JOURNAL OF COMPUTATIONAL PHYSICS **129,** 307–331 (1996)
ARTICLE NO. 0252

First- and Second-Order Aerodynamic Sensitivity Derivatives via Automatic Differentiation with Incremental Iterative Methods

Laura L. Sherman,* Arthur C. Taylor III,*,[1] Larry L. Green,† Perry A. Newman,†
Gene W. Hou,* and Vamshi Mohan Korivi*

*Department of Mechanical Engineering, Old Dominion University, Norfolk, Virginia 23529-0247; and †Multidisciplinary Design Optimization Branch, NASA Langley Research Center, Hampton, Virginia 23665-5525

Received February 17, 1995; revised April 1, 1996

The straightforward automatic-differentiation and the hand-differentiated incremental iterative methods are interwoven to produce a hybrid scheme that captures some of the strengths of each strategy. With this compromise, discrete aerodynamic sensitivity derivatives are calculated with the efficient incremental iterative solution algorithm of the original flow code. Moreover, the principal advantage of automatic differentiation is retained (i.e., all complicated source code for the derivative calculations is constructed quickly with accuracy). The basic equations for second-order sensitivity derivatives are presented, which results in a comparison of four different methods. Each of these four schemes for second-order derivatives requires that large systems are solved first for the first-order derivatives and, in all but one method, for the first-order adjoint variables. Of these latter three schemes, two require no solutions of large systems thereafter. For the other two for which additional systems are solved, the equations and solution procedures are analogous to those for the first-order derivatives. From a practical viewpoint, implementation of the second-order methods is feasible only with software tools such as automatic differentiation, because of the extreme complexity and large number of terms. First- and second-order sensitivities are calculated accurately for two airfoil problems, including a turbulent-flow example. In each of these two sample problems, three dependent variables (coefficients of lift, drag, and pitching-moment) and six independent variables (three geometric-shape and three flow-condition design variables) are considered. Several different procedures are tested, and results are compared on the basis of accuracy, computational time, and computer memory. For first-order derivatives, the hybrid incremental iterative scheme obtained with automatic differentiation is competitive with the best hand-differentiated method. Furthermore, it is at least two to four times faster than central finite differences, without an overwhelming penalty in computer memory.

1. INTRODUCTION

The use of advanced computational fluid dynamics (CFD) analysis codes in multidisciplinary design optimization studies and applications via sensitivity analysis requires the efficient and accurate calculation of individual-discipline sensitivity derivatives (SDs). The incremental iterative method (IIM) was proposed and demonstrated to provide such first-order SDs from a two-dimensional (2D) thin-layer Navier-Stokes (TLNS) flow code for both geometric (shape) and nongeometric (flow) design variables in Refs. [1, 2]. The IIM allows accurate, consistent discrete SDs to be obtained with computational efficiency (with respect to both computational time and memory requirements). Furthermore, the IIM also allows the use of approximate matrix operators for further efficiency, parallelization, or robustness, etc. Results for first-order SDs from an IIM for three-dimensional (3D) Euler codes (Refs. [3–5]) have also been presented. In all of the above cited works, the discretized flow residuals were differentiated by hand (also called the quasi-analytical (QA) method) and assembled to obtain the first-order SDs by an IIM.

In the present study, numerical results are given for the application of automatic differentiation (AD) [6–8] to obtain first-order aerodynamic SDs from an IIM for the same 2D TLNS code and sample problems studied in [1]. The numerical results are compared on the basis of accuracy and computational time and memory. Previous first-order SDs from the hand-differentiated IIM and the central finite-difference (CD) method [1] are compared with newly obtained AD results for both the straightforward and the incremental-iterative-form applications. This latter approach is new. That is, previous straightforward applications of AD to advanced CFD codes [9–12] did not result in an incremental iterative form, as will be discussed subsequently. This problem was recognized and noted in [2, 9–11], in which the use of AD in incremental iterative form was proposed.

An additional focus of this study is the development of the basic equations for computing second-order discrete aerodynamic SDs, which yields four methods. Where applicable, the incremental iterative forms of these equations are also given. Numerical results (i.e., second-order SDs) are shown for the same 2D sample problems for which first-order SDs are calculated.

[1] Correspondence should be addressed to Arthur C. Taylor, III, Dept. of Mechanical Engineering, KDH 238, Old Dominion University, Norfolk, VA 23529-0247.

0021-9991/96 $18.00

The second-order aerodynamic SDs are of interest for several reasons. For example, aerodynamic stability derivatives are required by the controls discipline as an input. Therefore, inclusion of controls in a gradient-based multidisciplinary design optimization procedure means that the sensitivities of stability derivatives are needed, which are second-order SDs. Second, in constructing function approximations for nonlinear flow behavior, the expansions that use first-order derivative information are only of limited usefulness. For N independent variables **D**, the truncated Taylor series

$$f(\mathbf{D} + \Delta\mathbf{D}) \simeq f(\mathbf{D}) + \sum_{j=1}^{N} \frac{\partial f}{\partial D_j} \Delta D_j \tag{1.1}$$

is a linear approximation. If the derivatives $\partial^2 f / \partial D_k \partial D_j$ are available, then

$$f(\mathbf{D} + \Delta\mathbf{D}) \simeq f(\mathbf{D}) + \sum_{j=1}^{N} \frac{\partial f}{\partial D_j} \Delta D_j + \frac{1}{2} \sum_{j=1}^{N} \sum_{k=1}^{N} \frac{\partial^2 f}{\partial D_k \partial D_j} \Delta D_j \Delta D_k \tag{1.2}$$

may exhibit much of the nonlinear behavior of f. Third, and more importantly, second-order optimization techniques can be employed.

The remainder of the paper is organized as follows: Section 2 discusses first-order derivatives, including development of the equations, methods, results (tabulated in Appendix A), and conclusions. Section 3 discusses second-order derivatives, including development of the equations, methods, results (tabulated in Appendix B), and conclusions. Section 4 presents a summary and final conclusions. A relatively large number of nonstandard acronyms are used in this article, in order to streamline the text and the tables of results. Therefore, for the convenience of the reader, the acronyms used throughout the article are collected in tables in Appendix C, as follows. General terminology has been collected in Table C.1. Acronyms which refer to methods for the first-order SDs have been gathered in Table C.2; for the second-order derivatives, the same is given in Table C.3.

2. FIRST-ORDER SENSITIVITY DERIVATIVES (FO SDs)

2.1. Basic Equations and Incremental Iterative Forms

A brief review is given of the basic equations of first-order (FO) discrete aerodynamic sensitivity analysis; also included is the incremental iterative form for solving the equations. A thorough discussion is given in [1, 2]. Reference [13], which is a review article, and the references therein provide an overview of recent advances in FO sensitivity analysis for modern, nonlinear CFD software.

After discretization, the nonlinear, multidimensional steady-state governing equations of fluid flow and the boundary conditions are approximated as a large system of coupled nonlinear algebraic equations as

$$\mathbf{R} = \mathbf{R}(\mathbf{Q}(\mathbf{b}), \mathbf{X}(\mathbf{b}), \mathbf{b}) = \mathbf{0}, \tag{2.1}$$

where **Q** is the vector of field variables, **X** is the computational grid, and **b** is the vector of independent input (design) variables. Similarly, the vector of aerodynamic output functions **F** is dependent on **Q**, **X**, and **b** as

$$\mathbf{F} = \mathbf{F}(\mathbf{Q}(\mathbf{b}), \mathbf{X}(\mathbf{b}), \mathbf{b}). \tag{2.2}$$

In Eqs. (2.1) and (2.2) and all subsequent equations, all applicable terms are evaluated at the steady-state flow conditions, unless explicitly superscripted with an appropriate iteration index.

2.1.1. *The Direct Differentiation (DD) Method*

Differentiation of Eqs. (2.2) and (2.1) with respect to **b** yields the respective matrix equations

$${}^{\mathrm{D}}\mathbf{F}' \equiv \frac{\partial \mathbf{F}}{\partial \mathbf{Q}} \mathbf{Q}' + \frac{\partial \mathbf{F}}{\partial \mathbf{X}} \mathbf{X}' + \frac{\partial \mathbf{F}}{\partial \mathbf{b}}, \tag{2.3}$$

$$\mathbf{R}' = \frac{\partial \mathbf{R}}{\partial \mathbf{Q}} \mathbf{Q}' + \frac{\partial \mathbf{R}}{\partial \mathbf{X}} \mathbf{X}' + \frac{\partial \mathbf{R}}{\partial \mathbf{b}} = \mathbf{0}, \tag{2.4}$$

where

$${}^{\mathrm{D}}\mathbf{F}' \equiv \frac{d\mathbf{F}}{d\mathbf{b}}; \quad \mathbf{R}' \equiv \frac{d\mathbf{R}}{d\mathbf{b}}; \quad \mathbf{Q}' \equiv \frac{d\mathbf{Q}}{d\mathbf{b}}; \quad \mathbf{X}' \equiv \frac{d\mathbf{X}}{d\mathbf{b}}.$$

The matrix ${}^{\mathrm{D}}\mathbf{F}'$ contains the SDs of interest; the superscript D denotes that they are obtained by the direct differentiation (DD) method. The matrix $\mathbf{Q}'$ represents the SDs of the field variables. The matrix $\mathbf{X}'$ represents the grid-sensitivity terms (which typically are obtained by differentiating the grid-generation code). The very large linear system (Eq. (2.4)) is solved first for $\mathbf{Q}'$ so that the SDs ${}^{\mathrm{D}}\mathbf{F}'$ can be calculated subsequently.

2.1.2. *The Adjoint-Variable (AV) Method*

As an alternative to solving Eq. (2.4) for $\mathbf{Q}'$, an adjoint-variable matrix **A** is introduced to combine Eqs. (2.3) and (2.4). The matrix **A** is then specified to ensure that the resulting coefficients of $\mathbf{Q}'$ vanish. The adjoint-variable (AV) method becomes

$$ {}^{\mathrm{A}}\mathbf{F}' \equiv \frac{\partial \mathbf{F}}{\partial \mathbf{X}}\mathbf{X}' + \frac{\partial \mathbf{F}}{\partial \mathbf{b}} + \mathbf{A}^{\mathrm{T}}\left(\frac{\partial \mathbf{R}}{\partial \mathbf{X}}\mathbf{X}' + \frac{\partial \mathbf{R}}{\partial \mathbf{b}}\right) \tag{2.5} $$

$$ \mathbf{G}' \equiv \left(\frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\right)^{\mathrm{T}}\mathbf{A} + \left(\frac{\partial \mathbf{F}}{\partial \mathbf{Q}}\right)^{\mathrm{T}} = \mathbf{0}. \tag{2.6} $$

The matrix ${}^{\mathrm{A}}\mathbf{F}'$ contains the sensitivity derivatives of interest; the superscript A denotes that they are obtained by the AV method. However, ${}^{\mathrm{D}}\mathbf{F}' = {}^{\mathrm{A}}\mathbf{F}' \equiv d\mathbf{F}/d\mathbf{b}$. The very large linear system of Eq. (2.6) is first solved for $\mathbf{A}$ in order that the SDs ${}^{\mathrm{A}}\mathbf{F}'$ can be calculated subsequently.

The dimension of $\mathbf{b}$ and, thus, the column dimension of $\mathbf{Q}'$ is the number of design variables (NDV). The dimension of $\mathbf{F}$ and, thus, the column dimension of $\mathbf{A}$ is the number of output functions (NOF). Therefore, if the NDV is greater than the NOF, then the solution of Eq. (2.6) is likely to be computationally less expensive than that of Eq. (2.4). (It will definitely be less expensive for a direct-solution procedure. Iterative methods are normally required, however, because of the extreme size of the coefficient matrix.)

2.1.3. *The Incremental Iterative Solution Method (IIM).*

As an alternative to pure Newton iteration, typical CFD codes employ what is sometimes called quasi-Newton iteration, which is an incremental iterative method (IIM), to solve the nonlinear flow system (Eq. (2.1)). This can be expressed as

$$ -\frac{\widetilde{\partial \mathbf{R}^n}}{\partial \mathbf{Q}}\Delta \mathbf{Q}^n = \mathbf{R}^{\mathrm{n}} \tag{2.7} $$

$$ \mathbf{Q}^{n+1} = \mathbf{Q}^n + \Delta \mathbf{Q}^n; \quad n = 1, 2, 3, ... \tag{2.8} $$

The left-hand-side (LHS) coefficient matrix operator $\widetilde{\partial \mathbf{R}^n}/\partial \mathbf{Q}$ of Eq. (2.7) is, in many CFD codes, at best only a rough approximation to the exact Jacobian matrix operator that is associated with true Newton iteration. Thus, Eqs. (2.7) and (2.8) are intended to represent the broad spectrum of iterative algorithms (either implicit or explicit) that are common to CFD software.

Numerous computational difficulties are associated with solving the FO linear sensitivity equations in the standard form given by either Eq. (2.4) (the DD method) or Eq. (2.6) (the AV method). These difficulties are documented, for example, in [1, 14]. Previous studies [1–5] have shown that these computational difficulties can be overcome (at least in part) by iteratively solving these equations in incremental iterative form. For the DD method (Eq. (2.4)), the IIM is

$$ -\frac{\widetilde{\partial \mathbf{R}}}{\partial \mathbf{Q}}\Delta \mathbf{Q}'^m = \mathbf{R}'^m \tag{2.9} $$

$$ \mathbf{Q}'^{m+1} = \mathbf{Q}'^m + \Delta \mathbf{Q}'^m; \quad m = 1, 2, 3, ..., \tag{2.10} $$

where

$$ \mathbf{R}'^m = \frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\mathbf{Q}'^m + \frac{\partial \mathbf{R}}{\partial \mathbf{X}}\mathbf{X}' + \frac{\partial \mathbf{R}}{\partial \mathbf{b}}. \tag{2.11} $$

In Eq. (2.9), the LHS coefficient matrix $\widetilde{\partial \mathbf{R}}/\partial \mathbf{Q}$ represents any convergent, computationally convenient approximation of the exact Jacobian matrix. In particular, the identical approximate LHS operator and algorithm that are used to solve the nonlinear flow equations can also be used to solve the linear sensitivity equations. Comparison of Eqs. (2.7) and (2.8) with Eqs. (2.9) and (2.10) reveals that the sensitivity equations are solved by interchanging the right-hand side (RHS) of Eq. (2.7) with that of Eq. (2.9) and "freezing" the LHS operator at the steady-state value. At convergence, the accuracy of the SDs is not compromised if the terms on the RHS of Eq. (2.9) are evaluated consistently. The use of the IIM is also applicable in the AV method to solve Eq. (2.6), in which case the LHS operator $\widetilde{\partial \mathbf{R}}/\partial \mathbf{Q}$ must be transposed. The IIM for the AV method becomes

$$ -\left(\frac{\widetilde{\partial \mathbf{R}}}{\partial \mathbf{Q}}\right)^{\mathrm{T}}\Delta \mathbf{A}^m = \mathbf{G}^m \tag{2.12} $$

$$ \mathbf{A}^{m+1} = \mathbf{A}^m + \Delta \mathbf{A}^m; \quad m = 1, 2, 3, ..., \tag{2.13} $$

where

$$ \mathbf{G}^m = \left(\frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\right)^{\mathrm{T}}\mathbf{A}^m + \left(\frac{\partial \mathbf{F}}{\partial \mathbf{Q}}\right)^{\mathrm{T}}, \tag{2.14} $$

and the superscript T indicates a matrix transpose.

