optimized airfoils for NACA64A410 airfoil optimization

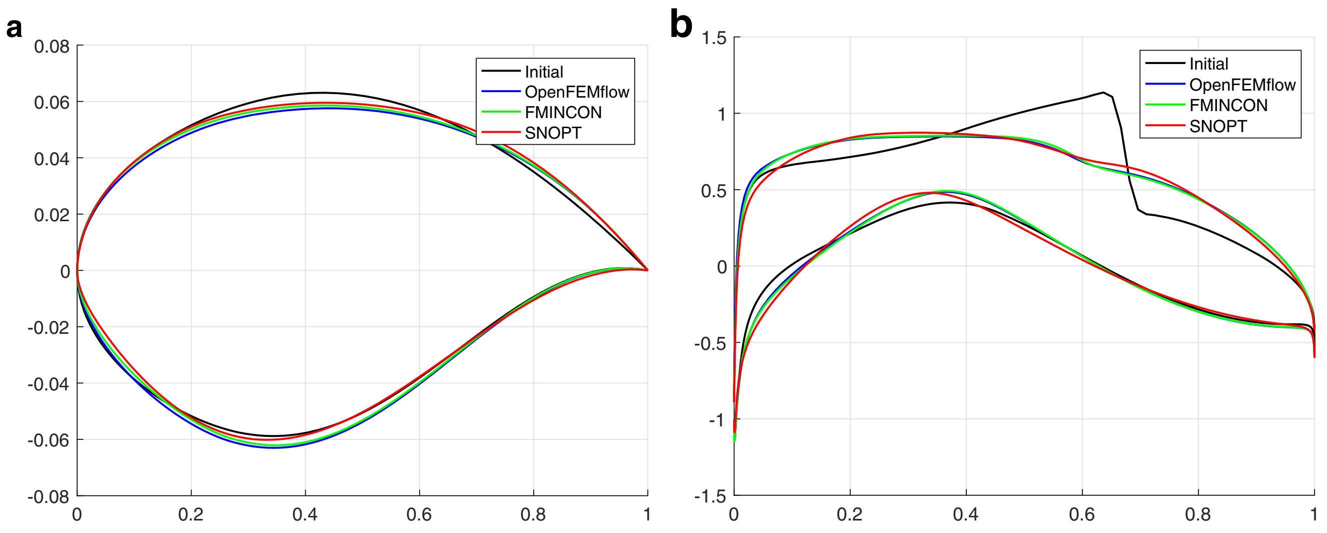

Fig. 19 **a**, **b** Comparision of the results of airfoil optimization using different optimization algorithms

Table 4 Comparison between results of FMINCON and OpenFEMflow build-in algorithms for airfoil optimization

	Initial	OpenFEMflow	FMINCON SQP	SNOPT
C_d	0.0159	0.0013	0.0013	0.0013
C_l	0.6008	0.6011	0.6014	0.6008
C_m	−0.1386	−0.1386	−0.1385	−0.1386
Iteration		5	5	15
Function call		22	85	37
Wall time (s)		197	596	408

computing the derivatives, while symbolic analysis can be used for integrating the element matrices, etc.

OpenFEMflow is developed to support educating students to develop adjoint finite element code for aerodynamic and/or structural analysis and optimization. The code is released as open-source, including the supporting codes to generate all the symbolic parts. It also includes a simple built-in SQP optimization algorithm for airfoil optimization. In addition to that, supporting codes are available to use MATLAB FMINCON for airfoil optimization.

