AIAA Journal
Vol. 46, No. 6, June 2008

Unsteady Discrete Adjoint Formulation for Two-Dimensional Flow Problems with Deforming Meshes

Karthik Mani[*] and Dimitri J. Mavriplis[†]
University of Wyoming, Laramie, Wyoming 82071-3295

DOI: 10.2514/1.29924

A method to apply the discrete adjoint for computing sensitivity derivatives in two-dimensional unsteady flow problems is presented. The approach is to first develop a forward or tangent linearization of the nonlinear flow problem in which each individual component building up the complete flow solution is differentiated against the design variables using the chain rule. The reverse or adjoint linearization is then constructed by transposing and reversing the order of multiplication of the forward problem. The developed algorithm is very general in that it applies directly to the arbitrary Lagrangian–Eulerian form of the governing equations and includes the effect of deforming meshes in unsteady flows. It is shown that an unsteady adjoint formulation is essentially a single backward integration in time and that the cost of constructing the final sensitivity vector is close to that of solving the unsteady flow problem integrated forward in time. It is also shown that the unsteady adjoint formulation can be applied to time-integration schemes of different orders of accuracy with minimal changes to the base formulation. The developed technique is then applied to three optimization examples, the first in which the shape of a pitching airfoil is morphed to match a target time-dependent load profile, the second in which the shape is optimized to match a target time-dependent pressure profile, and the last in which the time-dependent drag profile is minimized without any loss in lift.

I. Introduction

RECENTLY, the use of adjoint equations has become a popular approach for solving aerodynamic design optimization problems based on computational fluid dynamics [1–6]. Adjoint equations are a very powerful tool in the sense that they enable the computation of sensitivity derivatives of an objective functional to a set of given inputs at a cost that is essentially independent of the number of inputs. This is in contrast to the finite difference method, in which each input or design variable has to be perturbed individually to obtain a corresponding effect on the output functional. This is a tedious and costly process, which becomes impractical in the presence of large numbers of design variables or inputs.

Solutions of steady-state aerodynamic optimization problems using adjoint methods [4,7–10] are now fairly well established. However, relatively little work has been done in applying these methods for unsteady time-dependent problems [11–13]. In the context of unsteady flows, frequency-domain methods [14,15] have been investigated that allow for reduced computational expense, particularly for problems with strong periodic behavior. More recently, unsteady time-domain adjoint methods have been investigated by Rumpfkeil and Zingg [13], although this work is in the context of structured meshes using relatively simple mesh deformation strategies. The goal of this paper is to develop an efficient framework for computing sensitivity derivatives in unsteady flow problems operating directly in the time domain using unstructured meshes and including the effect of general mesh deformation techniques. It is important that dynamic deformation of unstructured meshes be addressed because such techniques are now commonplace in unsteady computations, particularly in the field of aeroelasticity. Additionally, although rigid-body motion in unsteady computations may be handled by overset-grid methods, shape deformation in such flows can be handled only via the use of mesh deformation equations.

The daunting computational expense associated with unsteady optimization problems has prompted the development of frequency-domain approaches, which have been shown to be more economical for problems with periodic behavior. Although time-domain methods may be more expensive for such problems, they can be expected to be more suitable in the general case for problems with no dominant periodic behavior, and the two approaches should be considered complementary to one another. In general, the total cost of an optimization is equal to the number of design iterations required to reach an optimum solution multiplied by the cost of a single design iteration. The cost of a single design iteration can be broken down into the cost of a flow solution and the cost of a sensitivity solution. Reduction of the total number of design cycles to obtain an optimum solution can be achieved by using sophisticated optimization algorithms. For our work, we use the reduced Hessian Broyden–Fletcher–Goldfarb–Shann (BFGS) optimization algorithm with bounds (LBFGS-B) developed in [16]. Considering the fact that unsteady analysis solutions themselves are inherently expensive, all possible methods to improve efficiency both in the case of the flow solver and the sensitivity solver must be exploited to the fullest extent. This necessitates the use of multigrid methods to reduce dependency on mesh resolution and also the use of high-order time-integration schemes to obtain high-quality, unsteady solutions with a minimal number of time steps [17]. Conversely, it is important that adjoint methods developed for unsteady problems be general enough that they are applicable to any order of time-integration scheme. Also, convergence acceleration methods, such as multigrid, must be easily extendable to the developed algorithms. The unsteady adjoint algorithm presented in this paper partially addresses both of these concerns. The algorithm takes into account high-order backward-difference formula (BDF) time-integration schemes and can be extended to higher-order implicit Runge–Kutta schemes without severe additional cost. The algorithm also uses the linear multigrid method [18] for convergence acceleration.

The optimization examples presented in the paper are based on a sinusoidally pitching airfoil under transonic conditions. Although this is a periodically steady-state problem, which can be treated using frequency-domain methods, we use the same case for demonstration of

Presented as Paper 60 at the 45th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, 8–11 January 2007; received 22 January 2007; revision received 25 January 2008; accepted for publication 7 March 2008.

[*]Graduate Student, Department of Mechanical Engineering; kmani@uwyo.edu. Member AIAA.

[†]Professor, Department of Mechanical Engineering; mavripl@uwyo.edu. Associate Fellow AIAA.

the algorithm by treating it directly in the time domain over one period of oscillation beginning with a steady-state solution. The primary motivation for this arises from the potential of applying the algorithm in aeroelastic design of rotorcraft blades although the time-domain approach developed in this work should be equally applicable to more complex unsteady motions such as aeroelastic flutter problems.

II. Analysis Problem Formulation

A. Governing Equations of Flow Problem in Arbitrary Lagrangian–Eulerian Form

The conservative form of the Euler equations is used in solving the flow problem. The paper is limited to inviscid flow problems because the primary focus is the development and verification of an unsteady adjoint method. Extension of the algorithm to viscous flow problems with the inclusion of turbulence models should prove to be relatively straightforward. The differences arise only in the linearization of the additional flux contributions and not in the base formulation presented in this paper. In vectorial form, the conservative form of the Euler equations may be written as

$$\frac{\partial \mathbf{U}(\mathbf{x},t)}{\partial t} + \nabla\cdot\mathbf{F}(\mathbf{U}) = 0 \tag{1}$$

where the state vector $\mathbf{U}$ of conserved variables and the Cartesian inviscid flux vector $\mathbf{F} = (\mathbf{F}^x, \mathbf{F}^y)$ are

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ E_t \end{pmatrix}, \quad \mathbf{F}^x = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ u(E_t + p) \end{pmatrix}, \quad \mathbf{F}^y = \begin{pmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ v(E_t + p) \end{pmatrix} \tag{2}$$

Here, ρ is the fluid density, (u, v) are the Cartesian fluid velocity components, p is the pressure, and E_t is the total energy. For an ideal gas, the equation of state relates the pressure to total energy by

$$p = (\gamma - 1)\left[E_t - \tfrac{1}{2}\rho(u^2 + v^2)\right] \tag{3}$$

where $\gamma = 1.4$ is the ratio of specific heats. Applying the divergence theorem and integrating over a moving control volume $A(t)$ that is bounded by the control surface $B(t)$ yields

$$\int_{A(t)} \frac{\partial \mathbf{U}}{\partial t}\,\mathrm{d}A + \int_{B(t)} \mathbf{F}(\mathbf{U})\cdot\mathbf{n}\,\mathrm{d}B = 0 \tag{4}$$

Using the differential identity

$$\frac{\partial}{\partial t}\int_{A(t)} \mathbf{U}\,\mathrm{d}A = \int_{A(t)} \frac{\partial \mathbf{U}}{\partial t}\,\mathrm{d}A + \int_{B(t)} (\dot{\mathbf{x}}\cdot\mathbf{n})\,\mathrm{d}B \tag{5}$$

Equation (4) is rewritten as

$$\frac{\partial}{\partial t}\int_{A(t)} \mathbf{U}\,\mathrm{d}A + \int_{B(t)} [\mathbf{F}(\mathbf{U}) - \dot{\mathbf{x}}\mathbf{U}]\cdot\mathbf{n}\,\mathrm{d}B = 0 \tag{6}$$

or, when considering cell-averaged values for the state $\mathbf{U}$, as

$$\frac{\partial A\mathbf{U}}{\partial t} + \int_{B(t)} [\mathbf{F}(\mathbf{U}) - \dot{\mathbf{x}}\mathbf{U}]\cdot\mathbf{n}\,\mathrm{d}B = 0 \tag{7}$$

This is the arbitrary Lagrangian–Eulerian (ALE) finite volume form of the Euler equations. The equations are required in ALE form because the problem involves deforming meshes in which mesh elements change in shape and size at each time step. Here, A refers to the area or volume of the element, $\dot{\mathbf{x}}$ is the vector of mesh face or edge velocities, $\mathbf{n}$ is the unit normal of the face or edge, and B refers to the area or length of the bounding surface or edge.

B. Temporal Discretization

The time derivative term in the Euler equations is discretized using either a first-order accurate backward-difference formula (BDF1) or a second-order accurate (BDF2) scheme. These discretizations are shown in Eqs. (8) and (9), respectively. The index n is used to indicate the current time level as the convention throughout the paper. The discretization of the BDF2 scheme shown in Eq. (9) is based on a uniform time-step size.

$$\frac{\partial A\mathbf{U}}{\partial t} = \frac{A^n\mathbf{U}^n - A^{n-1}\mathbf{U}^{n-1}}{\Delta t} \tag{8}$$

$$\frac{\partial A\mathbf{U}}{\partial t} = \frac{\frac{3}{2}A^n\mathbf{U}^n - 2A^{n-1}\mathbf{U}^{n-1} + \frac{1}{2}A^{n-2}\mathbf{U}^{n-2}}{\Delta t} \tag{9}$$

C. Spatial Discretization

The solver uses a cell-centered finite volume formulation in which the inviscid flux integral around a closed control volume is discretized as

$$\int_{\mathrm{d}B(t)} [F(\mathbf{U}) - \dot{\mathbf{x}}\mathbf{U}]\cdot\mathbf{n}\,\mathrm{d}B = \mathbf{S}_{\mathrm{cell}} = \sum_{i=1}^{n_{\mathrm{edge}}} F_i^{\perp}B_i - V_{e_i}\mathbf{U}_{\mathrm{cell}}B_i \tag{10}$$

where, for a 2-D problem, B_i is the edge length, V_{e_i} is the normal edge velocity, and $F_i^{\perp}$ is the normal flux across the edge. The normal flux across the edge is computed as

$$\mathbf{F}^{\perp}_{\mathrm{edge}} = \tfrac{1}{2}\left\{\mathbf{F}_L^{\perp}(\mathbf{q}_L) + \mathbf{F}_R^{\perp}(\mathbf{q}_R) + \left[\hat{A}\right](\mathbf{q}_L - \mathbf{q}_R)\right\} \tag{11}$$

where $\mathbf{q}_L$ and $\mathbf{q}_R$ are the left and right extrapolated state vectors, and $\mathbf{F}_L^{\perp}$ and $\mathbf{F}_R^{\perp}$ are the left and right normal fluxes for the edge computed as

$$\mathbf{F}^{\perp} = \begin{pmatrix} \rho V^{\perp} \\ \rho V^{\perp}u + \hat{n}_x p \\ \rho V^{\perp}v + \hat{n}_y p \\ (E_t + p)V^{\perp} \end{pmatrix} \tag{12}$$

The velocity normal to the edge $V^{\perp}$ is defined as $u\hat{n}_x + v\hat{n}_y$, where $\hat{n}_x$ and $\hat{n}_y$ are the unit edge normal vector components. The dissipative flux is computed using Roe's approximate Riemann solver [19], and, therefore, matrix $[\hat{A}]$ is the Roe-averaged flux Jacobian. Second-order spatial accuracy is achieved by using a gradient-based reconstruction of the state vector at the edges based on cell-centered values. The extrapolated state $\mathbf{q}$ is calculated as

$$\mathbf{q} = \mathbf{U} + \nabla\mathbf{U}\cdot\mathbf{r} \tag{13}$$

Here, $\mathbf{r}$ is the vector between the cell centroid and the edge midpoint, and $\nabla\mathbf{U}$ is the gradient of the state vector. The gradient is computed by first obtaining nodal state values via arithmetic averages of cell-centered values from elements that share the node, and then using Green–Gauss reconstruction to obtain the state gradients at the cell centers. An arithmetic average is used rather than more accurate polynomial-based interpolation methods to simplify the linearization procedure. No limiting procedure is employed in the reconstruction scheme as shown in Eq. (13).