2.2. Applications of ADIFOR

This section describes different applications of AD to assist in the efficient, accurate calculation of FO SDs from advanced CFD codes. In particular, the AD precompiler software tool ADIFOR (Automatic Differentiation of Fortran) of [9, 15–18] is used in this study. The ADIFOR precompiler tool is applied to the original FORTRAN program source code from which the SDs are to be obtained. The output of this precompiler procedure is a new, differentiated source code, which upon compilation and execution will compute (exactly, up to machine round-off) the numerical value(s) of the derivative(s) of any specified output function(s) with respect to any specified input variable(s). In addition, the new program will perform the function evaluations of the original code.

2.2.1. *Black-Box Applications*

Application of ADIFOR to FORTRAN coding of an iterative solution algorithm (e.g., CFD software) produces

a similar iterative algorithm for computing the exact derivatives. However, as noted in [2, 9–13, 19], this latter iterative algorithm obtained from a straightforward ''black box'' (BB) AD application may be neither computationally efficient nor robust. Furthermore, in general, it is not in the desired incremental iterative form, even if the original solution algorithm was in that form (as illustrated subsequently). The previous BB applications of ADIFOR to advanced CFD codes [9–12] produced iterative algorithms for SD calculations in which the entire flow-solution algorithm was differentiated.

From the discussion in [9, 10], this process whereby the SDs are iteratively calculated (following the straightforward BB use of AD) can be represented conceptually by first combining Eqs. (2.7) and (2.8) (i.e., the basic CFD flow-solution procedure) to yield

$$\mathbf{Q}^{n+1} = \mathbf{Q}^n - \mathbf{P}^n\mathbf{R}^n; \quad n = 1, 2, 3, ..., \tag{2.15}$$

where $\mathbf{P}^n \equiv (\widetilde{\partial \mathbf{R}^n}/\partial \mathbf{Q})^{-1}$. Differentiation with respect to $\mathbf{b}$ then yields

$$\mathbf{Q}'^{n+1} = \mathbf{Q}'^n - \mathbf{P}^n\mathbf{R}'^n - \mathbf{P}'^n\mathbf{R}^n; n = 1, 2, 3, ... \tag{2.16}$$

In contrast with Eq. (2.16), a hand-differentiated (HD) implementation of the DD method (i.e., Eqs. (2.2) and (2.3)) for the FO SDs can be expressed by combining Eqs. (2.9) and (2.10) (an IIM) to yield

$$\mathbf{Q}'^{m+1} = \mathbf{Q}'^m - \mathbf{P}\mathbf{R}'^m; \quad m = 1, 2, 3, ... \tag{2.17}$$

Symbolically, of course, Eqs. (2.16) and (2.17) are equivalent only at convergence of the flow solution, when $\mathbf{R}^n$ vanishes (which also results in the disappearance of $\mathbf{P}'^n\mathbf{R}^n$), and $\mathbf{P}^n$ becomes constant at the steady-state value. Computationally, however, Eq. (2.17) is potentially much more efficient for several reasons:

(1) If the AD-enhanced CFD flow code is executed after the original code has produced a well-converged flow solution, then the convergence rate of Eq. (2.16) might be accelerated somewhat. However, differentiation of the complete CFD solution algorithm and repeated calculation of these solution-algorithm derivatives (represented by $\mathbf{P}'^n$ in Eq. (2.16)), although unwanted and unnecessary, is not avoided. The computationally wasteful, repeated calculation of $\mathbf{P}'^n$ is likely a significant part of the total work represented by Eq. (2.16).

(2) In Eq. (2.17), the term $\mathbf{P}$ is constant. Thus, in principle, only a one-time calculation is required; thereafter, it should be stored in memory and reused repeatedly for all iterations. In Eq. (2.16), however, $\mathbf{P}^n$ (the CFD flow-solution algorithm) is updated at each iteration. (For 3D CFD calculations on large grids, the computer memory of currently available supercomputers might be too small to store the complete $\mathbf{P}$. In this case, $\mathbf{P}$ obviously cannot be frozen in memory and reused.)

(3) The AD-enhanced CFD code will continue to iterate on the solution to the nonlinear flow equations, regardless of whether or not they are already well converged.

(4) With the straightforward (BB) application of AD represented by Eq. (2.16), all parts of the term $\mathbf{R}'^n$ are forced inside the iteration loop and, thus, are calculated at each iteration. However, for the HD IIM represented by Eq. (2.17), most of the terms of $\mathbf{R}'^m$ can be placed judiciously outside the iteration loop. As seen from Eq. (2.11), only the matrix–matrix multiplication operation $(\partial \mathbf{R}/\partial \mathbf{Q})(\mathbf{Q}'^m)$ must be inside this loop.

(5) The ''vectorization'' properties of the AD-enhanced CFD code (Eq. (2.16)) for efficient operation on Cray-type computers may be severely degraded in comparison with those of the original code. In addition, depending on the approach used in the application of the ADIFOR tool and on how many SDs are calculated simultaneously, the computer memory requirements could become excessive.

Certain BB applications of AD, discussed subsequently, may enable the complete elimination of the second computational difficulty discussed above and greatly limit the impact of the first. Some CFD codes, particularly those of the 2D implicit type, are equipped with an optional computational-work (CW) saving strategy known as the ''frozen Jacobians'' (FJ) option. This scheme takes advantage of the fact that as the quasi-Newton flow-solution method of Eqs. (2.7) and (2.8) converges, the LHS operator of Eq. (2.7) becomes approximately constant. The FJ option provides the capability of freezing (not updating) these terms (represented by $\mathbf{P}^n$ in Eq. (2.15)) for a specified number of iterations. The result is typically a large savings in CW per iteration.

For an AD-enhanced CFD code with the FJ option (henceforth known as the BBFJ method), the potential for CW savings is very large (i.e., proportionally far greater than for the original CFD code because the unnecessary repeated update of $\mathbf{P}^n$ and of the unwanted $\mathbf{P}'^n$ can be avoided in Eq. (2.16)). The AD-enhanced CFD code is simply started with a well-converged flow solution, and the FJ option is set to activate for all iterations (after the first iteration).

With the BBFJ strategy, clearly the first computational inefficiency (discussed previously) is reduced significantly, but not eliminated. It is suggested that further improvements to the BBFJ method might be made by editing out these terms (i.e., $\mathbf{P}'^n\mathbf{R}^n$ of Eq. (2.16)) from the AD-enhanced code. If successful, this process could also have significant collateral benefits with respect to reduced computer memory requirements and restored vectorization.

These improvements are aimed at making the BB method (Eq. (2.16)) more like the IIM applied to the hand-differentiated DD method (given by either Eq. (2.17) or Eqs. (2.9) through (2.11)).

2.2.2. *Incremental Iterative Applications*

Earlier studies [2, 9–11] have proposed that many of the previously discussed computational inefficiencies associated with the BB application of ADIFOR to CFD codes (Eq. (2.16)) can be overcome (at least in part) by a more judicious application of ADIFOR. The goal of this approach is that the resulting SD calculations are (more nearly) in the IIM form of a HD application of the DD method (Eq. (2.17)). Specifically, in the present study, ADIFOR is applied to differentiate only the RHS of Eq. (2.7), which is the residual $\mathbf{R}$ of the nonlinear flow equations (Eq. (2.1)). These differentiated terms are assembled on the RHS of Eq. (2.9). The resulting scheme is thereby essentially that expressed by the efficient Eq. (2.17), which is the desired IIM. The construction of the required derivatives is now via AD rather than hand differentiation. This IIM scheme is henceforth known as the ADII method; it should effectively combine an existing, highly efficient, incremental iterative solution algorithm with a fast, accurate, reliable procedure for constructing all required terms.

The ADII strategy will bypass the most obvious computational inefficiencies of the BB strategy. For example, the unnecessary construction and repeated evaluation of the term $\mathbf{P}'^{n}$ in Eq. (2.16) is completely avoided, and the operator $\mathbf{P}$ is not updated at each iteration (at least, in principle, it need not be updated). Furthermore, evaluation of all derivative terms except $(\partial\mathbf{R}/\partial\mathbf{Q})\ \mathbf{Q}'^{m}$ can and should now be placed outside the iteration loop. With the ADII procedure, the computationally wasteful, repeated calculation of $\mathbf{R}^{n}$ is not avoided. Fortunately, however, the repeated full iteration on the nonlinear flow equations does not continue with this scheme. Despite these improvements with the ADII procedure, some important issues remain with respect to its implementation, efficiency, vectorization, and computer memory requirements. These issues are addressed subsequently, at least in part.

A more detailed discussion is provided in [20] of how ADIFOR should be applied to implement the ADII scheme. Only a few key highlights are mentioned here. Most important to note is that this scheme must be assembled with great care to ensure that contributions to the SDs from the boundary conditions are taken into account fully. Failure to do this will result in severe errors in the calculated SDs (Ref. [21]). In the present study, this involves separate applications of AD (i.e., applications to a master boundary-condition subroutine and applications to a master interior-cell-residual subroutine). Furthermore, these applications of AD are further subdivided to ensure the terms which must be calculated inside the iteration loop (i.e., $(\partial\mathbf{R}/\partial\mathbf{Q})\ \mathbf{Q}'^{m}$) can be separated from the remaining terms that should be placed outside the loop. Finally, the AD versions of these subroutines are then carefully interwoven to function as the ADII scheme.

One important and useful feature of the ADIFOR system for AD is that terms of the type $(\partial\mathbf{R}/\partial\mathbf{Q})\ \mathbf{Q}'$ or $(\partial\mathbf{R}/\partial\mathbf{X})\ \mathbf{X}'$ (recall Eqs. (2.4) and (2.11), for example) are calculated without the explicit calculation of the very large Jacobian matrices $\partial\mathbf{R}/\partial\mathbf{Q}$ or $\partial\mathbf{R}/\partial\mathbf{X}$, respectively, and without explicit postmultiplication by the matrices $\mathbf{Q}'$ or $\mathbf{X}'$, respectively. Of course, the AD-enhanced code, which can evaluate these complete expressions, will require increased memory over that of the original code. However, this increase is approximately equal only to the memory of the original code times the column dimension of $\mathbf{Q}'$ and $\mathbf{X}'$. For the present application, this is NDV, which is the dimension of $\mathbf{b}$ (or the dimension of that fraction of $\mathbf{b}$ for which SDs are to be concurrently via the ADII method). The final result is an extremely fortuitous conservation of computer memory. Without this conservation of memory, given the overwhelming size of $\partial\mathbf{R}/\partial\mathbf{Q}$ and $\partial\mathbf{R}/\partial\mathbf{X}$, the application of ADIFOR to advanced CFD codes would be infeasible. Despite these positive features with respect to computer memory, one should note that the CPU time associated with each repeated evaluation of $(\partial\mathbf{R}/\partial\mathbf{Q})\ \mathbf{Q}'^{m}$ via AD-generated code will be significantly larger than that which could be achieved (in principle) via hand differentiation and an efficient, hand-coded procedure for evaluation of these same terms.

In contrast with the preceding discussion, expressions of the form $(\partial\mathbf{R}/\partial\mathbf{Q})^{\mathrm{T}}\ \mathbf{A}$ (recall Eqs. (2.6) and (2.14) of the AV method) cannot currently be evaluated via applications of ADIFOR without the explicit calculation of the very large transposed Jacobian matrix $(\partial\mathbf{R}/\partial\mathbf{Q})^{\mathrm{T}}$ and the postmultiplication of it by the matrix $\mathbf{A}$. For modern CFD codes, this step is completely infeasible. Unfortunately, ADIFOR cannot be considered to assist in the construction of these terms in the IIM for the AV scheme (Eq. (2.14)). Nevertheless, it can be used to construct the remaining terms (Eq. (2.5)).

2.2.3. *Turbulence-Modeling Applications*

The presence of turbulence modeling presents a challenge in the calculation of aerodynamic SDs. The task of differentiating the turbulence-modeling terms to include their influence in the Jacobian matrices and other terms of the DD and AV methods can be exceedingly complex to do by hand. Symbolic manipulators could be used to differentiate the algebraic equations involved, with program-flow control by macros, and then to generate SD-

code. However, ADIFOR has the advantage of being able to directly work with the existing FORTRAN source code, with automatic program-flow control and global dependency checking.

In the earlier study of [1], a HD IIM version of the DD and AV schemes was created to compliment a 2D TLNS CFD code. (Henceforth, these two HD schemes are referred to as the DDII and AVII methods, respectively.) These two schemes were shown to generate very accurate SDs for constant-viscosity laminar flow but produced significantly erroneous SDs for turbulent flow. This discrepancy resulted because the turbulent viscosity terms (from the Baldwin–Lomax algebraic model (Ref. [22]) were not differentiated by hand because of their complexity. In the present study, ADIFOR is applied to correct this deficiency. That is, ADIFOR is applied to differentiate the turbulence-modeling terms only, and the results are incorporated as a correction in the HD SD-code. This correction is applied to the DDII scheme only, which results in a method known henceforth as DDIITC. This correction could not be added in full to the AVII scheme, for reasons discussed subsequently. Therefore, no AVIITC strategy currently exists.

