Initial Implementation of an Adjoint CFD Code for Aeroshell Shape Optimization

AE8900 MS Special Problems Report
Space Systems Design Lab (SSDL)
Guggenheim School of Aerospace Engineering
Georgia Institute of Technology
Atlanta, GA

Author:
Kevin W. Flaherty

Advisor:
Dr. Robert D. Braun

May 3, 2008

Initial Implementation of an Adjoint CFD Code for Aeroshell Shape Optimization

Kevin W. Flaherty*
Georgia Institute of Technology, Atlanta, GA, 30332-0150

Application of computational fluid dynamics to the optimization of aeroshell shapes usually entails high computational cost. Many converged solutions are required to generate gradients and optimize a shape with respect to very few design variables. The benefits of high-fidelity aerodynamic analysis can be reaped early in the design cycle with less computational cost if the traditional direct optimization problem is transformed to an indirect optimization, using optimal control theory. The indirect gradient formulation decouples the effects on the objective function of the design variables and the flow solution. Meaning, all derivatives used to compute the gradient can be generated from a single converged flow solution. Involved in the computation of the gradient is the solution of an adjoint system of PDEs. An incremental approach is developed for the implementation of an adjoint equation solver. The phased approach begins using inexact and computationally costly finite difference derivative calculations. Results are presented for a transonic airfoil and a supersonic wedge to demonstrate that the finite difference gradient is reasonably accurate, providing a meaningful validation as exact numerical derivatives are substituted later in the development cycle. Finally, a roadmap is presented for future implementation of indirect optimization capability for the Euler/Navier-Stokes CFD code, NASCART-GT.

Nomenclature

AD	= automatic differentiation	n	= number of output variables
α	= vector of design variables	υ	= vector of adjoint variables
C_d	= drag coefficient	NASCART-GT	= Numerical Aerodynamic Simulation via Cartesian Grid Techniques
CFD	= computational fluid dynamics		
C_l	= lift coefficient	p	= pressure
E	= energy state	PDEs	= partial differential equations
γ	= ratio of specific heats	q	= vector of flow state variables
GMRES	= generalized minimal residual	R	= vector of discretized residuals
J	= cost function	ρ	= density state
M	= computational mesh vector	T	= wetted surface triangulation vector
m	= number of input variables	u_x, u_y	= Cartesian velocity components
MDO	= multidisciplinary optimization	x, y	= Cartesian coordinates
N	= number of design variables		

I. Introduction

THE goal of this study is to establish a foundation for modifying the high-fidelity flow analysis tool NASCART-GT for use in an aeroshell shape optimization context. This paper briefly discusses the need for computational fluid dynamics (CFD) tools in the design and optimization of entry aeroshell shapes. Challenges to using CFD as a shape optimization tool are presented. Then, historical work on potential solutions is reviewed. In the Methodology section optimal control theory is used to develop an indirect shape optimization method and implementation issues are discussed for the modification of an existing CFD code. An application to a simple CFD code is presented as a proof-of-concept and results are shown for several cases. Conclusions are drawn regarding the viability of the method developed, and considerations are discussed for application to NASCART-GT, a Cartesian Euler and Navier-Stokes CFD code in development at Georgia Tech.

* Graduate Research Assistant, Daniel Guggenheim School of Aerospace Engineering, Georgia Institute of Technology, 270 Ferst Drive, Atlanta, GA 30332-0150.

A. Motivation

Aeroshells are structures designed to encapsulate payloads and safely deliver them to the lower atmosphere of a planetary body. An aeroshell must withstand the heating associated with the entry environment while providing aerodynamic and stability characteristics necessary to prepare for descent and landing events. The design of aeroshells must include analysis of these factors while considering system constraints such as size and shape of payload or diameter of launch vehicle. Using engineering approximations and low-fidelity tools these disciplines can be considered in a multidisciplinary optimization (MDO) environment, with aeroshell shape parameters being the design variables to optimize.[6] Realistic hypersonic entry regimes present complicated flow properties stemming from shock interactions, turbulence, and boundary layers that cannot be captured by low-fidelity methods, such as modified Newtonian theory. Thus, high-fidelity CFD analysis is now a mandatory step in the aeroshell design process.

MDO requires analysis tools that are automated and can execute rapidly. While development of CFD models can be a time-consuming and human-intensive process it is possible to reliably automate grid generation and perform repeated analyses with small perturbations to geometries or flow properties, thus satisfying MDO's requirement for automation. However, the greatest obstacle to using CFD for optimization is the computation time, which can range from minutes to days. In general, optimizers require many function calls to arrive at a solution. Considering a function call to be constituted by a converged CFD solution, having even a few variables creates an unmanageable optimization, even if the initial solution is close to optimal. Zero-order and second-order optimizers face similar problems.

The difficulty of using CFD for shape optimization is alleviated by using an indirect optimization formulation with adjoint equations that permit gradient calculation using portions of the existing CFD code. As a result, a gradient can be generated using numerical derivatives of a single converged flow solution. Additionally, an indirect optimization delivers more accurate gradient information, improving convergence time and producing a truly optimal solution. This study outlines an incremental approach to a fully adjoint code and provides a proof-of-concept for the first phase.

B. Background

Computer technology revolutionized aerodynamic design in the 1970's by enabling the creation of CFD programs to analyze complex configurations. While CFD rapidly became an important part of the design process, the computational effort it demanded did not easily lend itself to iterative shape optimizations. Early optimization strategies were pioneered by Hicks, Murman, and Vanderplaats for 2-D shapes using the potential flow equations, Hicks and Henne for full wing design, and Reuther et al. for supersonic wing-body configurations.[4] All of these researchers utilized finite difference methods to generate gradients for first-order numerical optimizers. Finite difference was easy to implement but it produced gradient inaccuracy and, more importantly, it required an additional flow solution for each design variable when applied to a direct optimization. This prohibitive computational effort led to use of shape parameters that could define geometries using a small number of design variables (generally less than ten). Finer control over the geometric shape required too much computational effort to be viable in an optimization framework, a problem exacerbated by increasing complexity of flow solvers.[9]

In 1988 Jameson used optimal control theory to lay the foundation for an indirect optimization methodology that would mitigate the computational costs associated with CFD optimization.[1] His methodology drew from Lions (1971) who outlined a theory for optimal control of systems of partial differential equations (PDEs) and Pironneau (1973) who first applied optimal control theory to fluid mechanics to model the drag imparted by a small bump on an otherwise smooth body surface in a viscous flow.[3] This application used adjoint, or "co-state", partial differential equations (PDEs) to infer aerodynamic properties as numerical schemes and computing power had not yet developed sufficiently to solve the Navier-Stokes equations that his formulation used.