D. Discrete Geometric Conservation Law

The discrete geometric conservation law (GCL) requires that a uniform flowfield be preserved when Eq. (7) is integrated in time. In other words, the deformation of the computational mesh should not introduce conservation errors in the solution of the flow problem. This translates into $\mathbf{U} = \mathrm{const}$ being an exact solution of Eq. (7). For a conservative scheme, the integral of the inviscid fluxes around a closed contour goes to zero when $\mathbf{U} = \mathrm{const}$. Applying these conditions to Eq. (7) results in the mathematical description of the GCL as stated here:

$$\frac{\partial A}{\partial t} - \int_{dB(t)} \dot{\mathbf{x}}\cdot\mathbf{n}\,\mathrm{d}B = 0 \tag{14}$$

For a first-order BDF1 time-integration scheme, this can be discretely represented using Eqs. (8) and (10) as

$$\left(\frac{A^n - A^{n-1}}{\Delta t}\right) - \sum_{i=1}^{n_{\text{edge}}} V_{e_i} B_i = 0 \tag{15}$$

Equation (15) implies that the change in area of an element in the mesh should be discretely equal to the area swept by the bounding edges of the element. The edge velocities V_{e_i} for each of the edges encompassing the element must therefore be chosen such that Eq. (15) is satisfied. Whereas various methods for computing the edge velocities satisfying the GCL have been developed for different temporal discretizations [17,20], a unifying approach applicable to first-, second-, and third-order BDF schemes, as well as higher-order accurate implicit Runge–Kutta (IRK) schemes, has been developed in [21]. This formulation is used exclusively in the current work. While the details of the formulation are not central to the current work, the functional form of the face-integrated edge velocities is important for the derivation of the adjoint equations. In our formulation, these values depend on the mesh coordinates $\mathbf{x}^n$ and $\mathbf{x}^{n-1}$ for BDF1 and $\mathbf{x}^n$, $\mathbf{x}^{n-1}$, and $\mathbf{x}^{n-2}$ for BDF2.

E. Mesh Deformation Strategy

Deformation of the mesh is achieved through the linear tension spring analogy [7,22], which approximates the mesh as a network of interconnected springs. The spring coefficient is assumed to be inversely proportional to the edge length. Two independent force balance equations are formulated for each node based on displacements of neighbors. This results in a nearest-neighbor stencil for the final linear system to be solved. The linear system that relates the interior node displacements in the mesh to known displacements on the boundaries is

$$[K]\delta\mathbf{x}_{\text{int}} = \delta\mathbf{x}_{\text{surf}} \tag{16}$$

where $[K]$ is the stiffness matrix assembled using the spring coefficients of each of the edges in the computational mesh.

F. Geometry Parameterization

Modification of the baseline geometry is achieved through displacements of the surface nodes defining the geometry. To ensure smooth geometry shapes, any displacement of a surface node is controlled by a bump function, which influences neighboring nodes with an effect that diminishes moving away from the displaced node. The bump function used for the work presented in this paper is the Hicks–Henne sine bump function [23]. The design variables or inputs for the optimization examples presented form a vector of weights controlling the magnitude of bump functions placed at various chordwise locations.

G. Analysis Procedure

The example problems chosen to demonstrate the unsteady adjoint algorithm involve a deformable airfoil that pitches sinusoidally. The procedure to determine the unsteady flow solution for such a case is as follows:

1) Determine the surface coordinates (shape) of the deformed airfoil as

$$\mathbf{x}^0_{\text{surf}} = \mathbf{x}^{\text{base}}_{\text{surf}} + \delta\mathbf{x}^{\text{base}}_{\text{surf}} \tag{17}$$

where $\delta\mathbf{x}^{\text{base}}_{\text{surf}}$ are the displacements of the surface nodes computed as a function of the design variables $\mathbf{D}$. In this case, the function refers to the Hicks–Henne sine bump function. The variable $\mathbf{x}^{\text{base}}_{\text{surf}}$ refers to the surface coordinates of the baseline nondeformed airfoil.

2) Solve mesh motion equations to determine displacements of interior nodes $\delta\mathbf{x}^0_{\text{int}}$. Add to interior node coordinates of baseline mesh to get the interior node distribution for the deformed airfoil as $\mathbf{x}^0_{\text{int}} = \mathbf{x}^{\text{base}}_{\text{int}} + \delta\mathbf{x}^0_{\text{int}}$.

3) Begin time integration at time $n = 0$.

4) Transform the deformed surface coordinates to their new orientation for the current time level n through multiplication of the rotation matrix $[\beta]^n$, which prescribes the motion of the pitching airfoil as a function of time. The center of rotation for the airfoil is taken to be the quarter-chord location. The rotation matrix $[\beta]^n$ is a function of the angle of attack at the current time level and can be determined using the sine function shown in Eq. (18). The transformed airfoil is then used to compute the surface displacement vector $\delta\mathbf{x}^n_{\text{surf}}$ for the current time level n as shown in Eq. (19).

$$\alpha^n = \alpha_0 + \alpha_{\max}\sin(\omega t^n) \tag{18}$$

$$\delta\mathbf{x}^n_{\text{surf}} = ([\beta]^n - [I])\mathbf{x}^0_{\text{surf}} \tag{19}$$

5) Solve the mesh motion equations again to obtain the interior node displacements $\delta\mathbf{x}^n_{\text{int}}$ at time level n. The interior node coordinates are then computed as $\mathbf{x}^n_{\text{int}} = \mathbf{x}^0_{\text{int}} + \delta\mathbf{x}^n_{\text{int}}$, where $\mathbf{x}^0_{\text{int}}$ are the interior node coordinates for the deformed airfoil at the time level $n = 0$.

6) Calculate edge velocities V_e in accordance with the GCL for the chosen time-integration scheme. For a BDF1 scheme, V_e is a function of coordinates $\mathbf{x}^n_{\text{int}}$ and $\mathbf{x}^{n-1}_{\text{int}}$, whereas, for a BDF2 scheme, V_e additionally depends on the coordinate values $\mathbf{x}^{n-2}_{\text{int}}$. Further details can be found in [21]. New cell areas A^n are also computed at this point using $\mathbf{x}^n_{\text{int}}$.

7) For an implicit time step, the nonlinear flow residual is defined as

$$\mathbf{R}^n = \frac{\partial(A\mathbf{U})}{\partial t} + \mathbf{S}(\mathbf{V}^n_e, \mathbf{n}^n, \mathbf{U}^n) = 0 \tag{20}$$

where the discretization of the time derivative is based on the chosen time-integration scheme. The second term is the spatial residual of the flow and is a function of the edge velocities $\mathbf{V}_e$, the edge normals $\mathbf{n}$, and the flow solution at the current time level $\mathbf{U}^n$.

8) Solve the nonlinear flow residual using Newton's method as shown in Eq. (21) in conjunction with an agglomeration multigrid scheme to accelerate convergence of the intermediate linear systems.

$$\left[\frac{\partial\mathbf{R}(\mathbf{U}^k)}{\partial\mathbf{U}^k}\right]\delta\mathbf{U}^k = -\mathbf{R}(\mathbf{U}^k) \qquad \mathbf{U}^{k+1} = \mathbf{U}^k + \delta\mathbf{U}^k \tag{21}$$

9) If the current time level is $n = 0$, obtain a steady-state solution for use as an initial condition by solving $\mathbf{S}(\mathbf{n}^n_e, \mathbf{U}^n) = 0$.

10) Compute lift and drag coefficients C^n_L and C^n_D for the current time level. The load coefficients are functions of both the current flow solution and the current mesh coordinates.

This analysis capability is very similar to previously developed unsteady Euler solvers and has been validated through mesh refinement and temporal refinement studies and comparisons with the two-dimensional unstructured mesh solver described in [24].

III. Sensitivity Formulation for Unsteady Flow Problem

A. Functional Dependence

The time-integrated objective functional L^g that is to be minimized can be written with intermediate dependencies as a function of the design variables $\mathbf{D}$ to assist with the chain rule differentiation process. For the problem of the unsteady pitching airfoil, the design variables form a vector of weights controlling the magnitudes of bumps placed at various surface node or chordwise locations. The intermediate dependencies for a flow solver based on a BDF1 temporal discretization can be written as

$$L^g = F\left[L^0, L^1, L^2, \ldots, L^n_f\right] \tag{22}$$

$$L^n = F\left[\mathbf{U}^n, \mathbf{x}^n_{\text{int}}\right] \tag{23}$$

$$\mathbf{U}^n = F\left[\mathbf{U}^{n-1}, \mathbf{x}^n_{\text{int}}, \mathbf{x}^{n-1}_{\text{int}}\right] \tag{24}$$

$$\mathbf{U}^n, \mathbf{U}^{n-1}, \mathbf{U}^{n-2} \cdots = F[\mathbf{D}] \tag{25}$$

$$\mathbf{x}^{n}_{\text{int}}, \mathbf{x}^{n-1}_{\text{int}}, \mathbf{x}^{n-2}_{\text{int}} \cdots = F\left[\mathbf{x}^{0}_{\text{surf}}\right] \tag{26}$$

$$\mathbf{x}^{0}_{\text{surf}} = F[\mathbf{D}] \tag{27}$$

where $L^n, L^{n-1} \cdots$ denote local functional values at each time level. Sequentially differentiating Eqs. (22–27) results in a general expression for the final sensitivity derivative $\mathrm{d}L^g/\mathrm{d}\mathbf{D}$ for the unsteady flow problem

$$\frac{\mathrm{d}L^g}{\mathrm{d}\mathbf{D}} = \sum_{n=0}^{n=n_f} \frac{\partial L^g}{\partial L^n}\left[\frac{\partial L^n}{\partial \mathbf{U}^n}\frac{\partial \mathbf{U}^n}{\partial \mathbf{D}} + \frac{\partial L^n}{\partial \mathbf{x}^n_{\text{int}}}\frac{\partial \mathbf{x}^n_{\text{int}}}{\partial \mathbf{D}}\right] \tag{28}$$

where several of the intermediate derivatives are not shown for simplicity.

B. Tangent or Forward Linearization

Consider the case of a single design variable D. Equations (16) and (20) form the constraint equations for the sensitivity problem and can be differentiated with respect to the design variable D in order to obtain convenient expressions for both the flow variable sensitivity $(\partial \mathbf{U}/\partial D)$ and the mesh-coordinate sensitivity $(\partial \mathbf{x}_{\text{int}}/\partial D)$ that appear in Eq. (28). For a BDF1 time discretization, the flow constraint equation (i.e., the nonlinear flow residual at a given time level) can be written as

$$\mathbf{R}^{n}\left(\mathbf{x}^n_{\text{int}}, \mathbf{x}^{n-1}_{\text{int}}, \mathbf{U}^n, \mathbf{U}^{n-1}\right) = 0 \tag{29}$$

Differentiating this with respect to the design variable D yields

$$\frac{\mathrm{d}\mathbf{R}^n}{\mathrm{d}D} = \frac{\partial \mathbf{R}^n}{\partial \mathbf{x}^n_{\text{int}}}\frac{\partial \mathbf{x}^n_{\text{int}}}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{x}^{n-1}_{\text{int}}}\frac{\partial \mathbf{x}^{n-1}_{\text{int}}}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^n}\frac{\partial \mathbf{U}^n}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^{n-1}}\frac{\partial \mathbf{U}^{n-1}}{\partial D} = 0 \tag{30}$$

Rearranging the equation in terms of the flow Jacobian results in an expression for the flow variable sensitivity as

$$\left[\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^n}\right]\frac{\partial \mathbf{U}^n}{\partial D} = -\left(\frac{\partial \mathbf{R}^n}{\partial \mathbf{x}^n_{\text{int}}}\frac{\partial \mathbf{x}^n_{\text{int}}}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{x}^{n-1}_{\text{int}}}\frac{\partial \mathbf{x}^{n-1}_{\text{int}}}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^{n-1}}\frac{\partial \mathbf{U}^{n-1}}{\partial D}\right) \tag{31}$$

The second constraint is the set of mesh motion equations. Differentiating these with respect to the design variable produces an expression for the mesh-coordinate sensitivity as

$$[K]\frac{\partial \mathbf{x}^n_{\text{int}}}{\partial D} = \frac{\partial \mathbf{x}^n_{\text{surf}}}{\partial D} \tag{32}$$

The stiffness matrix $[K]$ is based on the initial configuration of the mesh and does not change through the time integration or design optimization process and can therefore be treated as a constant in the differentiation. The procedure to obtain the sensitivity $\mathrm{d}L/\mathrm{d}D$ using the forward linearization can be summarized as follows:

1) Begin time-integration loop at $n = 0$.