Conceptually, the AD correction for the viscosity terms is added to the DDII scheme as

$$\frac{\partial \mathbf{F}}{\partial \mathbf{Q}} = \left(\frac{\partial \mathbf{F}}{\partial \mathbf{Q}}\right)_{\mathbf{V}} + \frac{\partial \mathbf{F}}{\partial \mathbf{V}}\frac{\partial \mathbf{V}}{\partial \mathbf{Q}} \tag{2.18}$$

$$\frac{\partial \mathbf{F}}{\partial \mathbf{X}} = \left(\frac{\partial \mathbf{F}}{\partial \mathbf{X}}\right)_{\mathbf{V}} + \frac{\partial \mathbf{F}}{\partial \mathbf{V}}\frac{\partial \mathbf{V}}{\partial \mathbf{X}} \tag{2.19}$$

$$\frac{\partial \mathbf{F}}{\partial \mathbf{b}} = \left(\frac{\partial \mathbf{F}}{\partial \mathbf{b}}\right)_{\mathbf{V}} + \frac{\partial \mathbf{F}}{\partial \mathbf{V}}\frac{\partial \mathbf{V}}{\partial \mathbf{b}}, \tag{2.20}$$

where $\mathbf{V}$ is a vector of viscosity terms (including the turbulent viscosity). The subscript $\mathbf{V}$ in the above indicates differentiation within the term with $\mathbf{V}$ held constant. Thus, terms with this subscript represent the original terms of the uncorrected, HD code. Substitution of the above into Eq. (2.3) of the DD method results in

$${}^{\mathrm{D}}\mathbf{F}' = \left(\frac{\partial \mathbf{F}}{\partial \mathbf{Q}}\right)_{\mathbf{V}}\mathbf{Q}' + \left(\frac{\partial \mathbf{F}}{\partial \mathbf{X}}\right)_{\mathbf{V}}\mathbf{X}' + \left(\frac{\partial \mathbf{F}}{\partial \mathbf{b}}\right)_{\mathbf{V}} + \frac{\partial \mathbf{F}}{\partial \mathbf{V}}\mathbf{V}' \tag{2.21}$$

where

$$\mathbf{V}' \equiv \frac{d\mathbf{V}}{d\mathbf{b}} = \frac{\partial \mathbf{V}}{\partial \mathbf{Q}}\mathbf{Q}' + \frac{\partial \mathbf{V}}{\partial \mathbf{X}}\mathbf{X}' + \frac{\partial \mathbf{V}}{\partial \mathbf{b}}. \tag{2.22}$$

Similar manipulations applied to Eq. (2.4) of the DD method yield

$$\mathbf{R}' = \left(\frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\right)_{\mathbf{V}}\mathbf{Q}' + \left(\frac{\partial \mathbf{R}}{\partial \mathbf{X}}\right)_{\mathbf{V}}\mathbf{X}' + \left(\frac{\partial \mathbf{R}}{\partial \mathbf{b}}\right)_{\mathbf{V}} + \frac{\partial \mathbf{R}}{\partial \mathbf{V}}\mathbf{V}' = \mathbf{0}. \tag{2.23}$$

Clearly, the viscosity-derivative correction terms to be added in Eqs. (2.21) and (2.23) are $(\partial\mathbf{F}/\partial\mathbf{V})\ \mathbf{V}'$ and $(\partial\mathbf{R}/\partial\mathbf{V})\ \mathbf{V}'$, respectively. In the present study, $\partial\mathbf{F}/\partial\mathbf{V}$ and $\partial\mathbf{R}/\partial\mathbf{V}$ are constructed by hand, and $\mathbf{V}'$ is constructed via ADIFOR. The three terms of $\mathbf{V}'$ on the RHS of Eq. (2.22) are constructed with separate applications of ADIFOR so that the terms $(\partial\mathbf{V}/\partial\mathbf{Q})\ \mathbf{Q}'$ can be separated from the others and placed inside the iteration loop. The terms $(\partial\mathbf{V}/\partial\mathbf{Q})\ \mathbf{Q}'$ and $(\partial\mathbf{V}/\partial\mathbf{X})\ \mathbf{X}'$ are assembled without the explicit computation of the Jacobian matrices $\partial\mathbf{V}/\partial\mathbf{Q}$ and $\partial\mathbf{V}/\partial\mathbf{X}$. Thus, these terms are evaluated without an excessive expansion of the computer memory, as discussed previously in regard to the AD-assisted evaluation of the terms $(\partial\mathbf{R}/\partial\mathbf{Q})\ \mathbf{Q}'$ and $(\partial\mathbf{R}/\partial\mathbf{X})\ \mathbf{X}'$.

The IIM for solving Eq. (2.23) becomes

$$-\left(\frac{\widetilde{\partial \mathbf{R}}}{\partial \mathbf{Q}}\right)_{\mathbf{V}}\Delta\mathbf{Q}'^{m} = \mathbf{R}'^{m} = \mathbf{R}'^{m}_{\mathbf{V}} + \frac{\partial \mathbf{R}}{\partial \mathbf{V}}\mathbf{V}'^{m}, \tag{2.24}$$

$$\mathbf{Q}'^{m+1} = \mathbf{Q}'^{m} + \Delta\mathbf{Q}'^{m}; \quad m = 1, 2, 3, ..., \tag{2.25}$$

where

$$\mathbf{R}'^{m}_{\mathbf{V}} = \left(\frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\right)_{\mathbf{V}}\mathbf{Q}'^{m} + \left(\frac{\partial \mathbf{R}}{\partial \mathbf{X}}\right)_{\mathbf{V}}\mathbf{X}' + \left(\frac{\partial \mathbf{R}}{\partial \mathbf{b}}\right)_{\mathbf{V}}, \tag{2.26}$$

$$\mathbf{V}'^{m} = \frac{\partial \mathbf{V}}{\partial \mathbf{Q}}\mathbf{Q}'^{m} + \frac{\partial \mathbf{V}}{\partial \mathbf{X}}\mathbf{X}' + \frac{\partial \mathbf{V}}{\partial \mathbf{b}}. \tag{2.27}$$

In the turbulent sample problem of this study, the term $(\partial\mathbf{V}/\partial\mathbf{Q})\ \mathbf{Q}'^{m}$ was a computationally expensive addition to the iteration loop, even after the ADIFOR-generated code was extensively "massaged" to restore vectorization and other features related to efficiency. Initially, the computational cost per iteration was about 3.62 times more costly per iteration when the correction was switched on, although the overall rate of convergence was not affected greatly. A CW saving strategy was proposed and tested, where the term $(\partial\mathbf{V}/\partial\mathbf{Q})\ \mathbf{Q}'^{m}$ of Eq. (2.27) was frozen (not updated) inside the iteration loop for a specified number of iterations. For 10 frozen iterations prior to each update of this term, the overall increase in average CW per iteration due to this turbulence-modeling correction was about 26.6% (compared with the CW per iteration with this correction switched off) with no major impact on the rate of total error reduction. Furthermore, the correction had little impact on the computer storage requirements.

The unsuccessful attempt to correct the AVII scheme for turbulent flow yielded the following formulation:

$$
{}^{A}\mathbf{F}' = \left(\frac{\partial \mathbf{F}}{\partial \mathbf{X}}\right)_{\mathbf{V}} \mathbf{X}' + \left(\frac{\partial \mathbf{F}}{\partial \mathbf{b}}\right)_{\mathbf{V}} + \mathbf{A}^{\mathrm{T}} \left[\left(\frac{\partial \mathbf{R}}{\partial \mathbf{X}}\right)_{\mathbf{V}} \mathbf{X}' + \left(\frac{\partial \mathbf{R}}{\partial \mathbf{b}}\right)_{\mathbf{V}} \right]
$$

$$
+ \left(\frac{\partial \mathbf{F}}{\partial \mathbf{V}} + \mathbf{A}^{\mathrm{T}} \frac{\partial \mathbf{R}}{\partial \mathbf{V}}\right) \left(\frac{\partial \mathbf{V}}{\partial \mathbf{X}} \mathbf{X}' + \frac{\partial \mathbf{V}}{\partial \mathbf{b}}\right), \tag{2.28}
$$

$$
-\left(\frac{\widetilde{\partial \mathbf{R}}}{\partial \mathbf{Q}}\right)_{\mathbf{V}}^{\mathrm{T}} \Delta \mathbf{A}^{m} = \mathbf{G}^{m} = \mathbf{G}_{\mathbf{V}}^{m} + \mathbf{G}_{\mathrm{TC}}^{m}, \tag{2.29}
$$

$$
\mathbf{A}^{m+1} = \mathbf{A}^{m} + \Delta \mathbf{A}^{m}; \quad m = 1, 2, 3, \ldots, \tag{2.30}
$$

where

$$
\mathbf{G}_{\mathbf{V}}^{m} = \left(\frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\right)_{\mathbf{V}}^{\mathrm{T}} \mathbf{A}^{m} + \left(\frac{\partial \mathbf{F}}{\partial \mathbf{Q}}\right)_{\mathbf{V}}^{\mathrm{T}}, \tag{2.31}
$$

$$
\mathbf{G}_{\mathrm{TC}}^{m} = \left(\frac{\partial \mathbf{V}}{\partial \mathbf{Q}}\right)^{\mathrm{T}} \left[\left(\frac{\partial \mathbf{R}}{\partial \mathbf{V}}\right)^{\mathrm{T}} \mathbf{A}^{m} + \left(\frac{\partial \mathbf{F}}{\partial \mathbf{V}}\right)^{\mathrm{T}} \right]. \tag{2.32}
$$

All parts of the turbulence correction can be constructed easily via ADIFOR, except $\mathbf{G}_{\mathrm{TC}}^{m}$ (Eq. (2.32)). The present version of ADIFOR can construct this term only by explicitly computing the large matrix $(\partial \mathbf{V}/\partial \mathbf{Q})^{\mathrm{T}}$ and postmultiplying it by the terms shown. As discussed previously for the matrix-matrix product $(\partial \mathbf{R}/\partial \mathbf{Q})^{\mathrm{T}}\ \mathbf{A}$, this procedure is not feasible for modern CFD codes.

2.2.4. *Vectorization and Memory Considerations*

Prior to the compilation and execution of any AD-enhanced FORTRAN source code, a parameter gp is specified within the code. For each execution of the code, this parameter determines the number of independent (design) variables with respect to which derivatives are concurrently computed. Thus, the user has the following options:

(1) Compute all required derivatives by executing the AD-enhanced code once for each independent variable (i.e., NDV code executions with gp = 1).

(2) Compute all required derivatives by executing the AD-enhanced code only once (i.e., one code execution, with gp = NDV).

(3) Set gp such that $1 \leq$ gp $\leq$ NDV; this requires multiple executions (less than NDV) of the AD-enhanced code, where subgroups of gp derivatives are concurrently computed for each code execution.

The specified value of gp has a significant impact on computational requirements in several critical ways. With respect to memory, for example, recall that the memory increase of the AD-enhanced code is approximately equal to gp times the memory of the original code. Thus, if this parameter is too large, the memory requirements of the code could be excessive.

The AD-enhanced code retains all do-loops and function evaluations of the original code. Within each original do-loop is inserted one or more new innermost do-loops. The length of each new do-loop is gp (e.g., DO 10 I = 1, gp). Inside these new loops, derivative calculations are made. The presence of these new innermost do-loops has a profound impact (frequently negative) on the vectorization characteristics for performance on Cray-type computers:

(1) The do-loops of the original code, which previously vectorized, will no longer be vectorized in the AD-enhanced version. An exception to this is when gp ≤ 5; the ''aggressive'' Cray compiler option will automatically ''unwind'' the new innermost loops and may restore the vectorization of the original loops, complete with the derivative calculations.

(2) For gp ≥ 6, vectorization of the original loops is not recovered, but with the aggressive compile option, the new innermost loops are vectorized. Nevertheless, overall code performance remains poor on Cray computers unless gp is large enough that the vector lengths become sufficiently long for efficient execution on these machines. At the same time, however, for large gp the computer memory requirements of the AD-enhanced CFD software can become excessively large.

Apart from the vectorization considerations discussed above, the number of arithmetic operations per concurrently computed derivative is always decreased as gp increases. This happens because, for each execution of an AD-enhanced code, part of the derivative calculations occur outside of the innermost loops, and the results are reused for all derivative calculations within the innermost loops. Furthermore, the complete function evaluations of the original code are performed only one (but, as needed, are thereafter used for the derivative calculations within the innermost loops).

The sample problems illustrate the consequences discussed previously; the results from these sample problems are to be given. For example, (except when gp is large) gp = 5 produces the highest computational efficiency per design variable, and this efficiency is progressively reduced as gp is reduced to 1. A particularly inefficient case is that of gp = 6 (thereafter efficiency gradually increases as gp increases). In the case with NDV = 6, rather than perform one code execution with gp = 6, two code executions, each with gp < 6 (e.g., the first execution with gp = 5 and the second execution with gp = 1), were significantly more efficient.

2.3. Computational Results: FO SDs

Two sample problems are considered here. They are identical in every way to those studied previously in [1],

where a more complete description is given. The first example is low-Reynolds-number ($\mathrm{Re} = 5 \times 10^3$) subsonic ($M_\infty = 0.6$), constant-viscosity laminar flow over an isolated NACA 1406 airfoil at an angle-of-attack, $\alpha = 1.0°$. The second example is similar, except the flow is a high-Reynolds-number ($\mathrm{Re} = 5 \times 10^6$) transonic ($M_\infty = 0.8$) turbulent flow. Flow calculations are made on a *C*-mesh with dimension 257×65 (circumferential $\times$ normal direction). The clustering of points near the airfoil's surface was tighter for the high-Reynolds-number example. Grid-sensitivity derivatives were produced with a unique scheme that was first reported in [14] and was subsequently applied to these sample problems in [1].

The CFD code applied here (and in [1]) solves the 2D TLNS equations with an upwind, cell-centered finite-volume formulation with a higher-order-accurate evaluation of all fluxes and the algebraic turbulence modeling of Baldwin and Lomax [22]. The code employs an implicit, spatially split approximate-factorization flow-solution algorithm. In addition to the nonlinear flow equations, this same algorithm is also used within all the subsequent IIM solutions of the linear sensitivity equations. Also available is the FJ option (discussed previously), where the entire approximate implicit operator (i.e., the complete set of LU-factored block-tridiagonal coefficient matrices) is stored in memory and repeatedly reused (not updated) for a specified number of iterations.

As in [1], the FO SDs of three aerodynamic output functions C_L, C_D, and C_M (the coefficients of lift, drag, and pitching-moment, respectively) are calculated with respect to three geometric shape variables T, C, and L (maximum thickness, maximum camber, and location of maximum camber, respectively) and with respect to three flow variables α, M_∞, and Re (each defined previously). Therefore, $\mathbf{F} \equiv (C_\mathrm{L}, C_\mathrm{D}, C_\mathrm{M})^\mathrm{T}$ and $\mathbf{b} \equiv (T, C, L, \alpha, M_\infty, \mathrm{Re})^\mathrm{T}$. The SDs are computed with a wide variety of different methods, including the method of central finite-differences (CD). The results are compared on the basis of accuracy and computational time and memory.

2.3.1. *Accuracy Comparisons*

The FO SDs are calculated for both sample problems with the methods CD, DDII, AVII, ADII, and BB. In addition, the DDIITC scheme is applied only to the turbulent example (because it is unnecessary for the laminar case). The application of the CD, DDII, and AVII schemes to these problems repeats the work of [1]; therein, the manner in which these schemes are applied is discussed in depth. Of course, the DDIITC, ADII, and BB schemes are the methods for which derivatives are calculated via applications of ADIFOR, either in part or in total (depending on the scheme).

The SDs that were calculated are presented in Table A.1 (in Appendix A) for the laminar example. The actual numerical values of the SDs are given for the CD method. For the other methods, SD *ratios* are given (i.e., each SD has been normalized by the respective SD calculated via the CD scheme). Table A.1 clearly shows excellent agreement among all these methods, as expected.