Jameson developed an optimal control formulation that enabled generation of a gradient with roughly the same computational effort as a single flow solution, regardless of the number of design variables. This was done by decoupling flow parameters from design parameters and will be further discussed under Methodology. Jameson's first papers on optimal control for aerodynamic design outlined adjoint formulations for potential flow, Euler equations, and Navier-Stokes equations.[1,38]

While Jameson's work was immediately applicable toward higher-fidelity design tools for the transonic regime, it was not until 1994 that Jameson and Reuther successfully implemented an indirect optimization and applied it to transonic wing design and optimizations of full aircraft configurations using wing design variables and simple parameters for wing and engine placement. Their initial implementation automatically optimized 2-D shapes and showed seven-fold improvement in computation time when compared to a finite difference method.[4]

A debate that has only recently begun to subside is the use of continuous versus discrete adjoint formulations. Jameson and Reuther's original implementation was discrete but they acknowledged that the continuous form had merits. With either formulation the resulting gradients will be very similar. The benefits of a continuous formulation include that the adjoint mathematical construct has a more apparent physical meaning (the adjoint equations' behavior at stagnation points and across shocks can be more easily understood) and the resulting adjoint code will require fewer lines and less memory to execute (because discrete adjoint codes make use of many intermediate processes).[2]

The continuous adjoint produces an exact gradient for the governing flow equations, but CFD codes are discretized so the adjoint also must be discretized for computation of the gradient. This "late" discretization will cause the continuous gradient to differ slightly from the fully discrete gradient, which is as accurate as the discretized flow solution. Generally, both gradients will be more accurate than finite differences; the benefits of this additional accuracy are demonstrated by Nadarajah and Jameson (2000) and Martins et al. and Courty et al.[25,31,33] More significantly, the discrete approach allows the manipulation of existing CFD code to generate derivatives, a benefit that can be exploited by several techniques outlined in the following sections to aid greatly in code development.

Another debate regards the use of complex variable differentiation or automatic differentiation to convert existing CFD code. Both methods produce exact gradients, unlike finite difference, but their usage is application-specific and will be discussed in the Methodology section. Two NASA centers have CFD codes under continuous development that have attempted to use these methods. Nielsen et al. (2005) have developed Fun3D at NASA LaRC and their code is capable of both forward mode differentiation using the complex variable method and reverse-mode automatic differentiation.[20] Aftosmis et al. (2005) are developing Cart3D at NASA Ames; their choice between complex-variable method and automatic differentiation is ambiguous. Cart3D emphasizes using adjoints for design variable manipulation and deformation of the computational mesh, saving grid generation time. Cart3D is also designed to optimize with a CAD program in the automated loop.[17,18] Both groups have published examples of successful optimizations including an optimized entry body shape from the NASA Ames group.[11,20,21,32]

Giles has contributed substantially to adjoint-based design through teaching tools. Some of his contributions include developing closed-form adjoint equations for simple problems, such as 1-D flow through a nozzle, and exploring the properties of the adjoint equations, such as continuity through shocks. In more recent years, Giles has presented solutions to common problems involved with application of adjoint equations for existing codes. His low-cost implementation methods for existing CFD codes make use of automatic differentiation tools.[2,23,27,29]

Praveen (2006) draws heavily from Giles' methods to develop an open-source Euler flow solver, Euler2D, with adjoint solving that can be applied to simple 2-D problems.[28] Both Giles and Praveen utilize the iterative scheme used by the flow solver because it is "highly optimized" for the adjoint equations as well as the governing equations. Praveen's code provided an excellent teaching tool that guided the initial investigations for modifying a more complex CFD code.

To demonstrate the capability of the automatic differentiation (AD) tool, TAPENADE, Martinelli (2007) worked with the tool's developers to calculate gradients and Hessians and he applied them to a quasi-Newton optimizer. As an alternative to modifying the flow solving scheme for adjoint equations, Courty et al. (2003) and Martinelli advocate using a hand-written adjoint solver for the arrays of derivatives generated using TAPENADE.[31,35]

Difficulties arose while attempting to apply Giles and Praveen's adjoint solution method; however, Courty and Martinelli's hand-written solver was applied successfully. They use a GMRES matrix-free algorithm for its ease of implementation, when compared to the iterative algorithm of Giles and Praveen. Mohammadi (1997) also addresses some practical concerns when optimizing many individual nodes that constitute a geometry instead of a small number of parameters and his methods are applied for this study. These methods include smoothing inconsistent derivatives and boundary treatments for 2-D optimizations.[8,41]

II. Methodology

An optimization framework is developed with the concurrent goals of rapid execution, validation for increasing levels of gradient fidelity, and ease of implementation. To achieve these goals, the traditional direct optimization is transformed into an indirect optimization. The following section describes the overall problem statement, the mathematical formulation for calculating gradients using an adjoint variable, and factors to consider prior to implementing the adjoint formulation.

A. Euler Equations

While adjoint formulations have been created for potential flow and Navier-Stokes flow, the Euler equations of gas dynamics are selected as the governing equations to yield higher-fidelity flow solutions and less complicated implementation than viscous alternatives. Despite the importance of flow chemistry, viscosity, and boundary layers in high-speed regimes, this study will consider the flow to be non-reacting and inviscid for ease of code development. The two-dimensional Euler equations in conservative form are as follows:

$$\frac{\partial U}{\partial t}+\frac{\partial f}{\partial x}+\frac{\partial g}{\partial y}=0 \tag{1}$$

$$U=\begin{bmatrix}\rho\\ \rho u_x\\ \rho u_y\\ E\end{bmatrix},\quad f=\begin{bmatrix}\rho u_x\\ p+\rho u_x^2\\ \rho u_x\rho u_y\\ (E+p)u_x\end{bmatrix},\quad g=\begin{bmatrix}\rho u_y\\ \rho u_x\rho u_y\\ p+\rho u_y^2\\ (E+p)u_y\end{bmatrix} \tag{2}$$

The system is closed with the following state equation:

$$E=\frac{p}{\gamma-1}+\frac{1}{2}\rho(u_x^2+u_y^2) \tag{3}$$

ρ is the fluid density, $u_{x \text{ and}}\, u_y$ are the x and y velocity components, p is the static pressure, and E is the total energy per unit volume.

B. Optimization problem statement

In balancing rapid execution and ease of implementation, a first-order optimizer will be used for this problem. Zeroth-order optimizers require too many function calls to be viable with CFD, and second-order optimizers require a more difficult implementation to produce Hessians from a CFD program.[40] Thus, the goal is to generate a gradient from a CFD code and use it in an optimization statement.

The direct optimization problem is defined as the minimization of a cost function based on aerodynamic forces for a converged flow solution:

$$\min_{\alpha} J(\alpha,q) \text{ such that } R(\alpha,q)=0 \tag{4}$$

For the supersonic example in the Results section, the cost function, J, is the drag coefficient and it is minimized for a blunt body. However, J can be any scalar output that is a function of shape parameters, α, and flow properties, q. The residual, R, is defined as the increment of flux between cells before an update scheme is applied. The subroutine to calculate the residual includes the flux routines and the boundary treatment. At a converged solution, the residual is essentially equal to zero.