Appendix

```
%% Defining symbols
% in this section the unkowns are defined as symbols

syms L1 L2
syms x1 x2 x3 y1 y2 y3
syms r u v p
syms r1 u1 v1 p1 r2 u2 v2 p2 r3 u3 v3 p3
syms dt

%% Defining shape functions for linear triangular elements

L3 = 1 - L1 - L2;

S1 = L1;
S2 = L2;
S3 = L3;

%% here the derivatives of the shape functions required for
% generating the stiffness and force matrices are derived

dx_dL1 = x1 - x3;
dx_dL2 = x2 - x3;
dy_dL1 = y1 - y3;
dy_dL2 = y2 - y3;

J = [dx_dL1  dy_dL1;  dx_dL2  dy_dL2];

detJ = (x1 - x3)*(y2 - y3) - (x2 - x3)*(y1 - y3);

dS1 = J\[diff(S1,L1); diff(S1,L2)]; dS1dx = dS1(1); dS1dy = dS1(2);
dS2 = J\[diff(S2,L1); diff(S2,L2)]; dS2dx = dS2(1); dS2dy = dS2(2);
dS3 = J\[diff(S3,L1); diff(S3,L2)]; dS3dx = dS3(1); dS3dy = dS3(2);

%% here the  Eq. (14) is derived symbilically

g = 1.4;

A1 = [u   r   0   0
      0   u   0   1/r
      0   0   u   0
      0   g*p 0   u];
```

```

A2 = [v  0  r  0
      0  v  0  0
      0  0  v  1/r
      0  0  g*p  v];

E = eye(4);


Ln1 = E*S1 + dt*A1*dS1dx + dt*A2*dS1dy;
Ln2 = E*S2 + dt*A1*dS2dx + dt*A2*dS2dy;
Ln3 = E*S3 + dt*A1*dS3dx + dt*A2*dS3dy;

L = [Ln1 Ln2 Ln3];
f = [r; u; v; p];

%% Here the stiffness and force matrices are derived symbolically

SK = transpose(L)*L;
K = int(int(SK*detJ,L2,0,1-L1),L1,0,1);

SF = transpose(L)*f;
F = int(int(SF*detJ,L2,0,1-L1),L1,0,1);

%% Here the residual R and its derivative with respect to the U
% vector (dR/dU) is derived symbolically

U = [r1; u1; v1; p1; r2; u2; v2; p2; r3; u3; v3; p3];
R = K*U-F;

dRdr = diff(R,r);
dRdu = diff(R,u);
dRdv = diff(R,v);
dRdp = diff(R,p);

dRdr1 = diff(R,r1) + dRdr*S1;
dRdr2 = diff(R,r2) + dRdr*S2;
dRdr3 = diff(R,r3) + dRdr*S3;

dRdu1 = diff(R,u1) + dRdu*S1;
dRdu2 = diff(R,u2) + dRdu*S2;
dRdu3 = diff(R,u3) + dRdu*S3;

dRdv1 = diff(R,v1) + dRdv*S1;
dRdv2 = diff(R,v2) + dRdv*S2;
dRdv3 = diff(R,v3) + dRdv*S3;

dRdp1 = diff(R,p1) + dRdp*S1;
dRdp2 = diff(R,p2) + dRdp*S2;
dRdp3 = diff(R,p3) + dRdp*S3;

dRdU1 = [dRdr1 dRdu1 dRdv1 dRdp1];
dRdU2 = [dRdr2 dRdu2 dRdv2 dRdp2];
dRdU3 = [dRdr3 dRdu3 dRdv3 dRdp3];

dRdU  = [dRdU1 dRdU2 dRdU3];

%% Here the derivatives of the residuals with respect to the x
% and y coordinates for a point at the center of the element is
% derived symbolically (dR/dx and dR/dy)
%  x = (x1+x2+x3)/3  and y = (y1+y2+y3)/3

syms x y

S1 = (x2*y3 - x3*y2)/(x1*y2 - x2*y1 - x1*y3 + x3*y1 + x2*y3 - ...
 x3*y2) - (y*(x2 - x3))/ (x1*y2 - x2*y1 - x1*y3 + x3*y1 + x2*y3 ...
 - x3*y2) + (x*(y2 - y3))/ (x1*y2 - x2*y1 - x1*y3 + x3*y1 + ...
 x2*y3 - x3*y2);
```

```

S2 = (y*(x1 - x3))/(x1*y2 - x2*y1 - x1*y3 + x3*y1 + x2*y3 - ...
x3*y2) - (x1*y3 - x3*y1)/ (x1*y2 - x2*y1 - x1*y3 + x3*y1 ...
+ x2*y3 - x3*y2) - (x*(y1 - y3))/ (x1*y2 - x2*y1 - x1*y3 + ...
x3*y1 + x2*y3 - x3*y2);