The SDs for the turbulent example are presented in Table A.2. The actual SDs are shown for the CD scheme; the remaining cases are shown as SD ratios. As expected, this table shows excellent agreement within the DDIITC, ADII, BB, and CD methods. The results for the DDII and AVII methods do not agree well (for some SDs) with the other schemes because of the turbulence-modeling terms, as discussed previously herein and in [1]. The erroneous results for the DDII and AVII schemes agree extremely well with each other, however, because the two are algebraically equivalent. For this reason, the results for these two methods are shown as a single result in Table A.2.

The good accuracy and agreement in the preceding results is due in part to the very tight convergence tolerances that were enforced on all calculations. The average total error was reduced to machine zero (a relative reduction of approximately 12 orders-of-magnitude (OM)) in the initial flow solution and in all 12 flow solutions that were required for the CD method (i.e., two solutions per design variable). Very small forward and backward perturbations $\Delta b_j = \pm 5.0E\text{-}6 \times b_j$ are made to each design variable to ensure good accuracy with the CD method. For each linear system that was solved to compute these SDs, the error was reduced at least eight OM. In each sample problem, this involved NDV = 6 solutions for the DDII, DDIITC, ADII, and BB methods, and NOF = 3 solutions for the AVII scheme.

2.3.2. *Computational Time and Memory Comparisons*

In this section, some of the methods discussed in the previous section are further subdivided. These subdivisions have little or no impact on the SDs that are calculated, but they can have a significant impact on the total computational efficiency of the method.

The CD method is subdivided into two methods, depending on whether or not the FJ option is activated. When active, 10 iterations of Eqs. (2.7) and (2.8) are specified prior to each update of the LHS operator. The methods without and with the FJ option are referred to as the CD and CDFJ methods, respectively.

Similarly, the BB method is subdivided into two methods, depending on whether the FJ option is activated (the BBFJ scheme) or not (the simple BB scheme). These two BB methods are further subdivided into additional methods, depending on how the parameter gp is applied. Two options were tested: first, all SDs were calculated with a single execution of the AD-enhanced code (i.e., this implies

gp = NDV = 6); and second, two executions of the code were made—the first with gp = 5 and the second with gp = 1. From these subdivisions, the methods are BB(6), BB(5 + 1), BBFJ(6), and BBFJ(5 + 1). Execution of the AD-enhanced code is started for all of these BB-type methods after the original code has produced the fully converged flow solution. (Recall that the fully converged flow solution is required at startup for the latter two methods.)

The ADII scheme also is subdivided into two methods, depending on the application of gp, to yield the ADII(6) and ADII(5 + 1) methods. For turbulent flow, the DDIITC method is further subdivided according to whether or not the option to freeze the turbulence-correction terms is activated. The terminology DDIITCFR indicates that this option is activated. (Ten frozen iterations are specified for each iteration that updates these terms.) The notation DDIITC indicates that the option is not activated.

Comparisons of the time (total CPU time) are shown in Table A.3 for both the laminar and turbulent sample problems. Results are given in seconds, with all calculations performed on a Cray-YMP computer. All reported timings do not include the cost of the initial flow solution. Note that a superscript * in the tables indicates an estimated result. These estimates are based on results in Table A.4, which shows a comparison of the computational times (the CD and CDFJ methods are excluded) in CPU time per iteration per linear system solved. The results from Table A.3 for the methods ADII(5 + 1), BB(5 + 1), and BBFJ(5 + 1) have been separated in Table A.4 to compare the *individual* effect of gp = 5 and gp = 1.

The total memory requirements for the different methods are compared in Table A.5, where the results are given in mega-words (Mw). No difference occurs in the results for the laminar and turbulent examples in this table. The computer memory requirements for the ADII scheme are less than that for the BB approach, particularly if the original flow code is of the type that uses a large amount of memory for storage of the terms of the LHS operator. (Recall that these terms are not differentiated with the ADII scheme, which results in a significant conservation of computer memory.)

2.4. Conclusions: FO SDs

Conclusions based on the calculations for FO SDs are

(1) The HD IIM schemes, although presently the most efficient, are very difficult and time consuming to construct accurately, even for relatively simple CFD codes. For more complex codes with features such as turbulence modeling, this approach is not feasible. ADIFOR is a reliable tool for the quick construction of accurate source code to evaluate all or parts of the SDs from complex CFD codes, but straightforward BB application of ADIFOR to CFD codes can be slower than even the CD method and require substantially more memory than the original code.

(2) ADIFOR can be used successfully to create corrections to HD SD codes, where only relatively small, previously undifferentiated parts of the original flow code (such as turbulence-modeling subroutines) are differentiated via ADIFOR. These corrections can be very costly with respect to the efficiency of the HD SD code, as seen in Tables A.3 and A.4 for the turbulent-flow problem. However, the cost in computer memory for these corrections is negligible, as shown in Table A.5.

(3) For all applicable methods, the computational penalty associated with the turbulence-modeling terms (all constructed via ADIFOR) is significant and disproportionately high. Detailed comparisons of the laminar and turbulent timings (for a given method) shown in Table A.4 reveal this penalty. The disproportionate cost is highest for the most efficient method and also for the application where gp = 1. The AD-correction for the turbulence-modeling terms in the DDIITC method represents an inefficient (gp = 1) application of ADIFOR; the inordinately large computational cost of this turbulence correction is thereby explained.

(4) The ADII scheme is not as efficient computationally as the HD IIM schemes, but it is more efficient than all other methods tested. For example, depending on the particular sample problem and application of gp, the ADII scheme produces computational improvements by a factor which varies from approximately 6 to 15 when comparisons are made with the simple BB scheme. Similar comparisons to the BBFJ scheme yield factors which vary from approximately 1.4 to 1.7. The most efficient ADII results were approximately 2 to 4 times more efficient than the results from the CDFJ scheme—even more efficient compared with the CD method.

(5) The ADII scheme is not as easy to implement as the BB methods. For example, particular care must be taken to ensure that the contributions from the boundary conditions are properly taken into account. However, when compared with the HD approach, the ADII scheme can be implemented easily with very accurate results, even for very advanced CFD codes. For example, the time required to develop the source code for some of these different methods is estimated: HD IIM (DDII, AVII, etc.)—6 man-months to 2 man-years, or even longer, depending on the complexity of the flow code; ADII—about 1 man-week; BB—about 1 man-day.

(6) The BBFJ strategy is no more difficult to implement than the simple BB approach, if the original flow code is equipped with the FJ option. A very large increase is noted in the computational efficiency (compared with the simple BB strategy) when this option is activated. This improvement was by an impressive factor which varied

from approximately 4 to 9, depending on the particular example problem and application of gp. The BBFJ strategy is not as efficient as the ADII scheme with respect to computational time or memory. However, with relatively minor code modifications, the method possibly could be made to function with nearly the efficiency of the ADII scheme. Therefore, when the FJ option is available, the BBFJ scheme is the simplest method to implement which also maintains reasonable efficiency (compared with the CDFJ scheme in particular); otherwise, the necessary extra effort should be invested to implement the ADII scheme.

(7) For the BB, BBFJ, and ADII schemes, the computational cost in terms of CPU time and computer storage is very sensitive to the value selected for the parameter gp, and this effect can vary significantly for different machines. The significance of gp is critical for a large NDV. On Cray computers with the aggressive compile option, the choice of gp = 5 seems to provide the most efficiency in terms of CPU time per iteration per linear system solved, with a manageable increase in memory.

(8) Currently, ADIFOR cannot be applied to construct the AV method in total. This limitation is a serious consideration, because great potential exists for efficiency with the AV scheme when the number of aerodynamic input variables of interest is much larger than is the number of output functions (i.e., when NDV is much larger than NOF).

3. SECOND-ORDER SENSITIVITY DERIVATIVES (SO SDs)

3.1. Basic Equations and Incremental Iterative Forms

A brief derivation is presented of the basic equations of second-order (SO) discrete aerodynamic sensitivity analysis. The result is four methods, denoted as (1) DD.DD, (2) AV.DD, (3) DD.AV, and (4) AV.AV. This notation roughly parallels the derivation and description given in [23] for SO shape sensitivity analysis applied to linear heat-conduction problems. In addition, the incremental iterative forms are given for solving the additional large linear systems that result from Methods (1) and (2).

For convenience and subsequent notational clarity, the key equations for the first-order (FO) derivatives are repeated, where only the terms for the ith aerodynamic output function (F_i) and for the jth design variable (b_j) are given here. Recall the FO direct differentiation (DD) approach is

$$ {}^{\mathrm{D}}F'_{i;j} \equiv \frac{dF_i}{db_j} = \frac{\partial F_i}{\partial \mathbf{Q}}\mathbf{Q}'_j + \frac{\partial F_i}{\partial \mathbf{X}}\mathbf{X}'_j + \frac{\partial F_i}{\partial b_j} \tag{3.1} $$

$$ \mathbf{R}'_j \equiv \frac{d\mathbf{R}}{db_j} = \frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\mathbf{Q}'_j + \frac{\partial \mathbf{R}}{\partial \mathbf{X}}\mathbf{X}'_j + \frac{\partial \mathbf{R}}{\partial b_j} = \mathbf{0}, \tag{3.2} $$

where $\mathbf{Q}'_j \equiv d\mathbf{Q}/db_j$; $\mathbf{X}'_j \equiv d\mathbf{X}/db_j$. The FO adjoint-variable (AV) approach is obtained by introduction of the adjoint-variable vector $\mathbf{A}_i$ into the preceding equations to eliminate $\mathbf{Q}'_j$, and the result is

$$ {}^{\mathrm{A}}F'_{i;j} \equiv \frac{dF_i}{db_j} = \frac{\partial F_i}{\partial \mathbf{X}}\mathbf{X}'_j + \frac{\partial F_i}{\partial b_j} + \mathbf{A}_i^{\mathrm{T}}\left(\frac{\partial \mathbf{R}}{\partial \mathbf{X}}\mathbf{X}'_j + \frac{\partial \mathbf{R}}{\partial b_j}\right), \tag{3.3} $$

$$ G_i \equiv \left(\frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\right)^{\mathrm{T}}\mathbf{A}_i + \left(\frac{\partial F_i}{\partial \mathbf{Q}}\right)^{\mathrm{T}} = \mathbf{0}. \tag{3.4} $$

The following differential operator $D(\,)/Db_k$ is defined for use in the derivations that follow

$$ \frac{D(\,)}{Db_k} = \frac{\partial(\,)}{\partial \mathbf{Q}}\frac{d\mathbf{Q}}{db_k} + \frac{\partial(\,)}{\partial \mathbf{X}}\frac{d\mathbf{X}}{db_k} + \frac{\partial(\,)}{db_k}. \tag{3.5} $$

Therefore, for example, comparison of this operator with Eqs. (3.1) and (3.2) yields, respectively,

$$ \frac{D(F_i)}{Db_j} = {}^{\mathrm{D}}F'_{i;j}; \quad \frac{D(\mathbf{R})}{Db_j} = \mathbf{R}'_j. \tag{3.6} $$

3.1.1. *Method* 1: DD.DD.

An inspection of Eqs. (3.1) and (3.2) reveals explicit dependencies in each on $\mathbf{Q}'_j$ and $\mathbf{X}'_j$, in addition to $\mathbf{Q}$, $\mathbf{X}$, and $\mathbf{b}$. This dependency is expressed as

$$ \frac{dF_i}{db_j} \equiv {}^{\mathrm{D}}F'_{i;j} = {}^{\mathrm{D}}F'_{i;j}(\mathbf{Q}'_j(\mathbf{b}), \mathbf{X}'_j(\mathbf{b}); \mathbf{Q}(\mathbf{b}), \mathbf{X}(\mathbf{b}), \mathbf{b}) \tag{3.7} $$

$$ \frac{d\mathbf{R}}{db_j} \equiv \mathbf{R}'_j = \mathbf{R}'_j(\mathbf{Q}'_j(\mathbf{b}), \mathbf{X}'_j(\mathbf{b}); \mathbf{Q}(\mathbf{b}), \mathbf{X}(\mathbf{b}), \mathbf{b}). \tag{3.8} $$

Differentiation of Eqs. (3.7) and (3.8) with respect to the kth design variable b_k yields, respectively,

$$ \frac{d^2F_i}{db_k db_j} = \frac{\partial F_i}{\partial \mathbf{Q}}\mathbf{Q}''_{j,k} + \frac{\partial F_i}{\partial \mathbf{X}}\mathbf{X}''_{j,k} + \frac{D({}^{\mathrm{D}}F'_{i;j})}{Db_k} \tag{3.9} $$

$$ \frac{d^2\mathbf{R}}{db_k db_j} = \frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\mathbf{Q}''_{j,k} + \frac{\partial \mathbf{R}}{\partial \mathbf{X}}\mathbf{X}''_{j,k} + \frac{D(\mathbf{R}'_j)}{Db_k} = \mathbf{0}, \tag{3.10} $$

where $\mathbf{Q}''_{j,k} \equiv d^2\mathbf{Q}/db_k db_j$ and $\mathbf{X}''_{j,k} \equiv d^2\mathbf{X}/db_k db_j$.

In the preceding differentiation, the chain rule is applied term by term to Eqs. (3.7) and (3.8). Equations (3.1) and (3.2) are used to produce the simplifications that result in the first two terms of Eqs. (3.9) and (3.10), respectively. Each of the third terms is very complex and has been simplified as a single term with the special differential

operator defined by Eq. (3.5). A more complete expansion of these terms is provided in [20].

Clearly, $\mathbf{X}''_{j,k}$ is the SO grid-sensitivity term, which is obtained in general (for geometric design variables only) by twice differentiating the mesh-generation code. These SO grid-sensitivity terms vanish in the example problems of the present study because of the linear nature of the particular remesh/grid-sensitivity scheme used. (See Refs. [1, 14].)

In all subsequent discussions, it is assumed that the *complete* Hessian matrix $d^2F_i/d\mathbf{b}^2$ is desired for each output function F_i. The DD.DD method requires *a priori* knowledge of the complete $\mathbf{Q}'$ matrix, which is obtained via NDV solutions of the FO DD system (Eq. (3.2)). (Recall NDV is the number of aerodynamic input variables of interest; NOF is the number of aerodynamic output functions.) Thereafter, a *maximum* of $(\text{NDV})^2$ solutions of the SO system (Eq. (3.10)), is required to determine all $\mathbf{Q}''_{j,k}$. However, if the identity $\mathbf{Q}''_{j,k} = \mathbf{Q}''_{k,j}$ is exploited computationally, then the *minimum* number of SO solutions is $[(\text{NDV})^2 + \text{NDV}]/2$. Thus, the DD.DD method requires a minimum total number of $\text{NDV} + [(\text{NDV})^2 + \text{NDV}]/2$ solutions of very large linear systems.