2) Determine the surface node sensitivities $\partial \mathbf{x}^0_{\text{surf}}/\partial D$ by differentiating the shape functions.

3) Multiply the surface node sensitivities with the rotation matrix $[\beta]^n$ to obtain the rotated surface node sensitivities for the current time level n.

4) Iteratively compute the mesh-coordinate sensitivities for the current time level as shown in Eq. (32).

5) Compute the flow variable sensitivities iteratively as shown in Eq. (31), in which the flow variable and the mesh-coordinate sensitivities from the previous time level are known quantities. In the case of the first time step, these are the outputs from the steady-state sensitivity solution (if the unsteady solution is restarted from the steady solution), or these are derivatives of the initial condition for the unsteady flow. The initial condition may be assumed to be uniform flow to remove dependence on the design variable D.

6) Continue integrating forward in time, performing matrix–vector or vector–vector inner products as per Eq. (28). At the end of the time-integration process, the complete sensitivity of the global objective functional L^g to the design variable D is available.

C. Adjoint or Reverse Linearization

For problems with multiple design variables and a single objective functional, Eq. (28) is transposed to obtain the adjoint or reverse linearization. Shape optimization problems require the adjoint linearization to obtain sensitivities because they typically involve numerous design variables and one or few objective functionals. The transpose of the forward linearization can be written as

$$\frac{\mathrm{d}L^{g^T}}{\mathrm{d}\mathbf{D}} = \sum_{n=0}^{n=n_f}\left[\frac{\partial \mathbf{U}^{n^T}}{\partial \mathbf{D}}\frac{\partial L^{n^T}}{\partial \mathbf{U}^n} + \frac{\partial \mathbf{x}^{n^T}_{\text{int}}}{\partial \mathbf{D}}\frac{\partial L^{n^T}}{\partial \mathbf{x}^n_{\text{int}}}\right]\frac{\partial L^{g^T}}{\partial L^n} \tag{33}$$

Expanding the summation backward,

$$\frac{\mathrm{d}L^{g^T}}{\mathrm{d}\mathbf{D}} = \underbrace{\left(\frac{\partial \mathbf{U}^{n^T}}{\partial \mathbf{D}}\frac{\partial L^{n^T}}{\partial \mathbf{U}^n} + \frac{\partial \mathbf{x}^{n^T}_{\text{int}}}{\partial \mathbf{D}}\frac{\partial L^{n^T}}{\partial \mathbf{x}^n_{\text{int}}}\right)\frac{\partial L^{g^T}}{\partial L^n}}_{(X)}$$
$$+ \underbrace{\left(\frac{\partial \mathbf{U}^{n-1^T}}{\partial \mathbf{D}}\frac{\partial L^{n-1^T}}{\partial \mathbf{U}^{n-1}} + \frac{\partial \mathbf{x}^{n-1^T}_{\text{int}}}{\partial \mathbf{D}}\frac{\partial L^{n-1^T}}{\partial \mathbf{x}^{n-1}_{\text{int}}}\right)\frac{\partial L^{g^T}}{\partial L^{n-1}}}_{(Y)}\cdots \tag{34}$$

Considering only term (X) in the expansion, we can now rearrange and transpose Eq. (31) to determine an expression that can be substituted for the flow variable sensitivity. Term (X) can now be rewritten as

$$(X) = -\left(\frac{\partial \mathbf{x}^{n^T}_{\text{int}}}{\partial \mathbf{D}}\frac{\partial \mathbf{R}^{n^T}}{\partial \mathbf{x}^n_{\text{int}}} + \frac{\partial \mathbf{x}^{n-1^T}_{\text{int}}}{\partial \mathbf{D}}\frac{\partial \mathbf{R}^{n^T}}{\partial \mathbf{x}^{n-1}_{\text{int}}} + \frac{\partial \mathbf{U}^{n-1^T}}{\partial \mathbf{D}}\frac{\partial \mathbf{R}^{n^T}}{\partial \mathbf{U}^{n-1}}\right)$$
$$\times\left[\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^n}\right]^{-T}\frac{\partial L^{n^T}}{\partial \mathbf{U}^n}\frac{\partial L^{g^T}}{\partial L^n} + \frac{\partial \mathbf{x}^{n^T}_{\text{int}}}{\partial \mathbf{D}}\frac{\partial L^{n^T}}{\partial \mathbf{x}^n_{\text{int}}}\frac{\partial L^{g^T}}{\partial L^n} \tag{35}$$

Here, $\partial L^g/\partial L^n$ is a scalar quantity, while $\partial L^n/\partial \mathbf{U}^n$ and $\partial L^n/\partial \mathbf{x}^n_{\text{int}}$ are vectors that are easily computable because L^n is a scalar quantity. These are merely linearizations with respect to the flow variables and the mesh coordinates. From Eq. (35), it is clear that the inverse of the transposed flow Jacobian matrix is required to proceed with the sensitivity computation. Because direct inversions of the flow Jacobian matrix are to be avoided, we introduce a flow adjoint variable defined as per Eq. (36) to enable an iterative approach:

$$\Lambda^n_{\mathrm{U}} = -\left[\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^n}\right]^{-T}\left[\frac{\partial L^n}{\partial \mathbf{U}^n}\right]^T\frac{\partial L^{g^T}}{\partial L^n} \tag{36}$$

This is a linear system that can be iteratively solved as

$$\left[\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^n}\right]^T\Lambda^n_{\mathrm{U}} = -\left[\frac{\partial L^n}{\partial \mathbf{U}^n}\right]^T\frac{\partial L^{g^T}}{\partial L^n} \tag{37}$$

The convergence of the linear system in Eq. (37) is similar to the convergence of any single linear system arising while solving the nonlinear flow constraint equation because both problems contain the same eigenvalues. Solving for the flow adjoint and substituting into Eq. (35) results in three terms: $(A)^n$, $(B)^n$, and $(C)^n$. The remaining preexisting contribution to term (X) is denoted term $(E)^n$.

$$(X) = \underbrace{\frac{\partial \mathbf{x}^{n^T}_{\text{int}}}{\partial \mathbf{D}}\frac{\partial \mathbf{R}^{n^T}}{\partial \mathbf{x}^n_{\text{int}}}\Lambda^n_{\mathrm{U}}}_{(A)^n} + \underbrace{\frac{\partial \mathbf{x}^{n-1^T}_{\text{int}}}{\partial \mathbf{D}}\frac{\partial \mathbf{R}^{n^T}}{\partial \mathbf{x}^{n-1}_{\text{int}}}\Lambda^n_{\mathrm{U}}}_{(B)^n} + \underbrace{\frac{\partial \mathbf{U}^{n-1^T}}{\partial \mathbf{D}}\frac{\partial \mathbf{R}^{n^T}}{\partial \mathbf{U}^{n-1}}\Lambda^n_{\mathrm{U}}}_{(C)^n}$$
$$+ \underbrace{\frac{\partial \mathbf{x}^{n^T}_{\text{int}}}{\partial \mathbf{D}}\frac{\partial L^{n^T}}{\partial \mathbf{x}^n_{\text{int}}}\frac{\partial L^{g^T}}{\partial L^n}}_{(E)^n} \tag{38}$$

Terms $(A)^n$ and $(E)^n$ can be combined and expanded in terms of the surface displacement sensitivity of the deformed airfoil as follows:

$$(A)^n + (E)^n = \frac{\partial \mathbf{x}_{\text{surf}}^{0^T}}{\partial \mathbf{D}} \frac{\partial \mathbf{x}_{\text{surf}}^{n^T}}{\partial \mathbf{x}_{\text{surf}}^{0}} \frac{\partial \mathbf{x}_{\text{int}}^{n^T}}{\partial \mathbf{x}_{\text{surf}}^{n}} \left\{ \frac{\partial \mathbf{R}^{n^T}}{\partial \mathbf{x}_{\text{int}}^{n}} \Lambda_{\mathbf{U}}^n + \frac{\partial L^{n^T}}{\partial \mathbf{x}_{\text{int}}^{n}} \frac{\partial L^{g^T}}{\partial L^n} \right\} \quad (39)$$

where the second term is the transpose of the rotation matrix $[\beta]^n$, and the third term can be written in terms of Eq. (32) as

$$\frac{\partial \mathbf{x}_{\text{int}}^{n^T}}{\partial \mathbf{x}_{\text{surf}}^{n}} = [K]^{-T} \quad (40)$$

Substituting Eq. (40) into the combination of terms $(A)^n$ and $(E)^n$, we can introduce a mesh adjoint variable $\Lambda_{\mathbf{x}}^n$ to avoid a direct inversion of the stiffness matrix $[K]$ as

$$\Lambda_{\mathbf{x}}^n = [K]^{-T} \left\{ \frac{\partial \mathbf{R}^{n^T}}{\partial \mathbf{x}_{\text{int}}^{n}} \Lambda_{\mathbf{U}}^n + \frac{\partial L^{n^T}}{\partial \mathbf{x}_{\text{int}}^{n}} \frac{\partial L^{g^T}}{\partial L^n} \right\} \quad (41)$$

This is again a linear system that can be iteratively solved as

$$[K]^T \Lambda_{\mathbf{x}}^n = \left\{ \frac{\partial \mathbf{R}^{n^T}}{\partial \mathbf{x}_{\text{int}}^{n}} \Lambda_{\mathbf{U}}^n + \frac{\partial L^{n^T}}{\partial \mathbf{x}_{\text{int}}^{n}} \frac{\partial L^{g^T}}{\partial L^n} \right\} \quad (42)$$

Here, the term $(\partial \mathbf{R}^{n^T}/\partial \mathbf{x}_{\text{int}}^n)\Lambda_{\mathbf{U}}^n$ transfers the effect of the flow adjoint variable residing at the cell center to the mesh coordinates, which are nodal quantities. This occurs first through the linearization of the cell-centered flow residual with respect to the flux integral around the edges of that cell and then via the linearization of the flux across each edge with respect to the edge normal from which it gets transferred to the nodes defining the edge normal. As in the case of the forward linearization, the surface displacement sensitivity is found by differentiating the shape functions and can be precomputed and stored. The first contribution to the total sensitivity vector $\mathrm{d}L/\mathrm{d}\mathbf{D}$ can be now be determined simply by

$$\frac{\mathrm{d}L^T}{\mathrm{d}\mathbf{D}} = \frac{\partial \mathbf{x}_{\text{surf}}^{0^T}}{\partial \mathbf{D}} [\beta]^{n^T} \Lambda_{\mathbf{x}}^n + \cdots \quad (43)$$

Moving onto the remaining terms $(B)^n$ and $(C)^n$, we first gather terms from time level $n-1$ that appear in term (Y) and rewrite in combined form as

$$(B)^n + (C)^n + (Y) = \frac{\partial \mathbf{x}_{\text{int}}^{n-1^T}}{\partial \mathbf{D}} \left\{ \frac{\partial \mathbf{R}^{n^T}}{\partial \mathbf{x}_{\text{int}}^{n-1}} \Lambda_{\mathbf{U}}^n + \frac{\partial \mathbf{x}_{\text{int}}^{n-1^T}}{\partial \mathbf{D}} \frac{\partial L^{n-1^T}}{\partial \mathbf{x}_{\text{int}}^{n-1}} \right\} + \frac{\partial \mathbf{U}^{n-1^T}}{\partial \mathbf{D}} \left\{ \frac{\partial \mathbf{R}^{n^T}}{\partial \mathbf{U}^{n-1}} \Lambda_{\mathbf{U}}^n + \frac{\partial L^{n-1^T}}{\partial \mathbf{U}^{n-1}} \frac{\partial L^{g^T}}{\partial L^{n-1}} \right\} \quad (44)$$

We next write an expression analogous to Eq. (31) but at time level $n-1$ and then transpose and rearrange results in a convenient form for $\partial \mathbf{U}^{n-1^T}/\partial \mathbf{D}$, which appears in Eq. (44). We can now introduce a flow adjoint variable $\Lambda_{\mathbf{U}}^{n-1}$ at time level $n-1$ and solve for it as

$$\left[\frac{\partial \mathbf{R}^{n-1}}{\partial \mathbf{U}^{n-1}}\right]^T \Lambda_{\mathbf{U}}^{n-1} = -\left\{ \frac{\partial \mathbf{R}^{n^T}}{\partial \mathbf{U}^{n-1}} \Lambda_{\mathbf{U}}^n + \frac{\partial L^{n-1^T}}{\partial \mathbf{U}^{n-1}} \frac{\partial L^{g^T}}{\partial L^{n-1}} \right\} \quad (45)$$

This gives rise to new terms at time level $n-1$, namely $(A)^{n-1}$, $(B)^{n-1}$, and $(C)^{n-1}$, of which $(A)^{n-1}$ can be combined with the remaining mesh-coordinate term in Eq. (44). This now permits the definition of a mesh adjoint variable $\Lambda_{\mathbf{x}}^{n-1}$ at time level $n-1$ that can be solved for iteratively as

$$[K]^T \Lambda_{\mathbf{x}}^{n-1} = \left\{ \frac{\partial \mathbf{R}^{n-1^T}}{\partial \mathbf{x}_{\text{int}}^{n-1}} \Lambda_{\mathbf{U}}^{n-1} + \frac{\partial \mathbf{R}^{n^T}}{\partial \mathbf{x}_{\text{int}}^{n-1}} \Lambda_{\mathbf{U}}^n + \frac{\partial \mathbf{x}_{\text{int}}^{n-1^T}}{\partial \mathbf{D}} \frac{\partial L^{n-1^T}}{\partial \mathbf{x}_{\text{int}}^{n-1}} \right\} \quad (46)$$

Once the mesh adjoint variable $\Lambda_{\mathbf{x}}^{n-1}$ at time level $n-1$ is available, the sensitivity vector $\mathrm{d}L/\mathrm{d}\mathbf{D}$ can be augmented with the contribution from time level $n-1$ with the product of the mesh adjoint variable with the rotation matrix $[\beta]^{n-1^T}$ and the surface displacement sensitivity $\partial \mathbf{x}_{\text{surf}}^{0^T}/\partial \mathbf{D}$.

A recurrence pattern is clearly seen at this point. Assembling the complete sensitivity vector $\mathrm{d}L/\mathrm{d}\mathbf{D}$ involves starting at the final time level and sweeping backward in time by solving for a flow adjoint variable and a mesh adjoint variable at each time level. The forms of the flow adjoint equation and the mesh adjoint equation to be solved at each time level are similar to Eqs. (45) and (46) with the appropriate modifications to the time indices. The backward sweep or recurrence terminates at the first time level $n = 0$ through the solutions of steady-state adjoint equations for both the flow and mesh coordinates. At this point, the complete sensitivity vector $\mathrm{d}L/\mathrm{d}\mathbf{D}$ is available.

D. Modification of Formulation for BDF2 Time-Integration Scheme

The nature of the previously described derivation permits the application of the same technique regardless of the employed temporal discretization. For a second-order accurate two-step backward-difference formula, the constraint equation has contributions from time indices n, $n-1$, and $n-2$. The new flow constraint equation can be written as

$$\mathbf{R}^n\left(\mathbf{x}_{\text{int}}^n, \mathbf{x}_{\text{int}}^{n-1}, \mathbf{x}_{\text{int}}^{n-2}\mathbf{U}^n, \mathbf{U}^{n-1}, \mathbf{U}^{n-2}\right) = 0 \quad (47)$$

The flow variable sensitivity at time level n obtained through differentiation of Eq. (47) with respect to design variables $\mathbf{D}$ becomes

$$\frac{\partial \mathbf{U}^n}{\partial D} = -\left[\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^n}\right]^{-1} \left(\frac{\partial \mathbf{R}^n}{\partial \mathbf{x}_{\text{int}}^n} \frac{\partial \mathbf{x}_{\text{int}}^n}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{x}_{\text{int}}^{n-1}} \frac{\partial \mathbf{x}_{\text{int}}^{n-1}}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{x}_{\text{int}}^{n-2}} \frac{\partial \mathbf{x}_{\text{int}}^{n-2}}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^{n-1}} \frac{\partial \mathbf{U}^{n-1}}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^{n-2}} \frac{\partial \mathbf{U}^{n-2}}{\partial D} \right) \quad (48)$$

The linearization at time level n now has additional contributions from time level $n-2$. For the BDF1 scheme, the unsteady adjoint derivation resulted in a two-point recurrence relation in time involving the time indices n and $n-1$. For the BDF2 time-integration scheme, the recurrence relation is based on three points in time, namely indices n, $n-1$, and $n-2$. Equations (45) and (46) will now have additional source terms as a consequence of this. It is interesting to note that irrespective of the number of backward steps required in the time-integration scheme, only one flow adjoint and one mesh adjoint solution are required at each time level.

IV. Implementation Details

A. Code Structure

The software is divided into three main components, namely, the flow solver, the adjoint/tangent solver, and the optimization routine. The flow solver is an unstructured cell-centered unsteady finite volume code with moving mesh capability that is spatially and temporally second-order accurate. The flow solver performs the time-integration process to compute an unsteady flow solution that is then written to disk periodically at each time level. The adjoint or tangent solver reads the solution from disk and computes the required sensitivities. A simple driver program is used as the optimization controller and calls the LBFGS-B optimization routine, the flow solver, and the adjoint/tangent solver. All data exchange between the flow and adjoint/tangent solvers occur via files written to disk. This is necessary because the adjoint solver performs a backward integration in time and thus requires the entire solution history. It would be impractical to hold the entire unsteady solution set in memory for later use by the adjoint solver; however, file I/O operations represent negligible overheads with respect to overall solution time.

B. Linearization for Second-Order Spatial Accuracy

Computation of the flow adjoint variable requires the linearization of the flow constraint or flow residual equation with respect to the flow state variables. Such a linearization results in the flow Jacobian matrix $\partial \mathbf{R}/\partial \mathbf{U}$ and is used in the Newton solver employed for the flow solution. In the case of a spatially first-order accurate discretization, this results in a nearest-neighbor stencil in which the flow residual $\mathbf{R}$ for any element in the mesh is a function of its own state variables and the state variables of its immediate neighbors. On triangular unstructured meshes, this translates into the flow Jacobian matrix being a sparse matrix with each row consisting of a dominant diagonal block element and three off-diagonal block elements. A simple edge-based data structure is sufficient for the purpose of constructing and storing the elements of the flow Jacobian matrix. In the case of second-order spatial accuracy, the flow residual of each element is no longer restricted to a nearest-neighbor stencil because the residual $\mathbf{R}$ depends not only on the state variables but also their spatial gradients. This is can be seen in Eq. (13), describing the reconstruction scheme used for second-order accuracy in which $\nabla\mathbf{U}$ is the spatial gradient of the state vector $\mathbf{U}$. Because the state variable gradients are functions of element states extending beyond the nearest-neighbor stencil, constructing and storing an exact flow Jacobian matrix quickly becomes impractical due to large memory requirements and the lack of a simple data structure. For the flow solver, it is not required that an exact linearization of the flow residual $\mathbf{R}$ be available because only the solution of $\mathbf{R}(\mathbf{U}) = 0$ is necessary. To achieve second-order accuracy, it is only required that the flow residual itself be constructed using second-order extrapolations of $\mathbf{U}$. An inexact Newton's method is employed in which the first-order accurate flow Jacobian matrix is used in solving second-order accurate residual equations $\mathbf{R}_2(\mathbf{U})$. The method, being inexact, no longer exhibits the quadratic rate of convergence typical of a Newton solver but greatly simplifies the solution procedure.

In the case of the adjoint solver, it has been shown that approximating a second-order accurate flow Jacobian matrix with a first-order accurate matrix leads to significant errors in the computed sensitivities [25]. The linear systems that involve the flow Jacobian matrix are Eqs. (37) and (45). Although computing and storing the exact second-order Jacobian is impractical, it is quite straightforward to construct the product of the second-order Jacobian and a vector using a two-pass approach [7,8]. This property is taken advantage of, and a defect correction method is used to solve the linear system involving the second-order flow Jacobian matrix. For example, consider the left-hand side of the linear system shown in Eq. (37). In the case in which the flow Jacobian matrix is second-order accurate, this can be split into the sum of two matrix-vector products as follows:

$$\left[\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^n}\right]_2^T \Lambda_{\mathrm{U}}^n = \left\{\left[\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^n}\right]_1 + \left[\frac{\partial \mathbf{R}^n}{\partial \nabla\mathbf{U}^n}\right]\left[\frac{\partial \nabla\mathbf{U}^n}{\partial \mathbf{U}^n}\right]\right\}^T \Lambda_{\mathrm{U}}^n \tag{49}$$

$$= \underbrace{\left[\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^n}\right]_1^T \Lambda_{\mathrm{U}}^n}_{(1)} + \underbrace{\left[\frac{\partial \nabla\mathbf{U}^n}{\partial \mathbf{U}^n}\right]^T \left[\frac{\partial \mathbf{R}^n}{\partial \nabla\mathbf{U}^n}\right]^T \Lambda_{\mathrm{U}}^n}_{(2)} \tag{50}$$

where $[\partial \mathbf{R}^n/\partial \mathbf{U}^n]_1$ is the first-order Jacobian. Term (1) can be easily computed because the first-order Jacobian is already stored. Term (2) can be computed via a series of matrix-vector products on the fly if Λ_{U}^n is available. The defect correction method is similar to a nonlinear solution method and can be expressed mathematically as

$$[P]\delta\Lambda_{\mathrm{U}}^k = -\left\{\left[\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^n}\right]_2^T \Lambda_{\mathrm{U}}^k - \left[\frac{\partial L^n}{\partial \mathbf{U}^n}\right]^T\left[\frac{\partial L^g}{\partial L^n}\right]^T\right\} \tag{51}$$

Here, $[P]$ refers to the preconditioner and $\delta\Lambda_U$ the correction for Λ_U. When the correct value for Λ_U has been achieved, the right-hand side of Eq. (51) goes to zero. In order for the system to converge, the preconditioner $[P]$ must be closely related to the second-order flow Jacobian matrix. Typically, the preconditioner is taken to be the transpose of the first-order flow Jacobian matrix [8,26]. The right-hand side includes the effect of the second-order flow Jacobian matrix and is evaluated as described by Eq. (50).

C. Optimization Algorithm

The LBFGS-B optimization routine from [16] is used for all of the examples presented in this paper. The optimization algorithm is an extension of the LBFGS algorithm, which does not require bounds on the design variables. The bounded algorithm was chosen to prevent the generation of invalid geometries that would cause code crashes in the flow and adjoint solvers. A simple program was used to adaptively select the bounds by calling the mesh deformation routine with different combinations of design variables and checking for degenerate elements in the resulting mesh.

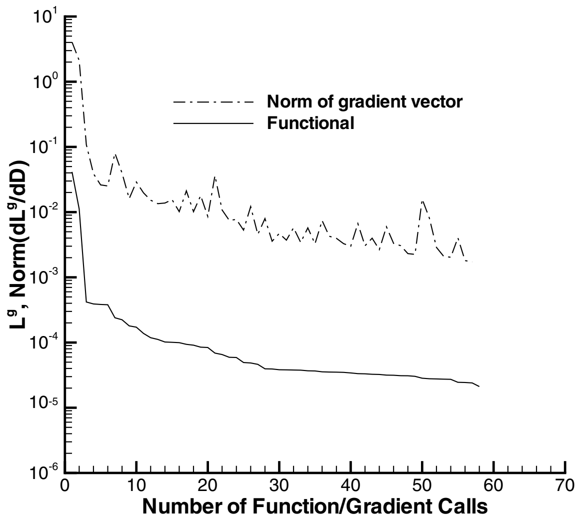


Fig. 2 Convergence of functional L^g and gradient for optimization case 1.

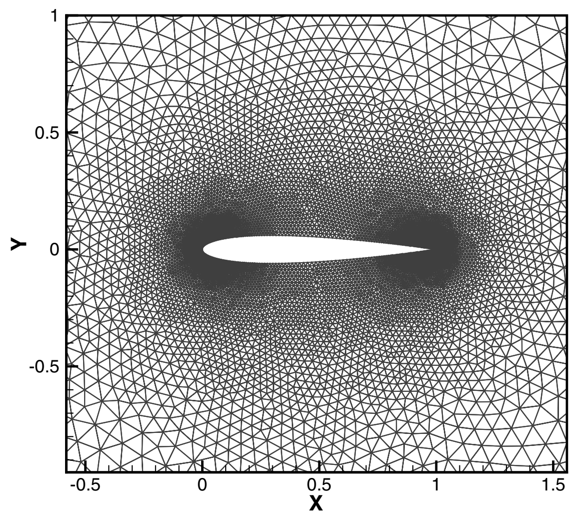


Fig. 1 Computational mesh of approximately 20,000 elements used in optimization examples.

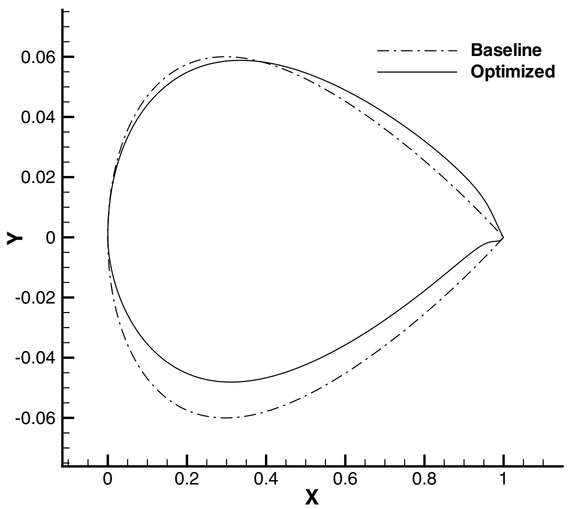
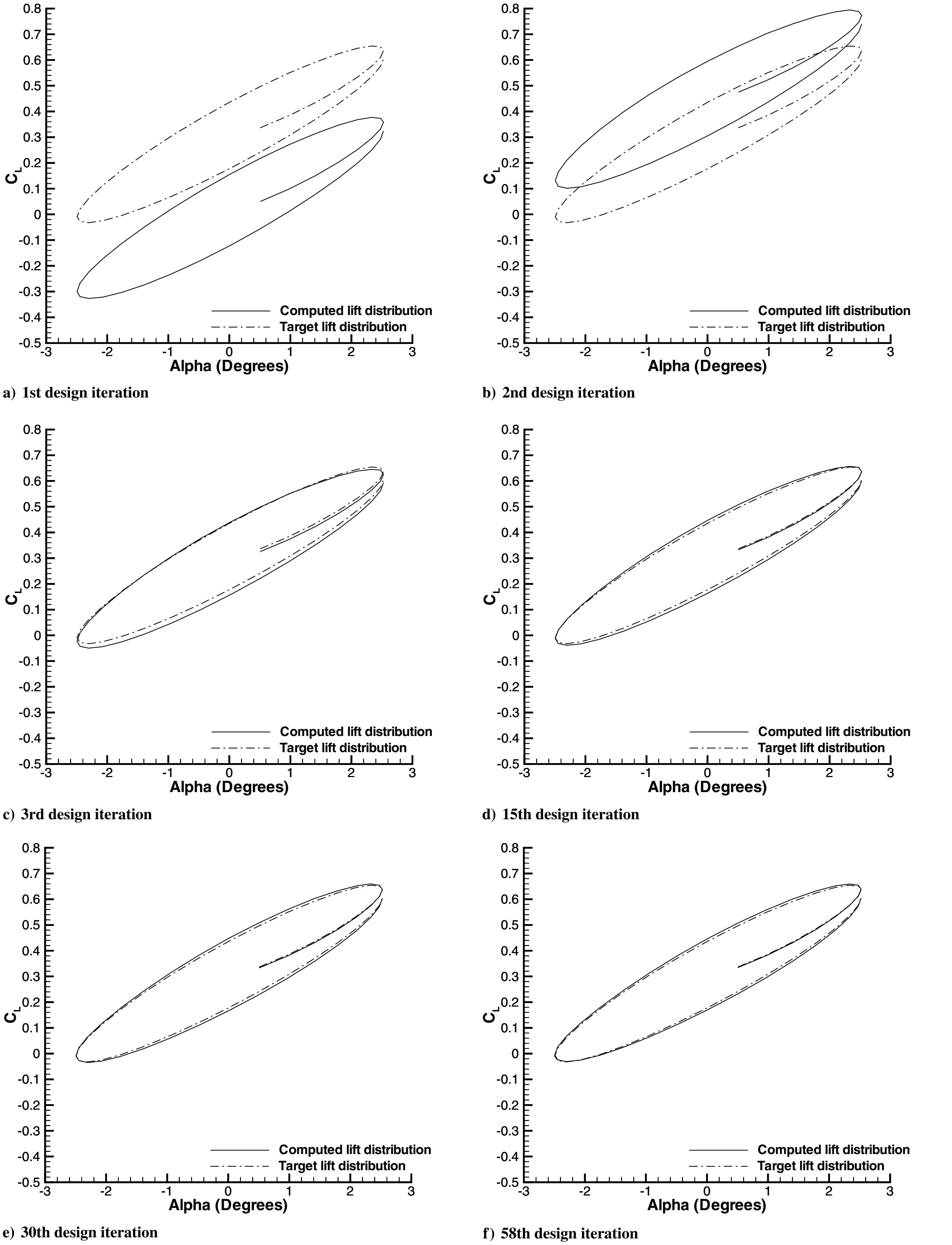


Fig. 3 Optimized airfoil for case 1 (exaggerated scale).

D. Verification of Adjoint Sensitivity

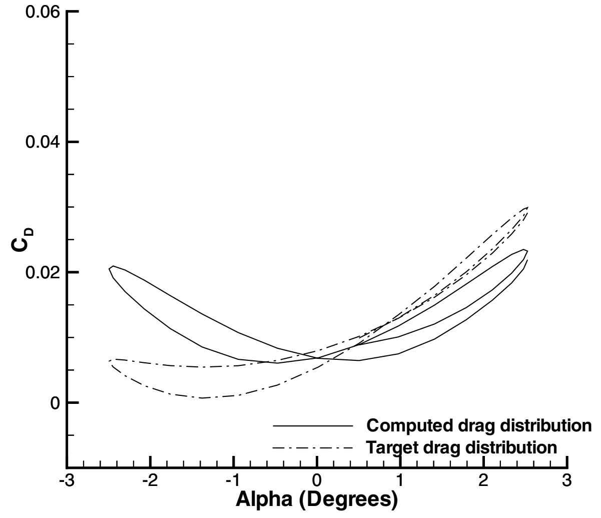
The procedure to validate the computed sensitivity is twofold. First, the sensitivity computed using the forward linearization is verified by comparing against finite differenced values. Then, duality [27] between transpose operations is invoked to assure that the sensitivity vector computed using the adjoint linearization matches that computed using the forward linearization. For finite differencing, each design variable is individually perturbed to record its effect on the global objective. Similarly, the forward linearization code is called $n_{\mathbf{D}}$ number of times to obtain the entire



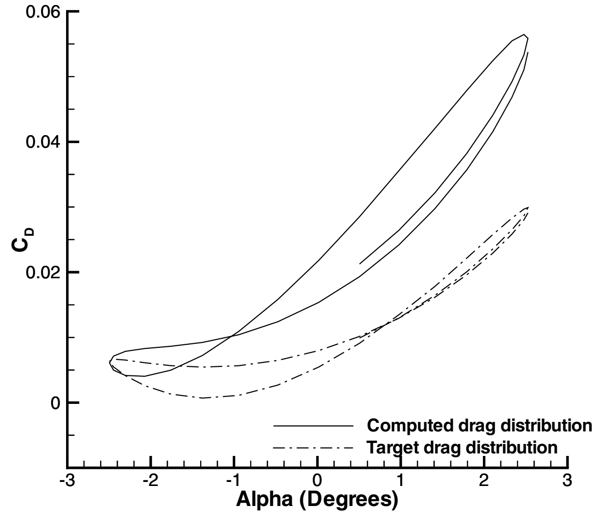
a) 1st design iteration
b) 2nd design iteration
c) 3rd design iteration
d) 15th design iteration
e) 30th design iteration
f) 58th design iteration

Fig. 4 Comparison of target and computed time-dependent lift profiles at various design iterations for case 1.

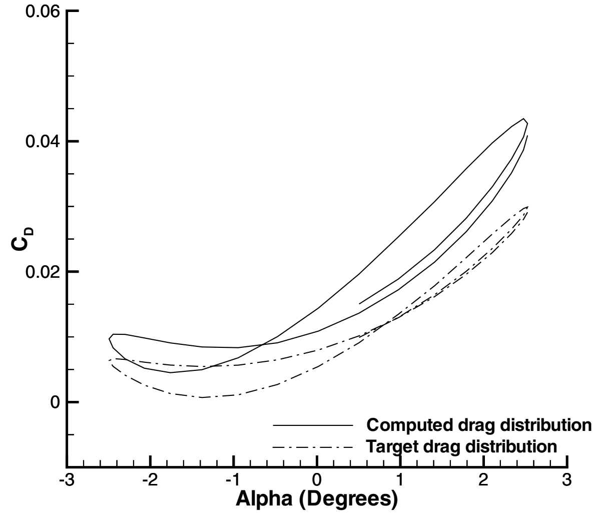
sensitivity vector. The finite differencing is done over a range of step sizes to obtain a converged value. The two-step procedure was employed over directly verifying the adjoint sensitivity with the finite differenced values because the forward linearization is far more intuitive than the adjoint linearization in which transposes of steps in the forward linearization are required. Determining the validity of the computed sensitivity and tracking down errors can be quite tedious when attempting to use the finite difference method to validate the adjoint solver directly. The gradients produced by the finite difference approach and the forward linearization approach were



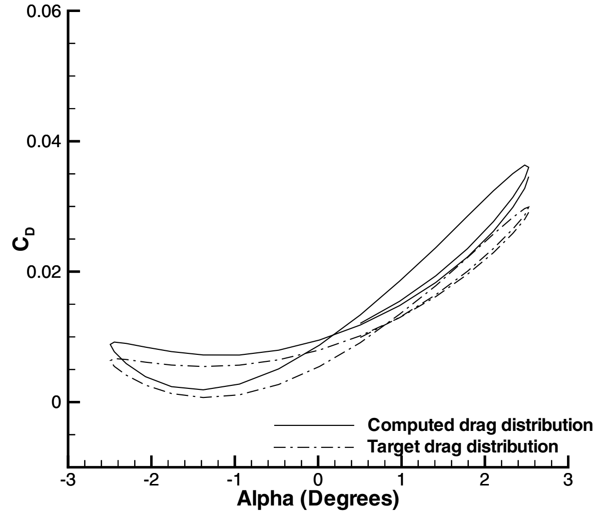
a) 1st design iteration



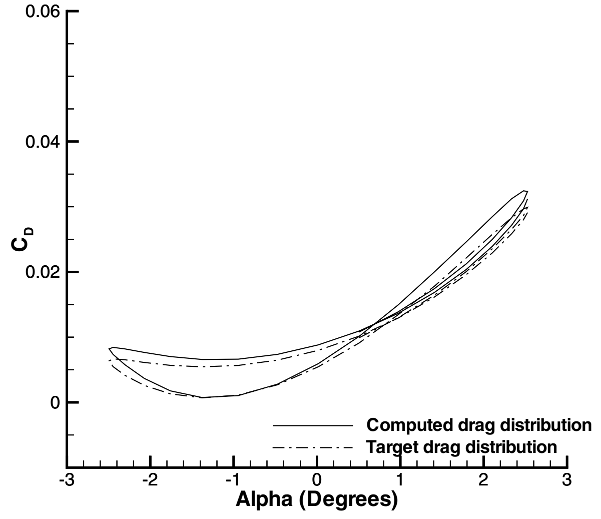
b) 2nd design iteration



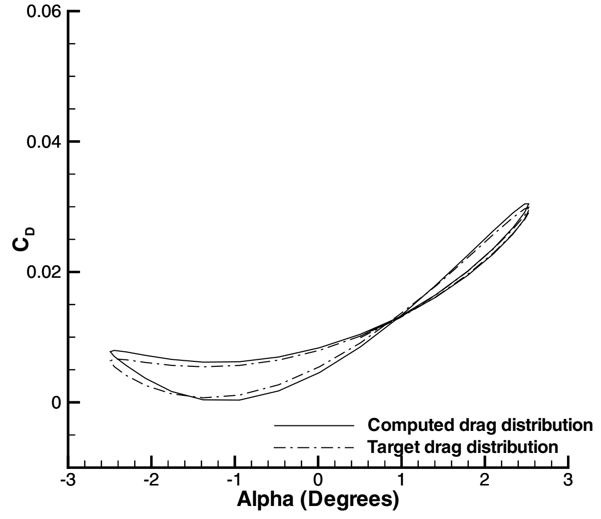
c) 3rd design iteration



d) 15th design iteration



e) 30th design iteration



f) 58th design iteration

Fig. 5 Comparison of target and computed time-dependent drag profiles at various design iterations for case 1.

verified in this manner to be in agreement to within four significant figures, whereas the forward linearization and adjoint values were verified to be in agreement to 15 significant figures, as expected.