S3 = (x1*y2 - x2*y1)/(x1*y2 - x2*y1 - x1*y3 + x3*y1 + ...
x2*y3 - x3*y2) - (y*(x1 - x2))/(x1*y2 - x2*y1 - x1*y3 +...
x3*y1 + x2*y3 - x3*y2) + (x*(y1 - y2))/(x1*y2 - x2*y1 -...
x1*y3 + x3*y1 + x2*y3 - x3*y2);

dS1_dx1 = diff(S1,x1) + diff(S1,x)*1/3;
dS2_dx1 = diff(S2,x1) + diff(S2,x)*1/3;
dS3_dx1 = diff(S3,x1) + diff(S3,x)*1/3;

dS1_dx2 = diff(S1,x2) + diff(S1,x)*1/3;
dS2_dx2 = diff(S2,x2) + diff(S2,x)*1/3;
dS3_dx2 = diff(S3,x2) + diff(S3,x)*1/3;

dS1_dx3 = diff(S1,x3) + diff(S1,x)*1/3;
dS2_dx3 = diff(S2,x3) + diff(S2,x)*1/3;
dS3_dx3 = diff(S3,x3) + diff(S3,x)*1/3;


dr_dx1 = r1*dS1_dx1 + r2*dS2_dx1+ r3*dS3_dx1 ;
dr_dx2 = r1*dS1_dx2 + r2*dS2_dx2+ r3*dS3_dx2 ;
dr_dx3 = r1*dS1_dx3 + r2*dS2_dx3+ r3*dS3_dx3 ;

du_dx1 = u1*dS1_dx1 + u2*dS2_dx1+ u3*dS3_dx1 ;
du_dx2 = u1*dS1_dx2 + u2*dS2_dx2+ u3*dS3_dx2 ;
du_dx3 = u1*dS1_dx3 + u2*dS2_dx3+ u3*dS3_dx3 ;

dv_dx1 = v1*dS1_dx1 + v2*dS2_dx1+ v3*dS3_dx1 ;
dv_dx2 = v1*dS1_dx2 + v2*dS2_dx2+ v3*dS3_dx2 ;
dv_dx3 = v1*dS1_dx3 + v2*dS2_dx3+ v3*dS3_dx3 ;

dp_dx1 = p1*dS1_dx1 + p2*dS2_dx1+ p3*dS3_dx1 ;
dp_dx2 = p1*dS1_dx2 + p2*dS2_dx2+ p3*dS3_dx2 ;
dp_dx3 = p1*dS1_dx3 + p2*dS2_dx3+ p3*dS3_dx3 ;


dRdx1 = diff(R,x1) + diff(R,r)*dr_dx1 + diff(R,u)*du_dx1 ...
+  diff(R,v)*dv_dx1 +diff(R,p)*dp_dx1 ;
dRdx2 = diff(R,x2) + diff(R,r)*dr_dx2 + diff(R,u)*du_dx2 ...
+ diff(R,v)*dv_dx2 +diff(R,p)*dp_dx2 ;
dRdx3 = diff(R,x3) + diff(R,r)*dr_dx3 + diff(R,u)*du_dx3 ...
+ diff(R,v)*dv_dx3 +diff(R,p)*dp_dx3 ;

dRdx = [dRdx1  dRdx2  dRdx3];

%
dS1_dy1 = diff(S1,y1) + diff(S1,y)*1/3;
dS2_dy1 = diff(S2,y1) + diff(S2,y)*1/3;
dS3_dy1 = diff(S3,y1) + diff(S3,y)*1/3;
```

```

dS1_dy2 = diff(S1,y2) + diff(S1,y)*1/3;
dS2_dy2 = diff(S2,y2) + diff(S2,y)*1/3;
dS3_dy2 = diff(S3,y2) + diff(S3,y)*1/3;

dS1_dy3 = diff(S1,y3) + diff(S1,y)*1/3;
dS2_dy3 = diff(S2,y3) + diff(S2,y)*1/3;
dS3_dy3 = diff(S3,y3) + diff(S3,y)*1/3;


dr_dy1 = r1*dS1_dy1 + r2*dS2_dy1+ r3*dS3_dy1 ;
dr_dy2 = r1*dS1_dy2 + r2*dS2_dy2+ r3*dS3_dy2 ;
dr_dy3 = r1*dS1_dy3 + r2*dS2_dy3+ r3*dS3_dy3 ;

du_dy1 = u1*dS1_dy1 + u2*dS2_dy1+ u3*dS3_dy1 ;
du_dy2 = u1*dS1_dy2 + u2*dS2_dy2+ u3*dS3_dy2 ;
du_dy3 = u1*dS1_dy3 + u2*dS2_dy3+ u3*dS3_dy3 ;

dv_dy1 = v1*dS1_dy1 + v2*dS2_dy1+ v3*dS3_dy1 ;
dv_dy2 = v1*dS1_dy2 + v2*dS2_dy2+ v3*dS3_dy2 ;
dv_dy3 = v1*dS1_dy3 + v2*dS2_dy3+ v3*dS3_dy3 ;

dp_dy1 = p1*dS1_dy1 + p2*dS2_dy1+ p3*dS3_dy1 ;
dp_dy2 = p1*dS1_dy2 + p2*dS2_dy2+ p3*dS3_dy2 ;
dp_dy3 = p1*dS1_dy3 + p2*dS2_dy3+ p3*dS3_dy3 ;


dRdy1 = diff(R,y1) + diff(R,r)*dr_dy1 + diff(R,u)*du_dy1 ...
+ diff(R,v)*dv_dy1 +diff(R,p)*dp_dy1 ;
dRdy2 = diff(R,y2) + diff(R,r)*dr_dy2 + diff(R,u)*du_dy2 ...
+ diff(R,v)*dv_dy2 +diff(R,p)*dp_dy2 ;
dRdy3 = diff(R,y3) + diff(R,r)*dr_dy3 + diff(R,u)*du_dy3 ...
+ diff(R,v)*dv_dy3 +diff(R,p)*dp_dy3 ;

dRdy = [dRdy1  dRdy2  dRdy3];
```