The coefficient matrices of the FO DD system and the SO DD.DD system are identical. Thus, when these systems are cast in incremental iterative form, both could be solved with the identical approximate left-hand-side (LHS) operator and algorithm that is also used to solve the nonlinear flow equations. The IIM for solving the SO Eq. (3.10) is

$$-\frac{\widetilde{\partial \mathbf{R}}}{\partial \mathbf{Q}}\,\Delta\mathbf{Q}''^{m}_{j,k} = \mathbf{R}''^{m}_{j,k} \tag{3.11}$$

$$\mathbf{Q}''^{m+1}_{j,k} = \mathbf{Q}''^{m}_{j,k} + \Delta\mathbf{Q}''^{m}_{j,k}; \quad m = 1, 2, 3, ..., \tag{3.12}$$

where

$$\mathbf{R}''^{m}_{j,k} = \frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\,\mathbf{Q}''^{m}_{j,k} + \frac{\partial \mathbf{R}}{\partial \mathbf{X}}\,\mathbf{X}''^{m}_{j,k} + \frac{D(\mathbf{R}'_j)}{Db_k}. \tag{3.13}$$

3.1.2. *Method* 2: AV.DD.

An inspection of Eq. (3.3) reveals explicit dependencies on $\mathbf{A}_i$ and $\mathbf{X}'_j$. In Eq. (3.4), however, explicit dependence on $\mathbf{A}_i$ occurs, but not on $\mathbf{X}'_j$. Both equations depend explicitly on $\mathbf{Q}$, $\mathbf{X}$, and $\mathbf{b}$. These complete dependencies are expressed as

$$\frac{dF_i}{db_j} \equiv {}^{\text{A}}F'_{i;j} = {}^{\text{A}}F'_{i;j}(\mathbf{A}_i(\mathbf{b}), \mathbf{X}'_j(\mathbf{b}); \mathbf{Q}(\mathbf{b}), \mathbf{X}(\mathbf{b}), \mathbf{b}), \tag{3.14}$$

$$\mathbf{G}_i = \mathbf{G}_i(\mathbf{A}_i(\mathbf{b}); \mathbf{Q}(\mathbf{b}), \mathbf{X}(\mathbf{b}), \mathbf{b}). \tag{3.15}$$

Differentiation of Eqs. (3.14) and (3.15) with respect to b_k yields, respectively,

$$\frac{d^2F_i}{db_k db_j} = \left(\frac{\partial R}{\partial \mathbf{X}}\,\mathbf{X}'_j + \frac{\partial \mathbf{R}}{\partial b_j}\right)^{\text{T}} \mathbf{A}'_{i;k} + \left(\frac{\partial F_i}{\partial \mathbf{X}} + \mathbf{A}_i^{\text{T}}\frac{\partial \mathbf{R}}{\partial \mathbf{X}}\right)\mathbf{X}''_{j,k} + \frac{D({}^{\text{A}}F'_{i;j})}{Db_k}, \tag{3.16}$$

$$\frac{d\mathbf{G}_i}{db_k} = \left(\frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\right)^{\text{T}} \mathbf{A}'_{i;k} + \frac{D(\mathbf{G}_i)}{Db_k} = \mathbf{0}, \tag{3.17}$$

where $\mathbf{A}'_{i;k} \equiv d\mathbf{A}_i/db_k$.

In the preceding differentiation, the chain rule is applied to Eqs. (3.14) and (3.15). Equations (3.3) and (3.4) are used to produce the simplifications that result in the first two terms of Eq. (3.16) and the first term of Eq. (3.17). Each of the last terms is very complex and has been simplified as a single term with the operator defined by Eq. (3.5). (See Ref. [20] for a more complete expansion of these terms.)

The AV.DD method requires *a priori* knowledge of the complete $\mathbf{Q}'$ and $\mathbf{A}$ matrices, which are obtained via NDV solutions of the FO DD system (Eq. (3.2)) and NOF solutions of the AV system (Eq. (3.4)), respectively. Thereafter, NDV × NOF solutions of the SO system (Eq. (3.17)) are required to determine all $\mathbf{A}'_{i;k}$. The AV.DD method thus requires a total of NDV + NOF + (NDV × NOF) solutions of very large linear systems.

The coefficient matrices of the FO AV system and the SO AV.DD system are identical. Thus, when these systems are cast in incremental iterative form, both could be solved using the identical approximate LHS operator and algorithm (the transpose of that which is also used to solve the nonlinear flow equations). The IIM for solving the SO Eq. (3.17) is

$$-\left(\frac{\widetilde{\partial \mathbf{R}}}{\partial \mathbf{Q}}\right)^{\text{T}} \Delta\mathbf{A}'^{m}_{i;k} = \mathbf{G}'^{m}_{i;k}, \tag{3.18}$$

$$\mathbf{A}'^{m+1}_{i;k} = \mathbf{A}'^{m}_{i;k} + \Delta\mathbf{A}'^{m}_{i;k}; \quad m = 1, 2, 3, ..., \tag{3.19}$$

where

$$\mathbf{G}'^{m}_{i;k} = \left(\frac{\partial \mathbf{R}}{\partial \mathbf{Q}}\right)^{\text{T}} \mathbf{A}'^{m}_{i;k} + \frac{D(\mathbf{G}_i)}{Db_k}. \tag{3.20}$$

3.1.3. *Method* 3: DD.AV

This method is derived by introducing an arbitrary adjoint-variable vector into the DD.DD method to combine Eqs. (3.9) and (3.10). The adjoint-variable vector is speci-

fied so that the resulting coefficient of $\mathbf{Q}''_{j,k}$ vanishes. The resulting DD.AV method

$$\frac{d^2F_i}{db_k db_j} = \left(\frac{\partial F_i}{\partial \mathbf{X}} + \mathbf{A}_i^{\mathrm{T}}\frac{\partial \mathbf{R}}{\partial \mathbf{X}}\right)\mathbf{X}''_{j,k} + \frac{D(^{\mathrm{D}}F'_{i;j})}{Db_k} + \mathbf{A}_i^{\mathrm{T}}\frac{D(\mathbf{R}'_j)}{Db_k}. \tag{3.21}$$

The adjoint-variable vector $\mathbf{A}_i$ in the derivation of Eq. (3.21) is identical to that of the FO AV method and is found by solving the FO Eq. (3.4).

The DD.AV method requires no simultaneous solutions of large systems of linear equations that involve SO terms (in contrast with the previous two schemes). An inspection of Eq. (3.21) reveals that the method requires knowledge of the complete $\mathbf{Q}'$ and $\mathbf{A}$ matrices. (Of course, $\mathbf{Q}'$ and $\mathbf{A}$ are obtained via NDV solutions of the FO DD system (Eq. (3.2)) and NOF solutions of the FO AV system (Eq. (3.4)), respectively.) Thus, the DD.AV method requires a total of NDV + NOF solutions of large simultaneous systems of linear equations, where only the systems for the FO derivatives are solved.

3.1.4. *Method* 4: AV.AV

This method is derived by introducing a new, arbitrary adjoint-variable vector into the AV.DD method to combine Eqs. (3.16) and (3.17). This new adjoint-variable vector is specified so that the resulting coefficient of $\mathbf{A}'_{i;j}$ vanishes. The resulting AV.AV method is

$$\frac{d^2F_i}{db_k db_j} = \left(\frac{\partial F_i}{\partial \mathbf{X}} + \mathbf{A}_i^{\mathrm{T}}\frac{\partial \mathbf{R}}{\partial \mathbf{X}}\right)\mathbf{X}''_{j,k} + \frac{D(^{\mathrm{A}}F'_{i;j})}{Db_k} + (\mathbf{Q}'_j)^{\mathrm{T}}\frac{D(\mathbf{G}_i)}{Db_k}. \tag{3.22}$$

The adjoint-variable vector (which results in the disappearance of terms that involve $\mathbf{A}'_{i;j}$) is seen in the derivation of Eq. (3.22) to be $\mathbf{Q}'_j$, which is found by solving the FO DD (Eq. 3.2)). Detailed demonstration of this result is given in [20].

No computational advantage is associated with the AV.AV method over the DD.AV method. In [20], these two schemes are shown to be term-by-term equivalent. That is,

$$(\mathbf{Q}'_j)^{\mathrm{T}}\frac{D(\mathbf{G}_i)}{Db_k} = \mathbf{A}_i^{\mathrm{T}}\frac{D(\mathbf{R}'_j)}{Db_k} \tag{3.23}$$

$$\frac{D(^{\mathrm{A}}F'_{i;j})}{Db_k} = \frac{D(^{\mathrm{D}}F'_{i;j})}{Db_k}. \tag{3.24}$$

The AV.AV method requires knowledge of the complete $\mathbf{Q}'$ and $\mathbf{A}$ matrices, which are obtained from solving Eqs. (3.2) and (3.4), respectively. Thus, the AV.AV method requires a total of NDV + NOF solutions of large systems of linear equations. Again, only the systems for FO derivatives are solved.

3.1.5. *Discussion*

An analysis was made to determine which of the preceding four methods would be potentially the least costly computationally by considering the *total* number of large simultaneous linear systems that must be solved to calculate the complete Hessian matrix $d^2F_i/d\mathbf{b}^2$ for all F_i. The conclusions of this study are:

(1) Methods (3) and (4) are computationally equivalent and are henceforth known as Method (3/4).

(2) Method (3/4) is unconditionally less costly than Method (2). Therefore, either Method (1) or Method (3/4) should be selected, depending on conclusion (3).

(3) If the equality $\mathbf{Q}''_{j,k} = \mathbf{Q}''_{k,j}$ is fully exploited, then Method (1) is less costly than Method (3/4) when NDV $\times$ (NDV + 1) is less than 2 $\times$ NOF. If the same equality is not exploited, then Method (1) is less costly than Method (3/4) when $(\mathrm{NDV})^2$ is less than NOF.

3.2. Applications of ADIFOR

In this section, various procedures are outlined whereby AD might be applied effectively to assist in computing the SO aerodynamics SDs. Many of the terms in each of the preceding four methods are exceedingly complex. In particular, this applies to the large, complex groups of terms that are symbolized compactly with the operator $D(\ )/Db_k$. Practically speaking, differentiation and coding by hand to construct these terms is impossible, even for the less complicated CFD codes (e.g., 2D Euler codes). Simply stated, without AD the equations for SO aerodynamic sensitivity analysis cannot be constructed.

3.2.1. *Noniterative Applications for Method* (3/4) (DD.AV)/(AV.AV)

Fortunately, ADIFOR is ideally suited for the quick and reliable generation of source code to accurately evaluate the required SO terms without an excessive expansion of the computer memory. For example, the ADIFOR-assisted construction of the SO Method (3) (DD.AV, Eq. (3.21)) involves creation of a source code that evaluates $D(\mathbf{R}'_j)/Db_k$. The source code can be created easily by the straightforward application of ADIFOR to an existing subroutine that evaluates $\mathbf{R}'_j$. This AD application is completely analogous to the AD-assisted creation of source code that evaluates $D(\mathbf{R})/Db_j$ (which is $\mathbf{R}'_j$) from an existing subroutine that evaluates $\mathbf{R}$. (Recall that the creation

of source code via ADIFOR to evaluate $\mathbf{R}'_j$ was an important requirement in the success of the FO ADII scheme.) Similar remarks, of course, apply in the AD-assisted creation of $D({}^{\mathrm{D}}F'_{i,j})/Db_k$ for use in Eq. (3.21). Therefore, the fact that the *a priori* calculation of the FO $\mathbf{Q}'$ matrix is required by all four SO methods (including Method (3)) is somewhat fortuitous, because this ensures that the FO source code to evaluate $\mathbf{R}'_j$ (and ${}^{\mathrm{D}}F'_{i,j}$) will be available for further use in the AD-assisted creation of the SO terms.

Except for the terms $\mathbf{X}''_{j,k}$ and $\mathbf{A}_i$, the remaining terms in the SO Eq. (3.21) are taken as is from the same FO equations that are used to calculate all $\mathbf{Q}'_j$. The term $\mathbf{X}''_{j,k}$ would be produced by twice differentiating the grid-generation code, and all $\mathbf{A}_i$ must be obtained by solving first the FO AV equations. After the required FO equations for the required $\mathbf{Q}'$ and $\mathbf{A}$ are solved, the SO Method (3) becomes a noniterative, computationally efficient scheme for computing SO SDs, where all required SO terms are easily constructed via straightforward applications of ADIFOR to key parts of the existing FO source code. The discussion of SO Method (3) in the preceding two paragraphs is easily extended to the SO Method (4) because, as noted previously, the two methods (Method (3/4)) are equivalent.

The requirement to first solve the FO AV equations is an important consideration for each SO method *except* Method (1) (DD.DD). Unique computational difficulties exist (discussed earlier in greater detail) which are associated with the use of ADIFOR to construct key parts of the AV methods. Thus, these difficulties are transmitted to these SO methods. This concern speaks to SO Method (3/4) directly, because it was concluded earlier that this method is unconditionally more efficient than SO Method (2) (AV.DD).

For the particular combination of NDV = 6 with NOF = 3 (which is applicable to the sample problems of this study), the SO Method (3/4) would be about 3 times less costly than the remaining choice Method (1); this projection is based on the previous discussion, which considers the comparative total number of large linear systems that must be iteratively solved (27 large system solutions for Method (1), compared to 9 for Method (3/4)). The actual implementation of the SO Method (3/4) is not included in the present study but is a topic of ongoing work.

3.2.2. *Black-Box and Incremental Iterative Applications for Method* (1) (DD.DD) *and Method* (2) (AV.DD)

The SO Method (1) has been projected to be less costly than Method (3/4) for certain combinations of NDV and NOF (which have been specified). An advantage unique to the SO Method (1) is that no adjoint-variable equations are to be solved. Thus, in principle, the entire scheme can be constructed from start to finish via the present version of ADIFOR. However, an important disadvantage also exists: in contrast with the SO Method (3/4), large linear systems must be iteratively solved for the SO $\mathbf{Q}''$ terms. Most of the remainder of this section focuses on ADIFOR-assisted black-box (BB) and/or IIM implementations of this SO method (DD.DD). The discussion and resulting methods are analogous to that seen earlier for ADIFOR-assisted implementations of the FO DD scheme.

In principle, SO aerodynamic SDs can be obtained by a BB application of the present version of the AD tool (ADIFOR 1.0) to an AD-enhanced version of the original flow code (called the BB.BB method). However, the BB.BB scheme can also be obtained by applying a future new version of the AD tool (ADIFOR 2.X) to the original flow code. This advanced version of ADIFOR is presently being developed with the new, optional capability of providing SO derivatives. For advanced CFD codes, the BB.BB method may be exceedingly inefficient computationally, for reasons that are discussed subsequently. In addition, the memory requirements could become prohibitively large. Therefore, this approach was abandoned early in the present study.