V. Optimization Examples

Three test cases are presented to demonstrate the unsteady adjoint algorithm. All three involve the sinusoidal pitching of a NACA0012 airfoil about its quarter-chord location. The flow and pitch conditions are identical for all cases with a freestream Mach number of 0.755 combined with a mean angle of attack of 0.016 deg. The time-dependent pitch criteria consists of an amplitude of 2.51 deg and a reduced frequency of 0.0814. The computational mesh consists of approximately 20,000 triangular elements as shown in Fig. 1, with 290 surface nodes defining the geometry. Although the algorithm is applicable in the time domain, a periodic problem with initial transient behavior is chosen for demonstration purposes. As described earlier, the design variables form a vector of weights that control the magnitude of bump functions placed at various chordwise locations of the airfoil, which is to be deformed. The chordwise locations are chosen such that they coincide with the x-coordinate locations of the nodes defining the surface of the airfoil. In general, we place a single bump at each one of the surface node locations, and, because the computational mesh consists of 290 surface nodes, this translates into an equal number of design variables.

A. Case 1: Time-Dependent Load Matching

The goal of this test case is to optimize the shape of a NACA0012 airfoil such that the time-dependent load profile matches that of a sinusoidally pitching NACA64210 airfoil. The time integration for this example is carried out from a steady-state solution to a nondimensional time of 48.2431. The temporal resolution consists of 40 time steps. The time-integrated global objective functional is defined as the mean of local functionals at each time level as

$$L^g = \left(\frac{1}{n_f+1}\right)\sum_{n=0}^{n=n_f} L^n \tag{52}$$

and the local objective functional at each time level is the linear combination of the differences between target and computed loads defined as

$$L^n = \tfrac{1}{2}\left(C_L^n - C_{L\text{target}}^n\right)^2 + \tfrac{10}{2}\left(C_D^n - C_{D\text{target}}^n\right)^2 \tag{53}$$

where the factor of 10 for the drag coefficient is introduced to equalize the difference in order of magnitude between the lift and drag coefficients. Note that there is no constraint in the functional that controls the shape of the optimized airfoil. The only guarantee based on the formulation is that when the functional goes to zero, the target and computed time-dependent load profiles match. The shape is constrained in a loose sense only through the bounds placed on the design variables such that degenerate geometries do not result during the course of the optimization. As described earlier, these are computed a priori and are read in by the optimization routine at the onset of the optimization process. It is not necessary that the optimized airfoil match the shape of the NACA64210 airfoil, although it is one of the possible solutions to the optimization problem. The convergence of the time-integrated functional and the norm of the gradient vector are shown in Fig. 2. Figure 3 shows the optimized airfoil for this case. A pronounced hook shape close to the trailing edge of the optimized airfoil is visible from the figure. Since the target airfoil has finite lift at zero angle of attack, the baseline symmetric airfoil must be deformed such that it gains some degree of camber. The trailing-edge flow exit angle, being the most influential on turning the flow from the freestream direction (i.e., camber), has pushed the optimization in that direction. In reality, a severely curved trailing edge will typically lead to flow separation and loss of lift with increase in drag. This is not the case for the presented example because the flow and adjoint solvers are based on the Euler equations and do not include the effect of viscous terms. It should be interesting to observe the results of a shape-unconstrained optimization based on the Navier–Stokes equations in comparison with those presented here.

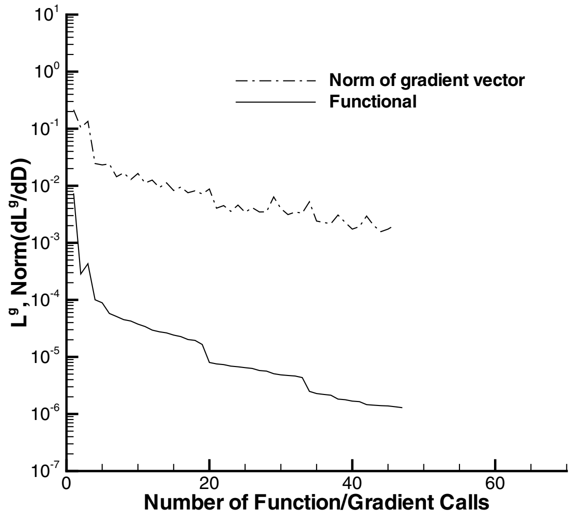


Fig. 7 Convergence of global objective function L^g for optimization case 2 using 290 design variables.

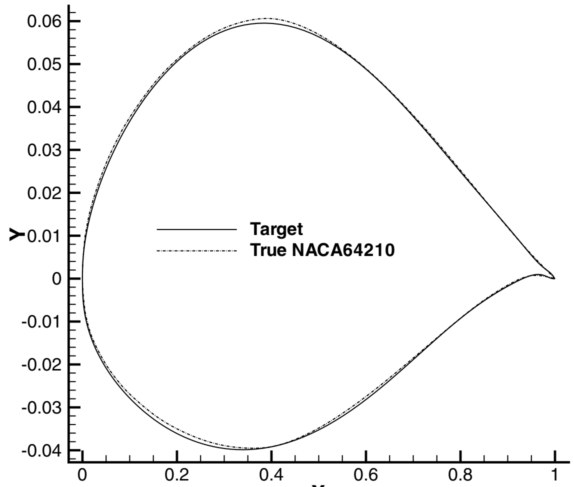


Fig. 6 Comparison of target airfoil and true NACA64210 airfoil.

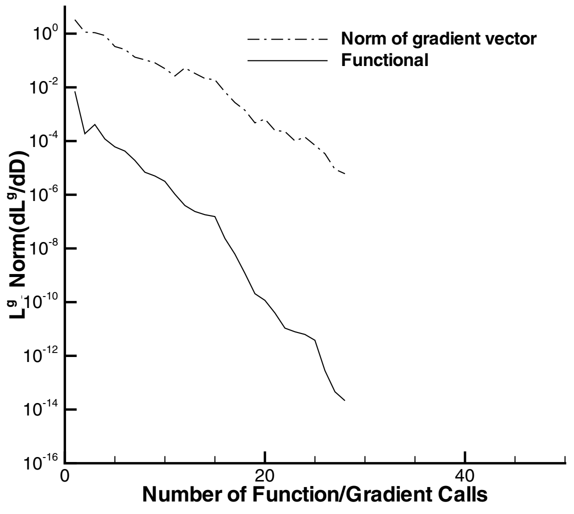
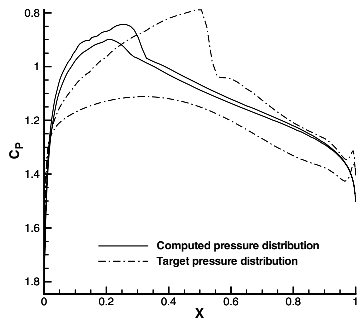
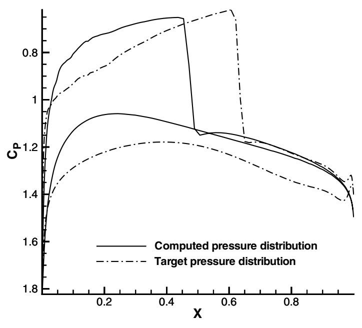


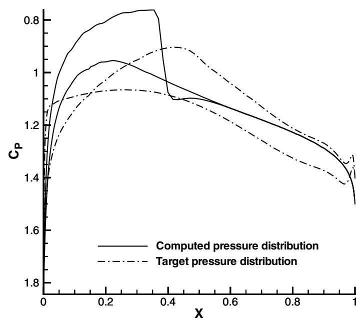
Fig. 8 Convergence of global objective function L^g for optimization case 2 using 14 design variables.



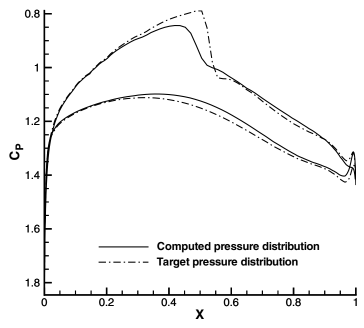
a) 1st design iteration (n=1)



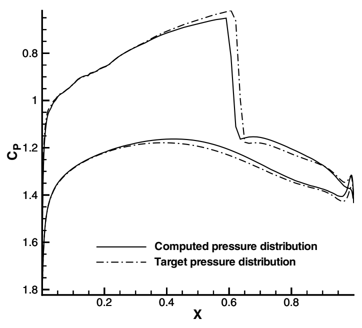
b) 1st design iteration (n=11)



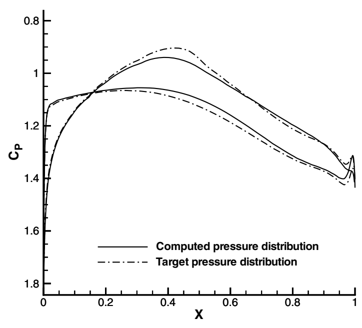
c) 1st design iteration (n=32)



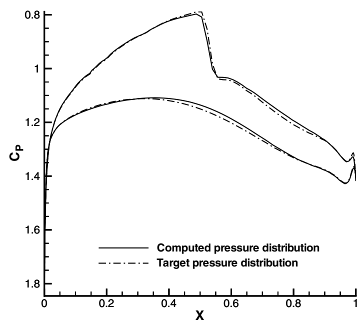
d) 3rd design iteration (n=1)



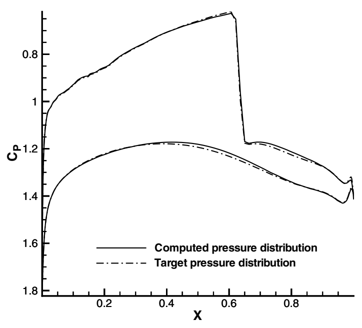
e) 3rd design iteration (n=11)



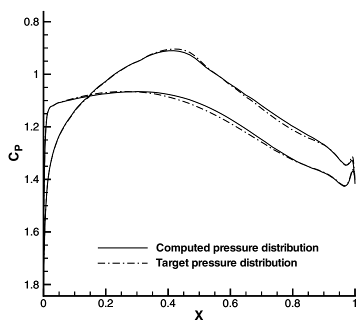
f) 3rd design iteration (n=32)



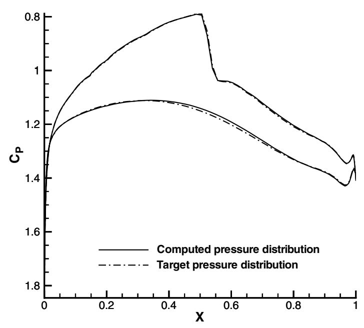
g) 15th design iteration (n=1)



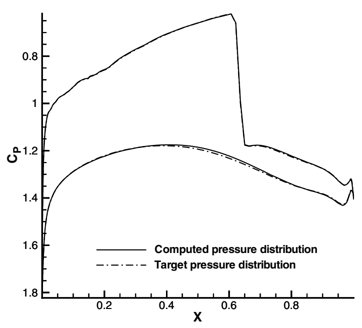
h) 15th design iteration (n=11)



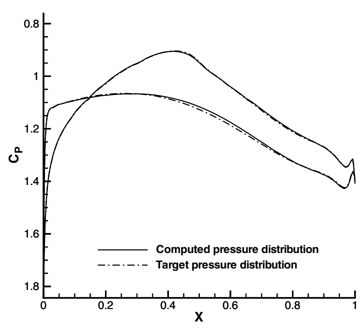
i) 15th design iteration (n=32)



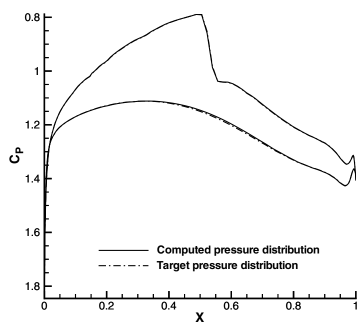
j) 25th design iteration (n=1)



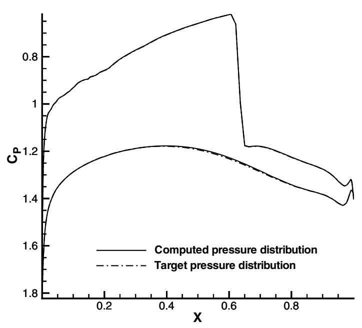
k) 25th design iteration (n=11)



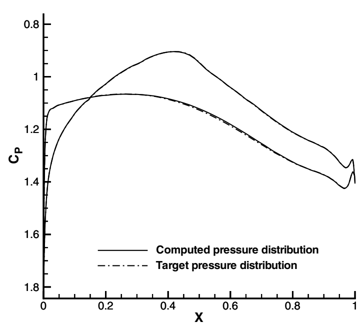
l) 25th design iteration (n=32)



m) 47th design iteration (n=1)

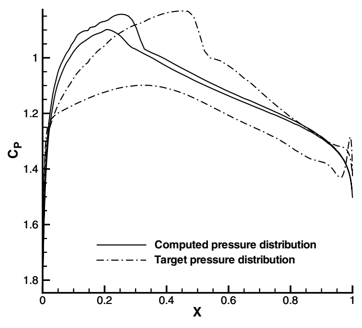


n) 47th design iteration (n=11)

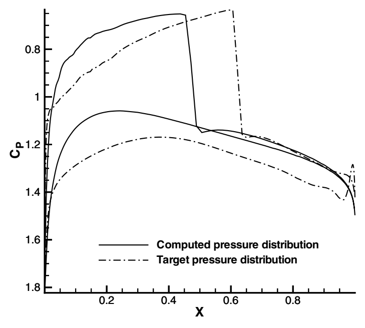


o) 47th design iteration (n=32)

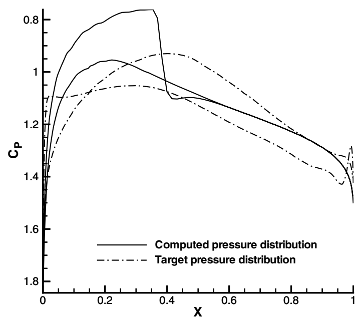
Fig. 9 Comparison of computed and target pressure profiles at different design iterations for three time levels, $n = 1$, $n = 11$, and $n = 32$, for optimization case 2 with 290 design variables.



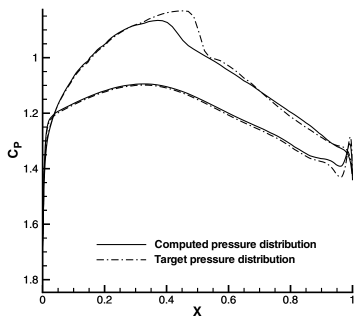
a) 1st design iteration (n=1)



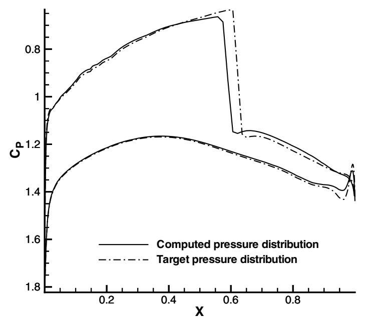
b) 1st design iteration (n=11)



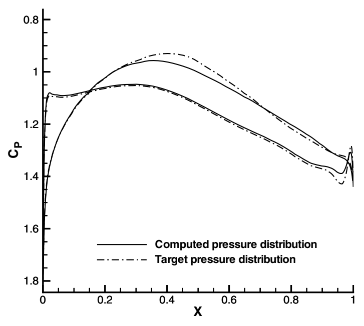
c) 1st design iteration (n=32)



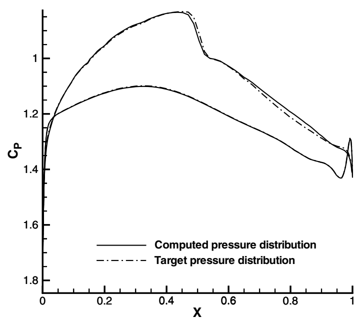
d) 3rd design iteration (n=1)



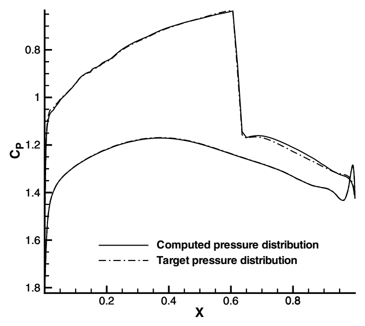
e) 3rd design iteration (n=11)



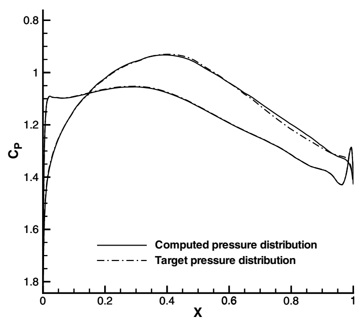
f) 3rd design iteration (n=32)



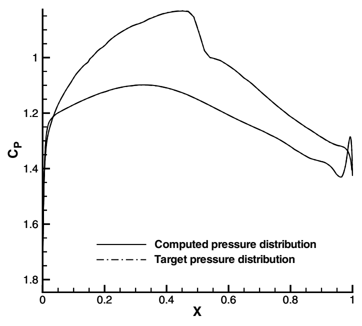
g) 7th design iteration (n=1)



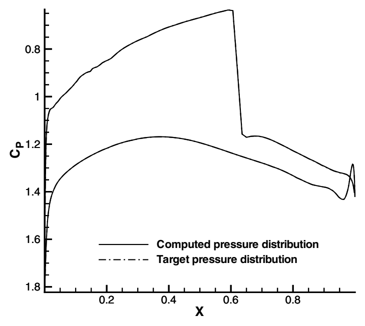
h) 7th design iteration (n=11)



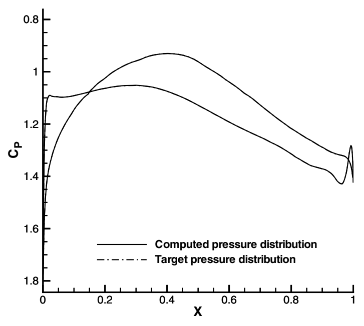
i) 7th design iteration (n=32)



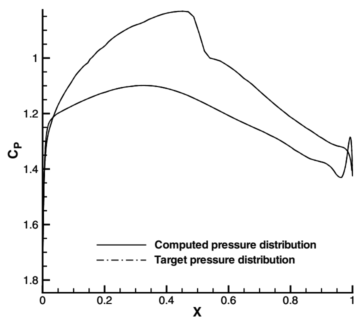
j) 15th design iteration (n=1)



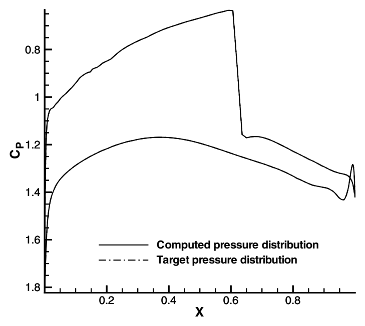
k) 15th design iteration (n=11)



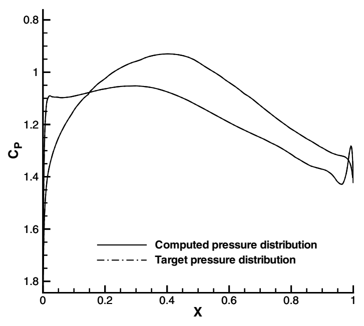
l) 15th design iteration (n=32)



m) 28th design iteration (n=1)



n) 28th design iteration (n=11)



o) 28th design iteration (n=32)

Fig. 10 Comparison of computed and target pressure profiles at different design iterations for three time levels, $n = 1$, $n = 13$, and $n = 27$, for optimization case 2 with 14 design variables.

Figures 4 and 5 compare the computed and targeted time-dependent lift and drag profiles at various stages in the optimization. The drag comparison figure indicates what appears to be a slightly larger error between the optimized and target values than the lift curves at the final design iteration because of the different scales of these coefficients. There is approximately a 2 orders of magnitude reduction in the functional and gradient within the first five design iterations beyond which the convergence rate greatly diminishes. No convergence specific stopping criteria was used except for total computational time to terminate the optimization. For this particular example, the optimization was terminated after 15 wall-clock hours, and this corresponded to 58 calls of the flow solver and the adjoint code.

B. Case 2: Time-Dependent Pressure Distribution Matching

The second optimization example is that of inverse design from a NACA0012 airfoil to a NACA64210 airfoil in the context of unsteady flows. The target time-dependent pressure profile for this example is that of a sinusoidally pitching NACA64210 type airfoil. Since inverse design is shape constrained, it is important to know a priori whether the geometric parametrization permits the target airfoil to lie within the design space of the problem. An optimization using only the shape parametrization independent of the flow equations was performed to check if this were true. It was determined that an exact NACA64210 was not achievable based on the current parametrization. As a consequence, the closest possible match from the result of this exploratory geometric optimization was used as the target because this guarantees an achievable shape. Figure 6 compares the airfoil that was used as the target against the true NACA64210 airfoil.

The objective functional for this example is similar to that of case 1, with the difference in computed and target loads replaced by the difference in computed and target time-dependent pressure profiles. The local objective functional at each time level for this example is defined as

$$L^n = \sum_{i=1}^{n_{\text{surf}}} \left\{ \frac{1}{2} \left(p_i^n - p_{i\text{target}}^n \right)^2 \right\} \tag{54}$$

Figure 7 shows the convergence of the global objective functional L^g. As in the previous case, the global objective functional drops by about 4 orders of magnitude before the convergence begins to stall. This is not an acceptable scenario because it is known that the target airfoil lies within the design space. Theoretically, the functional should be able to reach machine zero and the gradient to the square root of machine zero. The problem was suspected to be the complexity of the design space and the inability of the LBFGS-B algorithm to navigate through it. Because the optimization consists of 290 design variables, the problem was simplified by reducing the total number of design variables to 14. As previously described, a new target airfoil was chosen using the parametrization based on fewer design variables, and a second unsteady inverse design optimization was run. Figure 8 shows the convergence for this case. The results verify our suspicions because, in this case, the expected trend is observed. The functional reaches machine zero in 28 design iterations, whereas the gradient approximately reaches square root of machine precision. The termination criteria used in this case was a value of 10^{-14} for the functional.

Figures 9 and 10 compare the target and computed pressure profiles at different design iterations for three temporal locations for both the 290 and 14 design variables cases. Although qualitatively the results look impressive in both cases, it is clear from the convergence plots that the true optimum has been achieved only in the 14 design variable case.

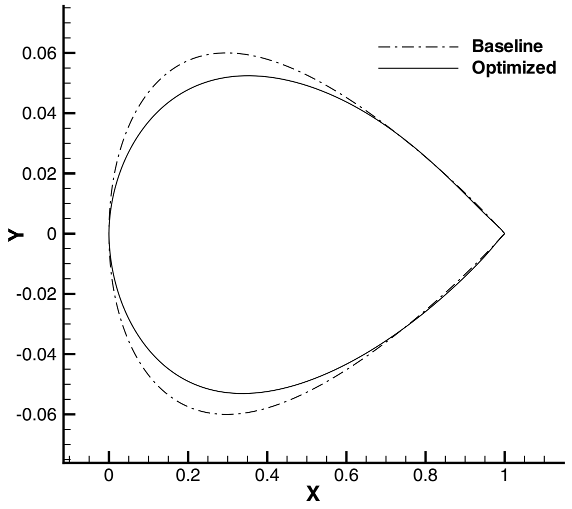


Fig. 12 Comparison of baseline NACA0012 and optimized airfoils for optimization case 3.

C. Case 3: Time-Dependent Drag Minimization

The final optimization example is that of lift-constrained time-dependent drag minimization. The objective formulation for this case is the same as that used for the time-dependent load matching problem. The difference lies in the target loads used in the formulation. Since this a lift-constrained drag minimization problem, the target time-dependent lift is set to be that of the baseline NACA0012 airfoil itself, whereas the target drag is set to be zero at

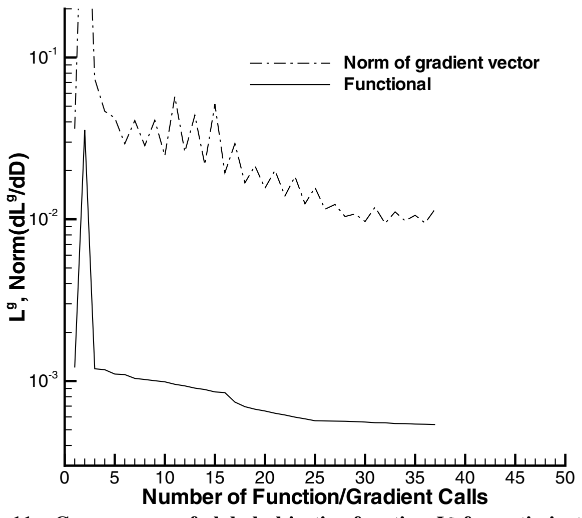


Fig. 11 Convergence of global objective function L^g for optimization case 3.

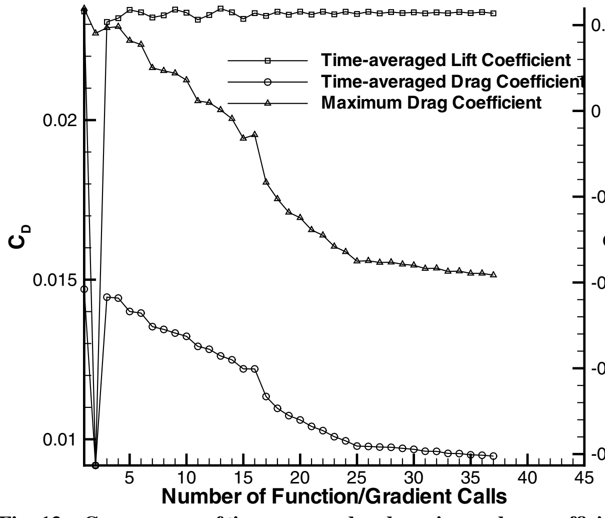


Fig. 13 Convergence of time-averaged and maximum drag coefficients for optimization case 3.

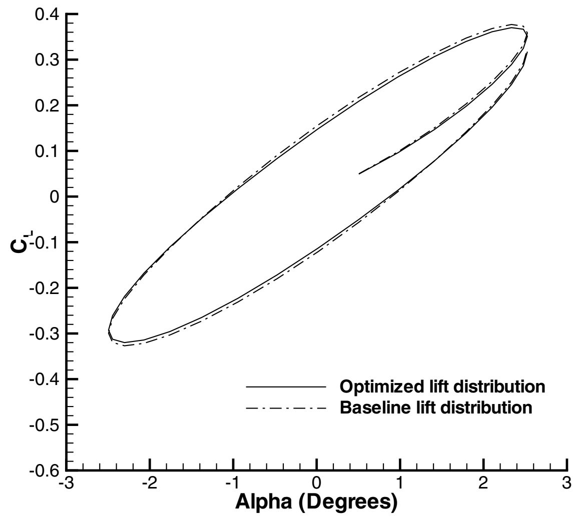


Fig. 14 Comparison of baseline and optimized time-dependent lift profiles for optimization case 3.

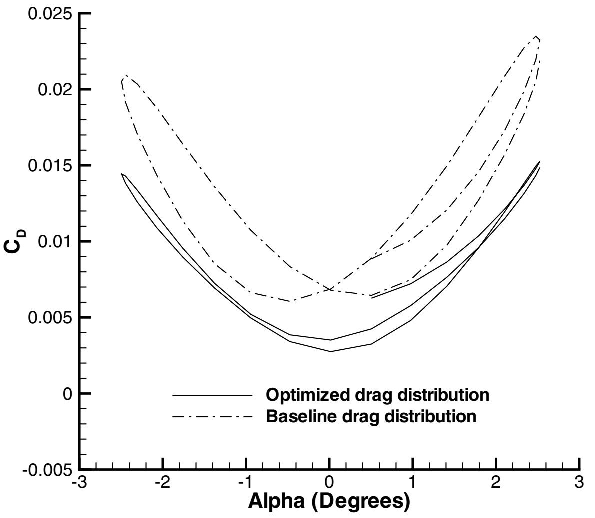


Fig. 15 Comparison of baseline and optimized time-dependent drag profiles for optimization case 3.

all temporal locations. The flow conditions and pitch criteria remain identical to the previous two examples. The objective functional cannot reach zero in this case because the drag acting on a pitching airfoil is always nonzero. Figure 11 shows the convergence of the functional and the gradient, and it is clear that the optimization levels out very quickly. Figure 12 shows the optimized airfoil in comparison with the baseline NACA0012 airfoil. Figure 13 shows the convergence of the time-averaged and maximum drag coefficients during the course of the optimization. An approximate reduction of 85 counts of drag in the maximum drag coefficient and 50 counts in the time-averaged drag coefficient is observed. Figures 14 and 15 compare the baseline and optimized time-dependent lift and drag profiles.

VI. Conclusions

An adjoint method for computing sensitivities in time-dependent flow problems based on an ALE formulation with deforming meshes was developed. The algorithm was demonstrated by applying it to time-dependent flow problems in the context of shape optimization. The method is feasible for substantially large cases because the solution history is written to disk and read in as and when required. Consequently, the memory overhead is never more than what is required for obtaining a single unsteady flow solution. Typical three-dimensional unsteady flow problems that run in parallel with approximately 200,000 elements per processor for about 2000 time steps would require roughly 15 GB of hard drive space per processor to hold the solution history. Such space requirements can be easily accommodated at low cost on today's computational clusters. Also, because the solution set is written piece by piece at the conclusion of each time step in the time-integration process, the speed of the I/O operation itself has little impact on the overall speed of computation. The other factor weighing in on feasibility when extended to large cases is the number of design iterations required to reach an optimum design. To this end, future work will explore the use of advanced optimization algorithms and will test the algorithm in three-dimensional problems.

References

[1] Jameson, A., "Aerodynamic Shape Optimization Using the Adjoint Method," *VKI Lecture Series on Aerodynamic Drag Prediction and Reduction*, von Karman Institute of Fluid Dynamics, Rhode St. Genese, Belgium, 2003.

[2] Jameson, A., and Vassberg, J., "Computational Fluid Dynamics for Aerodynamic Design: Its Current and Future Impact," AIAA Paper 2001-0538, 2001.

[3] Nadarajah, S., and Jameson, A., "A Comparison of the Continuous and Discrete Adjoint Approach to Automatic Aerodynamic Optimization," AIAA Paper 2000-0667, 2000.

[4] Nielsen, E., and Anderson, W., "Recent Improvements in Aerodynamic Design Optimization on Unstructured Meshes," *AIAA Journal*, Vol. 40, No. 6, June 2002, pp. 1155–1163.

[5] Jameson, A., Alonso, J., Reuther, J., Martinelli, L., and Vassberg, J., "Aerodynamic Shape Optimization Techniques Based on Control Theory," AIAA Paper 98-2538, 1998.

[6] Giles, M., Duta, M., and Muller, J., "Adjoint Code Developments Using Exact Discrete Approach," AIAA Paper 2001-2596, 2001.

[7] Mavriplis, D. J., "Multigrid Solution of the Discrete Adjoint for Optimization Problems on Unstructured Meshes," *AIAA Journal*, Vol. 44, No. 1, Jan. 2006, pp. 42–50. doi:10.2514/1.15696

[8] Mavriplis, D. J., "Discrete Adjoint-Based Approach for Optimization Problems on Three-Dimensional Unstructured Meshes," *AIAA Journal*, Vol. 45, No. 4, April 2007, pp. 740–750. doi:10.2514/1.22743

[9] Elliott, J., and Peraire, J., "Practical Three-Dimensional Aerodynamic Design and Optimization Using Unstructured Meshes," *AIAA Journal*, Vol. 35, No. 9, 1997, pp. 1479–1485.

[10] Soto, R., and Yang, C., "An Adjoint-Based Design Methodology for CFD Optimization Problems," AIAA Paper 2003-0299, 2003.

[11] Ghayour, K., and Baysal, O., "Limit-Cycle Shape Optimization using Time-Dependent Transonic Equation," AIAA Paper 99-3375, 1999.

[12] Nadarajah, S., and Jameson, A., "Optimal Control of Unsteady Flows using A Time Accurate Method," AIAA Paper 2002-5436, 2002.

[13] Rumpfkeil, M., and Zingg, D., "A General Framework for the Optimal Control of Unsteady Flows with Applications," AIAA Paper 2007-1128, Jan. 2007.

[14] Nadarajah, S., McMullen, M., and Jameson, A., "Non-Linear Frequency Domain Based Optimum Shape Design for Unsteady Three-Dimensional Flow," AIAA Paper 2006-387, 2006.

[15] Nadarajah, S., McMullen, M., and Jameson, A., "Optimal Control of Unsteady Flows Using Time Accurate and Non-Linear Frequency Domain Methods," AIAA Paper 2003-3875, June 2003.

[16] Byrd, R. H., Lu, P., and Nocedal, J., "A Limited Memory Algorithm for Bound Constrained Optimization," *SIAM Journal on Scientific and Statistical Computing*, Vol. 16, No. 5, 1995, pp. 1190–1208. doi:10.1137/0916069

[17] Yang, Z., and Mavriplis, D. J., "Higher-Order Time Integration Schemes for Aeroelastic Applications on Unstructured Meshes," *AIAA Journal*, Vol. 45, No. 1, Jan. 2007, pp. 138–150. doi:10.2514/1.22847

[18] Mavriplis, D., "Multigrid Techniques for Unstructured Meshes," *Notes Prepared for the 26th Computational Fluid Dynamics Lecture Series Program*, von Karman Institute of Fluid Dynamics, Rhode St. Genese, Belgium, 1995.

[19] Roe, P., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," *Journal of Computational Physics*, Vol. 43, No. 2, 1981, pp. 357–372. doi:10.1016/0021-9991(81)90128-5

[20] Geuzaine, P., Grandmont, C., and Farhat, C., "Design and Analysis of ALE Schemes with Provable Second-Order Time-Accuracy for Inviscid and Viscous Flow Simulations," *Journal of Computational Physics*, Vol. 191, No. 1, 2003, pp. 206–227. doi:10.1016/S0021-9991(03)00311-5

[21] Mavriplis, D., and Yang, Z., "Construction of the Discrete Geometric Conservation Law for High-Order Time Accurate Simulations on Dynamic Meshes," *Journal of Computational Physics*, Vol. 213, No. 2, 2006, pp. 557–573. doi:10.1016/j.jcp.2005.08.018

[22] Batina, J. T., "Unsteady Euler Airfoil Solutions Using Unstructured Dynamic Meshes," *AIAA Journal*, Vol. 28, No. 8, Aug. 1990, pp. 1381–1388.

[23] Hicks, R., and Henne, P., "Wing Design by Numerical Optimization," *Journal of Aircraft*, Vol. 15, No. 7, 1978, pp. 407–412.

[24] Valarezo, W. O., and Mavriplis, D. J., "Navier–Stokes Applications to High-Lift Airfoil Analysis," *Journal of Aircraft*, Vol. 32, No. 3, 1995, pp. 618–624.

[25] Dwight, R., and Brezillon, J., "Effect of Various Approximations of the Discrete Adjoint on Gradient-Based Optimization," AIAA Paper 2006-0690, 2006.

[26] Nielsen, E., Lu, J., Park, M., and Darmofal, D., "An Exact Dual Adjoint Solution Method for Turbulent Flows on Unstructured Grids," AIAA Paper 2003-0272, 2003.

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

Z. Wang
Associate Editor