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Replication of results The results presented in this study can be replicated by using the open source code OpenFEMflow.

References

Antunes AP, Azevedo JLF (2014) Studies in aerodynamic optimization based on genetic algorithms. J Aircr 51(3):1002–1012

Baysal O, Ghayour K (2001) Continuous adjoint sensitivities for optimization with general cost functionals on unstructured meshes. AIAA J 39(1):48–55

Brezillon J, Gauger NR (2004) 2D And 3D aerodynamic shape optimisation using the adjoint approach. Aerosp Sci Technol 8:715–727

Bochev PB, Gunzburger MD (2009) Least-squares finite element methods. Springer, New York

Carpentieri G, Koren B, van Tooren MJL (2007) Adjoint-based aerodynamic shape optimization on unstructured meshes. J Comput Phys 224:267–287

Elham A, La Rocca G, van Tooren MJL (2013) Development and implementation of an advanced, design-sensitive method for wing weight estimation. Aerosp Sci Technol 29:100–113

Elham A, van Tooren MJL (2016) Coupled adjoint aerostructural wing optimization using quasi-three-dimensional aerodynamic analysis. Struct Multidiscip Optim 54:889–906

Elham A, van Tooren MJL (2014) Weight indexing for wing-shape multi-objective optimization. AIAA J 52(2):320–337

Farrell PE, Ham DA, Funke S, Rognes M (2012) Automated derivation of the adjoint of high-level transient finite element programs. SIAM J Sci Comput 35(4)

Gill P, Murray W, Saunders M (2005) SNOPT: an SQP algorithm for large-scale constrained optimization. SIAM Rev 47(1):99–131

Grienwank A, Walther A (2008) Evaluating drivatives, principles and techniques of algorithmic differentiation. SIAM

Hovland P, Mohammadi B, Bischof C (1998) Automatic differentiation of Navier-Stokes computations, computational methods for optimal design and control. Boston 265–284

Jiang BN (1998) The least squares finite element methods, theory and applications in computational fluid dynamics and electromagnetics. Springer

Jiang BN, Carey GF (1990) Least-squares finite element methods for compresible euler equations. Int J Numer Methods Fluids 10:557–568

Kenway GKW, Mader CA, He P, Martins JRRA (2019) Effective adjoint approaches for computational fluid dynamics. Prog Aerosp Sci 110

Korelc J, Wriggers P (2016) Automation of finite element methods. Springer

Krishnamoorthy CS (1994) Finite element analysis, theory and programming, 2nd edn. McGraw Hill Education, New Delhi

Kulfan B (2008) Universal parametric geometry representation method. J Aircr 45(1):142–158

Lefebvre D, Peraire J, Morgan K (1993) Finite elmenet least squares solution of the euler equatsions using linear and quadratic approximations. Int J Comput Fluid Dyn 1:1–23

Mader CA, Martins JRRA, Alonso JJ, van der Weide E (2008) ADjoint: an approach for the rapid development of discrete adjoint solvers. AIAA J 46(4):863–873

Mariens J, Elham A, van Tooren MJL (2014) Quasi three-dimensional aerodynamic solver for multidisciplinary design optimization of lifting surfaces. J Aircr 51(2):547–558

Martins JRRA, Hwang JT (2013) Review and unification of methods for computing derivatives of multidisciplinary computational models. AIAA J 51(11):2582–2598

Nocedal J, Wright S (2006) Numerical optimization. Springer

Palacios F, Economon TD, Wendorf AD, Alonso JJ (2015) Large-scale aircraft design using SU2. 53rd AIAA Aerospace Sciences Meeting, Kissimmee, Florida, USA

Reddy JN, Gartling DK (2001) The finite element method in heat transfer and fluid dynamics. CRC Press, Boca Raton

Sederberg TW, Parry SR (1986) Free-Form Deformation of solid geometric models. In: Proceedings of SIGGRAPH '86, computer graphics, vol 20, pp 151–160

Viviand H (1985) Numerical solutions of two-dimensional reference test cases, in test cases for inviscid flow field methods AGARD-AR-211