A convenient symbolic representation of the BB.BB scheme is obtained by differentiating Eq. (2.16) with respect to $\mathbf{b}$ to yield

$$\mathbf{Q}''^{n+1} = \mathbf{Q}''^{n} - \mathbf{P}^{n}\mathbf{R}''^{n} - 2\mathbf{P}'^{n}\mathbf{R}'^{n} - \mathbf{P}''^{n}\mathbf{R}^{n}; \quad n = 1, 2, 3, ... \tag{3.25}$$

In contrast, the IIM for the SO Method (1) (DD.DD) can be represented compactly by combining Eqs. (3.11) and (3.12). The result is (by dropping the subscripts j, k)

$$\mathbf{Q}''^{m+1} = \mathbf{Q}''^{m} - \mathbf{P}\mathbf{R}''^{m}; \quad m = 1, 2, 3, ... \tag{3.26}$$

Clearly, Eqs. (3.25) and (3.26) are symbolically equivalent only at convergence of the flow equations *and* the FO sensitivity equations. Computationally, however, Eqs. (3.25) and (3.26) are by no means equivalent; the potential for greater efficiency is with Eq. (3.26). This potential is seen from the previous discussion of the FO BB application of AD, where Eqs. (2.16) and (2.17) are compared. The potential for computational inefficiency is even greater now than previously for Eq. (2.16) because of the additional presence of the unwanted term $\mathbf{P}''^{n}\mathbf{R}^{n}$ in Eq. (3.25) and because the unwanted term $2\mathbf{P}'^{n}\mathbf{R}'^{n}$ represents a double computational evaluation of $\mathbf{P}'^{n}\mathbf{R}'^{n}$.

It is suggested that ADIFOR can be used to assist in the creation of the potentially efficient scheme represented by Eq. (3.26) in a manner that is completely analogous to

the implementation of the AD-assisted FO method, ADII. Thus, the resulting method, known here as the ADII.SO scheme, involves the AD of all terms on the right-hand side (RHS) only of Eq. (2.9), which is the residual $\mathbf{R}'$, of the linear FO sensitivity equations (Eq. (2.4)). These differentiated terms are then judiciously assembled for efficient operation on the RHS of the identical approximate LHS operator and algorithm that were also used to efficiently solve the nonlinear flow and the linear FO sensitivity equations.

The ADII.SO scheme is potentially the most efficient implementation of the SO Method (1) (DD.DD) that is feasible, because hand-differentiated (HD) construction of the scheme is too complex to be practical. It represents the only true SO IIM which, in principle, can be constructed in total with the present various of ADIFOR. (Recall that the SO Methods (2) and (3/4) require construction of the FO AV equations, in which the construction of some terms is not feasible via ADIFOR.) Actual implementation of the ADII.SO scheme is not included in the present study, but it is a topic of ongoing work.

An alternative AD-assisted SO strategy is the BB application of ADIFOR to the scheme represented by Eq. (2.17). After differentiation with respect to $\mathbf{b}$, the representation becomes

$$\mathbf{Q}''^{m+1} = \mathbf{Q}''^{m} - \mathbf{P}\mathbf{R}''^{m} - \mathbf{P}'\mathbf{R}'^{m}; \quad m = 1, 2, 3, ... \tag{3.27}$$

From the previous discussions of the analogous FO Eqs. (2.16) and (2.17), Eq. (3.27) clearly represents a significant improvement over Eq. (3.25) with respect to computational efficiency. However, Eq. (3.27) will also surely be less efficient than the true SO IIM (Eq. (3.26)). Symbolically (but not computationally), the two are equivalent only at convergence when $\mathbf{R}'$ vanishes.

The preceding method is constructed by the BB AD of the source code which computes FO SDs via an IIM (except the AVII scheme, for the present application). In principle, this code could be the source code for the previously discussed ADII scheme. The source code for the HD DDII method was selected in the sample problems, however, because it was more efficient. Results of the SO method are known here as the DDII.BB scheme; by extension of this terminology, the FO DDIITC and DDIITCFR methods become the SO DDIITC.BB and DDIITCFR.BB schemes, respectively. By considering the latter two schemes, it is interesting to note that part of the original source code differentiated by ADIFOR is that which calculates the turbulence-modeling correction terms (i.e., those terms that were originally created by ADIFOR are then successfully differentiated by ADIFOR). This is the only example in the present work where this was actually attempted.

For completeness, a final SO scheme is introduced here, called the AVII.BB method, which can be constructed via the BB application of ADIFOR to the source code for the FO AVII scheme. The AVII.BB scheme would yield inaccurate SO SDs for the turbulent example problem because the full AD-generated correction of the turbulence-modeling terms could not be added to the FO AVII scheme. Symbolically, the AVII.BB scheme is represented by combining Eqs. (2.12) and (2.13) and differentiating with respect to $\mathbf{b}$ to yield

$$\mathbf{A}'^{m+1} = \mathbf{A}'^{m} - (\mathbf{P})^{\mathrm{T}}\mathbf{G}'^{m} - (\mathbf{P}')^{\mathrm{T}}\mathbf{G}^{m}; \quad m = 1, 2, 3, ... \tag{3.28}$$

In contrast, a similar representation of the IIM for the SO Method (2) (AV.DD) is (recall Eqs. (3.18) and (3.19))

$$\mathbf{A}'^{m+1} = \mathbf{A}'^{m} - (\mathbf{P})^{\mathrm{T}}\mathbf{G}'^{m}; \quad m = 1, 2, 3, ... \tag{3.29}$$

Comparisons of the two schemes represented by Eqs. (3.28) and (3.29) yield conclusions with respect to computational efficiency that are analogous to those made previously for other FO and SO methods that employ BB applications of ADIFOR.

The AVII.BB scheme, which essentially is an AD-assisted form of the SO Method (2) (AV.DD), is not pursued further in this study because the SO Method (2) is always less efficient than the SO Method (3/4) (DD.AV/AV.AV) as demonstrated earlier.

3.3. Computational Results: SO SDs

The FO SD results for the previous sample problems are extended here to include calculation of the complete Hessian matrices $d^2C_{\mathrm{L}}/d\mathbf{b}^2$, $d^2C_{\mathrm{D}}/d\mathbf{b}^2$, and $d^2C_{\mathrm{M}}/d\mathbf{b}^2$. These SO SDs are calculated with different methods, and the results are compared on the basis of accuracy and computational time and memory. For the methods tested, considerable CW could have been saved by taking advantage computationally of the symmetry of the Hessian matrices. However, this was not done here in order to exploit this symmetry as an additional internal accuracy check.

3.3.1. *Accuracy Comparisons*

The SO SD Hessian matrices are calculated for both the laminar and turbulent example problems with two basic methods known here as QA.CD and DDII.BB. The latter method has been described previously herein. More pre-

cisely specified, the DDII.BB scheme is applied to the laminar example, and the DDIITC.BB scheme is applied to the turbulent example.

The QA.CD method can be described as a hybrid quasi-analytical/central finite-difference scheme, which is implemented in the following manner:

(1) Recall that, for the FO CD method, 12 machine-zero-converged solutions of the nonlinear flow equations were generated: two perturbations for each design variable, a forward and a backward perturbation of $\Delta b_j = \pm 5.0\text{E} - 6 \times b_j$.

(2) For each of the 12 perturbed nonlinear flow solutions, the complete set of FO SDs were calculated using the FO HD QA (i.e., first-order, hand-differentiated, quasi-analytical) code. Specifically, the DDII and DDIITCFR methods were used for the laminar and turbulent examples, respectively. A reduction of 10 OM or better in the error of each linear system is specified; 72 linear systems are solved for those FO SDs.

(3) The complete SO Hessian matrices were calculated with central finite-difference approximations using the QA FO derivatives calculated via that of the preceding discussion.

Thus, the QA.CD method for the SO SDs performs the first differentiation exactly, and the second differentiation via a CD approximation.

The SO SDs that were calculated via the QA.CD method are presented in Tables B.1a and B.1b (in Appendix B) for the laminar and turbulent sample problems, respectively. In these results, the actual numerical values of the derivatives are given for the main-diagonal and above-main-diagonal terms. The results presented for the below-main-diagonal terms are SD ratios, where each result shown has been divided by its equivalent above-main-diagonal term. (Clearly this calculation is an internal accuracy check of all the off-main-diagonal terms.) As expected, these below-main-diagonal SD ratios are all unity to at least three and usually four or more significant digits. In these tables, the quasi-analytical first differentiation is with respect to b_j (shown horizontally), followed by the CD approximate second differentiation with respect to b_k (shown vertically).

The SO results that were generated via the DDII.BB method are shown for the laminar example in Table B.2a. Similar results with the DDIITC.BB scheme for the turbulent example are shown in Table B.2b. All of these results are given as SO SD ratios, where the numerical value of each result has been normalized by the numerical value of the respective term calculated via the QA.CD method. These tables clearly show the excellent agreement among the results obtained by these two methods, as expected.

Prior to execution of the AD-enhanced code for the DDII.BB-type methods, the FO solution for the complete $\mathbf{Q}'$ matrix should be calculated first as input for the subsequent SO SD calculations. This initial $\mathbf{Q}'$ is calculated with the DDII-type methods (i.e., the original code from which the AD-enhanced SO code was created). An initial total of six linear systems is solved; a reduction of 10 OM in the error of each was specified. Subsequent execution of the AD-enhanced code produces the solutions of 36 linear systems for the complete SO SDs $\mathbf{Q}''$; an average reduction of 8 OM in the error of each system is specified in this case. These tight convergence tolerances should ensure accurate results, but at great expense in the computational timing comparisons presented subsequently.

In addition to the 36 solutions for the SO terms, execution of the AD-enhanced code results in the solution of six linear systems for the FO SDs $\mathbf{Q}'$. This is a computationally wasteful, repeated solution for these terms, but their calculation cannot be avoided here because it is the function of the original code. The initial solution for $\mathbf{Q}'$ as input to the AD-enhanced code, discussed previously, can be avoided under certain conditions. However, such avoidance may not necessarily be efficient with respect to convergence rates and computer memory. This implementation is always possible and is also straightforward to invoke if the original code solves the FO sensitivity equations *concurrently*; the initial input of the complete $\mathbf{Q}'$ matrix is replaced directly by the dynamic calculation of these same terms, because they also evolve concurrently during the SO SD calculations. The original code used here solves the FO sensitivity equations sequentially, however. Consequently, avoidance of the initial $\mathbf{Q}'$ solution becomes a more complicated issue; in some cases where the complete Hessian matrices are not calculated, it is not even possible.

The complete set of terms on the main diagonal of the Hessian matrices can be calculated via the CD.CD method (i.e., the pure central finite-difference approximation of these terms) with the initial flow solution and the 12 perturbed flow solutions calculated previously for application in the FO CD and SO QA.CD schemes. An approximation of the complete set of cross-derivative terms in this manner is also possible, of course, but it would require many additional perturbed flow solutions. Comparison of the results for these terms from the applications of the CD.CD and QA.CD methods is presented in Tables B.3a and B.3b for the laminar and turbulent examples, respectively. The results shown here are SO SD ratios, where the reported CD.CD calculations have been normalized by the respective QA.CD results of Tables B.1a and B.2b.

The two tables, B.3a and B.3b, show poor agreement among the results of these two methods. This discrepancy is attributed to inaccuracy in the CD.CD calculations. It occurs whenever a finite-difference perturbation is too

small, the consequence of which is a necessity for accuracy in the function evaluations that is beyond the capability of the finite-arithmetic machine. With the present perturbation $\Delta b_j = \pm 5.0\text{E} - 6 \times b_j$, this machine limitation was not exceeded for the accuracy desired in the FO CD results. For the SO CD.CD results, $(\Delta b_j)^2$ is now the divisor in the finite-difference expressions. Compared to the FO CD approximations, six or more additional significant digits of accuracy are required in the function evaluations to achieve the first digit of accuracy in the SO CD.CD results (with this perturbation). Of course, significant competing accuracy considerations emerge as the perturbation size is progressively increased. Larger perturbations were not tried for this demonstration because of the inordinate amount of CPU time required to produce SO derivatives, and the lack of any guarantee that the CD.CD results would be any better. Therefore, when the simple finite-difference method is applied, the selection of a good numerical perturbation size becomes progressively more difficult for higher-order derivatives.

3.3.2. *Computational Time and Memory Comparisons*

In this section, the SO methods of the previous section are further subdivided in a manner similar to that which was done previously for the various FO methods that were tested. As before, these subdivisions have no impact on the SDs that are calculated but can greatly impact the computational time and memory requirements.

The QA.CD method is subdivided into two methods, depending on whether or not the FJ option (previously discussed) is activated. This refers to the first phase of the method only, where the 12 nonlinear flow solutions are obtained. When this option is active, the method becomes the QA.CD(FJ) scheme; the method remains the simple QA.CD scheme when the option is inactive.

The DDII.BB and DDIITC.BB methods, including the previously described DDIITCFR.BB efficiency option that is associated with the latter method, are subdivided into the following six schemes: DDII.BB(6), DDII.BB(5 + 1), DDIITC.BB(6), DDIITC.BB(5 + 1), DDIITCFR.BB(6), and DDIITCFR.BB(5 + 1). These subdivisions are made based on different applications of the parameter gp in the AD-enhanced code, as described previously for the FO results.

Comparisons of total CPU times are shown in Table B.4 for the laminar and turbulent sample problems. All reported timings do not include the cost of the initial flow solution. For the QA.CD and QA.CD(FJ) schemes, the cost for the 12 perturbed nonlinear flow solutions and the 72 linear system derivative solutions is included in the reported timings. For the AD-assisted DDII.BB-type schemes, the computational cost is included for initially solving the FO equations for all $\mathbf{Q}'$. (Recall that this FO solution is required input at the outset of the execution of the AD-enhanced code for the SO SDs.) Table B.5 shows comparisons (excluding the QA.CD and QA.CD(FJ) methods) of CPU time per iteration per SO linear system solved. The cost of the initial FO solution for $\mathbf{Q}'$ is *not* included in the results of this table. As described previously for the FO results, the individual effects of gp = 5 and gp = 1 are shown in Table B.5. Finally, the total computer memory requirements for the different methods are compared in Table B.6.

3.4. Conclusions: SO SDs

Conclusions based on the calculations for the SO SDs are:

(1) The calculation of SO aerodynamic SDs by a pure finite-difference scheme is much more sensitive to the selection of a proper perturbation size than is the calculation of the FO SDs. This difficulty is noted in addition to the extreme computational cost of the method, particularly if the complete Hessian matrices (with cross-derivatives) are computed. These difficulties for the SO derivatives can be mitigated to a large extent if the FO derivatives, at least, are calculated via one of the HD or AD methods discussed previously.

(2) The good agreement that is seen in the AD-assisted SO results (when compared with the results from the QA.CD method) confirms that ADIFOR can be successfully used as a tool to construct and implement the SO methods from source code that evaluates the FO derivatives. In principle, the source code for FO derivatives can be constructed either in whole or in part by either hand differentiation or AD. Without ADIFOR, the SO schemes could not be implemented even for simple CFD codes. Construction of the source code by hand would not be feasible to evaluate the extremely large number of very complex SO terms that are involved.

(3) The previously discussed limitations of ADIFOR when the FO AVII methods are constructed carries over at least indirectly in all the SO methods except Method (1) (DD.DD) because the other three SO methods require the FO AV equations to be solved. This requirement is significant because Method (3/4) is potentially by far the most efficient method for particular combinations of NDV and NOF.

(4) As expected, the computational results, although accurate, were very costly in terms of CPU time and (for the DDII.BB-type methods) computer memory. The possibility exists that these computational costs can be significantly reduced, however, as described in the subsequent conclusions. This possibility is also a topic of ongoing study.

(5) For the particular combination of NDV and NOF studies here, Method (3/4) is potentially far more efficient than the methods actually tested. It is projected to cost only slightly more than the cost of solving the large systems of equations for the FO DD and AV schemes—a total of only nine large systems. This cost, taken from the HD FO results, was a total of only 583 and 1014 CPU seconds for the laminar and turbulent-flow examples, respectively. The results for the turbulent-flow example would have inaccuracies traced to the FO AVII scheme (which could not be corrected via AD for all of the turbulence-modeling terms).

(6) In contrast with the nine large system solutions that would be required for Method (3/4), the DDII.BB type of implementation of Method (1) included the solution of 42 large systems—six for FO derivatives and 36 relatively inefficient (as discussed subsequently) solutions for the SO derivatives. Note that by simply taking advantage computationally of the symmetry of the Hessian matrices, the number of solutions for SO derivatives could be reduced from 36 to 21.

(7) As discussed previously, the DDII.BB-type approach is not the most efficient implementation of Method (1) because it is not a true IIM. It is believed that the true SO IIM implementation, known here as the ADII.SO approach, would be the most efficient feasible implementation of Method (1). The improvement in CPU time and memory requirements that results from the ADII.SO implementation (compared to the DDII.BB-type approach) is projected to be roughly proportional to those improvements seen in a comparison of the two FO methods ADII and BBFJ (that is, ADII.SO could be possibly 40%–70% faster than DDII.BB).

(8) The computational efficiency, in terms of CPU time and memory, is affected greatly by the selection of the parameter gp in the AD-assisted SO schemes. This effect is similar to that noted previously in the FO results.

4. SUMMARY AND FINAL CONCLUSIONS

A number of different approaches for computing first-order (FO) aerodynamic sensitivity derivatives (SDs) have been tested, and the results have been compared on the basis of accuracy, computational time, and computer memory requirements. The methods represent a broad spectrum of choices: finite-difference methods, hand-differentiated (HD) incremental iterative method (IIM) schemes, and black-box (BB) automatic differentiation (AD) methods. In addition, combinations and variations of these are possible.

The automatically differentiated incremental iterative (ADII) scheme stands out as a very well-rounded combination of the best features of the other methods. Although it is not yet as efficient as the HD schemes, it is more efficient than all other methods tested—substantially more efficient than the simple BB and central finite-difference (CD) approaches. The ADII method is not quite as easy to implement as the BB approach. However, when compared with the HD methods, the ADII scheme can be implemented quickly, reliably, and with very accurate results, even for very complex computational fluid dynamics (CFD) codes. The ADII scheme also requires less expansion of computer memory than some of the other AD-assisted methods. When compared with the best CD method, the most efficient implementation of the ADII scheme improved computational efficiency by a factor of approximately 3.6 for the laminar example and 1.7 for the turbulent example. Additionally, the memory increase was about 1.6 times that of the original flow code. The computational penalty which is associated with the AD-assisted differentiation of the turbulence-modeling terms is significant and disproportionately large for all applicable methods; this requires additional examination for possible improvements.

A complete procedure has been described by which the discrete second-order (SO) aerodynamic SDs can be calculated from modern CFD codes. The assistance of AD is required for the implementation, because of the extremely large number of very complex terms that are required. Initially, four SO methods were presented. This selection was then reduced effectively to only two, because SO Methods (3) and (4) were shown to be equivalent and unconditionally more efficient than Method (2). Thereafter, the choice between either Method (1) or Method (3/4) is dependent upon the relative size of NDV and NOF (i.e., number of design variables and number of output functions, respectively); a large NOF favors Method (1). The specific criterion to be used for this selection was defined herein.

The SO Method (3/4) requires that the large systems for the FO DD *and* AV schemes are solved first. Thereafter, the computation of the SO SDs is noniterative (i.e., additional simultaneous solutions of large systems are not needed). Method (1) requires that the large systems for the FO DD scheme are solved first; the FO AV systems are not solved, which can be a significant advantage over Method (3/4) for reasons that have been detailed herein. Thereafter, the SO SDs are computed by solving large systems for SO terms. However, these equations and the efficient solution methods for them are analogous to those for the FO DD method. The computational results for the SO SDs were highly accurate, which confirms that ADIFOR is a reliable tool in constructing these SO methods. Significantly improved results with respect to computational efficiency for SO SDs are expected in the future from ongoing work.

APPENDIX A: TABLES FOR FIRST-ORDER RESULTS

TABLE A.1

First-Order Sensitivity Derivatives and Sensitivity-Derivative Ratios; Laminar Example

Solution method	Design variable b_j	$\frac{dC_L}{d\mathbf{b}}$	$\frac{dC_D}{d\mathbf{b}}$	$\frac{dC_M}{d\mathbf{b}}$
Central differences method (CD)	T	−1.392E + 00	+2.019E − 01	+1.805E − 01
SD_{CD}	C	+6.583E + 00	+7.583E − 02	−2.240E + 00
	L	−1.154E − 02	+5.540E − 05	−2.122E − 02
	α	+6.122E + 00	+9.181E − 02	−3.166E − 02
	M_∞	+5.438E − 03	+1.628E − 02	−4.732E − 03
	Re	+5.958E − 06	−4.912E − 06	−6.563E − 07
Direct differentiation incremental iterative method (DDII)	T	1.0000	1.0000	1.0000
	C	1.0000	1.0000	1.0000
$\frac{SD_{DDII}}{SD_{CD}}$	L	1.0000	1.0009	1.0000
	α	1.0000	1.0000	1.0004
	M_∞	1.0000	1.0000	1.0000
	Re	1.0000	1.0000	1.0000
Adjoint-variable incremental iterative method (AVII)	T	1.0000	1.0000	1.0000
	C	1.0000	1.0000	1.0000
$\frac{SD_{AVII}}{SD_{CD}}$	L	1.0000	1.0009	1.0000
	α	1.0000	1.0000	1.0004
	M_∞	1.0000	1.0000	1.0000
	Re	1.0000	1.0000	1.0000
Automatic differentiation in incremental iterative form (ADII)	T	1.0000	1.0000	1.0000
	C	1.0000	1.0000	1.0000
$\frac{SD_{ADII}}{SD_{CD}}$	L	1.0000	1.0009	1.0000
	α	1.0000	1.0000	1.0004
	M_∞	1.0000	1.0000	1.0000
	Re	1.0000	1.0000	1.0000
Automatic differentiation, "black box" method (BB)	T	1.0000	1.0000	1.0000
	C	1.0000	1.0000	1.0000
$\frac{SD_{BB}}{SD_{CD}}$	L	1.0000	1.0009	1.0000
	α	1.0000	1.0000	1.0004
	M_∞	1.0001	1.0000	1.0000
	Re	1.0000	1.0000	1.0000

TABLE A.2

First-Order Sensitivity Derivatives and Sensitivity-Derivative Ratios; Turbulent Example

Solution method	Design variable b_j	$\frac{dC_L}{d\mathbf{b}}$	$\frac{dC_D}{d\mathbf{b}}$	$\frac{dC_M}{d\mathbf{b}}$
Central differences method (CD)	T	+7.919E − 01	+2.744E − 01	−4.153E − 01
SD_{CD}	C	+2.063E + 01	+6.776E − 01	−5.770E + 00
	L	+1.108E − 01	−1.174E − 02	−5.350E − 02
	α	+1.300E + 01	+4.346E − 01	−6.328E − 01
	M_∞	+2.040E + 00	+1.969E − 01	−5.972E − 01
	Re	−1.185E − 09	−2.829E − 10	+1.497E − 10
Direct differentiation **and** adjoint-variable incremental iterative methods, uncorrected (DDII and AVII)	T	0.2874	0.9672	0.7523
	C	0.9415	0.9609	0.9560
$\frac{SD_{DDII}}{SD_{CD}}$ and $\frac{SD_{AVII}}{SD_{CD}}$	L	1.2077	0.9806	1.0447
	α	0.9221	0.9663	0.7388
	M_∞	0.8690	0.9754	0.9093
	Re	−3.4966	1.7251	−2.9367
Direct differentiation incremental iterative method, corrected (DDIITC)	T	1.0000	1.0000	1.0000
	C	1.0000	1.0000	1.0000
$\frac{SD_{DDIITC}}{SD_{CD}}$	L	1.0000	1.0000	1.0000
	α	1.0000	1.0000	1.0000
	M_∞	1.0000	1.0000	1.0000
	Re	1.0000	1.0000	1.0000
Automatic differentiation in incremental iterative form (ADII)	T	1.0000	1.0000	1.0000
$\frac{SD_{ADII}}{SD_{CD}}$	C	1.0000	1.0000	1.0000
	L	1.0000	1.0000	1.0000
	α	1.0000	1.0000	1.0000
	M_∞	1.0000	1.0000	1.0000
	Re	1.0000	1.0000	1.0000
Automatic differentiation, "black box" method (BB)	T	1.0000	1.0000	1.0000
$\frac{SD_{BB}}{SD_{CD}}$	C	1.0000	1.0000	1.0000
	L	1.0000	1.0000	1.0000
	α	1.0000	1.0000	1.0000
	M_∞	1.0000	1.0000	1.0000
	Re	1.0000	1.0000	1.0000

TABLE A.3

Computational Timing Comparisons: Total CPU Time (Seconds)

Method	Laminar	Turbulent
CD	9,800[a]	12,000[a]
CDFJ	2,962	4,057
AVII	324	300
DDII	259	590
DDIITC	Not applicable	2,000[a]
DDIITCFR	Not applicable	714
ADII(6)	1,722	4,879
ADII(5 + 1)	820[a]	2,300[a]
BB(6)	14,000[a]	27,000[a]
BB(5 + 1)	12,000[a]	24,000[a]
BBFJ(6)	2,696	6,964
BBFJ(5 + 1)	1,400[a]	3,600[a]

[a] Projected result based on timings from Table A.4.

TABLE A.4

Computational Timing Comparisons: CPU Time (Seconds)/Iteration/Linear System Solved

Method	Laminar	Turbulent
AVII	0.06196	0.06293
DDII	0.06325	0.06314
DDIITC	Not applicable	0.2285
DDIITCFR	Not applicable	0.07996
ADII(6)	0.4196	0.5432
ADII(5)\ADII(1)	0.1705\0.3453	0.2096\0.5107
BB(6)	3.448	3.570
BB(5)\BB(1)	2.033\8.010	2.050\8.096
BBFJ(6)	0.7694	0.9046
BBFJ(5)\BBFJ(1)	0.3039\0.8827	0.3469\1.046

TABLE A.5

Total Computer Memory Comparisons

Method	Total memory (Mw)
CD, CDFJ	5.27
DDII, AVII, DDIITC, DDIITCFR	7.39
ADII(6)	9.07
ADII(5)\ADII(1)	8.36\5.09
BB(6), BBFJ(6)	34.65
BB(5), BBFJ(5)\BB(1), BBFJ(1)	29.94\10.27

APPENDIX B: TABLES FOR SECOND-ORDER RESULTS

TABLE B.1a

Second-Order Sensitivity Derivatives and Sensitivity-Derivative Ratios: QA.CD Method; Laminar Example

	$b_k \backslash b_j$	T	C	L	α	M_∞	Re
$\frac{d^2C_L}{d\mathbf{b}_k d\mathbf{b}_j}$	T	−2.280E + 01	−8.390E + 01	−6.579E − 01	−3.661E + 01	−4.728E + 00	+3.752E − 06
SD	C	1.0000	−6.708E + 01	+1.029E − 01	−3.048E + 01	−5.473E + 00	+3.680E − 04
SD ratios	L	1.0000	1.0000	−1.079E − 02	+1.803E − 01	−1.351E − 01	+4.409E − 06
	α	1.0000	1.0000	1.0000	−2.376E + 01	+1.700E + 00	+1.043E − 04
	M_∞	1.0000	1.0000	1.0000	1.0000	−1.222E − 01	+7.902E − 06
	Re	1.0001	1.0000	1.0000	1.0000	1.0000	−1.446E − 09
$\frac{d^2C_D}{d\mathbf{b}_k d\mathbf{b}_j}$	T	+1.180E + 00	+1.017E + 00	−2.651E − 02	−4.182E − 01	+1.690E − 01	−5.236E − 06
SD	C	1.0000	+9.983E + 00	−6.234E − 02	−1.352E + 00	+2.412E − 01	−2.481E − 06
SD ratios	L	1.0000	1.0000	+4.952E − 03	+3.478E − 02	−3.804E − 03	+6.470E − 08
	α	1.0000	1.0000	1.0001	+5.606E + 00	+9.573E − 02	+2.326E − 06
	M_∞	1.0000	1.0000	1.0000	1.0001	+4.447E + 02	−3.945E − 07
	Re	1.0000	1.0001	1.0000	1.0000	1.0000	+1.606E − 09
$\frac{d^2C_M}{d\mathbf{b}_k d\mathbf{b}_j}$	T	+2.965E + 00	+1.269E + 01	+1.907E − 01	+3.047E + 00	+6.144E − 01	+4.748E − 07
SD	C	1.0000	+8.244E + 00	+1.233E − 02	+3.545E + 00	−6.423E − 01	−5.805E − 05
SD ratios	L	1.0000	1.0000	−1.076E − 03	+3.947E − 02	+1.459E − 02	−1.481E − 06
	α	1.0000	1.0000	1.0000	+2.206E + 00	+2.385E − 01	+6.425E − 07
	M_∞	1.0000	1.0000	1.0000	0.9999	−3.726E − 02	−2.892E − 07
	Re	1.0000	1.0000	1.0000	0.9999	0.9998	+1.547E − 10

TABLE B.1b

Second-Order Sensitivity Derivatives and Sensitivity-Derivative Ratios: QA.CD Method; Turbulent Example

	$b_k \backslash b_j$	T	C	L	α	M_∞	Re
$\frac{d^2C_L}{d\mathbf{b}_k d\mathbf{b}_j}$	T	−1.587E + 02	−5.480E + 02	−5.168E − 01	−1.883E + 02	−1.266E + 02	−1.664E − 09
SD	C	1.0000	−1.780E + 03	−2.415E + 00	−6.383E + 02	−4.057E + 02	−3.025E − 08
SD ratios	L	1.0000	1.0000	−5.723E − 02	+3.011E − 01	−7.414E − 01	−1.044E − 09
	α	1.0000	1.0000	1.0000	−1.772E + 02	−1.068E + 02	−1.355E − 08
	M_∞	1.0000	1.0000	1.0000	1.0000	−1.020E + 02	+6.374E − 09
	Re	1.0000	1.0000	1.0000	1.0000	1.0000	+1.512E − 16
$\frac{d^2C_D}{d\mathbf{b}_k d\mathbf{b}_j}$	T	+7.245E + 00	+1.763E + 01	−2.782E − 01	+1.278E + 01	+6.118E + 00	−7.085E − 10
SD	C	1.0000	+6.397E + 01	−6.362E − 01	+3.285E + 01	+1.730E + 01	−3.797E − 09
SD ratios	L	1.0000	1.0000	+1.683E − 02	−5.967E − 01	−2.184E − 01	+1.675E − 11
	α	1.0000	1.0000	1.0000	+2.879E + 01	+1.235E + 01	−1.916E − 09
	M_∞	1.0000	1.0000	1.0000	1.0000	+5.029E + 00	−1.054E − 09
	Re	1.0000	1.0000	1.0000	1.0000	1.0000	+1.104E − 16
$\frac{d^2C_M}{d\mathbf{b}_k d\mathbf{b}_j}$	T	+4.993E + 00	+1.492E + 01	+3.707E − 01	−1.283E + 01	−5.695E + 00	+3.513E − 09
SD	C	1.0000	−3.965E + 00	+8.127E − 01	−4.331E + 01	−2.995E + 01	+1.753E − 08
SD ratios	L	1.0000	1.0000	−3.088E − 03	+1.308E + 00	+3.977E − 01	+2.034E − 10
	α	1.0000	1.0000	1.0000	−3.227E + 01	−2.348E + 01	+4.208E − 09
	M_∞	1.0000	1.0000	1.0000	1.0000	−1.173E + 01	+2.259E − 09
	Re	1.0000	1.0000	1.0000	1.0000	1.0000	−1.668E − 17

TABLE B.2a

Second-Order Sensitivity-Derivative Ratios: DDII.BB Method, Normalized by Results from the QA.CD Method;
Laminar Example

	$b_k \backslash b_j$	T	C	L	α	M_∞	Re
$\frac{d^2C_L}{d\mathbf{b}_k d\mathbf{b}_j}$	T	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	C	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
SO	L	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
SD	α	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
ratios	M_∞	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Re	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
$\frac{d^2C_D}{d\mathbf{b}_k d\mathbf{b}_j}$	T	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	C	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
SO	L	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
SD	α	1.0000	1.0000	1.0000	1.0000	1.0001	1.0000
ratios	M_∞	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Re	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
$\frac{d^2C_M}{d\mathbf{b}_k d\mathbf{b}_j}$	T	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	C	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
SO	L	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
SD	α	1.0000	1.0000	1.0000	1.0000	0.9999	0.9999
ratios	M_∞	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Re	1.0000	1.0000	1.0000	1.0000	1.0002	1.0000

TABLE B.2b

Second-Order Sensitivity-Derivative Ratios: DDIITC.BB Method, Normalized by Results from the QA.CD Method;
Turbulent Example

	$b_k \backslash b_j$	T	C	L	α	M_∞	Re
$\frac{d^2C_L}{d\mathbf{b}_k d\mathbf{b}_j}$	T	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	C	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
SO	L	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
SD	α	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
ratios	M_∞	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Re	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
$\frac{d^2C_D}{d\mathbf{b}_k d\mathbf{b}_j}$	T	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	C	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
SO	L	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
SD	α	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
ratios	M_∞	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Re	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
$\frac{d^2C_M}{d\mathbf{b}_k d\mathbf{b}_j}$	T	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	C	1.0000	0.9999	1.0000	1.0000	1.0000	1.0000
SO	L	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
SD	α	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
ratios	M_∞	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Re	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

TABLE B.3a

Second-Order Sensitivity-Derivative Ratios; CD.CD Method Normalized by Results from the QA.CD Method; Main-Diagonal Terms Only; Laminar Example

	T	C	L	α	M_∞	Re
$\frac{d^2C_L}{d\mathbf{b}_j^2}$	−0.7096	−18.31	−71.90	−18.78	+2.138	−3.023
$\frac{d^2C_D}{d\mathbf{b}_j^2}$	+1.803	+4.341	+5.158	+3.027	+1.214	+1.095
$\frac{d^2C_M}{d\mathbf{b}_j^2}$	+1.017	+5.160	−18.00	+0.6872	+0.9608	+0.6613

TABLE B.3b

Second-Order Sensitivity-Derivative Ratios; CD.CD Method Normalized by Results from the QA.CD Method; Main-Diagonal Terms Only; Turbulent Example

	T	C	L	α	M_∞	Re
$\frac{d^2C_L}{d\mathbf{b}_j^2}$	+0.8072	+0.5389	−12.55	−1.274	+1.000	+31.81
$\frac{d^2C_D}{d\mathbf{b}_j^2}$	+1.028	+0.8928	+1.390	+1.072	+0.9992	+1.353
$\frac{d^2C_M}{d\mathbf{b}_j^2}$	+1.569	−35.16	−12.87	+0.2042	+0.9966	−17.68

TABLE B.4

Computational Timing Comparisons: Total CPU Time (Seconds)

Method	Laminar	Turbulent
QA.CD	14,000[a]	23,000[a]
QA.CD(FJ)	7,018	14,941
DDII.BB(6)	14,543	Not tested
DDII.BB(5 + 1)	4,300[a]	Not tested
DDIITC.BB(6)	Not applicable	64,000[a]
DDIITC.BB(5 + 1)	Not applicable	51,000[a]
DDIITCFR.BB(6)	Not applicable	33,899
DDIITCFR.BB(5 + 1)	Not applicable	14,000[a]

[a] Projected result based on timings from Table B.5.

TABLE B.5

Computational Timing Comparisons: CPU Time (Seconds)/Iteration/Linear System Solved

Method	Laminar	Turbulent
DDII.BB(6)	0.57562	Not tested
DDII.BB(5)\(1)	0.1409\0.2675	Not tested
DDIITC.BB(6)	Not applicable	1.2295
DDIITC.BB(5)\(1)	Not applicable	0.6630\2.5500
DDIITCFR.BB(6)	Not applicable	0.6474
DDIITCFR.BB(5)\(1)	Not applicable	0.2016\0.5216

TABLE B.6

Total Computer Memory Comparisons

Method	Total memory (Mw)
QA.CD, QA.CD(FJ)	7.39
DDII.BB(6), DDIITC.BB(6), DDIITCFR.BB(6)	48.85
DDII.BB(5)\(1), DDIITC.BB(5)\(1), DDIITCFR.BB(5)\(1)	42.38\14.55

APPENDIX C: TABLES OF ACRONYMS

TABLE C.1

General Terms

2D	Two-dimensional
3D	Three-dimensional
AD	Automatic differentiation
ADIFOR	Automatic differentiation of Fortran (software tool)
CFD	Computational fluid dynamics
CPU	Central processing unit (of computer)
CW	Computational work (CPU time used)
FJ	Frozen Jacobian (option for fast convergence of a CFD solution)
FO	First-order (sensitivity derivatives)
FR	Indicates that the turbulence correction terms are frozen for a specified number of iterations
gp	Parameter in all ADIFOR-generated FORTRAN source code governing the length of do-loops to calculate a set of derivatives
HD	Hand-differentiated (and hand-coded, as opposed to the finite-difference approach or automatic differentiation)
IIM	Incremental iterative method (as opposed to standard methods for solution of linear matrix equations)
LU	Lower-upper (factorization)
LHS	Left-hand side (of an equation)
NDV	Number of aerodynamic design (input) variables
NOF	Number of aerodynamic output functions
OM	Order of magnitude (reduction of error)
QA	Quasi-analytical (differentiation of discretized equations by calculus, or ''analytically,'' and by hand)
RHS	Right-hand side (of an equation)
SD(s)	Sensitivity derivative(s)
SO	Second-order (sensitivity derivatives)
TLNS	Thin-layer Navier-Stokes (equations or code)
TC	Indicates that AD-generated turbulence-correction terms are included

TABLE C.2

First-Order Formulations and Solution Strategies

ADII	Automatic differentiation in incremental iterative form
AV	Adjoint-variable (formulation for derivative equations)
AVII	Adjoint-variable incremental iterative (strategy)
AVIITC	AVII with AD-generated turbulence-modeling correction (not currently possible)
BB	Black-Box (straightforward application of AD, as opposed to an application in incremental iterative form)
BBFJ	Black-box with a frozen Jacobian
CD	Central finite differences
CDFJ	Central finite differences with a frozen Jacobian
DD	Direct differentiation (as opposed to adjoint-variable formulation)
DDII	Direct differentiation incremental iterative (strategy)
DDIITCFR	DDIITC with the AD-generated turbulence-modeling correction frozen
DDIITC	DDII with AD-generated turbulence-modeling correction
[a](6)	Implementation for six design variables concurrently
[a](5 + 1)	Implementation first for five design variables concurrently, followed by one design variable

[a] Indicates each of the methods ADII, BB, and BBFJ.

TABLE C.3

Second-Order Formulations and Solution Strategies

[a]ADII.SO	SO automatic differentiation in incremental iterative form
AV.DD	FO adjoint-variable approach, then SO direct differentiation
AV.AV	FO adjoint-variable approach, then SO adjoint-variable approach
[a]AVII.BB	FO AVII, then black-box automatic differentiation
[a]BB.BB	Black-box automatic differentiation applied twice
CD.CD	Second-order central finite differences
DD.AV	FO direct differentiation, then SO adjoint-variable approach
DD.DD	FO direct differentiation, then SO direct differentiation
DDII.BB	FO DDII, then black-box automatic differentiation
DDIITC.BB	FO DDIITC, then black-box automatic differentiation
DDIITCFR.BB	FO DDIITCFR, then black-box automatic differentiation
QA.CD	FO quasi-analytical differentiation, then first-order central finite differences
QA.CD(FJ)	QA.CD with a frozen Jacobian
[b](6)	Implementation for six design variables concurrently
[b](5 + 1)	Implementation first for five design variables concurrently, followed by one design variable

[a] Not implemented.

[b] Indicates each of the methods DDII.BB, DDIITC.BB, and DDIITCFR.BB.

ACKNOWLEDGMENTS

The work of L.L.S., A.C.T. III, G.J.-W.H., and V.M.K. is supported by NASA Grant NAG-1-1265 from NASA Langley Research Center; Dr. Henry Jones is the technical monitor. We acknowledge Dr. Chris Bischof of Argonne National Laboratory and Dr. Alan Carle of Rice University for consultations on the use of ADIFOR and the assistance of Kara Haigler in helpful discussions regarding vectorization improvements.

REFERENCES

1. V. M. Korivi, A. C. Taylor, III, P. A. Newman, G. J.-W. Hou, and H. E. Jones, *J. Comput. Phys.* **113,** No. 2, 336 (1994); in *Proceedings, Fourth AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization, AIAA, Cleveland, OH, 1992,* p. 465; AIAA Paper 92-4746-CP, Sept. 1992.
2. P. A. Newman, G. J.-W. Hou, H. E. Jones, A. C. Taylor, III, and V. M. Korivi, "Observations on Computational Methodologies for Use in Large-Scale Gradient-Based Multidisciplinary Design," in *Proceedings, Fourth AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization, AIAA, Cleveland, OH, 1992,* p. 531. AIAA Paper 92-4753-CP, Sept. 1992.
3. V. M. Korivi, A. C. Taylor, III, G. J. -W. Hou, P. A. Newman, and Jones, H. E., "Sensitivity Derivatives for Three-Dimensional Supersonic Euler Code using Incremental Iterative Strategy," *A Collection of Technical Papers, Part 2, 11th AIAA Computational Fluid Dynamics Conference, AIAA, Orlando, FL, 1993,* p. 1053; expanded version, *AIAA J.* **32,** No. 6, 1319 (1994).
4. G. W. Burgreen, and O. Baysal, AIAA Paper 94-0094, January 1994 (unpublished).
5. V. M. Korivi, P. A. Newman, and A. C. Taylor, III, AIAA Paper 94-4270, September 1994 (unpublished).
6. A. Griewank and G. F. Corliss (Eds.), *Automatic Differentiation of Algorithms: Theory, Implementation, and Applications* (SIAM, Philadelphia, 1991).
7. A. Griewank, "On Automatic Differentiation," in *Mathematical Programming: Recent Developments and Applications,* edited by M. Iri and K. Tanabe (Kluwer Academic, Boston, 1989), p. 83.
8. L. B. Rall, "Automatic Differentiation: Techniques and Applications," *Lecture Notes in Computer Science, Mathematical Programming: Recent Developments and Applications,* Vol. 120, (Springer-Verlag, Berlin, 1981).
9. C. H. Bischof, and A. Griewank, "ADIFOR: A Fortran System for Portable Automatic Differentiation," in *Fourth AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Optimization, Cleveland, OH,* p. 433; AIAA 92-4744-CP, Sept. 1992.
10. C. Bischof, G. Corliss, L. Green, A. Griewank, K. Haigler, and P. Newman, *Comput. Systems Eng.* **3**(6) (1993); presented at the *Symposium on High-Performance Computing for Flight Vehicles, Dec. 7–9, 1992, Arlington, VA.*
11. L. Green, P. Newman, and K. Haigler, AIAA 93-3321, 1993; *J. Comput. Phys.* **125,** 313 (1996).
12. L. Green, C. Bischof, A. Carle, A. Griewank, K. Haigler, and P. Newman, "Automatic Differentiation of Advanced CFD Codes With Respect to Wing Geometry Parameters for MDO," in *Abstracts from Second U.S. National Congress on Computational Mechanics, August 16–18, 1993, Washington, D.C.,* p. 136.
13. A. C. Taylor, III, P. A. Newman, Hou, G. J.-W., and H. E. Jones, "Recent Advances in Steady Compressible Aerodynamic Sensitivity Analysis," in *Volumes in Mathematics and Its Applications* (*IMA*),

Vol. 68, *Flow Control*, edited by Max. D. Gunzburger (Springer-Verlag, New York/Berlin, 1995), p. 341.

14. A. C. Taylor, III, G. W. Hou, and V. M. Korivi, AIAA Paper 91-3083, September 1991 (unpublished).
15. C. H. Bischof, A. Carle, G. F. Corliss, A. Griewank, and P. Hovland, *Sci. Programming* **1**(1), 1 (1992).
16. C. H. Bischof, and P. