45th AIAA Aerospace Sciences Meeting and Exhibit
8 - 11 January 2007, Reno, Nevada

AIAA 2007-60

45th AIAA Aerospace Sciences Meeting and Exhibit, January 8–11, 2007, Reno, NV

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

Karthik Mani [*]

Department of Mechanical Engineering, University of Wyoming, Laramie, Wyoming 82071-3295

Dimitri J. Mavriplis [†]

Department of Mechanical Engineering, University of Wyoming, Laramie, Wyoming 82071-3295

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 non-linear flow problem where 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 (ALE) 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 where the shape of a pitching airfoil is morphed to match a target time-dependent load profile, the second where the shape is optimized to match a target time-dependent pressure profile, and the last where the time-dependent drag profile is minimized without any loss in lift.

I. Introduction

In the recent past, 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 allow the computation of sensitivity derivatives of an objective function to a set of given inputs at a cost which is essentially independent of the number of inputs. This is in contrast to the brute-force finite-difference method, where each input or design variable has to perturbed individually to obtain a corresponding effect on the output. This is a tedious and costly process which is of little use when there a large number of design variables or inputs.

Adjoint equations may be utilized in two forms, either continuous or discrete. In the continuous approach, the governing non-linear flow equations are differentiated with respect to the design variables and then discretized before obtaining a solution, whereas in the case of the discrete adjoint approach, this order is reversed. In the discrete approach the governing equations are discretized and the discrete version then differentiated against the design variables. Theoretically, both methods converge to the same solution as the limit of the discretization resolution tends to infinity. The continuous adjoint approach offers significant flexibility in the discretization process, however, the discrete adjoint approach provides exact sensitivity derivatives of the original discretized governing equations. Also, the boundary conditions in the case of the discrete adjoint are straightforward derivatives of existing boundary conditions for the governing flow equations. Another significant advantage in using the discrete approach comes from the fact that the governing flow equations in many cases are solved implicitly, thus requiring a linearization. This linearization is therefore readily available for use in computing the discrete adjoint.

The use of adjoint equations either in continuous or discrete form for solving steady-state aerodynamic optimization problems[4,7–10] is now fairly well established. However, relatively little work has been done in applying these methods for unsteady time-dependent problems.[11,12] In the context of unsteady flows, frequency domain methods[13,14]

[*]Graduate Student, AIAA Member; email: kmani@uwyo.edu.
[†]Professor, AIAA Associate Fellow; email: mavripl@uwyo.edu.

have been investigated that allow for reduced computational expense particularly for problems with strong periodic behavior. The goal of this paper is to develop an efficient framework for the computation of sensitivities for unsteady flow problems operating directly in the time domain. Although applicable to any type of unsteady problem, the method may be more costly when compared with the frequency domain approach. However, both methods are complimentary to one and other. Most importantly the developed algorithm takes into account the effect of deforming meshes which are becoming commonplace in unsteady computations. 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. This is especially true in the field of aeroelasticity. Ultimately the procedure may be extended to applications such as unsteady adaptive mesh refinement built on current adjoint based mesh refinement in steady problems.[15,16] There is also potential in using the core of the algorithm in time-step size adaptation for unsteady flows.

II. Analysis Problem Formulation

A. Governing Equations of the Flow Problem in ALE Form

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

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

Applying the divergence theorem and integrating over a moving control volume $A(t)$ yields:

$$\int_{A(t)} \frac{\partial \mathbf{U}}{\partial t} dA + \int_{dB(t)} \mathbf{F}(\mathbf{U}) \cdot \mathbf{n} dB = 0 \tag{2}$$

Using the differential identity:

$$\frac{\partial}{\partial t} \int_{A(t)} \mathbf{U} dA = \int_{A(t)} \frac{\partial \mathbf{U}}{\partial t} dA + \int_{dB(t)} (\dot{\mathbf{x}} \cdot \mathbf{n}) dB \tag{3}$$

equation (2) is rewritten as:

$$\frac{\partial}{\partial t} \int_{A(t)} \mathbf{U} dA + \int_{dB(t)} [\mathbf{F}(\mathbf{U}) - \dot{\mathbf{x}}\mathbf{U}] \cdot \mathbf{n} dB = 0 \tag{4}$$

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

$$\frac{\partial A\mathbf{U}}{\partial t} + \int_{dB(t)} [\mathbf{F}(\mathbf{U}) - \dot{\mathbf{x}}\mathbf{U}] \cdot \mathbf{n} dB = 0 \tag{5}$$

This is the Arbitrary-Lagrangian-Eulerian (ALE) finite-volume form of the Euler equations. The equations are required in ALE form since the problem involves deforming meshes where mesh elements change in shape and size at each time-step. Here A refers to the area of the control volume, $\dot{\mathbf{x}}$ is the vector of mesh face or edge velocities, and $\mathbf{n}$ is the unit normal of the face or edge. The state vector $\mathbf{U}$ of conserved variables and the cartesian inviscid flux vector $\mathbf{F} = (\mathbf{F}^x, \mathbf{F}^y)$ are:

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ E_t \end{pmatrix}, \quad \mathbf{F}^x = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u(E_t + p) \end{pmatrix}, \quad \mathbf{F}^y = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ v(E_t + p) \end{pmatrix}, \tag{6}$$

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:

$$p = (\gamma - 1)\left[E_t - \frac{1}{2}\rho(u^2 + v^2)\right] \tag{7}$$

where $\gamma = 1.4$ is the ratio of specific heats.

B. Temporal Discretization

The time derivative term in the Euler equations is discretized using both a first-order accurate backward-difference-formula (BDF1), and a second-order accurate BDF2 scheme. These discretizations are shown in equations (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 equation (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 where the inviscid flux integral around a closed control volume is discretized as:

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

where 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:

$$\mathbf{F}^{\perp}_{\mathbf{edge}} = \frac{\mathbf{1}}{\mathbf{2}}\{\mathbf{F}^{\perp}_{\mathbf{L}}(\mathbf{q_L}) + \mathbf{F}^{\perp}_{\mathbf{R}}(\mathbf{q_R}) + [\hat{\mathbf{A}}]\,(\mathbf{q_L} - \mathbf{q_R})\} \tag{11}$$

where $\mathbf{q_L}$, $\mathbf{q_R}$ are the left and right extrapolated state vectors and $\mathbf{F}_L^{\perp}$, $\mathbf{F}_R^{\perp}$ 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[17] and therefore matrix $[\hat{A}]$ is the Roe averaged flux Jacobian. Second-order spatial accuracy is achieved by using a gradient based reconstruction scheme to determine 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 center and the edge center, 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 final state gradients at the cell-centers. An arithmetic average is used rather than more accurate polynomial based interpolation methods in order simplify the linearization procedure.

D. The Discrete Geometric Conservation Law (GCL)

The discrete geometric conservation law (GCL) requires that a uniform flow field be preserved when equation (5) 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 $\mathbf{U} =$*constant* being an exact solution of equation (5). For a conservative scheme, the integral of the inviscid fluxes around a closed contour goes to zero when $\mathbf{U} =$*constant*. Applying these conditions to equation (5) results in the mathematical description of the GCL as stated below.

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

Equation (14) implies that the change in area of a control volume should be discretely equal to the area swept by the boundary of the control volume. The vector of cartesian edge velocities $\dot{x}$ for each of the edges encompassing the control volume must therefore be chosen such that equation (14) is satisfied. The dot product of the cartesian edge velocities and the edge normal vector is represented by the term $\mathbf{V_e}$. There are several methods to compute the edge velocities while satisfying the GCL, but in previous work it has been shown that satisfaction of the GCL is not a sufficient condition to achieve the underlying accuracy of the chosen time-integration scheme.[18, 19] The GCL has been specifically derived for first, second and third-order backward difference (BDF) time-integration schemes and also for the fourth-order accurate IRK64 implicit Runge-Kutta scheme.[19] The form of the GCL described in Ref.[19] is used exclusively in this work.

E. Mesh Deformation Strategy

Deformation of the mesh is achieved through the linear tension spring analogy which approximates the mesh as a network of inter-connected 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 vertex displacements in the mesh to known displacements on the boundaries is:

$$[K]\delta\mathbf{x}_{int} = \delta\mathbf{x}_{surf} \tag{15}$$

The stiffness matrix [K] is a sparse block matrix that can be treated as a constant since it is based on the initial configuration of the mesh and remains unchanged through the time-integration process. For each node in the mesh, the displacement can be solved as

$$\delta\mathbf{x}_{int_i} = \frac{\delta\mathbf{x}_{surf_i} - \sum_{j=1}^{n_j}[O]_{ij}\delta\mathbf{x_j}}{[D]_i} \tag{16}$$

where the off-diagonal term $[O]_{ij}$ and diagonal term $[D]_i$ are 2×2 diagonal matrices as follows:

$$[O]_{ij} = -k_{ij}[I] \qquad \text{for } i \neq j \tag{17}$$

$$[D]_i = \sum_{j=1}^{n_j} k_{ij}[I] \tag{18}$$

where k_{ij} refers to the stiffness of the corresponding edge. The mesh motion equations are elliptic in nature and can be solved using conventional smoothers such as Gauss-Seidel or Gauss-Jacobi iterations. The convergence of the system is a function of the mesh resolution and becomes prohibitively expensive as the mesh resolution becomes large. An agglomeration multigrid[20] approach is used to accelerate the convergence of the linear system. Being elliptic in nature the system is particularly well suited for acceleration using multigrid.

F. Geometry Parametrization

Modification of the baseline geometry is achieved through displacements of the surface nodes defining the geometry. In order 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 results presented in this paper is the Hicks-Henne Sine bump function.[21] It is defined as:

$$b_i(x) = a \cdot sin^4(\pi x^{m_i}) \tag{19}$$

$$m_i = ln(0.5)/ln(x_{M_i}) \tag{20}$$

Here x_{M_i} refers to the location where the bump is to be placed and is typically the x coordinate of the surface node which is to be displaced. $b_i(x)$ are the displacements of points with coordinates ranging from 0 to 1 as a result of the displacement of the point at x_{M_i} in accordance with the sine bump function. a is the magnitude of the bump placed at x_{M_i}. The design variables for the examples presented in the paper are the magnitudes of bumps placed at surface nodes. The bump function is valid in the range of $0 \leq x, x_{M_i} < 1$, and is ideally suited for airfoil problems. For geometries extending beyond this range, the bump function may be mapped locally over some predetermined radius about the surface node of interest. The superposition of bump functions placed at each surface node is added to the shape of the baseline airfoil in order to deform its surface. Figure (1) shows a series of bumps with a magnitude of 0.1 placed between $0 \leq x < 1$.

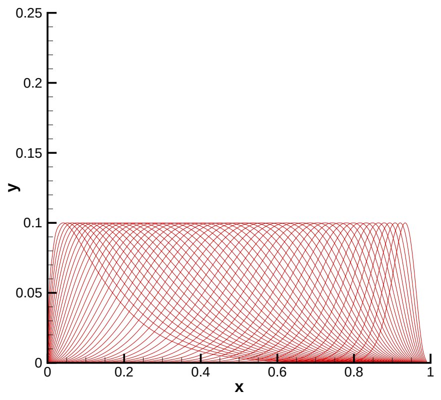


Figure 1. Hick-Henne Sine bump functions.

III. Analysis Procedure

The example problems chosen to demonstrate the unsteady adjoint algorithm involves 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

$$\delta\mathbf{x}_{surf}^{base} = f(D + \delta D) \tag{21}$$

$$\mathbf{x}_{surf}^{0} = \mathbf{x}_{surf}^{base} + \delta\mathbf{x}_{surf}^{base} \tag{22}$$

   where $\delta\mathbf{x}_{surf}^{base}$ are the displacements of the surface nodes computed as a function of the design variables D. In this particular case the functional operator refers to the Hicks-Henne Sine bump function described in the previous section. $\mathbf{x}_{surf}^{base}$ refers to the surface coordinates of the baseline non-deformed airfoil. In the case of an optimization $\mathbf{x}_{surf}^{base}$ refers to the airfoil from the previous design iteration rather than the original baseline configuration.

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

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 defines the prescribed 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 equation (23). The transformed airfoil is then used to compute the surface displacement vector $\delta\mathbf{x}_{surf}^{n}$ for the current time-level as shown in equation (25).

$$\alpha^n = \alpha_o + \alpha_{max} sin(\omega t^n) \tag{23}$$

$$[\beta]^n = \begin{pmatrix} cos\alpha^n & sin\alpha^n \\ -sin\alpha^n & cos\alpha^n \end{pmatrix} \tag{24}$$

$$\delta\mathbf{x}_{surf}^{n} = ([\beta]^n - [I])\,\mathbf{x}_{surf}^{0} \tag{25}$$

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

6. Calculate edge velocities $\mathbf{V_e}$ in accordance with the GCL for the time-integration scheme used. For a BDF1 scheme these are functions of coordinates $\mathbf{x}^n_{int}$ and $\mathbf{x}^{n-1}_{int}$ and functions of coordinates $\mathbf{x}^n_{int}$, $\mathbf{x}^{n-1}_{int}$ and $\mathbf{x}^{n-2}_{int}$ for the BDF2 scheme. Further details can be found in Ref.[19] New cell areas A^n are also computed at this point using $\mathbf{x}^n_{int}$.

7. For an implicit time-step the non-linear flow residual is defined as:

$$\mathbf{R}^n \quad = \quad \frac{\partial(A\mathbf{U})}{\partial t} + \mathbf{S}(\mathbf{V}^n_e, \mathbf{n}^n_e, \mathbf{U}^n) = 0 \tag{26}$$

   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 $\mathbf{n}_e$ and the flow solution at the current time-level $\mathbf{U}^n$.

8. The non-linear flow residual is then solved using Newton's method as shown in equation (27) in conjunction with an agglomeration multigrid scheme to accelerate convergence of the intermediate linear systems.

$$\left[\frac{\partial \mathbf{R}(\mathbf{U}^k)}{\partial \mathbf{U}^k}\right] \delta\mathbf{U}^k = -\mathbf{R}(\mathbf{U}^k) \tag{27}$$

$$\mathbf{U}^{k+1} = \mathbf{U}^k + \delta\mathbf{U}^k \tag{28}$$

$$\delta\mathbf{U}^k \to 0, \mathbf{U}^{k+1} = \mathbf{U}^n \tag{29}$$

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

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

IV. Sensitivity Formulation

A. Generalized Sensitivity Analysis

Consider a non-linear operation that requires an input D and computes an output L. Mathematically this can be represented as L=F(D), where F is the non-linear operator acting on the input D. For example, in the case of a steady-state flow problem, the output L may be the drag acting on an airfoil, and the input D may be a vector of variables representing the shape of the airfoil. If the drag on the airfoil is to be minimized by changing the shape of the airfoil, the sensitivity of the drag to the shape is required. The output, which in this case is the drag is defined as the objective function L that requires minimization. The conventional method of determining the sensitivity $\frac{dL}{dD}$ is to obtain the output L for a baseline configuration with some known input D and then to perturb the shape of the baseline configuration by δD. This results in a change of the output L by δL, by which the sensitivity $\frac{dL}{dD}$ is approximated as $\delta L/\delta D$. This procedure describes the finite-difference method and in the case of multiple inputs and/or multiple outputs involves perturbing each input D and recording their effect on each L. It is obvious that the finite-difference procedure would require the numerical evaluation of the non-linear operator F for each input D in the problem. For multiple design variables and multiple outputs the sensitivities form a matrix of size n_L by n_D where n_L is the number of objectives and n_D is the number of design variables. The finite-difference method has been used effectively and extensively to develop the sensitivity matrix in many commercial optimization packages such as iSight[22] and ModelCenter.[23]

In order to determine $\frac{dL}{dD}$ it is possible to differentiate the analytical form of F(D) with respect to D. This describes the continuous approach. In the case of a flow problem based on the Euler equations it involves the analytical derivative of the Euler equations. The other option is to sequentially differentiate each discrete operation used to evaluate F(D) numerically. This is done by directly differentiating the code used to solve the discretized Euler equations which requires D as the input and computes L as the output. This is the discrete approach and is the focus of this paper. If the intermediate functional dependencies of the output L on the input D are written as:

$$\mathbf{L}(\mathbf{D}) = \mathbf{L}(F_{n-1}(F_{n-2}(....F_2(F_1(\mathbf{D}))....)))) \tag{30}$$

then the sensitivity of L on D can be computed by differentiating the equation using the chain rule as:

$$\frac{d\mathbf{L}}{d\mathbf{D}} = \frac{\partial \mathbf{L}}{\partial F_{n-1}} \cdot \frac{\partial F_{n-1}}{\partial F_{n-2}} ..... \frac{\partial F_2}{\partial F_1} \cdot \frac{\partial F_1}{\partial \mathbf{D}} \tag{31}$$

For the case of a single input D and multiple outputs L, the sensitivity $\frac{dL}{dD}$ is a vector of size $[n_L,1]$. The product of the last two terms in equation (31) is either a matrix-vector multiplication or a vector-vector inner product that results in a vector. This implies that the final sensitivity $\frac{dL}{dD}$ can be computed as a series matrix-vector or vector-vector products. There are no expensive matrix-matrix multiplications required in this procedure. For cases with more than one input D, the series of multiplications must be performed for each input D. In situations with a single output L and multiple inputs D it becomes inefficient to use the forward linearization since the cost of obtaining the final sensitivity $\frac{dL}{dD}$ is directly proportional to the number of inputs D. This arises from the fact that the procedure has to be carried out for each input variable D. The problem is eliminated by transposing equation (31) to obtain a reverse linearization as shown below.

$$\frac{d\mathbf{L}}{d\mathbf{D}}^T = \frac{\partial F_1}{\partial \mathbf{D}}^T \cdot \frac{\partial F_2}{\partial F_1}^T \cdot \frac{\partial F_{n-1}}{\partial F_{n-2}}^T \cdots\cdots \frac{\partial \mathbf{L}}{\partial F_{n-1}}^T \tag{32}$$

The last term in equation (32) is now a vector of size $[n_L,1]$, and the resulting sequence of multiplications again consist of either matrix-vector products or vector-vector inner products. In contrast with the forward procedure, for each additional output L the sequence of operations in equation (32) has to be repeated to obtain the matrix of sensitivities $\frac{dL}{dD}$.