The disadvantage of the direct optimization problem is that the gradient, $dJ/d\alpha$, couples flow parameters with design parameters. This coupling requires a new flow solution for every change in a design parameter, even if it is only a small finite difference perturbation. The adjoint formulation that is developed decouples these computations so that partial derivatives can be generated without new flow solutions.[1]

Implementing the adjoint formulation into an existing flow solver produces a converged flow solution and an exact gradient of the sensitivity of the cost function to design parameters. For this implementation, all x, y coordinates describing the exterior boundary of the aeroshell shape are considered design parameters.

This optimization problem is subject to the same limitations of any gradient-based optimization. All design variables must be continuous, and multiple maxima or discontinuities in J can prevent convergence on a global maximum. To mitigate these limitations, it is necessary to use an appropriate initial shape, especially for an indirect formulation in which to converging design space tends to narrow.[36] Additionally, enforcing geometric constraints is a difficult prospect with adjoint equations; it is better implemented through a penalty function added to J. For the sake of simplicity, this study considers only unconstrained optimization.

C. Adjoint Formulation

1. Continuous/Discrete

As described in the background research, most development efforts of an adjoint solver for indirect optimization would benefit from selection of the discrete adjoint method. Many tools and methods exist to aid in the development of a discrete adjoint code, and the phased implementation is possible only for a discrete method.

2. Gradient derivation

To derive the adjoint formulation for calculating gradients, the cost function is added to the flow residual (which is zero for a converged solution) multiplied by a Lagrange Multiplier (called an "adjoint variable" in this context):

$$J(\alpha, q) = J + \upsilon^T R \tag{5}$$

Then, the equation is differentiated:

$$dJ = \frac{\partial J}{\partial \alpha} d\alpha + \frac{\partial J}{\partial q} dq + \upsilon^{\mathrm{T}} \underbrace{\left(\frac{\partial R}{\partial \alpha} d\alpha + \frac{\partial R}{\partial q} dq \right)}_{= 0 \text{ when converged}} \tag{6}$$

Rearrange the sensitivity equation to separate the design variable terms and flow terms:

$$dJ = \left(\frac{\partial J}{\partial \alpha} + \upsilon^T \frac{\partial R}{\partial \alpha} \right) d\alpha + \left(\frac{\partial J}{\partial q} + \upsilon^T \frac{\partial R}{\partial q} \right) dq \tag{7}$$

Then, the sensitivity to the design variables is isolated by solving for an adjoint variable that drives the *dq* term to zero:

$$\frac{\partial J}{\partial q} + \upsilon^T \frac{\partial R}{\partial q} = 0 \quad \Rightarrow \quad \left(\frac{\partial R}{\partial q} \right)^T \upsilon = -\left(\frac{\partial J}{\partial q} \right)^T \tag{8}$$

This is a system of adjoint equations that must be satisfied, among other conditions, to achieve an optimum solution. Note that the resulting adjoint equation can be solved iteratively and is independent of the number of design variables. With the *dq* term zeroed, the remaining sensitivity forms the gradient and the complete indirect optimization statement:

$$\underset{\alpha}{\nabla} J = G = \frac{\partial J}{\partial \alpha} + \upsilon^T \frac{\partial R}{\partial \alpha} \quad \text{s.t.} \quad \left(\frac{\partial R}{\partial q} \right)^T \upsilon = -\left(\frac{\partial J}{\partial q} \right)^T \tag{9}$$

When the gradient is minimized to 0, and the adjoint equations are satisfied, the design variables are optimal. Note that the design variable derivatives are evaluated separately from the flow derivatives, indicating that portions of the code can be used for derivative generation, thus avoiding the effort associated with differentiation of the entire CFD code.

D. Adjoint Solving

1. Gradient Components

As was shown in the gradient derivation, there are four derivatives required to solve for the adjoint variable and compute the gradient:

$$\frac{\partial J}{\partial \alpha}, \quad \frac{\partial J}{\partial q}, \quad \frac{\partial R}{\partial \alpha}, \quad \frac{\partial R}{\partial q}$$

R: residual, J: cost function, α: geometric design variables, q: flow solution

If N is the number of grid points on the surface, the derivative arrays will have the following dimensions: (Nx2), (Nx4), (2xNx4), (4xNx4), respectively. Note that 2-D geometry with four state variables is assumed. In order to extract the derivatives it is best if the CFD code flows as follows:

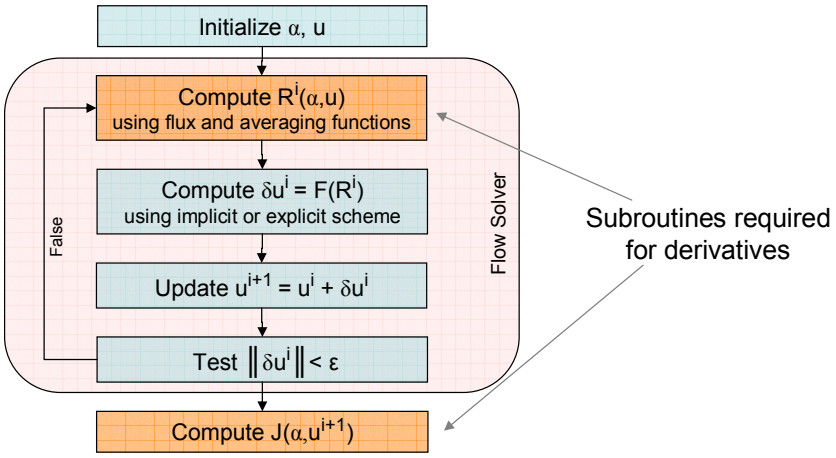


Figure 1 Preferred flow of a CFD code to be differentiated.

Four methods exist for the computation of these derivatives in the context of a CFD code:

1. Manual differentiation – Mathematical derivatives are generated by hand offline and then coded into the CFD program. This method can be exact and coded efficiently for simpler computations. However, it can be time-consuming, error-prone, and inflexible to changes in the original code, especially for complicated CFD codes.

2. Finite Difference (FD) – Calculates a derivative by perturbing a design variable by a small value and measuring the change in the output. FD is very simple to implement and does not require access to source code. However, it is inexact due to the order assumption (linear, parabolic, etc.) and cancellation error. Additionally, FD for complicated functions can yield unstable results based on selected step-size.

$$\frac{\partial R}{\partial \alpha} = \lim_{h \to 0} \frac{R(\alpha + h, q) - R(\alpha, q)}{h} \tag{10}$$

3. Complex Difference – Calculates a derivative using a form similar to FD but with a complex step:

$$\frac{\partial R}{\partial \alpha} = \lim_{\varepsilon \to 0} \frac{imag\{R(\alpha + i\varepsilon, q)\}}{\varepsilon} \tag{11}$$

The result is a true second-order derivative with no cancellation error, significantly improving upon FD. Additionally, the sensitivity of the derivative to step size is far more robust.[25] This method is generally easy to implement by declaring all variables in a code to be complex and ensuring that the mathematical and logical functions can operate on complex numbers. Depending on the application, execution can be slow because complex operations generally require more computational effort than real operations.

4. Automatic Differentiation (AD) – AD tools operate on source code to create derivative versions of the code. This can be done by parsing a code into simple mathematical functions, differentiating them, and summing a derivative's components using the chain rule. "Operator overloading" is one method that involves converting each real variable into a tandem real variable, including the original variable and the derivative of the original. Then, a modified library of mathematical functions simultaneously operates on the original value and the derivative. The other method is "source transformation," which creates new code to compute derivative variables based on the original code.

2. Forward/Reverse Mode

Most numerical differentiations are considered "forward mode," meaning that a single input variable's effect is propagated to generate derivatives of *n* output variables. "Reverse mode" means that effects are propagated backward from a single output variable to determine the derivatives from *m* input variables. Thus, the reverse mode is most appropriate if $m \gg n$, as is the case for our many design points forming a geometry that is evaluated with a single metric J.

To further illustrate the rationale for using reverse mode, generating a single gradient for *m* input variables would require *m*+1 function evaluations using any forward mode differentiation. The same gradient requires only 1 function evaluation if a reverse mode differentiation is used. Finite difference, complex difference, and operator overloading AD are all forward-mode differentiations. The only options for reverse mode are manual differentiation and source transformation AD. Manual differentiation has essentially been ruled out due to its inflexibility and complicated nature. On the other hand, source transformation AD is a technique that has been gaining popularity for this purpose, and several tools are available to assist in modifying codes in various languages.

The software tool recommended for this application is TAPENADE 2.2.3, developed by Project TROPICS at INRIA - Sophia Antipolis, France. TAPENADE is recommended because it operates on FORTRAN code, the language used most for CFD codes, including NASCART-GT. Additionally, TAPENADE is the tool that has appeared most frequently in recent literature for other adjoint applications. TAPENADE utilizes source transformation and it can produce forward-mode and reverse-mode differentiated codes.

3. Adjoint Solving and Program Flow

Two methods are generally applied toward solving adjoint PDEs for indirect shape optimization. Method I utilizes the same iterative numerical scheme as the flow solver. This is possible because the adjoint variables (one for each Euler state) are discretized over the same grid as the flow solution. Method I is beneficial because the scheme is highly optimized for accelerating convergence of large systems of PDEs. This method can also be formulated as "duality preserving" meaning that an adjoint solution has a convergence rate guaranteed to be asymptotically equivalent to that of the flow solution.[22] The downside is that AD is essentially required for this form, and the user must have extensive knowledge of the flux computations (especially for viscous flux) and the numerical update scheme.

When Praveen applies method I, AD is applied to the aerodynamic force and residual functions. The residual function is comprised of five flux and averaging subroutines that are reverse-mode differentiated. They are rearranged in reverse order within an iterative loop to maintain reverse derivative computation, as shown in Figure 3. It is especially difficult to validate this method while in development because each differentiated phase of the program contributes to the gradient. In other words, the derivatives produced by the residual subroutines are confounded with the design variable derivatives.

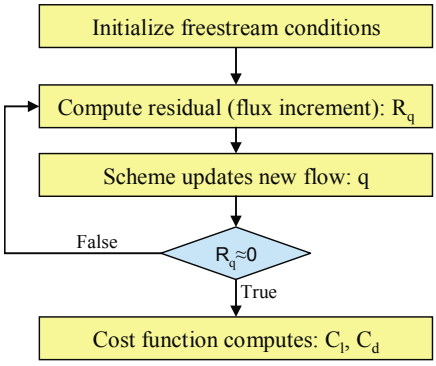
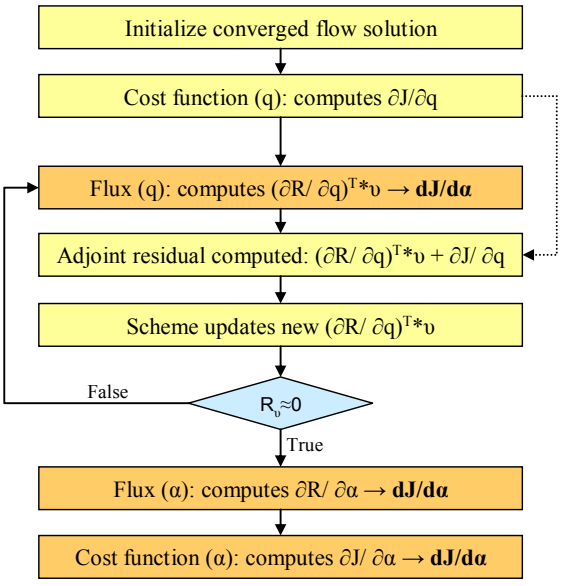


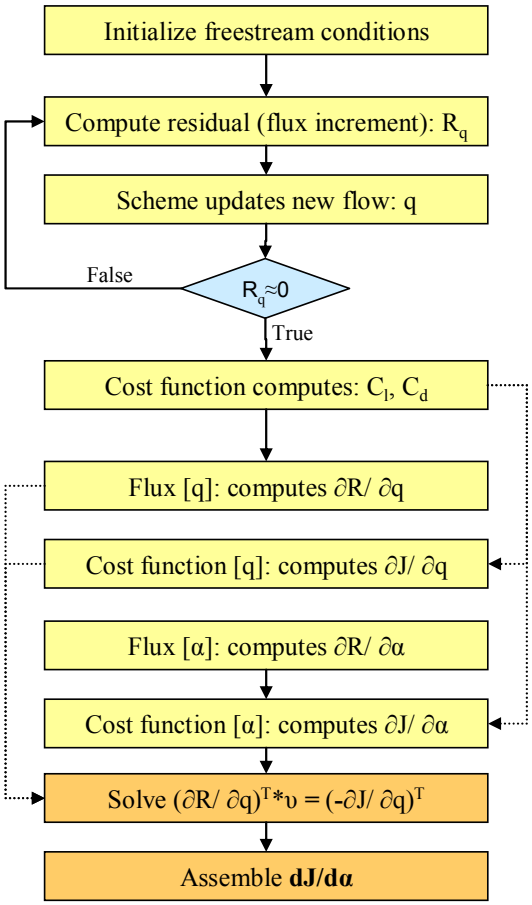
Figure 2 Standard CFD program flow.



(x) denotes that AD applied to a subroutine with respect to x

Figure 3 Adjoint code with iterative adjoint solver.

To mitigate derivative confounding, and enable piecewise validation of the gradient components, method II has been developed to use an alternative program flow as shown in Figure 4. After a flow solution has converged, derivative components are extracted individually using any of the aforementioned numerical differentiation methods. Note that (8) is in the form $Ax=b$. As a result, a linear algebra solver calculates the adjoint variables with respect to the flow derivatives. Method II leads to costly adjoint solutions if a linear algebra solver is applied to millions of cells. To reduce storage requirements, matrix-free iterative solvers, such as GMRES, are recommended to reduce the number of intermediate arrays stored while obtaining a solution. [35]



[x] denotes any differentiation method with respect to x

Figure 4 Program flow of adjoint code using linear algebra adjoint solver.

III. Implementation

NASCART-GT is a complex CFD code that can perform aerodynamic analyses using Euler equations with an integral boundary layer calculation and Navier-Stokes equations. As a result, transforming the code using AD, or even the more straightforward complex variable method, presents significant challenges. The goal of this section is to outline the first step of a phased approach to implementing a fully adjoint code in a fashion that provides a validation framework at each increasing level of complexity. The first phase framework is validated using the Euler2D flow and adjoint solver and is demonstrated for a transonic airfoil and a supersonic blunt body.

A. Adjoint Solving Method

NASCART-GT was initially examined with the intention of implementing adjoint solving method I. It was quickly discovered that NASCART's program flow did not easily lend itself to applying AD and rearrangement of flux calculations to create a reverse iterative solver. As a result, method II is developed for the gradient computation. Method II enables the gradient computation to be built around components of the flow solver, without having to manipulate the lowest levels of computational code. Furthermore, FD is selected for generating derivatives in the

preliminary phase. Application of FD is well-understood for complicated functions, and can provide sufficient numerical accuracy to validate the resulting gradients. Additionally, application of FD allows us to forgo manipulation of the residual functions, as would be required for AD and complex difference methods.

B. CFD Modification

Euler2D's flow solver is used as the testbed for the FD implementation of an adjoint solver. Euler2D's adjoint solver provides validation for the final gradient produced by the FD code. Additionally, the simple structure of Euler2D permitted creation of a complex difference version that was used to validate individual FD step sizes.

The first step toward the FD implementation is to isolate the aerodynamic force subroutine and the residual update subroutine. After a flow solution converges, forces and residuals are calculated separately for two coordinate perturbations (x, y), and four flow perturbations (ρ, ρu_x, ρu_y, E). Perturbations are performed only for surface nodes and adjacent cells. Gradients could be generated for every grid node if mesh deformation were desired, but only the surface nodes are considered, for the sake of simplicity. If N is the number of design variables, and thus the number of panels forming the solid surface, the perturbed computations are the equivalent of 6N extra iterations of flow updates. Unlike reverse mode derivatives, which are calculated in a single iteration, the FD approach requires more computational power and scales linear with increasing number of design variables. For a simple 2D model, the extra computational effort is acceptable for validation purposes.

Selection of FD step sizes requires consideration. For Euler2D, variable step sizes are computed based on the geometry and flow state of the panel in question. Node coordinate perturbations are calculated as 10^{-4} times the total distance from the nearest node. This ensures that perturbations do not overlap neighboring nodes. Also, by using total distance, adequate step sizes are calculated even if neighboring nodes have similar x or y values. Flow perturbations are calculated as 10^{-3} times the current state value in a cell. Trial and error was required to select the coefficients. The complex difference code is used to confirm step sizes as shown for several derivatives in Figure 5.

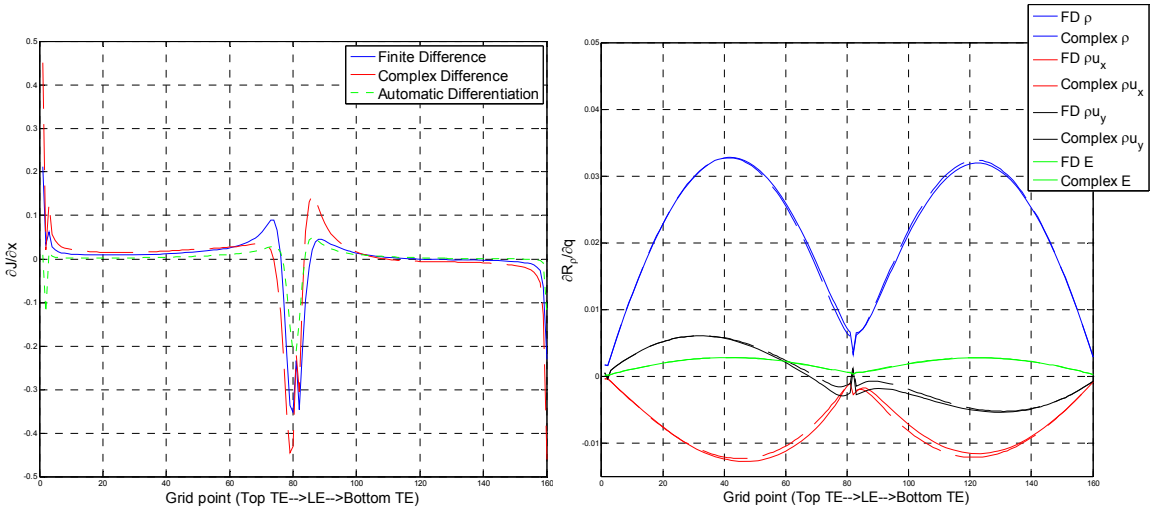


Figure 5 Comparison of exact and inexact numerical differentiation techniques for cost function and residual derivatives. Shown for a NACA 0012 airfoil.

For the proof-of-concept code, all derivatives are exported to an offline MATLAB code and a built-in GMRES algorithm with preconditioning is used to solve for the adjoint variable. Then, the gradient is assembled and the original geometry is manually deformed using information from the gradient. The optimization could be performed automatically if appropriate geometric constraints were included in the aerodynamic force subroutine to robustly handle arbitrary shapes, and if GMRES was written into Euler2D. Automatic optimization was unnecessary for this proof-of-concept because a single gradient is sufficient to validate the adjoint calculation.

Figure 6 compares an FD gradient to Praveen's AD gradient and complex difference derivatives computed by applying complex perturbations and converging new flow solutions.

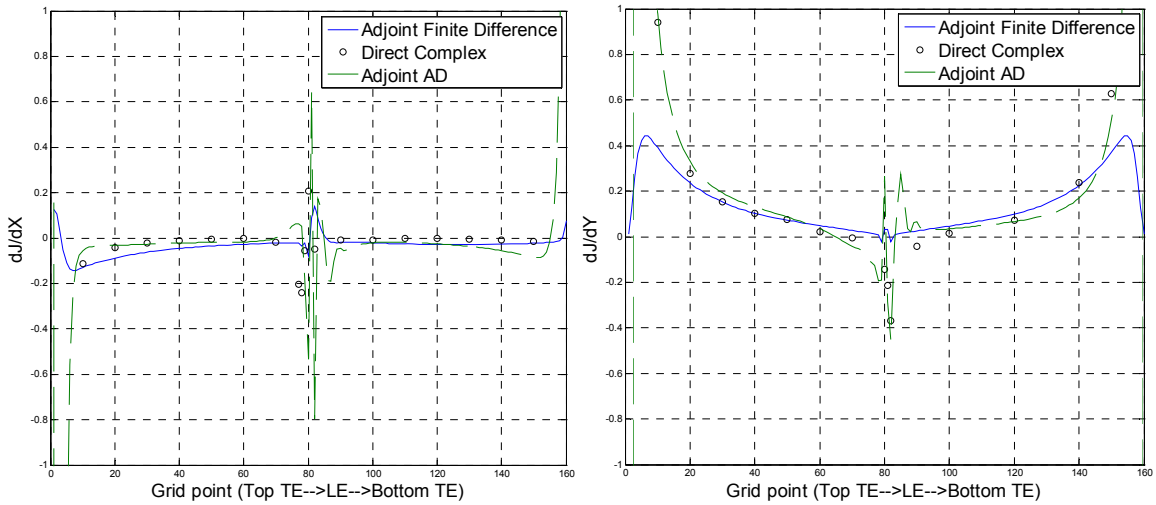


Figure 6 Comparison of inexact gradient calculation when compared to exact indirect (adjoint) and exact direct (new flow solutions) methods.

The FD gradient reasonably matches the other modes. It is expected that significant inaccuracy will occur at discontinuous geometry, such as the trailing edge. However, Giles shows that adjoint computations have inaccuracy as well.[45] As a result, a practical application of automatic optimization may benefit from the smoothing effect FD has on the final gradient.

IV. Results

The FD adjoint solution and gradient calculation is applied to two examples where we know *a priori* what shape the geometry should take. The first example is applied to a NACA 0012 airfoil in transonic flow. The second example is a wedge in supersonic flow. The resulting gradients are applied to show geometry change after one iteration. The modified shapes agree with expected results; however, some important lessons are learned for future application.

A. Transonic NACA 0012

The symmetric NACA 0012 airfoil is analyzed at 3° angle-of-attack, Mach 0.8. The shape change resulting from lift maximization is shown in Figure 7.

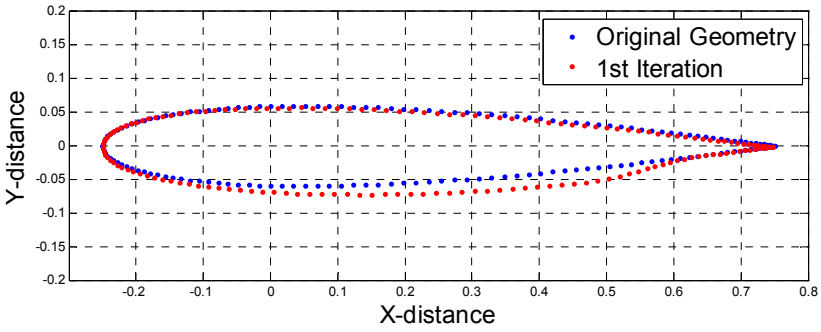


Figure 7 NACA 0012, AoA=3°, M=0.8. The next design iteration assumes features of a supercritical airfoil.

The modified shape begins to resemble a supercritical airfoil, a shape optimized for high-lift in the transonic regime. This result resembles Jameson's investigation of a similar optimization problem.[43]

The transonic airfoil at an angle-of-attack was selected as an example case after initially attempting to maximize lift for a NACA 0012 in level flight. Zero lift from a level, symmetric airfoil meant that numerical error was large relative to the lift force. This caused erratic finite difference gradients. If the initial shape is at an angle-of-attack or

has a slight camber, the results are more reasonable because numerical error is negligible relative to the non-zero value of lift force. This is a shortcoming of finite difference gradients not experienced when using complex difference or AD derivatives, which are less susceptible to the value of the cost function.

B. Supersonic Wedge

A wedge is analyzed for zero angle-of-attack, Mach 1.4. Drag minimization is performed so that results can be compared to a von Kármán ogive, an analytically-derived shape that minimizes wave drag.

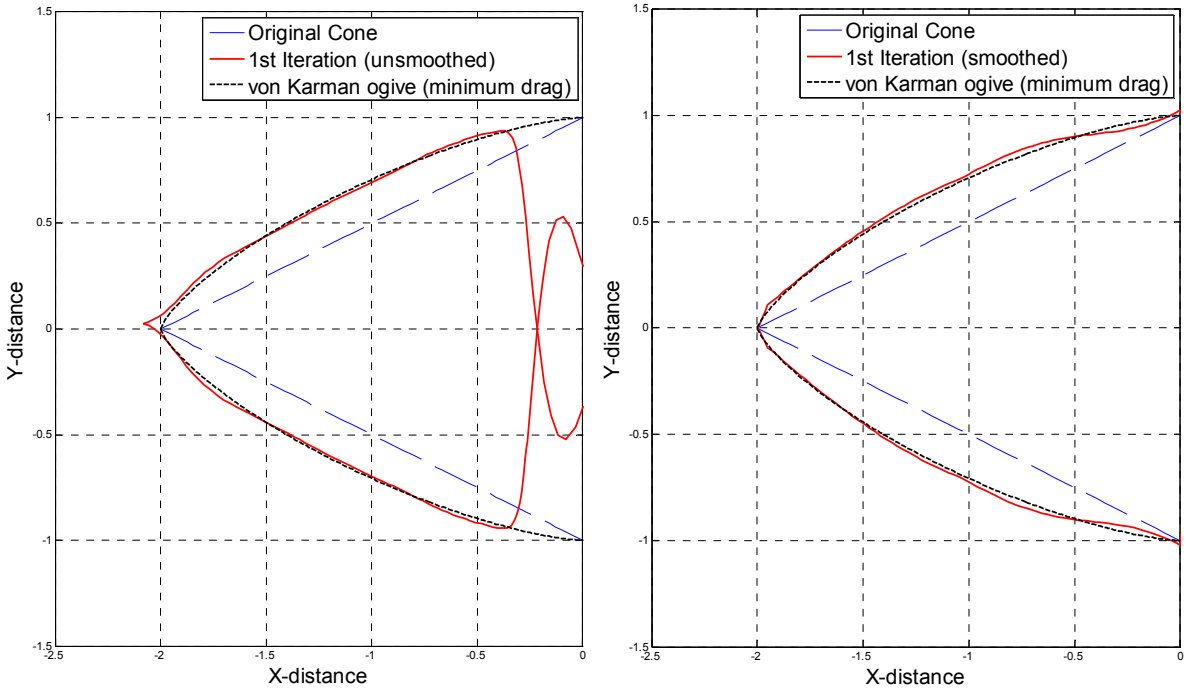


Figure 8 Wedge, AoA=0°, M=1.4. The next design iteration begins to resemble portions of the drag-minimum ogive. After smoothing the gradient, the next design iteration closely approximates the ogive.

The resulting wedge matches most of the ogive with good accuracy. The sharpening of the nosetip could be caused by several effects. For example, a more slender body would reduce drag. Also, moving the lead point forward extends the nosetip into a lower pressure region. Because the flow derivatives are decoupled from the geometry, it is possible that the pressure effect of extending the nose is not adequately captured. Small numerical error causes the nosetip to be shifted slightly above the centerline. Nominally, the gradient and resulting geometry should be symmetric.

When a smoothing function is applied to the gradient, the resulting shape closely matches the ogive. For this case, the smoothing is performed using a moving average function; other applications might make use of an elliptic equation scheme to smooth the gradient.[8] Smoothing the gradient and fixing the nosetip position resolves the issues of the unmodified gradient. This case is intended to illustrate the importance of *a priori* knowledge of the solution. Essentially, large geometry changes between iterations cannot be trusted implicitly.

V. Modification of NASCART-GT

Adjoint-based gradients have slowly found their way into a few research CFD codes. However, development generally requires years to produce a robust adjoint CFD code. While numerical difficulties complicate later phases of development, the first hurdle that must be cleared is rewriting the code in a format that permits computation of exact derivatives. For a finite difference implementation, NASCART-GT's residual update and aerodynamic force functions must be isolated as separate subroutines. Once this is done, FD itself is a trivial task if the correct values are perturbed. NASCART-GT represents geometries with a Cartesian grid, meaning that flow calculations are performed on a Cartesian approximation of the original surface. This presents an additional layer of complexity for the design variable derivatives which may or may not have a direct effect on the computational grid seen by the flow

solver. Aftosmis et al. solve this problem by including in the gradient computation the sensitivity of the computational mesh with respect to the original design variables.[11] This is performed by isolating the gridding functions and using one of the aforementioned numerical differentiation techniques. The gradient calculation assumes the following form:

$$G = \frac{\partial J}{\partial \alpha} + \frac{\partial J}{\partial M}\frac{\partial M}{\partial T}\frac{\partial T}{\partial \alpha} + \upsilon^T\left(\frac{\partial R}{\partial \alpha} + \frac{\partial R}{\partial M}\frac{\partial M}{\partial T}\frac{\partial T}{\partial \alpha}\right) \quad \text{s.t.} \quad \left(\frac{\partial R}{\partial q}\right)^T \upsilon = -\left(\frac{\partial J}{\partial q}\right)^T \tag{12}$$

where M = computational mesh = f[T(α)]
T = wetted surface triangulation

The mesh sensitivity approach can work with both types of design variables: individually defined nodes that form a shape, or parameters that shape generation functions use to create discrete nodes (i.e. maximum diameter, nose angle, splines, or Bezier curve parameters). If future implementation into NASCART uses 2-D shape parameters, the number of design variables will be small enough to enable complex difference derivatives, which is generally a simpler implementation. Development using complex differences would require that all variables be declared to be complex double precision. All operations within the residual and aerodynamic force calculations need to handle complex math, and logical operators must be modified to operate on the real portion of variables. Because complex operations take about twice the computational effort, it is advisable to use the real code to converge the flow solution and then use complex residual and force functions for the adjoint portion.

3-D shapes can require extremely complicated spline and curve generation functions, so use of hundreds or thousands of geometric nodes as the design variables could ease development (as shown in the Euler2D example).[8] Such a high number of design variables would necessitate the use of reverse-mode AD. The obstacle to applying AD is NASCART's use of a common external file for storing arrays, including the state vector. While FD and complex difference can perform perturbations on common arrays, AD requires that differentiable variables be passed as arguments between functions. All of the function calls within NASCART would need to be rewritten in a format acceptable for AD tools. In order to ease code rewriting, a developer will need to have experience with how TAPENADE parses FORTRAN code, and how it generates flow charts of interactions with active variables.

A summary of required tasks for future development of NASCART-GT are provided below:

1. Decide whether design variables will be comprised of nodes or shape parameters. Implement a shape generation function if necessary.
2. Rewrite isolated residual and force subroutines as functions of the design variables and the flow for use outside of the iterative flow solver.
3. Choose whether gradients will be calculated as a function of the computational surface or the original body surface. Isolate mesh generation subroutines.
4. Implement FD to calculate mesh derivatives, flow derivatives, and design variable derivatives for cells adjacent to the surface.
5. Convert the aerodynamic force subroutine into a complex code and use complex differences to validate FD step sizes. This relatively easy compared to making the residual function complex. Additionally, AD already has been successfully applied to the aerodynamic force function.
6. Program a linear algebra solver, such as GMRES, to solve the adjoint equation. Then assemble the gradient using the FD components. Perturb individual design variables and converge a new flow solution to check the validity of the gradient. Also, simple variables like freestream Mach number and angle-of-attack can be used to validate the gradient.
7. Once the FD code is validated, derivative components can be replaced individually. As complex difference or AD routines are implemented the gradients will become more exact. Validate the resulting derivatives and gradient as each module is replaced.
8. Watch for problems in the adjoint solution specific to the CFD model; singularities and linear instabilities are frequent problems.

Further avenues for expanding NASCART's capabilities for hypersonic shape optimization are including chemistry states in the governing equations, and application of Navier-Stokes equations. Unsteady solutions can be considered but many researchers have reported difficulties with solving unsteady adjoint equations.

VI. Conclusions

The problem of direct shape optimization has been reformulated as an indirect problem using adjoint equations. The indirect formulation decouples the effects of flow derivatives and shape derivatives on the gradient, permitting rapid gradient computation from a single flow solution. The result is an optimization method that is suitable for early design phases. The complexity of modifying an existing CFD code is an obstacle to implementing an indirect formulation. In an attempt to ease the code development effort, a discrete adjoint system is solved using a linear algebra solver instead of the more complicated numerical scheme used to converge the flow solution. While complex differences and AD can provide exact derivatives, finite differences are used to further ease the initial implementation. The phased implementation uses FD derivatives as validation for individual gradient components as more complex numerical derivatives are applied. The examples shown verified that FD could produce sufficient accuracy for checking preliminary adjoint solvers. The next step will be to apply this framework to the more complicated CFD code, NASCART-GT, and develop a high-fidelity shape optimization capability for hypersonic aeroshells.

References

[1] Jameson, A., "Aerodynamic Design via Control Theory," *Journal of Scientific Computing*, Vol. 3, No. 3, 1988.

[2] Giles, M.B., Pierce, N.A., "An Introduction to the Adjoint Approach to Design," *Flow, Turbulence, and Combustion*, Vol. 65, 2000, pp. 393-415.

[3] Pironneau, O., "On Optimum Design in Fluid Mechanics," *Journal of Fluid Mechanics*, Vol. 64, pp. 97-110, 1974.

[4] Reuther, J., Jameson, A., "Supersonic Wing and Wing-Body Shape Optimization Using an Adjoint Formulation," *ASME International Mechanical Engineering Congress and Exposition*, San Francisco, CA, November 1995.

[5] Giles, M.B., Pierce, N.A., "Adjoint equations in CFD: Duality, boundary conditions and solution behaviour," AIAA Paper 97-1850, 1997.

[6] Thesinger, J., Braun, R.D., "Hypersonic Entry Aeroshell Shape Optimization," MS Special Problems Report, Georgia Institute of Technology, 12 September 2007.

[7] Jameson, A., "Optimum aerodynamic design using CFD and control theory," AIAA Paper 95-1729-CP, 1995.

[8] Mohammadi, B., "Optimal shape design, reverse mode of automatic differentiation and turbulence," AIAA Paper 97-0099, 1997.

[9] Reuther, J., Jameson, A., "Control based airfoil design using the Euler equations," AIAA Paper 94-4272-CP, 1994.

[10] Iollo, A., Salas, M.D., Ta'asan, S., "Shape Optimization Governed by the Euler Equations using an Adjoint Method," ICASE Report 93-78, 1993.

[11] Nemec, M., Aftosmis, M.J., "Aerodynamic Shape Optimization Using a Cartesian Adjoint Method and CAD Geometry," *AIAA Applied Aerodynamics Conference*, San Francisco, CA, 5-8 June 2006.

[12] Anderson, W.K., Venkatakrishnan, V., "Aerodynamic Design Optimization on Unstructured Grids with a Continuous Adjoint Formulation," ICASE Report 97-9, 1997.

[13] Borggaard, J., Burns, J.A., Cliff, E., Gunzburger, M., "Sensitivity Calculation for a 2-D, Inviscid, Supersonic Forebody Problem," ICASE Report 93-13, 1993.

[14] Baysal, O., Ghayour, K., "Continuous Adjoint Sensitivities for Optimization with General Cost Functionals on Unstructured Meshes," *AIAA Journal*, No. 1, 2000, pp. 48-55.

[15] Xie, L., "Gradient-Base Optimum Aerodynamic Design Using Adjoint Methods," Ph.D thesis, Virginia Polytechnic Institute, Blacksburg, VA, 2002.

[16] Beux, F., Dervieux, A., "Exact-Gradient Shape Optimization of a 2-D Euler Flow," *Finite Elements in Analysis and Design*, Vol. 12, 1992, pp. 281-302.

[17] Nemec, M., Aftosmis, M.J., "Adjoint Algorithm for CAD-Based Shape Optimization Using a Cartesian Method," *AIAA Computational Fluid Dynamics Conference*, Toronto, ON, 6-9 June 2005.

[18] Nemec, M., Aftosmis, M.J., Murman S.M., Pulliam, T.H., "Adjoint Formulation for an Embedded-Boundary Cartesian Method," *AIAA Aerospace Sciences Meeting*, Reno, NV, 10-13 January 2005.

[19] Borggaard, J., Burns, J., "A PDE Sensitivity Equation Method for Optimal Aerodynamic Design," *Journal of Computation Physics*, Vol. 136, No. 2, 1997, pp. 366-384.

[20]Nielsen E.J., "Adjoint-Based Algorithms for Adaptation and Design Optimization on Unstructured Grids," NASA LaRC.
[21]Nielsen, E.J., Kleb, W.L., "Efficient Construction of Discrete Adjoint Operators on Unstructured Grids by Using Complex Variables," AIAA Paper 2005-324, 2005.
[22]Nielsen, E.J., Lu, J., Park, M.A., Darmofal, D.L., "An Implicit, Exact Dual Adjoint Solution Method for Turbulent Flows on Unstructured Grids," *Computers & Fluids*, Vol. 33, 2004, pp. 1131-1155.
[23]Giles, M.B., "On the Iterative Solution of Adjoint Equations," Oxford University Computing Laboratory.
[24]Giering, R., Kaminski, T., "Recipes for Adjoint Code Construction," *ACM Transactions on Mathematical Software*, Vol. 24, No. 4, December 1988, pp. 437-474.
[25]Nadarajah, S.K., Jameson, A., "A Comparison of the Continuous and Discrete Adjoint Approach to Automatic Aerodynamic Optimization," AIAA Paper 2000-0667, 2000.
[26]Nadarajah, S.K., "The Discrete Adjoint Approach to Aerodynamic Shape Optimization," Ph.D thesis, Stanford University, January 2003.
[27]Giles, M.B., Duta, M.C., Muller, J., Pierce, N.A., "Algorithm Developments for Discrete Adjoint Methods," *AIAA Journal*, Vol. 41, No. 2, February 2003, pp. 198-205.
[28]Praveen, C., "Adjoint Code Development and Optimization Using Automatic Differentiation," India National Aerospace Laboratories, Report No. PD CF 0604, April 2006.
[29]Giles, M.B., Ghate, D., Duta, M.C., "Using Automatic Differentiation for Adjoint CFD Code Development," *SAROD Workshop*, 2005.
[30]Hascoët, L., Pascual, V., "TAPENADE 2.1 User's Guide," INRIA, Université of Nice – Sophia Antipolis, September 2004.
[31]Courty, F., Dervieux, A., Koobus, B., Hascoët, L., "Reverse Automatic Differentiation for Optimum Design: from Adjoint State Assembly to Gradient Computation," July 29, 2003.
[32]Nemec, M., Aftosmis, M.J., "Reentry-Vehicle Shape Optimization Using a Cartesian Adjoint Method and CAD Geometry," *AIAA Applied Aerodynamics Conference*, San Francisco, CA, 5-8 June 2006.
[33]Martins, J.R.R.A., Kroo, I.M., Alonso, J.J., "An Automated Method for Sensitivity Analysis Using Complex Variables," AIAA Paper 2000-0689, 2000.
[34]De Pauw, D.J.W., Vanrolleghem, P.A., "Avoiding the Finite Difference Sensitivity Analysis Deathtrap by Using the Complex-step Derivative Approximation Technique," *International Environmental Modelling and Software Society*, 2006.
[35]Martinelli, M., "Sensitivity Evaluation in Aerodynamic Optimal Design," Ph.D thesis, Université of Nice – Sophia Antipolis, 2007.
[36]Betts, J., "Survey of Numerical Methods for Trajectory Optimization," *Journal of Guidance, Control, and Dynamics*, Vol. 21, No. 2, March-April 1998, pp. 193-207.
[37]Carpentieri, G., Koren, B., van Tooren, M.J.L., "Adjoint-based Aerodynamic Shape Optimization on Unstructured Meshes," *Journal of Computation Physics*, Vol. 224, 2007, pp. 267-287.
[38]Reuther, J., Jameson, A., "Control Theory Based Airfoil Design for Potential Flow and Finite Volume Discretization", AIAA Paper 94-0499, 1994.
[39]Newman III, J.C., Taylor III, A.C., Barnwell, R.W., Newman, P.A. and Hou, G.J.-W., "Overview of sensitivity analysis and shape optimization for complex aerodynamic configurations" *Journal of Aircraft,* Vol. 36, No. 1, 1999, pp. 87–96.
[40]Ghate, D., Giles, M.B., "Hessian Calculations using Automatic Differentiation," 6 December 2006.
[41]Kim, H., Sasaki, D., Obayashi, S., Nakahashi, K., "Aerodynamic Optimization of Supersonic Transport Wing Using Unstructured Adjoint Method," *AIAA Journal*, Vol. 39, No. 6, June 2001, pp. 1011-1020.
[42]Thomas, J.P., Hall, K.C., Dowell, E.H., "A Discrete Adjoint Approach for Modeling Unsteady Aerodynamic Design Sensitivities," AIAA Paper 2003-0041, 2003.
[43]Jameson, A., "Aerodynamic Shape Optimization Using the Adjoint Method," Lecture at Stanford University, 3 February 2003.
[44]Mader, C.A., Martins, J.R.R.A., Alonso, J.J., van der Weide, E., "Adjoint: An Approach for the Rapid Development of Discrete Adjoint Solvers," *AIAA Journal*, Vol. 46, No. 4, April 2008.
[45]Giles, M.B., Pierce, N.A., "Adjoint Equations in CFD: Duality, Boundary Conditions, and Solution Behavior," AIAA Paper 97-1850, 1997.