To summarize, it is efficient to invoke the forward linearization for cases with multiple outputs L and a single or few inputs D, while the adjoint or reverse linearization is more economical for cases with a single or few outputs L and multiple inputs D.

B. Objective Function Formulation for the Unsteady Problem

The objective function for the first unsteady optimization example is defined such that the time-dependent lift and drag profiles of the optimized airfoil match target or required time-dependent profiles. The objective function is based on the difference between the target and computed objective values at each time-level n. Figures (2(a)) and (2(b)) exemplify the formulation of the unsteady objective function. If C_L^n and C_D^n refer to the lift and drag coefficients computed on the airfoil at time-level n during the pitch cycle, then the local objective at time-level n is defined as:

$$L^n = (\delta C_L^n)^2 + 10(\delta C_D^n)^2 \tag{33}$$

$$\delta C_L^n = (C_L^n - C_{Ltarget}^n) \tag{34}$$

$$\delta C_D^n = (C_D^n - C_{Dtarget}^n) \tag{35}$$

where the factor of 10 for the drag coefficient is introduced to equalize its order of magnitude with the lift coefficient. The global or time-integrated objective is taken to be the root-mean-square (RMS) average of the local objective functions:

$$L^g = \sqrt{\frac{\sum_{n=0}^{n=n_f} L^n}{n_f+1}} \tag{36}$$

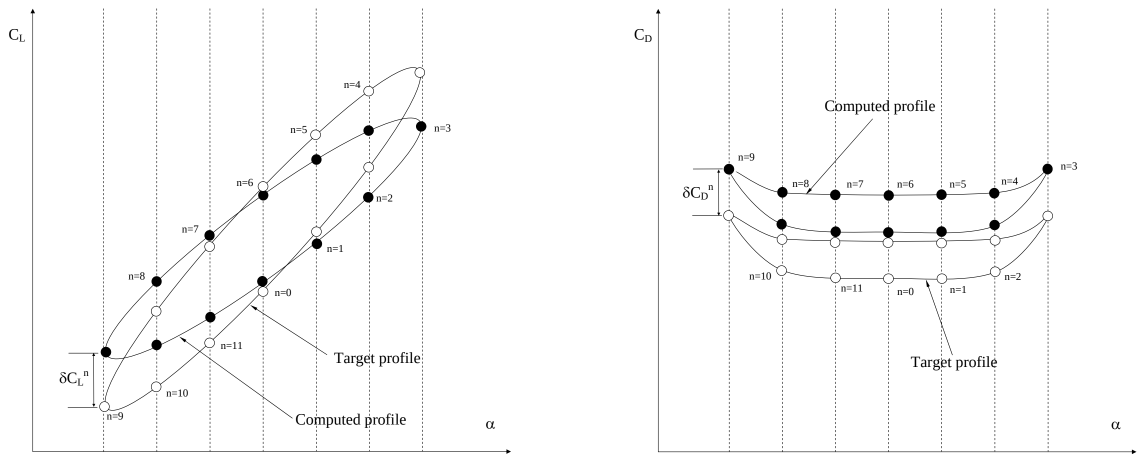
When the RMS error goes to zero, the target and computed load profiles match. This can be interpreted as the discretization of a time-dependent objective $L(t)$ defined as:

$$L(t) = \delta C_L(t) + 10\delta C_D(t) \tag{37}$$

For the second optimization example the local objectives are defined as the RMS difference between a target and computed pressure profile at each time-level, and the global or time-integrated objective is taken to be the arithmetic mean of the local objectives. It would be sufficient to target only the pressure profile at the end of the time-integration process since all other pressure profiles would automatically match up if any one of the profiles match. This arises from the fact that the pressure profile is a function of the airfoil shape and this shape remains fixed through the time-integration process for each design iteration. However, for the purpose of testing the algorithm and to study its behavior the entire set of time-dependent pressure distributions is used.

$$L^n = \left\{ \frac{\sum_{i=1}^{i=n_{surf}} \left[p_i^n - p_{itarget}^n\right]^2}{n_{surf}} \right\}^{\frac{1}{2}} \tag{38}$$

$$L^g = \frac{\sum_{n=0}^{n=n_f} L^n}{n_f+1} \tag{39}$$



(a) Time dependent lift profile (b) Time dependent drag profile

Figure 2. Time dependent load profiles

The third and last optimization example uses the same objective formulation as the first example with the target time-dependent lift profile set as that of the baseline configuration. The target time-dependent drag is taken to be zero since this allows for its reduction when the objective is minimized.

Unlike in the case of a sinusoidally pitching airfoil, a more general unsteady problem of interest may not exhibit periodic behavior. In such cases, the objective function L is defined as some quantity of interest that is integrated in time or is based on flow variables and mesh coordinates at the final time-level n_f. For example, such situations are commonplace in computational weather forecasting,[24] where some global quantity such as the average temperature or moisture in the future over some region is computed on the basis of known initial conditions from the present. This is a case where the quantity of interest is determined through integration of flow variables in time. The sensitivity of this quantity at the end of the time-integration cycle against the known initial conditions can then be computed using the unsteady adjoint method.

C. Sensitivity Formulation for the Unsteady Flow Problem

The objective function L can be written with complete intermediate dependencies as a function of the design variables D in order to assist with the chain rule differentiation process. For the problem of the unsteady pitching airfoil the design variables are defined as a vector of weights controlling the magnitudes of bumps placed at surface node locations. For the purpose of deriving the unsteady adjoint, the objective function definition of the first optimization example shown in equation (36) is used. The intermediate dependencies for a BDF1 based scheme can be written as follows:

$$L^g = F\left[L^0, L^1, L^2 .....L^n_f\right] \quad (40)$$

$$L^n = F\left[C_L^n, C_D^n\right] \quad (41)$$

$$C_L^n, C_D^n = F\left[\mathbf{U}^n, \mathbf{n}_e^n\right] \quad (42)$$

$$\mathbf{U}^n = F\left[A^n, A^{n-1}, \mathbf{U}^{n-1}, \mathbf{S}^n\right] \quad (43)$$

$$\mathbf{S}^n = F\left[\mathbf{U}^n, \mathbf{V}_e^n, \mathbf{n}_e^n\right] \quad (44)$$

$$\mathbf{n}_e^n = F\left[\mathbf{x}_{int}^n\right] \quad (45)$$

$$A^n = F\left[\mathbf{x}_{int}^n\right] \quad (46)$$

$$\mathbf{V}_e^n = F\left[\mathbf{x}_{int}^n, \mathbf{x}_{int}^{n-1}\right] \quad (47)$$

$$\mathbf{x}_{int}^n = F\left[\mathbf{x}_{int}^0, \mathbf{x}_{surf}^n\right] \quad (48)$$

$$\mathbf{x}_{surf}^n = F\left[\mathbf{x}_{surf}^0\right] \quad (49)$$

$$\mathbf{x}_{surf}^0 = F\left[\mathbf{D}\right] \quad (50)$$

where L_g is the time-integrated objective function and the superscript n denotes values at the n^{th} time-level. Sequentially differentiating equations (40) through (50) results in a general expression for the final sensitivity derivative $\frac{dL}{dD}$ for the unsteady flow problem:

$$\frac{dL^g}{dD} = \sum_{n=0}^{n=n_f} \frac{dL^g}{dL^n} \left[\frac{\partial L^n}{\partial \mathbf{U}^n} \frac{\partial \mathbf{U}^n}{\partial D} + \frac{\partial L^n}{\partial \mathbf{x}^n_{int}} \frac{\partial \mathbf{x}^n_{int}}{\partial D} \right] \tag{51}$$

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

D. Tangent or Forward Linearization

Consider the case of a single design variable D. Equation (26) and equation (15) form the constraint equations for the sensitivity problem and can be differentiated with respect to the design variable D in order obtain convenient expressions for both the flow variable and the mesh-coordinate sensitivities. For a BDF1 time discretization the flow constraint equation (i.e. the non-linear flow residual at a given time-level) can be written as:

$$\mathbf{R}^n = \frac{A^n\mathbf{U}^n - A^{n-1}\mathbf{U}^{n-1}}{\Delta t} + \mathbf{S}^n\left(\mathbf{U}^n, V_e^n, \mathbf{n}^{\mathbf{n}}_{\mathbf{e}}\right) = 0 \tag{52}$$

$$\tag{53}$$

Based on the functional dependencies described previously, all terms appearing in the flow residual equation can be written in terms of the mesh coordinates and the flow state variables. In simplified functional form for a BDF1 time-integration scheme this is expressed as:

$$\mathbf{R}^n = F\left[\mathbf{x}^n_{int}, \mathbf{x}^{n-1}_{int}, \mathbf{U}^n, \mathbf{U}^{n-1}\right] = 0 \tag{54}$$

Differentiating this with respect to the design variable D yields:

$$\frac{d\mathbf{R}^n}{dD} = \frac{\partial \mathbf{R}^n}{\partial \mathbf{x}^n_{int}} \frac{\partial \mathbf{x}^n_{int}}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{x}^{n-1}_{int}} \frac{\partial \mathbf{x}^{n-1}_{int}}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^n} \frac{\partial \mathbf{U}^n}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^{n-1}} \frac{\partial \mathbf{U}^{n-1}}{\partial D} = 0 \tag{55}$$

Rearranging the equation in terms of the flow Jacobian results in a convenient expression for the flow variable sensitivities.

$$\frac{\partial \mathbf{U}^n}{\partial D} = -\left[\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^n}\right]^{-1} \left(\frac{\partial \mathbf{R}^n}{\partial \mathbf{x}^n_{int}} \frac{\partial \mathbf{x}^n_{int}}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{x}^{n-1}_{int}} \frac{\partial \mathbf{x}^{n-1}_{int}}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^{n-1}} \frac{\partial \mathbf{U}^{n-1}}{\partial D} \right) \tag{56}$$

Substituting equation (56) into equation (51) yields the final form of the forward linearization of the flow problem.

$$\frac{dL^g}{dD} = \sum_{n=1}^{n=n_f} \frac{dL^g}{dL^n} \left\{ -\frac{\partial L^n}{\partial \mathbf{U}^n} \left[\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^n}^{-1}\right] \left(\frac{\partial \mathbf{R}^n}{\partial \mathbf{x}^n_{int}} \frac{\partial \mathbf{x}^n_{int}}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{x}^{n-1}_{int}} \frac{\partial \mathbf{x}^{n-1}_{int}}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^{n-1}} \frac{\partial \mathbf{U}^{n-1}}{\partial D} \right) + \frac{\partial L^n}{\partial \mathbf{x}^n_{int}} \frac{\partial \mathbf{x}^n_{int}}{\partial D} \right\} \tag{57}$$

The second constraint is the set of mesh motion equations. The stiffness matrix $[K]$ remains a constant and therefore does not affect the derivative.

$$[K]\frac{\partial \mathbf{x}^n_{int}}{\partial D} = \frac{\partial \mathbf{x}^n_{surf}}{\partial D} \tag{58}$$

$$\frac{\partial \mathbf{x}^n_{int}}{\partial D} = [K]^{-1}\frac{\partial \mathbf{x}^n_{surf}}{\partial D} \tag{59}$$

This can be substituted in place of the mesh derivative terms appearing in equation (57). The procedure to obtain the sensitivity $\frac{dL}{dD}$ using the forward linearization is as follows:

1. Begin time-integration loop at $n = 0$
2. Determine the derivative $\frac{\partial x^0_{surf}}{\partial D}$ 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.

4. Iteratively compute mesh sensitivity for current time-level as:

$$[K]\frac{\partial \mathbf{x}_{int}^n}{\partial D} = \frac{\partial \mathbf{x}_{surf}^n}{\partial D} \tag{60}$$

5. Compute flow residual sensitivity as:

$$\frac{\partial \mathbf{R}^n}{\partial D} = \left(\frac{\partial \mathbf{R}^n}{\partial \mathbf{x}_{int}^n}\frac{\partial \mathbf{x}_{int}^n}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{x}_{int}^{n-1}}\frac{\partial \mathbf{x}_{int}^{n-1}}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^{n-1}}\frac{\partial \mathbf{U}^{n-1}}{\partial D} \right) \tag{61}$$

   where the flow variable sensitivities and the mesh sensitivities for the previous time-step 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 conditions for the unsteady flow. The initial condition may be assumed to be uniform flow in order to remove dependence on the design variable D.

6. Compute flow variable sensitivities for the current time-level by iteratively solving the linear system:

$$\left[\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^n}\right]\frac{\partial \mathbf{U}^n}{\partial D} = -\frac{\partial \mathbf{R}^n}{\partial D} \tag{62}$$

7. Perform vector-vector inner products as per equation (57) to determine the local objective sensitivity to the design variable D. Multiply this quantity by the global-to-local objective sensitivity and augment previously computed dL/dD

8. Continue integrating forward in time. At the end of the time-integration process, the complete sensitivity of the global objective L^g to the design variable D is available.

E. Adjoint or Reverse Linearization

For problems with multiple design variables and a single objective function, equation (57) is transposed to obtain the adjoint or reverse linearization. Shape optimization problems require the adjoint linearization to obtain sensitivities since there are typically numerous design variables and one or few objectives. The transpose of the forward linearization can be written as:

$$\left[\frac{dL^g}{dD}\right]^T = \sum_{n=0}^{n=n_f}\left[\frac{dL^n}{dD}\right]^T\left[\frac{dL^g}{dL^n}\right]^T \tag{63}$$

$$\left[\frac{dL^n}{dD}\right]^T = \sum_{n=0}^{n=n_f}\left\{-\left(\frac{\partial \mathbf{x}_{int}^n}{\partial D}^T\frac{\partial \mathbf{R}^n}{\partial \mathbf{x}_{int}^n}^T + \frac{\partial \mathbf{x}_{int}^{n-1}}{\partial D}^T\frac{\partial \mathbf{R}^n}{\partial \mathbf{x}_{int}^{n-1}}^T + \frac{\partial \mathbf{U}^{n-1}}{\partial D}^T\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^{n-1}}^T\right)\left[\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^n}\right]^{-T}\frac{\partial L^n}{\partial \mathbf{U}^n}^T + \frac{\partial \mathbf{x}_{int}^n}{\partial D}^T\frac{\partial L^n}{\partial \mathbf{x}_{int}^n}^T\right\} \tag{64}$$

1. *Case-A : Non-Time-Integrated Objective Function*

Consider the situation where the objective function is defined as a function of only the state variables and mesh coordinates at the end of the time-integration process. For such a case the derivative $\partial L^n/\partial D|_{n=n_f}$ is not just the local objective sensitivity to the design variables D at $n = n_f$, but is also the final sensitivity required. The vectors $\partial L^n/\partial \mathbf{U}^n$ and $\partial L^n/\partial \mathbf{x}_{int}^n$ are easily computable since L^n is a scalar quantity. From equation (64) it is clear that the transpose of the inverse flow Jacobian matrix is required in order to proceed with the sensitivity computation. Since direct inversions of the flow Jacobian matrix are to be avoided due to cost reasons, a new variable called the flow adjoint is defined as per equation (65) to enable an iterative approach:

$$\Lambda_{\mathbf{U}}^n = -\left[\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^n}\right]^{-T}\left[\frac{\partial L^n}{\partial \mathbf{U}^n}\right]^T \tag{65}$$

$$\tag{66}$$

which can also be written as:

$$\left[\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^n}\right]^T \Lambda^n_{\mathbf{U}} = -\left[\frac{\partial L^n}{\partial \mathbf{U}^n}\right]^T \tag{67}$$

The flow adjoint can be obtained by iteratively solving the linear system shown in equation (67). The solution convergence is similar to the convergence of any single linear system arising while solving the non-linear flow constraint equation since both problems contain the same eigenvalues. Solving for the flow adjoint and substituting in equation (64) results in three terms $(A)^n$,$(B)^n$ and $(C)^n$. The remaining preexisting contribution to $\frac{\partial L^n}{\partial D}$ is denoted term $(E)^n$.

$$\underbrace{\frac{\partial \mathbf{x}^n_{int}}{\partial D}^T \frac{\partial \mathbf{R}^n}{\partial \mathbf{x}^n_{int}}^T \Lambda^n_{\mathbf{U}}}_{(A)^n} + \underbrace{\frac{\partial \mathbf{x}^{n-1}_{int}}{\partial D}^T \frac{\partial \mathbf{R}^n}{\partial \mathbf{x}^{n-1}_{int}}^T \Lambda^n_{\mathbf{U}}}_{(B)^n} + \underbrace{\frac{\partial \mathbf{U}^{n-1}}{\partial D}^T \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^{n-1}}^T \Lambda^n_{\mathbf{U}}}_{(C)^n} + \underbrace{\frac{\partial \mathbf{x}^n_{int}}{\partial D}^T \frac{\partial L^n}{\partial \mathbf{x}^n_{int}}^T}_{(E)^n} \tag{68}$$

Terms $(A)^n$, $(B)^n$ and $(E)^n$ can be expressed in terms of the surface displacement sensitivities of the deformed airfoil as:

$$(A)^n = \frac{\partial \mathbf{x}^0_{surf}}{\partial D}^T \frac{\partial \mathbf{x}^n_{surf}}{\partial \mathbf{x}^0_{surf}}^T \frac{\partial \mathbf{x}^n_{int}}{\partial \mathbf{x}^n_{surf}}^T \frac{\partial \mathbf{R}^n}{\partial \mathbf{x}^n_{int}}^T \Lambda^n_{\mathbf{U}} \tag{69}$$

$$(B)^n = \frac{\partial \mathbf{x}^0_{surf}}{\partial D}^T \frac{\partial \mathbf{x}^{n-1}_{surf}}{\partial \mathbf{x}^0_{surf}}^T \frac{\partial \mathbf{x}^{n-1}_{int}}{\partial \mathbf{x}^{n-1}_{surf}}^T \frac{\partial \mathbf{R}^n}{\partial \mathbf{x}^{n-1}_{int}}^T \Lambda^n_{\mathbf{U}} \tag{70}$$

$$(E)^n = \frac{\partial \mathbf{x}^0_{surf}}{\partial D}^T \frac{\partial \mathbf{x}^n_{surf}}{\partial \mathbf{x}^0_{surf}}^T \frac{\partial \mathbf{x}^n_{int}}{\partial \mathbf{x}^n_{surf}}^T \frac{\partial \mathbf{L}^n}{\partial \mathbf{x}^n_{int}}^T \tag{71}$$

The second quantity in the expanded versions of these terms are merely transposes of the corresponding rotation matrices $[\beta]$. This can be derived starting from equation (25) as:

$$\delta \mathbf{x}^n_{surf} = \{[\beta]^n - [I]\}\mathbf{x}^0_{surf} \tag{72}$$

$$\mathbf{x}^n_{surf} = [\beta]^n \mathbf{x}^0_{surf} \tag{73}$$

$$\frac{\partial \mathbf{x}^n_{surf}}{\partial \mathbf{x}^0_{surf}} = [\beta]^n \tag{74}$$

$$\frac{\partial \mathbf{x}^n_{surf}}{\partial \mathbf{x}^0_{surf}}^T = [\beta]^{nT} \tag{75}$$

The mesh motion constraint equation provides the third quantity in these expressions.

$$[K]\,\delta \mathbf{x}_{int} = \delta \mathbf{x}_{surf} \tag{76}$$

$$[K]\frac{\partial \mathbf{x}_{int}}{\partial \mathbf{x}_{surf}} = [I] \tag{77}$$

$$\frac{\partial \mathbf{x}_{int}}{\partial \mathbf{x}_{surf}}^T = [K]^{-T} \tag{78}$$

Terms $(A)^n$,$(B)^n$ and $(E)^n$ can now be written as:

$$(A)^n = \frac{\partial \mathbf{x}^0_{surf}}{\partial D}^T [\beta]^{nT}[K]^{-T} \frac{\partial \mathbf{R}^n}{\partial \mathbf{x}^n_{int}}^T \Lambda^n_{\mathbf{U}} \tag{79}$$

$$(B)^n = \frac{\partial \mathbf{x}^0_{surf}}{\partial D}^T [\beta]^{n-1^T}[K]^{-T} \frac{\partial \mathbf{R}^n}{\partial \mathbf{x}^{n-1}_{int}}^T \Lambda^n_{\mathbf{U}} \tag{80}$$

$$(E)^n = \frac{\partial \mathbf{x}^0_{surf}}{\partial D}^T [\beta]^{nT}[K]^{-T} \frac{\partial \mathbf{L}^n}{\partial \mathbf{x}^n_{int}}^T \tag{81}$$

The computation of the second quantity in term $(C)^n$ is trivial since by examination of equation (52) it is obvious that $\mathbf{U}^{n-1}$ occurs linearly. The product of the second and third quantities in term $(C)^n$ is denoted $\lambda_{\mathbf{U}}^n$.

$$\lambda_{\mathbf{U}}^n = \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^{n-1}}^T \Lambda_{\mathbf{U}}^n = -\frac{A^{n-1}}{\Delta t} \Lambda_{\mathbf{U}}^n \tag{82}$$

In order to determine $\frac{\partial \mathbf{U}^{n-1}}{\partial D}$ in term $(C)^n$, a new expression that is analogous to equation (56) is written and transposed.

$$\frac{\partial \mathbf{U}^{n-1}}{\partial D} = -\left[\frac{\partial \mathbf{R}^{n-1}}{\partial \mathbf{U}^{n-1}}\right]^{-1} \left(\frac{\partial \mathbf{R}^{n-1}}{\partial \mathbf{x}_{int}^{n-1}} \frac{\partial \mathbf{x}_{int}^{n-1}}{\partial D} + \frac{\partial \mathbf{R}^{n-1}}{\partial \mathbf{x}_{int}^{n-2}} \frac{\partial \mathbf{x}_{int}^{n-2}}{\partial D} + \frac{\partial \mathbf{R}^{n-1}}{\partial \mathbf{U}^{n-2}} \frac{\partial \mathbf{U}^{n-2}}{\partial D} \right) \tag{83}$$

$$\frac{\partial \mathbf{U}^{n-1}}{\partial D}^T \lambda_{\mathbf{U}}^n = -\left(\frac{\partial \mathbf{x}_{int}^{n-1}}{\partial D}^T \frac{\partial \mathbf{R}^{n-1}}{\partial \mathbf{x}_{int}^{n-1}}^T + \frac{\partial \mathbf{x}_{int}^{n-2}}{\partial D}^T \frac{\partial \mathbf{R}^{n-1}}{\partial \mathbf{x}_{int}^{n-2}}^T + \frac{\partial \mathbf{U}^{n-2}}{\partial D}^T \frac{\partial \mathbf{R}^{n-1}}{\partial \mathbf{U}^{n-2}}^T \right) \left[\frac{\partial \mathbf{R}^{n-1}}{\partial \mathbf{U}^{n-1}}\right]^{-T} \lambda_{\mathbf{U}}^n \tag{84}$$

The flow adjoint variable Λ_U^{n-1} for the time-level $n-1$ can now be defined and solved as:

$$\left[\frac{\partial \mathbf{R}^{n-1}}{\partial \mathbf{U}^{n-1}}\right]^T \Lambda_{\mathbf{U}}^{n-1} = -\lambda_{\mathbf{U}}^n \tag{85}$$

The final form of term $(C)^n$ is therefore:

$$(C)^n = \underbrace{\frac{\partial \mathbf{x}_{int}^{n-1}}{\partial D}^T \frac{\partial \mathbf{R}^{n-1}}{\partial \mathbf{x}_{int}^{n-1}}^T \Lambda_{\mathbf{U}}^{n-1}}_{(A)^{n-1}} + \underbrace{\frac{\partial \mathbf{x}_{int}^{n-2}}{\partial D}^T \frac{\partial \mathbf{R}^{n-1}}{\partial \mathbf{x}_{int}^{n-2}}^T \Lambda_{\mathbf{U}}^{n-1}}_{(B)^{n-1}} + \underbrace{\frac{\partial \mathbf{U}^{n-2}}{\partial D}^T \frac{\partial \mathbf{R}^{n-1}}{\partial \mathbf{U}^{n-2}}^T \Lambda_{\mathbf{U}}^{n-1}}_{(C)^{n-1}} \tag{86}$$

As in equation (80), $(A)^{n-1}$ from equation (86) can again be written in terms of the surface displacement sensitivity of the deformed airfoil as:

$$(A)^{n-1} = \frac{\partial \mathbf{x}_{surf}^0}{\partial D}^T [\beta]^{n-1^T} [K]^{-T} \frac{\partial \mathbf{R}^{n-1}}{\partial \mathbf{x}_{int}^{n-1}}^T \Lambda_{\mathbf{U}}^{n-1} \tag{87}$$

Combining terms $(A)^{n-1}$ and $(B)^n$ a mesh motion adjoint variable $\Lambda_{\mathbf{x}}^{n-1}$ is defined and solved for as shown in equation (88).

$$[K]^T \Lambda_{\mathbf{x}}^{n-1} = \left\{ \left(\frac{\partial \mathbf{R}^n}{\partial \mathbf{x}_{int}^{n-1}}\right)^T \Lambda_{\mathbf{U}}^n + \left(\frac{\partial \mathbf{R}^{n-1}}{\partial \mathbf{x}_{int}^{n-1}}\right)^T \Lambda_{\mathbf{U}}^{n-1} \right\} \tag{88}$$

The mesh motion adjoint variable can be iteratively solved for with cost similar to a mesh motion solution. The coefficient matrices again for both sets of equations are the same with the exception of the transpose. Note that although technically the combination of $(A)^{n-1}$ and $(B)^n$ to form a single mesh adjoint is not required, this step reduces the number of mesh adjoint solutions per time-level from two to one. The combined term $(A)^{n-1} + (B)^n$ can now be computed as:

$$(A)^{n-1} + (B)^n = \frac{\partial \mathbf{x}_{surf}^0}{\partial D}^T [\beta]^{n-1^T} \Lambda_{\mathbf{x}}^{n-1} \tag{89}$$

A recurrence pattern is clearly seen from equation (86). $\partial \mathbf{U}^{n-2}/\partial D$ in equation (86) will have to be solved for by substituting an expression analogous to equation (83) and defining an adjoint variable $\Lambda_{\mathbf{U}}^{n-2}$. This will again yield a term $(A)^{n-2}$ that can be combined with $(B)^{n-1}$ and a mesh adjoint $\Lambda_{\mathbf{x}}^{n-2}$ can be defined and solved. The procedure described above is simply an integration backward in time from time-level $n = n_f$ to the starting time-level $n = 0$. When the reverse time-integration reaches the first time-level, the recurrence phenomenon stops since the quantity $\partial \mathbf{U}^0/\partial D$ is defined based on initial conditions or can be solved for by obtaining a steady-state adjoint solution. Also, just as $(B)^n$ and $(A)^{n-1}$ were combined to define and solve the mesh adjoint $\Lambda_{\mathbf{x}}^{n-1}$, $(A)^n$ and $(E)^n$ are combined at the final time-level n to define and solve a mesh adjoint $\Lambda_{\mathbf{x}}^n$.

$$[K]^T \Lambda_{\mathbf{x}}^n = \left\{ \left(\frac{\partial L^n}{\partial \mathbf{x}_{int}^n}\right)^T + \left(\frac{\partial \mathbf{R}^n}{\partial \mathbf{x}_{int}^n}\right)^T \Lambda_{\mathbf{U}}^n \right\} \tag{90}$$

This can then be multiplied to get the final answer for the combination of $(A)^n$ and $(E)^n$.

$$(A)^n + (E)^n = \frac{\partial \mathbf{x}^0_{surf}}{\partial D}^T [\beta]^{nT} \Lambda^n_{\mathbf{x}} \tag{91}$$

Adding all four terms yields the final sensitivity vector:

$$\frac{\partial L^n}{\partial D} = (A)^n + (B)^n + (C)^n + (E)^n \tag{92}$$

The procedure for computing the sensitivity vector $\frac{dL}{d\mathbf{D}}$ using the adjoint approach is as follows:

1. Begin backward time-integration at $n = n_f$.
2. Compute vectors $\frac{\partial L^n}{\partial \mathbf{U}^n}$ and $\frac{\partial L^n}{\partial \mathbf{x}^n_{int}}$.
3. Solve for flow adjoint $\Lambda^n_{\mathbf{U}}$ as:

$$\left[\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^n}\right]^T \Lambda^n_{\mathbf{U}} = -\left[\frac{\partial L^n}{\partial \mathbf{U}^n}\right]^T \tag{93}$$

4. Solve for mesh adjoint $\Lambda^n_{\mathbf{x}}$ as per equation (90) and multiply by surface displacement sensitivities of deformed airfoil and also by rotation matrix $[\beta]^n$ to get sum of $(A)^n$ and $(E)^n$ as per equation (91).
5. Multiply $\Lambda^n_{\mathbf{U}}$ by $-\frac{A^{n-1}}{\Delta t}$ to get $\lambda^n_{\mathbf{U}}$.
6. Solve for flow adjoint at previous time-level $n-1$ as:

$$\left[\frac{\partial \mathbf{R}^{n-1}}{\partial \mathbf{U}^{n-1}}\right]^T \Lambda^{n-1}_{\mathbf{U}} = -\lambda^n_{\mathbf{U}} \tag{94}$$

7. Solve for mesh adjoint $\Lambda^{n-1}_{\mathbf{x}}$ as per equation (88) and determine $(B)^n$.
8. Recursively integrate backward in time all the way to the first time-level in order to assemble $(C)^n$, solving for a flow adjoint and a mesh adjoint at each time-level.
9. At the end of the reverse integration process the complete sensitivity vector is available.

2. *Case-B: Time-Integrated Objective Function*

For a situation where the final objective function is obtained through integration in time or in discrete terms a function of some local objective computed at each time-level, it appears that the procedure described above would have to be repeated for each time-level to obtain $\frac{dL^n}{dD}$ after which $\frac{dL^g}{dD}$ can be assembled. A slight modification in the derivation of the unsteady adjoint can circumvent this tedious process. In the previous case the reverse time-integration was initiated at the final time-level and proceeded to the first time-level in order to compute the sensitivity of a global objective whose definition was based only on the solution and mesh coordinates at the final time-level. If the global objective L^g is redefined as a quantity integrated in time, or as a quantity dependent on the solutions at each time-level $n = n_f, n_f - 1, ...2, 1, 0$, the effect of these local sensitivities can be gathered in a single backward integration beginning at the final time-level, thus eliminating the need to evaluate these independently for each time-level n. Expanding equation (63) and reversing the summation order yields:

$$\frac{dL^g}{d\mathbf{D}}^T = \frac{dL^n}{d\mathbf{D}}^T \frac{dL^g}{dL^n}^T + \frac{dL^{n-1}}{d\mathbf{D}}^T \frac{dL^g}{dL^{n-1}}^T \cdots\cdots\cdots\cdots + \frac{dL^0}{d\mathbf{D}}^T \frac{dL^g}{dL^0}^T \tag{95}$$

Further expanding the local objective sensitivities:

$$\frac{dL^g}{d\mathbf{D}}^T = \underbrace{\left(\frac{\partial \mathbf{U}^n}{\partial \mathbf{D}}^T \frac{\partial L^n}{\partial \mathbf{U}^n}^T + \frac{\partial \mathbf{x}^n_{int}}{\partial \mathbf{D}}^T \frac{\partial L^n}{\partial \mathbf{x}^n_{int}}^T\right) \frac{dL^g}{dL^n}^T}_{(X)} + \underbrace{\left(\frac{\partial \mathbf{U}^{n-1}}{\partial \mathbf{D}}^T \frac{\partial L^{n-1}}{\partial \mathbf{U}^{n-1}}^T + \frac{\partial \mathbf{x}^{n-1}_{int}}{\partial \mathbf{D}}^T \frac{\partial L^{n-1}}{\partial \mathbf{x}^{n-1}_{int}}^T\right) \frac{dL^g}{dL^{n-1}}^T}_{(Y)} \cdots\cdots \tag{96}$$

Case-A presented a method to compute term (X). Equation (96) falls back to Case-A if the terms $\partial L^g/\partial L^n$ for $n = n_f - 1, \ldots.0$ vanish and $\partial L^g/\partial L^{n_f} = 1$. In order to avoid computing term (Y) independently of term (X), the derivation from Case-A is modified to include its effect. This applies to all terms recursively appearing in the expansion shown in equation (95). The modifications to the derivation are:

1. Firstly the flow adjoint variable indicated in equation (65) is redefined as:

$$\Lambda_{\mathbf{U}}^n = -\left[\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^n}\right]^{-T}\left[\frac{\partial L^n}{\partial \mathbf{U}^n}\right]^T\left[\frac{dL^g}{dL^n}\right]^T \tag{97}$$

2. $(E)^n$ in equation (81) is modified to include the global-to-local objective sensitivity:

$$(E)^n \quad = \quad \frac{\partial \mathbf{x}_{surf}^0}{\partial D}^T [\beta]^{nT}[K]^{-T}\frac{\partial \mathbf{L}^n}{\partial \mathbf{x}_{int}^n}^T \frac{dL^g}{dL^n}^T \tag{98}$$

3. Previously term $(C)^n$ was defined as:

$$(C)^n = \frac{\partial \mathbf{U}^{n-1}}{\partial \mathbf{D}}^T \lambda_{\mathbf{U}}^n \tag{99}$$

   A crucial difference in the definition of term $(C)^n$ is to be emphasized at this juncture. The basis of the previous derivation was the substitution of a suitable expression for the flow variable sensitivities $\partial \mathbf{U}^{n-1}/\partial \mathbf{D}$ in order to proceed with the backward integration. While this remains true for Case-B, examination of term (Y) in equation (96) indicates an additional contribution to $\lambda_{\mathbf{U}}^n$ from time-level $n-1$. Modifying the definition of $\lambda_{\mathbf{U}}^n$ to include this contribution results in:

$$\lambda_{\mathbf{U}}^n = \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^{n-1}}^T \Lambda_{\mathbf{U}}^n + \frac{\partial L^{n-1}}{\partial \mathbf{U}^{n-1}}^T \frac{dL^g}{dL^{n-1}}^T \tag{100}$$

   It is clear that although the integration was initiated at the final time-level n_f, the effect of the local-to-global sensitivity from time-level $n_f - 1$ has been included via the second term, thus eliminating the necessity to evaluate this independently.

4. Similarly the mesh adjoint $\Lambda_{\mathbf{x}}^{n-1}$ is redefined such that it includes objective information from time-level $n-1$. The additional contribution in this case is due to the appearance of $\partial \mathbf{x}_{int}^{n-1}/\partial \mathbf{D}$ in term (Y) of equation (96).

$$[K]^T \Lambda_{\mathbf{x}}^{n-1} = \left\{\left(\frac{\partial \mathbf{R}^n}{\partial \mathbf{x}_{int}^{n-1}}\right)^T \Lambda_{\mathbf{U}}^n + \left(\frac{\partial \mathbf{R}^{n-1}}{\partial \mathbf{x}_{int}^{n-1}}\right)^T \Lambda_{\mathbf{U}}^{n-1} + \left(\frac{\partial L^{n-1}}{\partial \mathbf{x}^{n-1}}\right)^T\left(\frac{dL^g}{dL^{n-1}}\right)^T\right\} \tag{101}$$

5. The mesh adjoint at the final time-level $\Lambda_{\mathbf{x}}^n$ is also changed to include the global-to-local objective relation.

$$[K]^T \Lambda_{\mathbf{x}}^n = \left\{\left(\frac{\partial L^n}{\partial \mathbf{x}_{int}^n}\right)^T\left(\frac{dL^g}{dL^n}\right)^T + \left(\frac{\partial \mathbf{R}^n}{\partial \mathbf{x}_{int}^n}\right)^T \Lambda_{\mathbf{U}}^n\right\} \tag{102}$$

6. Additionally, since there are local objectives computed at each time-level, the linearizations of these with respect to the mesh coordinates and flow solution at the respective time-levels are required. This translates into evaluating the vectors $\frac{\partial L^n}{\partial \mathbf{U}^n}$ and $\frac{\partial L^n}{\partial \mathbf{x}_{int}^n}$ at each time-level n.

F. Modification of Formulation for BDF2 Time-Integration Scheme

The unique character of the derivation permits the application of the same technique regardless of the number of backward steps required in the time discretization. For a two-step backward difference formula, the constraint equation has contributions from time indices n,$n-1$ and $n-2$. The new constraint equation written in functional form is therefore:

$$\mathbf{R}^n = F\left[\mathbf{x}_{int}^n, \mathbf{x}_{int}^{n-1}, \mathbf{x}_{int}^{n-2}\mathbf{U}^n, \mathbf{U}^{n-1}, \mathbf{U}^{n-2}\right] = 0 \tag{103}$$

The flow variable sensitivity at time-index n obtained through differentiation of equation (103) with respect to design variables $\mathbf{D}$ is:

$$\frac{\partial \mathbf{U}^n}{\partial D} = -\left[\frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^n}\right]^{-1}\left(\frac{\partial \mathbf{R}^n}{\partial \mathbf{x}_{int}^n}\frac{\partial \mathbf{x}_{int}^n}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{x}_{int}^{n-1}}\frac{\partial \mathbf{x}_{int}^{n-1}}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{x}_{int}^{n-2}}\frac{\partial \mathbf{x}_{int}^{n-2}}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^{n-1}}\frac{\partial \mathbf{U}^{n-1}}{\partial D} + \frac{\partial \mathbf{R}^n}{\partial \mathbf{U}^{n-2}}\frac{\partial \mathbf{U}^{n-2}}{\partial D}\right) \tag{104}$$

The linearization at any 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 (100) and (101) will now have additional source terms from time-index $n-2$. 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 is required at each time-level.

V. Implementation Details

A. Code Structure

The software is divided into three main components, namely the flow solver, the adjoint/tangent solver, and the optimization script. 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 adjoint and tangent solvers consist of linearizations of the physics routines used in the flow solver and iterative solution routines retained from the flow solver. The flow solver performs the time-integration process to compute an unsteady flow solution that is then written to disk. The adjoint or tangent solver reads the solution from disk and computes the required sensitivities. A simple *bash* script is used as the optimization controller and calls the flow and adjoint/tangent solvers. All data exchange between the flow and adjoint/tangent solvers occur via files written to disk. This is necessary since the adjoint solver performs a backward integration in time and thus requires the 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 consume trivial amount of time. In contrast to the adjoint solver, the tangent solver has the ability to compute sensitivities simultaneously with the flow solver but with a lag of one time-level. This can be done since the tangent solver also performs a forward integration in time and can discard old values as it proceeds forward in time.

B. Linearization for $2^{nd}-$ 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 $\partial R/\partial U$ and is used in the Newton solver employed to obtain the flow solution. In the case of a spatially first-order accurate discretization this results in a nearest neighbor stencil where the flow residual 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 since the residual depends not only on the state variables but also their spatial gradients. The gradients are functions of interpolated nodal state values which in turn are functions of the states of elements that share that node. 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 be available since only the solution of $R(U) = 0$ is necessary. In order to achieve second-order accuracy, it is only required that the flow residual itself be constructed using second-order extrapolations of U. An inexact Newton's method is employed where the first-order accurate flow Jacobian matrix is used in solving second-order accurate residual equations $R(U)$. 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] This becomes especially true in regions of the flow where second-order accuracy is important or necessary. The linear systems that involve the flow Jacobian matrix are equations (62) and (67). 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 equation (67). In the case where the flow Jacobian matrix is second-order accurate, this can be split into the sum of two matrix-vector products as follows:

$$\left[\frac{\partial R}{\partial U}\right]_2^T \Lambda_U = \left\{ \left[\frac{\partial R}{\partial U}\right]_1 + \left[\frac{\partial R}{\partial \nabla U}\right]\left[\frac{\partial \nabla U}{\partial U}\right] \right\}^T \Lambda_U \tag{105}$$

$$= \underbrace{\left[\frac{\partial R}{\partial U}\right]_1^T \Lambda_U}_{(1)} + \underbrace{\left[\frac{\partial \nabla U}{\partial U}\right]^T \left[\frac{\partial R}{\partial \nabla U}\right]^T \Lambda_U}_{(2)} \tag{106}$$

where $[\partial R/\partial U]_1$ is the first-order Jacobian. Term (1) can be easily computed since 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 is available. The defect correction method is similar to a non-linear solution method and can mathematically be expressed as:

$$[P]\delta\Lambda_U^k = -\left\{ \left[\frac{\partial R}{\partial U}\right]_2^T \Lambda_U^k - \left[\frac{\partial L}{\partial U}\right]^T \right\} \tag{107}$$

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 equation (107) 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.[26] The right-hand-side includes the effect of the second-order flow Jacobian matrix and is evaluated as described by equation (106).

C. Linearization of Roe-Averaged Flux Jacobian Matrix

The flux model used in the analysis code is based on Roe's approximate Riemann solver. Typically the flux Jacobian $[\hat{A}]$ is assumed to be a constant during the flow residual linearization process, since in most cases its linearization has little effect on the overall accuracy of the flow Jacobian matrix $\partial R/\partial U$.[25] Considering that prior work on the importance of including the exact linearization of the $[\hat{A}]$ matrix was done in the context of steady-state problems, it was decided that its effect in unsteady flows must be studied. Several methods to include the effect of the $[\hat{A}]$ matrix were investigated in the process. These were:

1. Hybrid: finite-difference + linearization (FDL) of flux routines
2. Hybrid: complex variable differentiation + linearization (CVDL) of flux routines
3. Exact linearization

The easiest method was to approach the problem from a hybrid standpoint, where an exact linearization is used to compute all components of the flow Jacobian matrix except the $[\hat{A}]$ matrix which is finite-differenced or differentiated using the complex variable method. The hybrid techniques are relatively easier in the sense that a tedious linearization of the dissipative flux model need not be undertaken. The disadvantage of using FDL or CVDL to obtain the linearization of the $[\hat{A}]$ matrix is that it involves five or four function calls of the Roe flux routine respectively, rather than just one function call of the linearized routine. Additionally the step-size for the finite difference in FDL remains a variable, and it would be uneconomical to perform finite-differencing with several different step-sizes to choose a suitable value during the course of an optimization. The CVDL method is more involved than the FDL method since it requires the redefinition of variables used in functional evaluations as complex variables. The CVDL method also requires a step-size but is not as sensitive as the FDL method to the step-size itself. Details on the implementation of this method can be found in the following subsection. An exact linearization of the dissipative flux model was also performed but no comparisons in terms of cost were done. From an accuracy standpoint it would appear that an exact linearization would be more appropriate since the choice of step-size for the hybrid methods remains a variable.

D. Complex Variable Based Differentiation[27,28]

Any function operator $f(x)$ that operates on a real variable x can be utilized to compute the derivative $f'(x)$ by redefining the input variable x and all intermediate variables used in $f(x)$ as a complex variables. For a complex input, the function when redefined as described produces a complex output. The derivative of the real function $f(x)$ can be computed by expanding the complex operator $f(x+ih)$. The expansion of the complex operator can be written as:

$$f(x+ih) = f(x) + ihf'(x) + \cdots \tag{108}$$

The derivative $f'(x)$ is then simply:

$$f'(x) = \frac{Im\left[f\left(x+ih\right)\right]}{h} \tag{109}$$

In the case of the finite-difference method the step-size is of extreme importance. A large step-size will lead to inaccuracies in the computed slope, while on the other end, too small a step-size will generate errors due to machine precision. There is usually a small range of step-sizes over which the computed slope values converge, but in several cases no such region can be identified. The complex variable method overcomes this challenge although it is similar to the finite-difference method in the sense that it also requires a step size h. The advantage over the finite-difference method is that it is not sensitive to extremely small step sizes. The complex variable method is useful in validating sections of the linearized code when there are significant discrepancies between the finite-differenced and adjoint-based values computed for the sensitivity.

E. Optimization Procedure

The optimization algorithm used in the example problems is the simple steepest-descent method. The optimization process can be broken down into the following steps:

1. Input the baseline or deformed geometry.
2. Call the flow solver to obtain an unsteady solution.
3. Evaluate the objective function based on the unsteady solution.
4. Call the adjoint solver to obtain the sensitivity vector $\frac{dL}{dD}$.
5. Compute the perturbation of the design variables using the sensitivity as:

$$\delta\mathbf{D} = -\lambda\frac{dL}{d\mathbf{D}} \tag{110}$$

   where λ is the step size and the negative sign indicates the direction for minimizing the objective.
6. Deform the airfoil geometry using computed perturbation and the deformation function.
7. Repeat the process until an optimum solution is achieved.

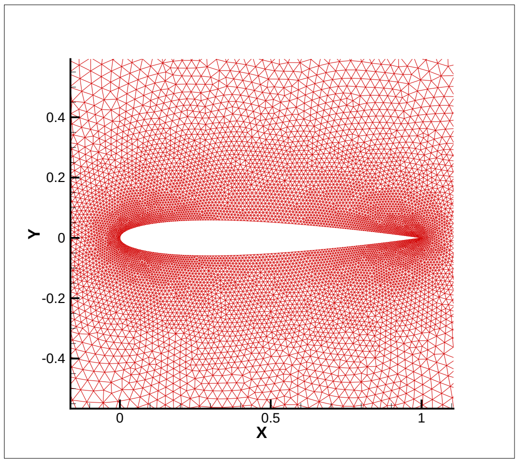
F. Linear Multigrid

A linear geometric agglomeration multigrid[20] strategy is used to accelerate convergence of all linear systems encountered in the solution process, be either, the flow, mesh motion, flow adjoint or mesh adjoint equations. The linear multigrid method uses a correction scheme where coarser levels recursively provide corrections to the fine grid solution. The coarse level meshes are built by repeated agglomeration or merging of neighboring control volumes to form a single control volume. Figures (3(a)) through (3(f)) show an unstructured triangular mesh around a NACA0012 airfoil and the agglomerated coarser levels. Consider a linear system developed by discretizing and linearizing a non-linear equation on the fine level mesh.

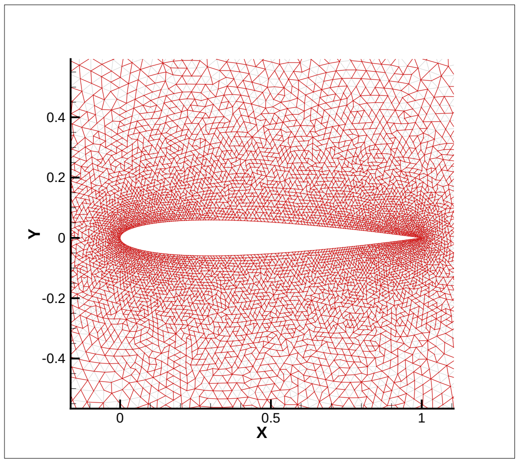
$$[A]_h\mathbf{x}_h = \mathbf{b}_h \tag{111}$$

An approximate solution to the linear system can be obtained by using conventional iterative methods such as Jacobi or Gauss-Seidel iterations. The residual of the linear system can then be written as:

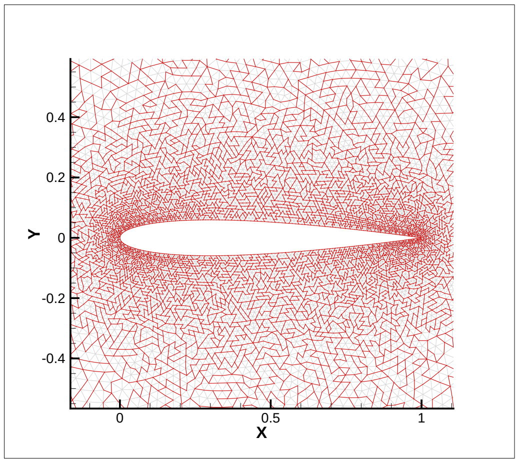
$$[A]_h\bar{\mathbf{x}}_h - \mathbf{b}_h = \mathbf{r}_h \tag{112}$$



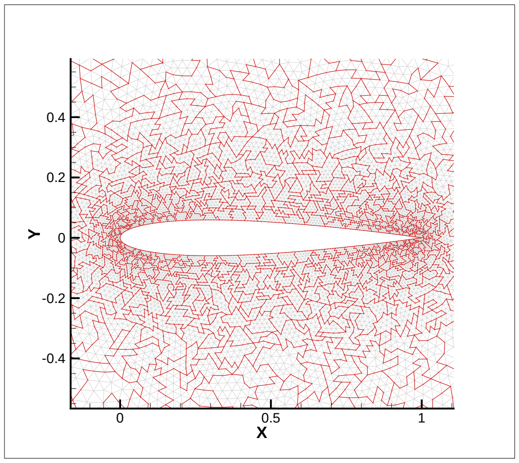
(a) Triangular fine level mesh



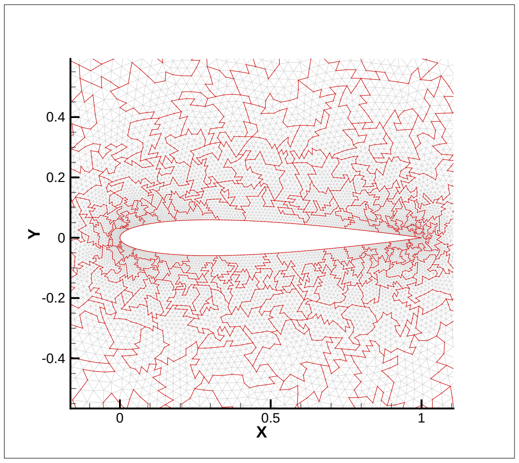
(b) Multigrid mesh level 2



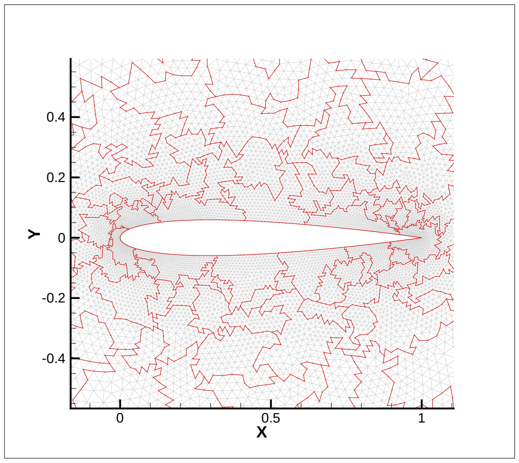
(c) Multigrid mesh level 3



(d) Multigrid mesh level 4



(e) Multigrid mesh level 5



(f) Multigrid mesh level 6

Figure 3. Agglomerated coarse level multigrid meshes

where $\bar{\mathbf{x}}_h$ is the approximate solution to the linear system. A correction $\mathbf{x}'_h$ to the approximate solution is defined such that the residual of the linear system on the fine level mesh goes to zero.

$$\mathbf{x}_h = \bar{\mathbf{x}}_h + \mathbf{x}'_h \tag{113}$$

By taking the difference between the correction equation and the residual equation on the fine mesh, a new linear system is defined as:

$$[A]_h \mathbf{x}'_h = -\mathbf{r}_h \tag{114}$$

Transferring this equation to a coarse level to form a coarse level correction equation yields:

$$[A]_H \mathbf{x}'_h = -I_h^H \mathbf{r}_h \tag{115}$$

where the subscript H refers to the coarse level and I_h^H is the fine-to-coarse level restriction operator. This system can now be iteratively solved approximately or exactly if H is coarse enough. If H is not coarse enough a new approximate solution is obtained and the residual of this system is then transferred to a coarser level. This process is repeated until H is coarse enough to obtain a numerically exact solution to the system. The exact or approximate solutions obtained from the coarse levels are used to correct the fine grid solution as:

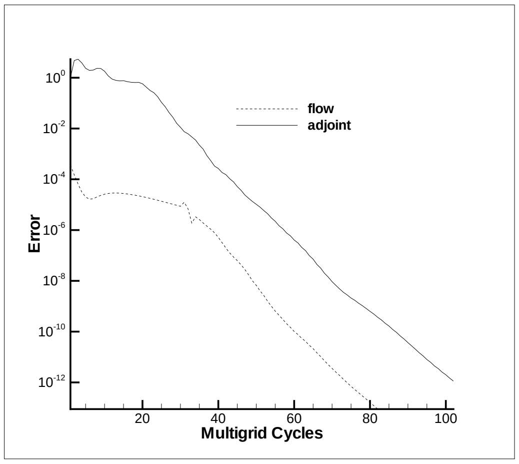
$$\bar{\mathbf{x}}_h^{new} = \bar{\mathbf{x}}_h + I_H^h \mathbf{x}'_H \tag{116}$$

where I_H^h refers to the coarse-to-fine grid prolongation operator. For the example problems, summation of parent residuals is used as the restriction operator for the right-hand-side of the linear system. For the flow problem, the coefficient matrix $[A]_H$ (flow Jacobian matrix) for the coarse levels are constructed by first restricting the fine level flow solution through a parent element area weighted approach, and then using the restricted flow solution to build the flow Jacobian matrix. In the case of mesh motion, summation of the edge spring coefficients is used as the restriction operator to construct the coarse level stiffness matrix $[K]_H$. For all cases direct injection of coarse level corrections into parent elements is chosen to be the prolongation operator.

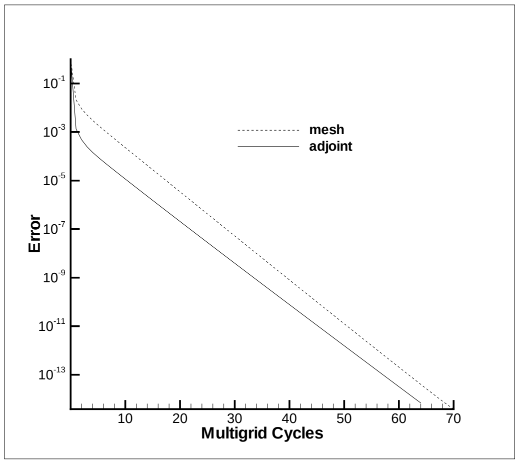
G. Efficiency

The total cost of an optimization is equal to the number of design iterations required 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 an adjoint solution. Reduction of the total number of design cycles to obtain an optimum solution can be achieved by utilizing more sophisticated optimization algorithms than the basic steepest-descent method. Since this is beyond the scope of this paper, the only approach to improve overall efficiency is to reduce the cost per design cycle.

For spatially first-order accurate schemes, the cost of obtaining a sensitivity solution is cheaper than the cost of a flow solution. This is because only linear solutions are required in the adjoint solver whereas a non-linear solution is necessary in the case of the flow equations. For spatially second-order accurate schemes the sensitivity solutions are slightly more expensive than in the first-order accurate case due to the defect-correction approach described earlier. Under any circumstance, the worst case scenario is that the cost of obtaining a sensitivity solution is the same as the cost of a flow solution. Additionally, considering the fact that unsteady solutions are inherently expensive, all possible methods to improve efficiency both in the case of the flow solver and the adjoint solver must be exploited to the fullest extent. This necessitates the use of multigrid methods to reduce dependency on mesh resolution and also necessitates the use of high-order time-integration schemes to obtain high quality unsteady solutions with minimal number of time-steps.[29] Following the same argument, it is important that adjoint methods developed for unsteady problems be applicable to any order of time-integration scheme used. 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 problems. The algorithm takes into account high-order BDF time-integration schemes and can be extended to IRK schemes without severe additional cost. The algorithm also utilizes the linear multigrid method for convergence acceleration. Figure (4(a)) compares the convergence of the non-linear flow equations and the defect-correction scheme to compute the flow adjoint when using multigrid. The similarity in convergence rates of both systems is evidenced by the comparable slopes of both curves. Figure (4(b)) shows the equivalent plots for the mesh motion equations and mesh adjoint equations.



(a) Convergence of flow equations and flow-adjoint



(b) Convergence of mesh equations and mesh-adoint

Figure 4. Convergence rate comparisons

VI. Validation of Adjoint Sensitivity

A. Basic Validation Procedure

The procedure to validate the computed sensitivity is two-fold. First the sensitivity computed using the forward linearization is verified by comparing against finite-differenced values. Then duality between transpose operations is invoked to assure that the sensitivity vector computed using the adjoint linearization matches that computed using the forward linearization. The test case chosen for sensitivity validation was a pitching NACA0012 airfoil under transonic conditions. The mesh resolution was chosen to be particularly coarse to allow for rapid solutions. For finite-differencing, each surface node on the airfoil was perturbed and its effect on the global objective was recorded. Similarly the forward linearization code was called n_{surf} number of times to obtain the entire sensitivity vector. The finite-differencing was done over a range of step-sizes in order to obtain a converged value.

B. Duality of Transpose Operations

The forward linearization is far more intuitive than the adjoint linearization where 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. A more tangible approach is to ensure accuracy of the forward solution via finite-differencing and then to invoke the principle of duality to track down errors in the reverse linearization.

Consider two operations on a vector based on the same coefficient matrix. In the case of the forward problem this can be described simply as a matrix-vector product. In the case of the adjoint problem it can be recast as the product of the transposed coefficient matrix and the vector. The duality principle relates the results of the the two product operations to each other as follows:

$$[A]x = f \tag{117}$$

$$[A]^T y = g \tag{118}$$

$$x^T g = f^T y \tag{119}$$

Sections of the code where transpose operations are performed are validated by using random input vectors x and y to determine the corresponding outputs f and g. Equation (119) relates two scalar values and in the case of perfectly dual operations should match to machine precision. Although duality using a particular set of input vectors is a necessary test to ensure accuracy of the transpose operation, it is by no means sufficient. As a consequence the duality test is performed using several random input vectors in order to validate the operation.

C. Validity of Finite-Difference Method

It was noticed during the validation process that under transonic or supersonic conditions there were significant deviations of the adjoint based sensitivity from finite-differenced values especially in the vicinity of shockwaves. The issue was narrowed down to the Roe-averaged flux Jacobian matrix $[\hat{A}]$. Figure (5) shows the sensitivity distribution for the validation case computed using different methods. The shockwave for the validation case occurs in the vicinity of node $ID = 110$. At this location and around it there is large difference between the adjoint based sensitivity and the finite-difference based values. Utilizing the hybrid FDL method for the Roe flux collapses the overall finite-difference curve and the hybrid curve almost onto each other, thus affirming that it is indeed the $[\hat{A}]$ matrix that is the source of the problem. Surprisingly, the exact linearization of the $[\hat{A}]$ matrix produces results not very different from the adjoint sensitivity based on a frozen or constant $[\hat{A}]$ matrix. This leads to questioning the entire premise of utilizing finite-differencing for the purpose of adjoint sensitivity validation. In order to determine the validity of the finite-differenced values themselves, the hybrid CVDL approach was invoked. Figure (5) clearly indicates that the sensitivity computed using the hybrid adjoint solver based on the CVDL method matches very well with the exact linearization of the $[\hat{A}]$ matrix, thus confirming the validity of the exact linearization.

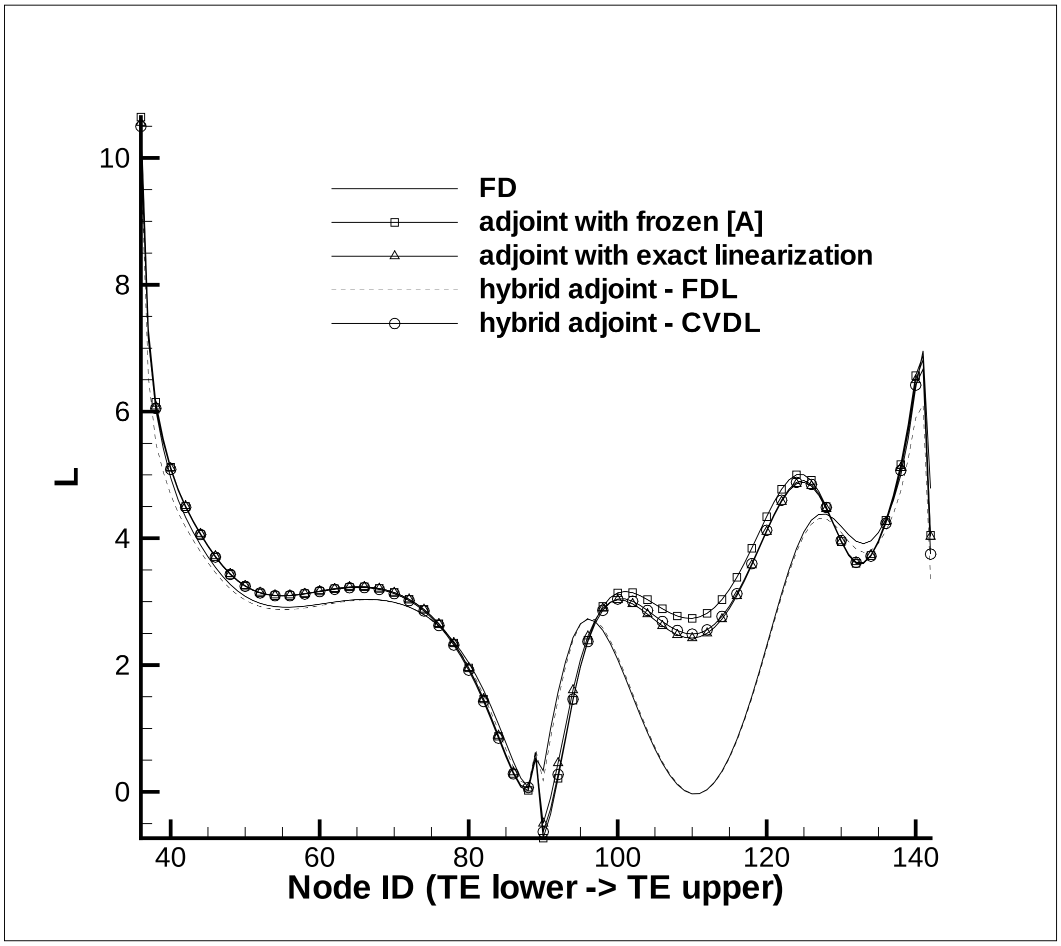


Figure 5. Comparison of sensitivities computed using different methods

VII. Optimization Examples

Both optimization examples presented involve a pitching airfoil under identical transonic conditions. The flow conditions chosen were:

1. Free-stream Mach number of $M_\infty = 0.755$

2. Mean angle-of-attack of $\alpha_0 = 0.016^o$
3. Amplitude of pitch of $\alpha_{max} = 2.51^o$
4. Reduced frequency of $\omega = 0.1628$
5. Time-integration from $t = 0$ to $t = 54$ with 27 time-steps and a uniform time-step size of $dt = 2.0$

As described earlier the design variables are the weights controlling the magnitudes of bumps placed at surface nodes of the baseline airfoil. The examples utilized the temporally and spatially second-order accurate cell-centered finite-volume code in order to obtain the unsteady flow solution, and the corresponding linearized adjoint solver to compute the sensitivities. It should be noted that an exact linearization was used for the Roe-averaged flux Jacobian although a linearization with a frozen $[\hat{A}]$ matrix would have sufficed. The computational mesh consisted of approximately 20,000 triangular elements with 289 nodes defining the surface of the airfoil, which translates into an equal number of design variables. Figure (6) shows a zoomed-in view of the computational mesh used in the optimization examples.

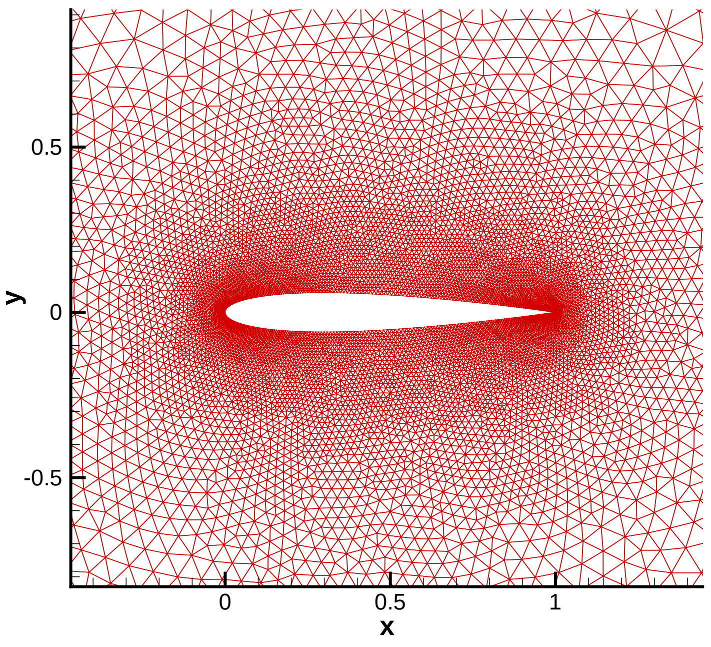


Figure 6. Computational mesh of approximately 20,000 elements used in optimization examples

A. Case-1: Time-dependent Load Matching

The target time-dependent load profile in this case was that of the NACA64210 airfoil undergoing sinusoidal pitching with the prescribed flow conditions. The global objective function was based on an RMS difference between the targeted and computed time-dependent load profiles. The undeformed baseline airfoil used as the starting point for the optimization was the symmetric NACA0012 airfoil. The optimization step-size for this particular example was scaled at each design iteration by the objective computed at the previous design iteration raised to the power of $3/2$ in addition to being chosen such that it always produces an improved design. This was implemented in order to automatically use smaller step-sizes when close to the optimum solution. The optimization was stopped after 90 design iterations due to a diminished rate of convergence and all results presented are from this point. Figure (7) shows the optimized airfoil from the final design iteration. Since only the matching of loads was required in this case, the optimized airfoil bears no similarity to a NACA64210 airfoil on which the target load profile was based on. A pronounced hook shape close to the trailing edge of the optimized airfoil is clearly 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 free-stream 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 since 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 an optimization based on the Navier-Stokes equations in comparison with those presented here. The sensitivity computed by the adjoint solver was smoothed via a few passes of Jacobi iterations in order to remove or mitigate spikes that occur close to the trailing edge. The trailing edge itself is a singularity and it is customary to ignore the sensitivity computed at that location in order to physically constrain it. Figure (8) shows the convergence of the global objective function L^g. There is almost an order of magnitude reduction in the objective at the end of the very first design iteration, but from thereon the convergence falls to a slower rate. Figures (9) and (10) 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 due to the different scales of these coefficients. Figure (11) shows the distribution of the computed sensitivity at different stages in the optimization. The peaks occurring close to the trailing edge at the first design iteration indicate the strong sensitivity of shape in this region to the objective and explains the hook shape. Although there is almost an order of magnitude reduction in the objective at the very first design iteration it is clear that the optimum solution was overshot at this point and this is evidenced by the change in sign of the sensitivity curve for all design variables.

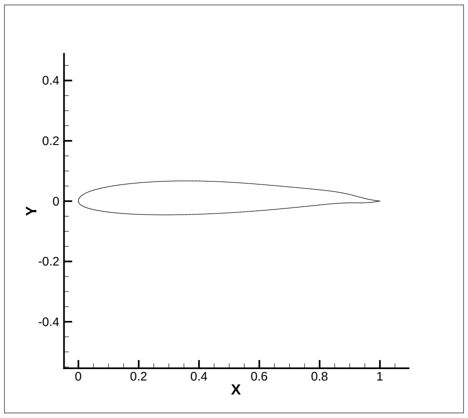


Figure 7. Optimized airfoil for Case-1

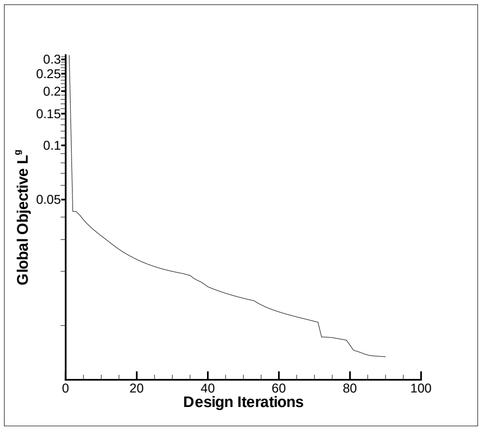
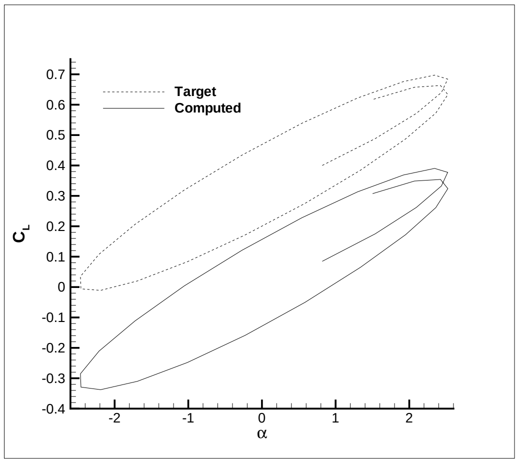
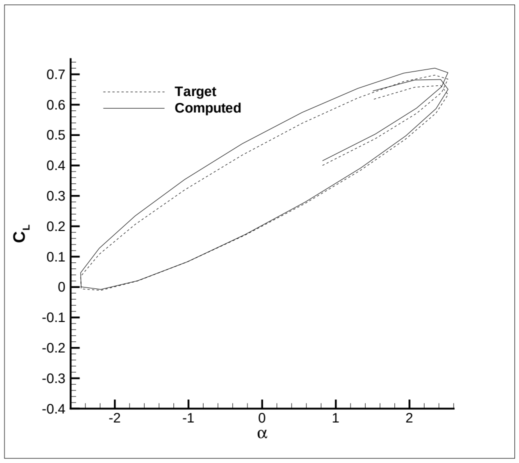


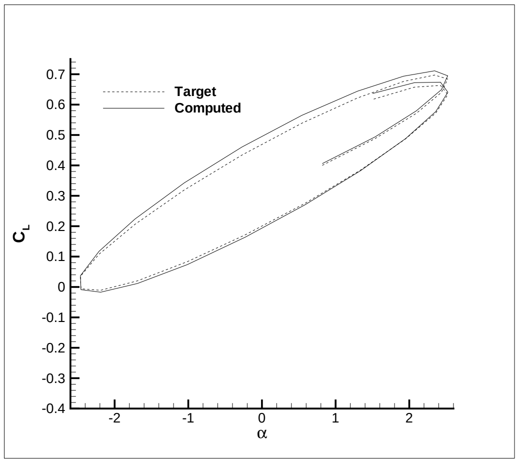
Figure 8. Convergence of global objective function L^g for optimization example 1



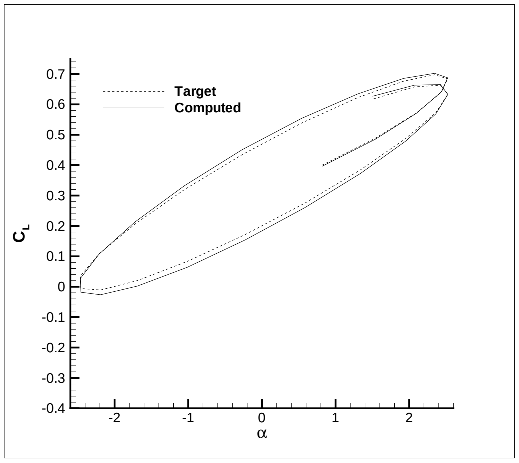
(a) 1st Design Iteration



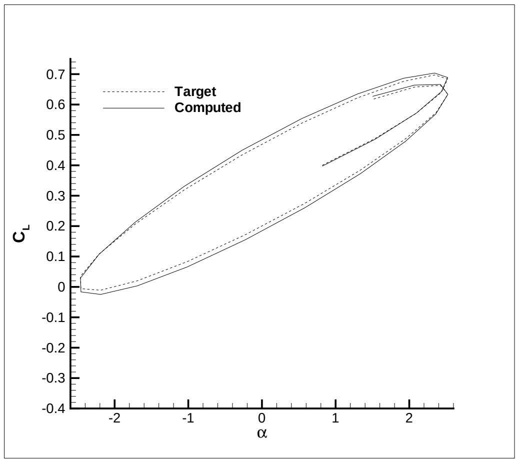
(b) 15th Design Iteration



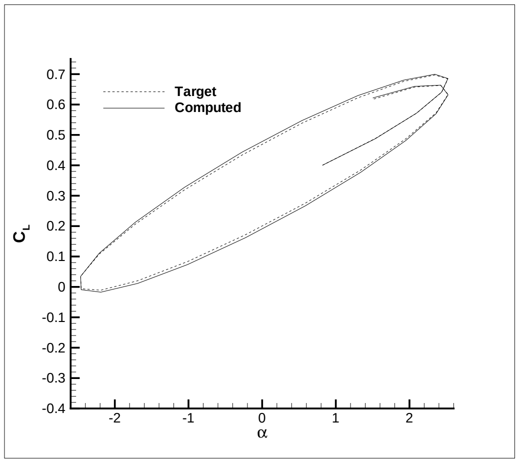
(c) 30th Design Iteration



(d) 45th Design Iteration

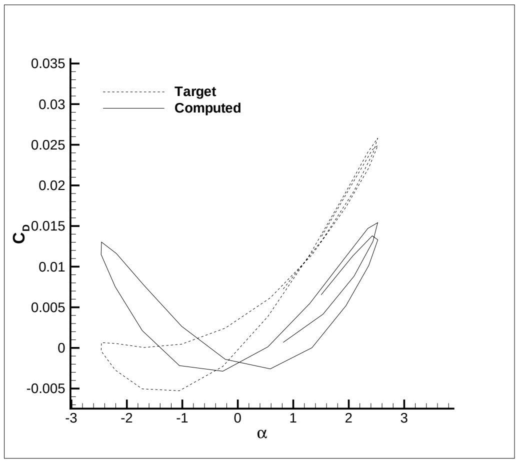


(e) 60th Design Iteration

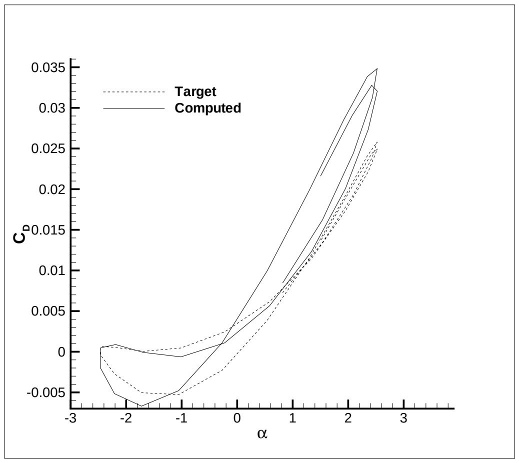


(f) 90th Design Iteration

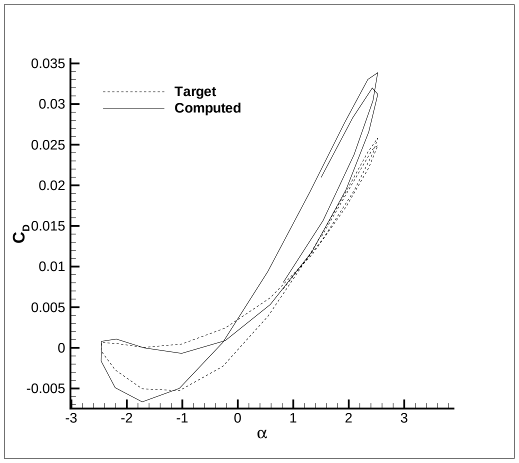
Figure 9. Comparison of target and computed time-dependent lift profiles at various design iterations for Case-1



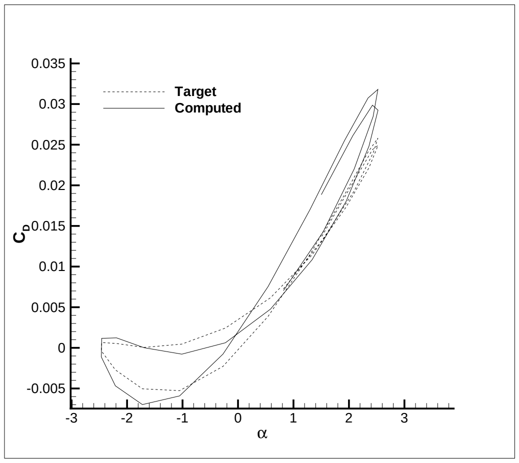
(a) 1st Design Iteration



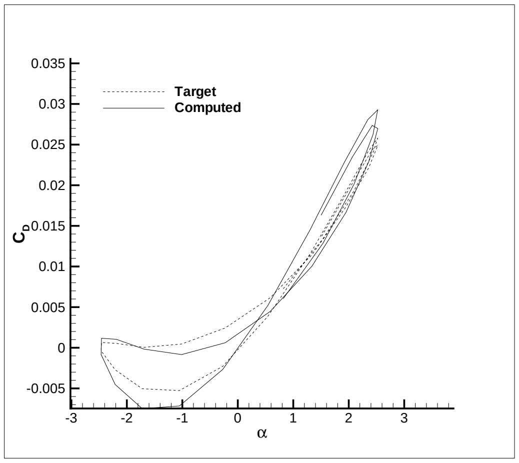
(b) 15th Design Iteration



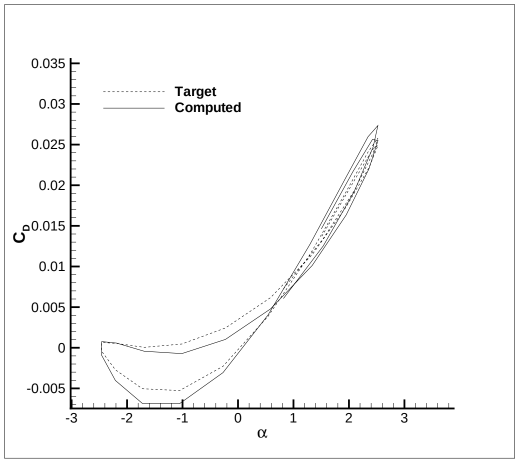
(c) 30th Design Iteration



(d) 45th Design Iteration

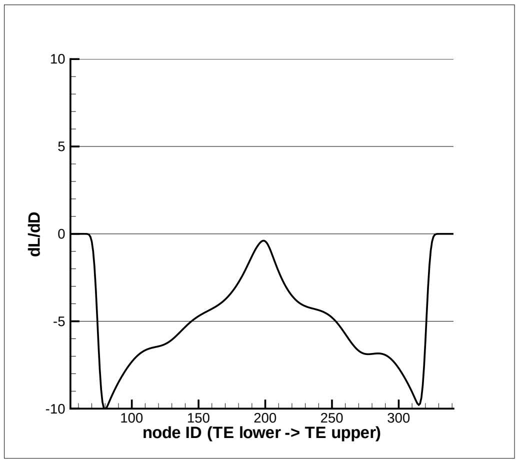


(e) 60th Design Iteration

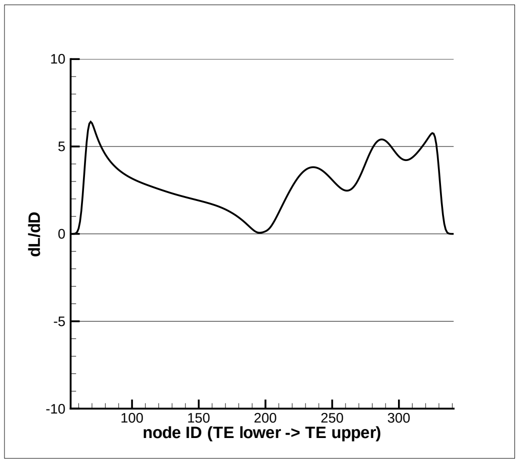


(f) 90th Design Iteration

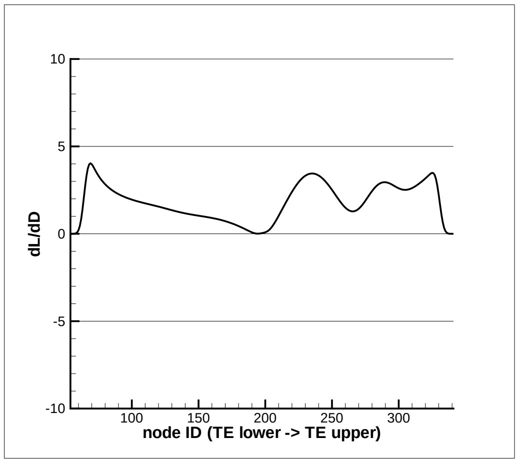
Figure 10. Comparison of target and computed time-dependent drag profiles at various design iterations for Case-1



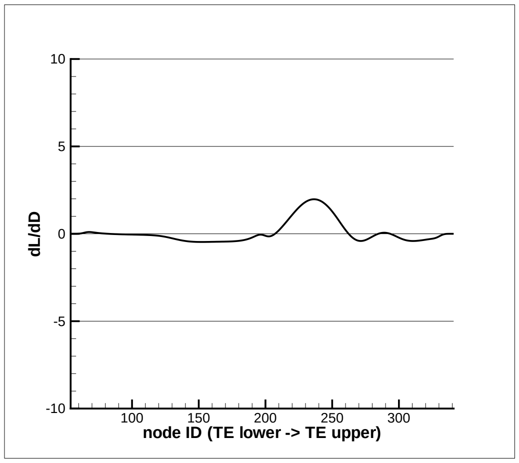
(a) 1st Design Iteration



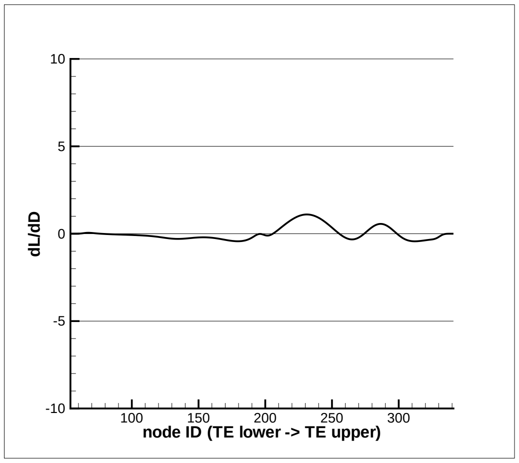
(b) 15th Design Iteration



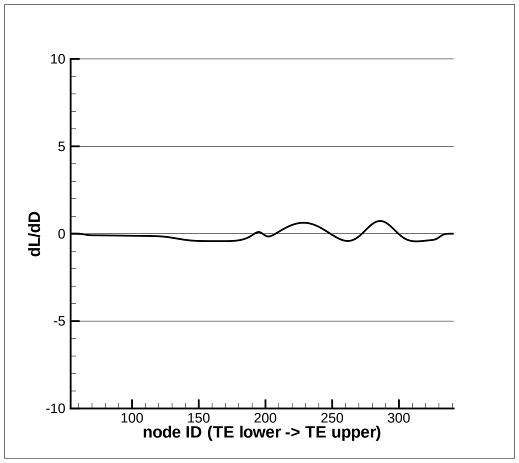
(c) 30th Design Iteration



(d) 45th Design Iteration



(e) 60th Design Iteration



(f) 90th Design Iteration

Figure 11. Sensitivity distribution at various design iterations for Case-1

B. Case-2: Time-Dependent Pressure Distribution Matching

The target time-dependent pressure profile for this example was that of a sinusoidally pitching NACA64210 type airfoil. The baseline airfoil which was to be deformed to match the time-dependent pressure profile of the target airfoil was again the symmetric NACA0012 airfoil. Since pressure distribution is a function of airfoil curvature, it is important to have prior knowledge about whether the geometric parametrization allows for the target airfoil shape to lie within the design space. An optimization utilizing only the shape parametrization independent of the flow equations was performed in order to check if this were true. It was determined that an exact NACA64210 was not achievable based on the current parametrization but the RMS difference between the computed airfoil and the true NACA64210 airfoil was within four orders of magnitude. The computed airfoil rather than the true NACA64210 airfoil was taken to be the target in the unsteady flow optimization case since it is guaranteed to lie within the design space. Figure (12) shows the airfoil that was used as the target and a true NACA64210 airfoil.

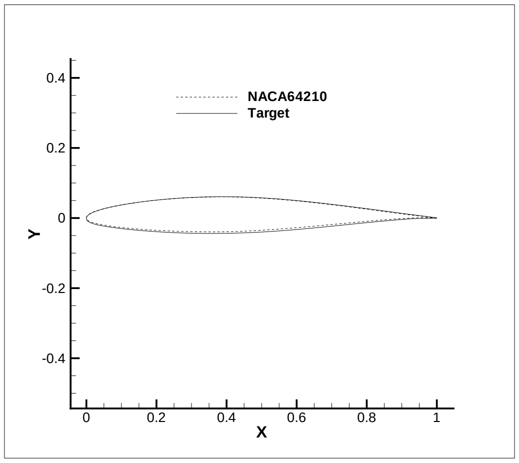
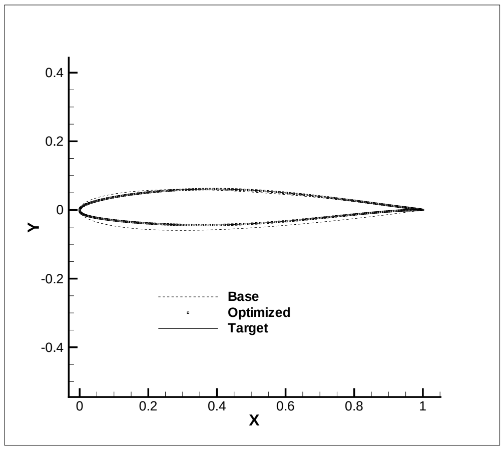
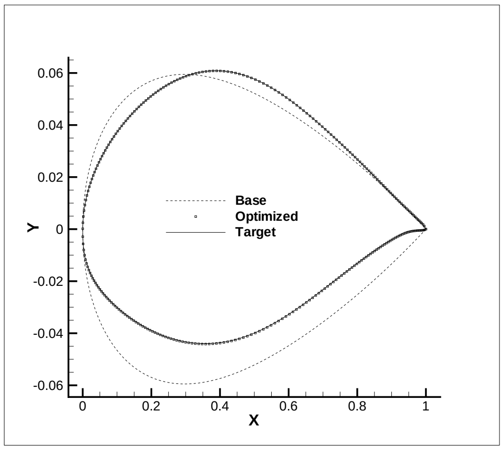


Figure 12. Comparison of target airfoil and true NACA64210 airfoil

The objective function for this example was based on the RMS difference between computed and targeted time-dependent pressure profiles. Since a strong transient shockwave appears on the upper surface of the target airfoil during the time-integration process, the RMS difference between the computed and targeted pressures was weighted more heavily on the lower surface. This is necessary since subtle changes to the shape of the upper surface lead to changes in the chordwise location of the shockwave, and unless the shock locations match exactly between the target and the computed airfoil shapes, the RMS difference between the target and computed pressures will always remain large. Figures (13(a)) and (13(b)) compare the target, baseline and optimized airfoils using a 1:1 scale and an exaggerated scale to show detail. Figure (14) shows the convergence of the global objective function L^g. As in the previous case, the global objective drops by an order of magnitude within the first five design iterations, and from Figure (15) it is clear that the pressures at three different time locations are quite close to the corresponding target values at just the fifth design iteration. By the 36^{th} design iteration the difference between the computed and target pressures is no longer visible to the naked eye. By the 64^{th} design iteration the global objective has reduced by more than two orders of magnitude and the optimization was stopped. The selection of the step-size at each design iteration was somewhat ad-hoc and was based only on achieving the largest reduction in the objective function at that design iteration. Even with the biased weighting of the objective on the lower surface, the upper surface dominated the selection of the step-size. Large step sizes resulted in a growing instability in the vicinity of the shockwave, while small step-sizes although stable resulted in extremely slow convergence rates. This emphasizes the importance for high quality objective formulations and also the necessity for more rigorous methods to determine the step-size at each design iteration. Both of these factors contribute significantly to the reduction of the overall number of design iterations required to reach an optimum solution. The plateau-like regions visible in Figure (14) are locations where small step-sizes were chosen to recover from growing instabilities setup by preceding large step-size design iterations. The existence of this phenomenon is visible from the distributions of the computed sensitivity shown in Figures (16(a)) through (16(f)). The 5^{th}, 36^{th} and 64^{th} design iterations correspond to locations immediately following design iterations with large step-sizes. A spike in the sensitivity near the shockwave (node $ID \approx 275$) is clearly visible at these design iterations while this is absent in the other plots which correspond to regions with a small step-size.



(a) 1:1 scale



(b) Exaggerated scale

Figure 13. Comparison of Target and Computed Airfoil Shapes for Case-2

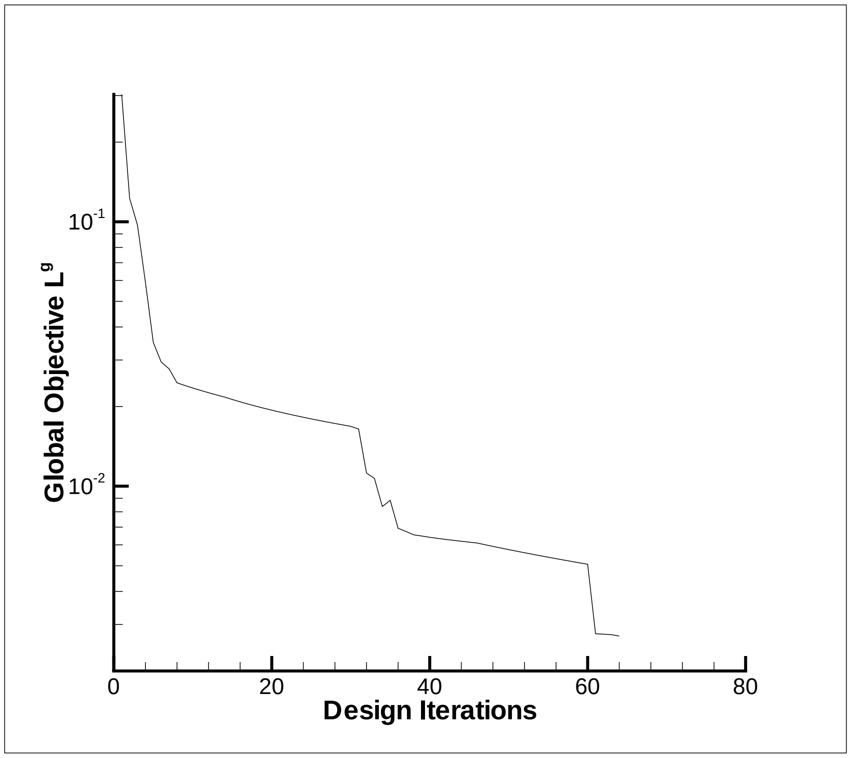
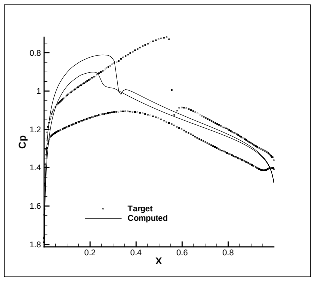
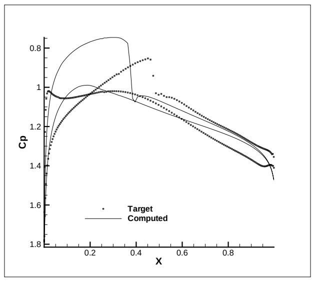


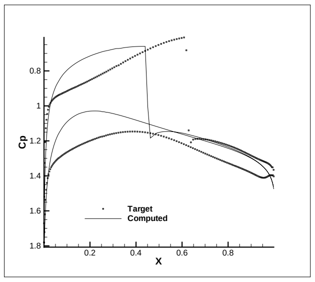
Figure 14. Convergence of global objective function L^g for optimization Case-2



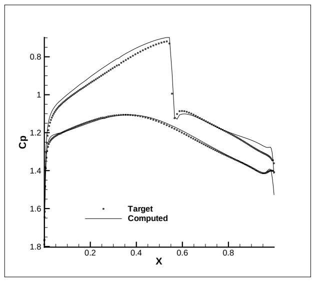
(a) 1st Design Iteration (n=1)



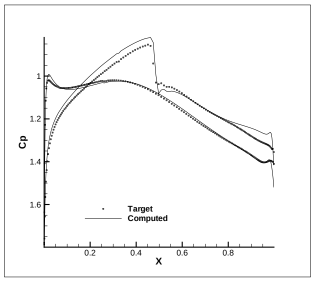
(b) 1st Design Iteration (n=13)



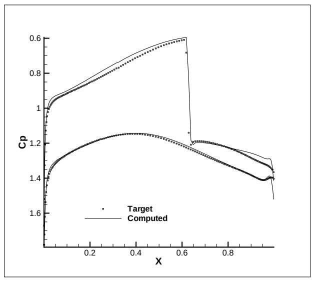
(c) 1st Design Iteration (n=27)



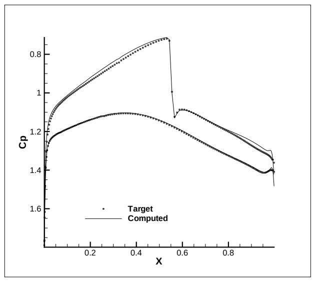
(d) 5th Design Iteration (n=1)



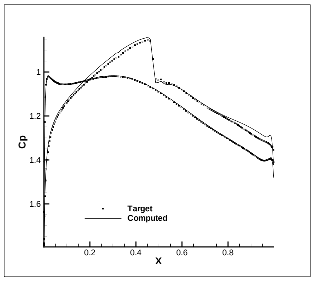
(e) 5th Design Iteration (n=13)



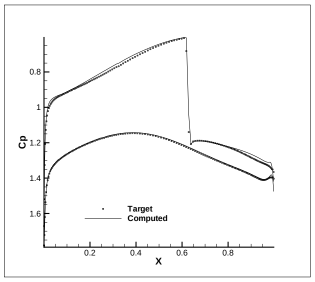
(f) 5th Design Iteration (n=27)



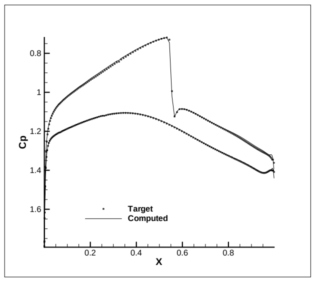
(g) 31st Design Iteration (n=1)



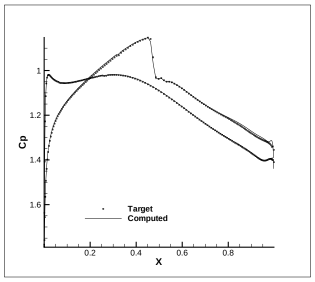
(h) 31st Design Iteration (n=13)



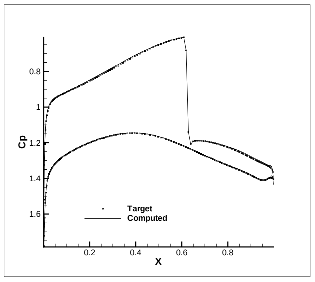
(i) 31st Design Iteration (n=27)



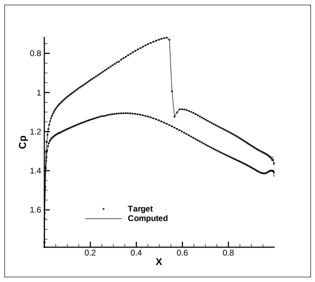
(j) 36th Design Iteration (n=1)



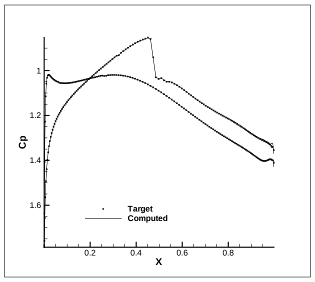
(k) 36th Design Iteration (n=13)



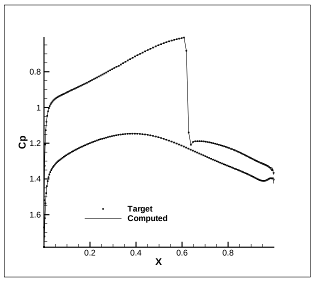
(l) 36th Design Iteration (n=27)



(m) 64th Design Iteration (n=1)

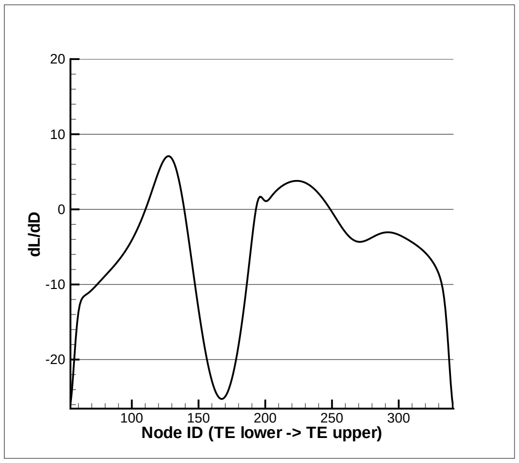


(n) 64th Design Iteration (n=13)

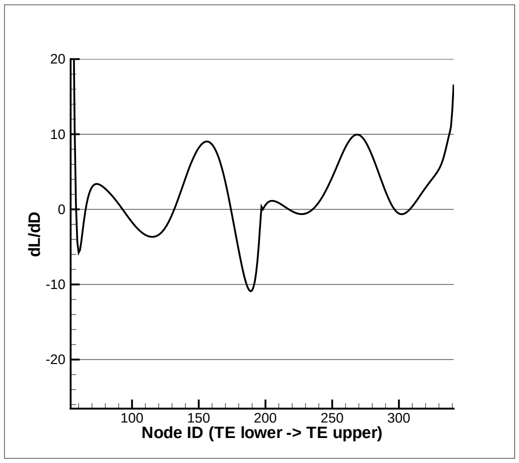


(o) 64th Design Iteration (n=27)

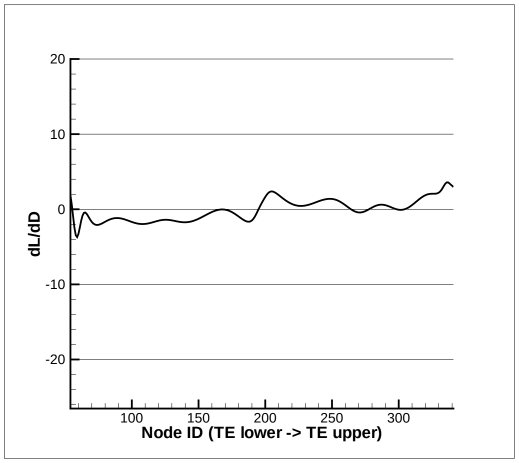
Figure 15. Comparison of computed and target pressure profiles at different design iterations for three time-levels $n = 1$, $n = 13$, and $n = 27$ for optimization Case-2



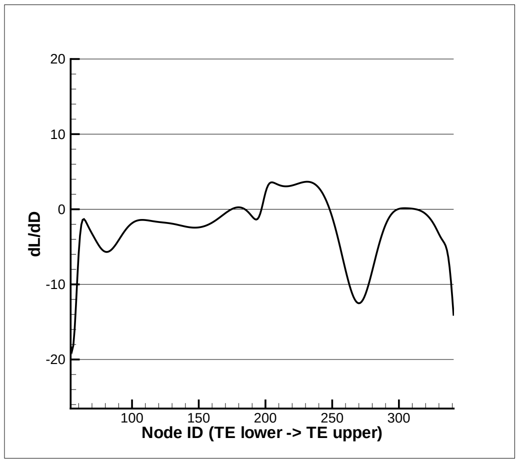
(a) 1st Design Iteration



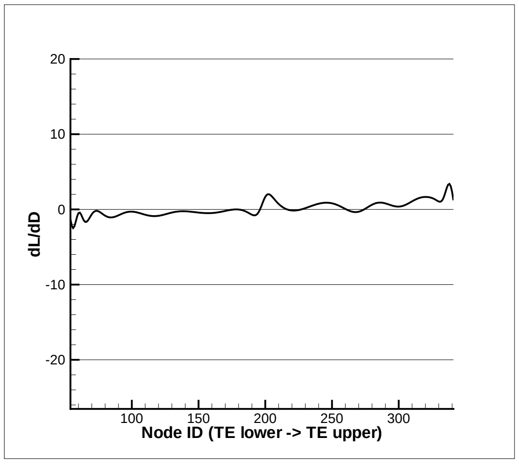
(b) 5th Design Iteration



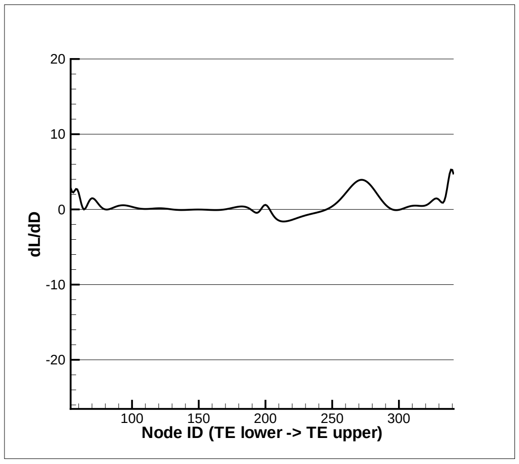
(c) 31st Design Iteration



(d) 36th Design Iteration



(e) 60th Design Iteration



(f) 64th Design Iteration

Figure 16. Sensitivity distribution at different design iterations for optimization Case-2

C. Case-3: Time-Dependent Drag Minimization

The objective formulation for this case was the same as that used for the time-dependent load matching problem. The flow conditions and the baseline configuration for the optimization problem also remained unchanged from the previous two examples. In order to perform a lift constrained drag minimization, the target time-dependent lift profile was taken to be the same as that of the baseline configuration (NACA0012), and the target time-dependent drag profile was set as zero at all locations. The objective function cannot reach zero in this case since the drag acting on a pitching airfoil is always non-zero. Figure (17) shows the optimized airfoil in comparison with the baseline NACA0012 airfoil. Figure (18) shows the convergence of the global objective function and it is clear that the optimization levels out very quickly. Figure (19) shows the convergence of the time-averaged and maximum drag coefficients during the course of the optimization. An approximate reduction of 64 counts of drag in the maximum drag coefficient and 36 counts in the time-averaged drag coefficient is observed. Figure (20) compares the baseline and optimized time-dependent drag profiles.

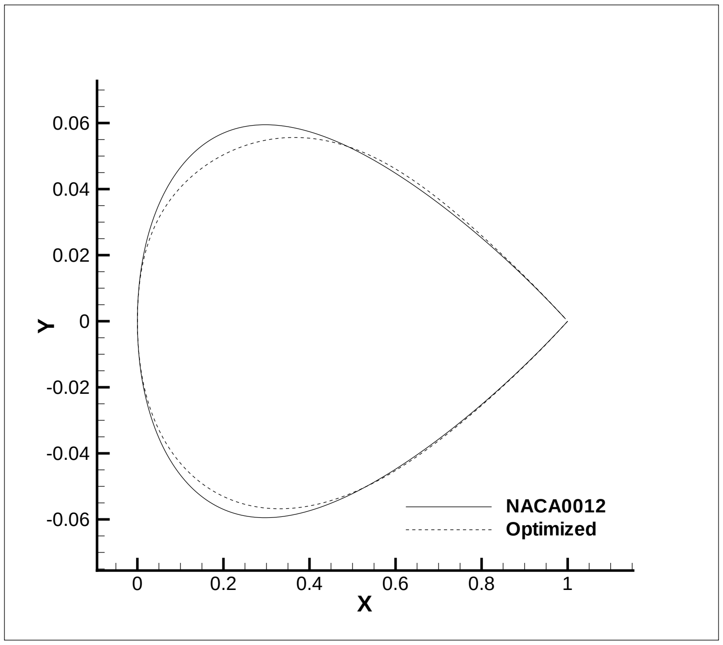


Figure 17. Comparison of baseline NACA0012 and optimized airfoils for optimization Case-3

VIII. Concluding Remarks

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 efficient in the sense that the cost of obtaining the sensitivity vector is less than or equal to the cost of obtaining an unsteady flow solution. The method is feasible for substantially large cases since 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 2,000 time-steps would require roughly 15 gigabytes 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, since 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 testing the algorithm in three-dimensional problems.

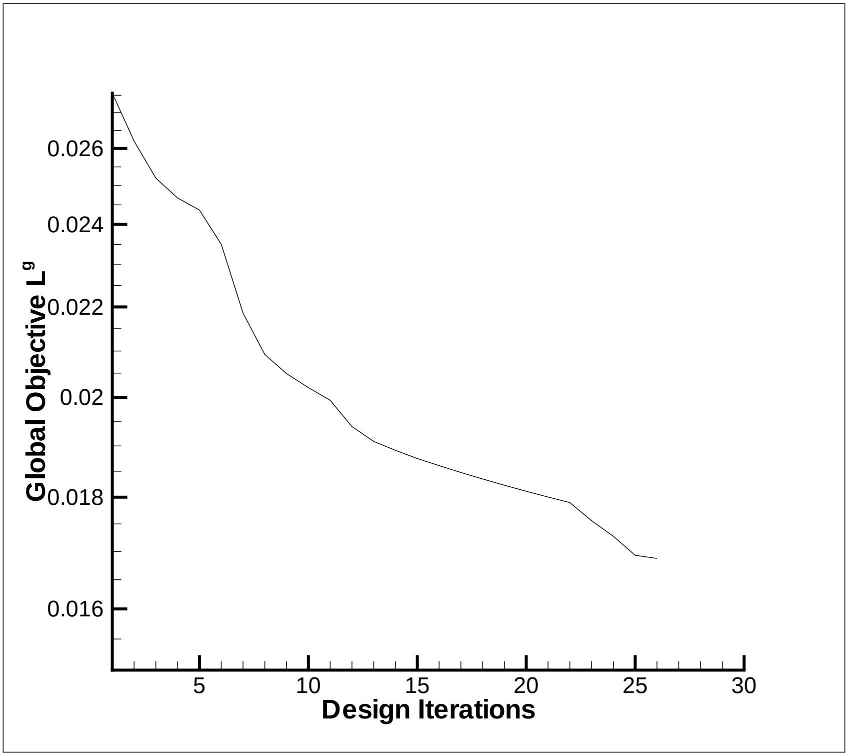


Figure 18. Convergence of global objective function L^g for optimization Case-3

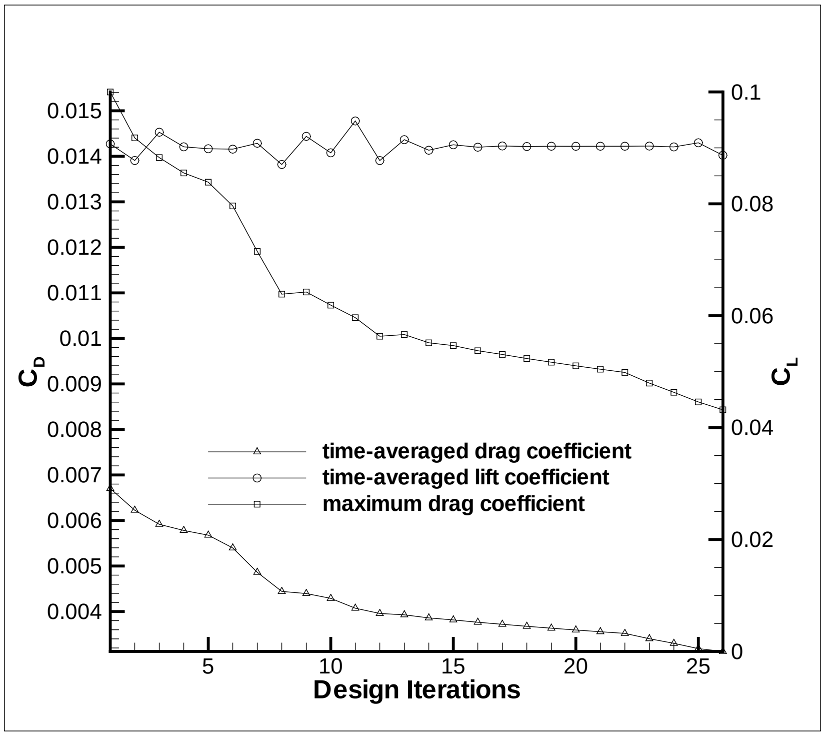


Figure 19. Convergence of time-averaged and maximum drag coefficients for optimization Case-3

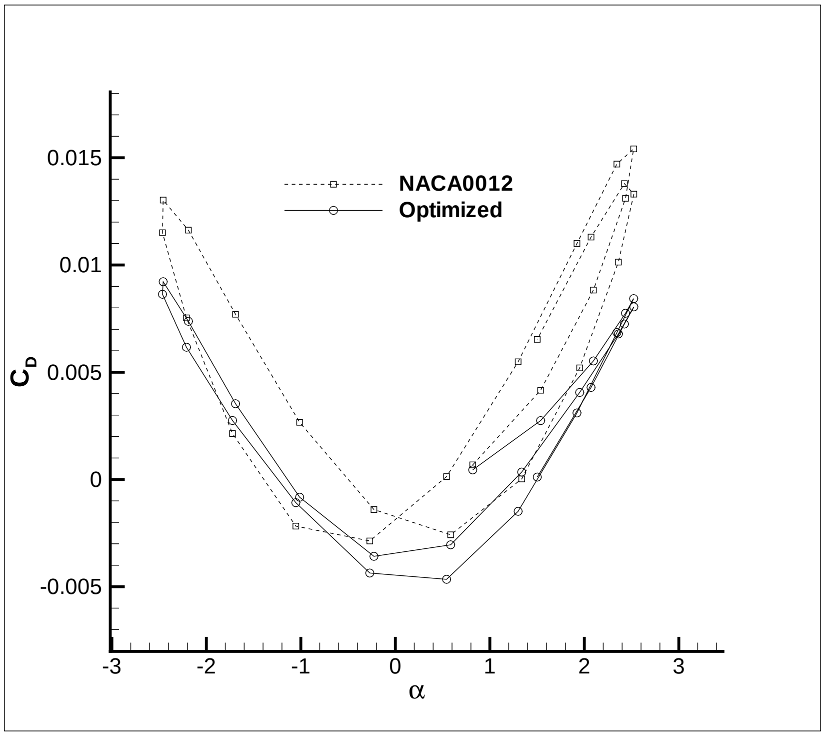


Figure 20. Comparison of baseline and optimized time-dependent drag profiles for optimization Case-3

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.*

[2]Jameson, A. and Vassberg, J., "Computational Fluid Dynamics for Aerodynamic Design: Its Current and Future Impact," *Proceedings of the 39th Aerospace Sciences Meeting and Exhibit, Reno NV*, 2001, AIAA Paper 2001–0538.

[3]Nadarajah, S. and Jameson, A., "A Comparison of the Continuous and Discrete Adjoint Approach to Automatic Aerodynamic Optimization," *Proceedings of the 38th Aerospace Sciences Meeting and Exhibit, Reno NV*, 2000, AIAA Paper 2000–0667.

[4]Nielsen, E. and Anderson, W., "Recent Improvements in Aerodynamic Optimization of Unstructured Meshes," *AIAA Journal*, Vol. 40-6, 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," 1998, AIAA Paper 98–2538.

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

[7]Mavriplis, D., "Multigrid Solution of the Discrete Adjoint for Optimization Problems on Unstructured Meshes," Accepted for Publication in AIAA Journal, 2005.

[8]Mavriplis, D., "A Discrete Adjoint-Based Approach for Optimization Problems on Three-Dimensional Unstructured Meshes," *Proceedings of the 44th Aerospace Sciences Meeting and Exhibit, Reno NV*, 2006, AIAA Paper 2005–0050.

[9]Elliot, J. and Peraire, J., "Practical three-dimensional aerodynamic design by optimization," *AIAA Journal*, Vol. 35-9, 1997, pp. 1479–1485.

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

[11]Ghayour, K. and Baysal, O., "Limit-Cycle Shape Optimization using Time-Dependent Transonic Equation," *Proceedings of the 14th Computational Fluid Dynamics Conference, Norfolk VA*, 1999, AIAA Paper 99–3375.

[12]Nadarajah, S. and Jameson, A., "Optimal Control of Unsteady Flows using A Time Accurate Method," *Proceedings of the 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization Conference, Atlanta GA*, 2002, AIAA Paper 2002–5436.

[13]Nadarajah, S., McMullen, M., and Jameson, A., "Non-Linear Frequency Domain Based Optimum Shape Design for Unsteady Three-Dimensional Flow," *Proceedings of the 44th Aerospace Sciences Meeting and Exhibit, Reno NV*, 2006, AIAA Paper 2006–387.

[14]Nadarajah, S., McMullen, M., and Jameson, A., "Optimal Control of Unsteady Flows Using Time Accuraate and Non-Linear Frequency Domain Methods," *33rd AIAA Fluid Dynamics Conference and Exhibit, Orlando, FL, June 23-26,*, 2003, AIAA Paper 2003–3875.

[15]Vendetti, D. and Darmofal, D., "Grid Adaptation for Functional Outputs: Application to Two-Dimensional Inviscid Flows," *Journal of Computational Physics*, Vol. 176, 2002, pp. 40–69.

[16]Vendetti, D. and Darmofal, D., "Anisotropic grid adaptation for functional outputs: application to two-dimensional viscous flows," *Journal of Computational Physics*, Vol. 187, 2003, pp. 22–46.

[17]Roe, P., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," *Journal of Computational Physics*, Vol. 43-2, 1981, pp. 357–372.

[18]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, 2003, pp. 206–227.

[19]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, 2006, pp. 557–573.

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

[21]Hicks, R. and Henne, P., "Wing Design by Numerical Optimization," *Journal of Aircraft*, Vol. 15-7, 1978, pp. 407–413.

[22]"iSight," Process Integration and Design Optimization Software, Engineous Software Inc.

[23]"ModelCenter," Process Integration and Design Optimization Software, Phoenix Integration Inc.

[24]Errico, R. M., "What is an Adjoint Model ?" *Bulletin of the American Meteorological Society*, Vol. 8, No. 11, 1997, pp. 2577–2591.

[25]Dwight, R. and Brezillon, J., "Effect of Various Approximations of the Discrete Adjoint on Gradient-Based Optimization," *Proceedings of the 44th Aerospace Sciences Meeting and Exhibit, Reno NV*, 2006, AIAA Paper 2006–0690.

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

[27]Nielsen, E. J. and Kleb, W. L., "Efficient Construction of Discrete Adjoint Operators on Unstructured Grids by Using Complex Variables," *AIAA Journal*, Vol. 44-4, January 2005, pp. 827–836.

[28]Martins, J. R. R. A., Kroo, I. M., and Alonso, J. J., "A Method for Sensitivity Analysis using Complex Variables," *38th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, January*, 2000, AIAA Paper 2000–0689.

[29]Yang, Z. and Mavriplis, D., "High-Order Time Integration Schemes for Aeroelastic Applications on Unstructured Meshes," *44th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, January*, 2006, AIAA Paper 2006–0441.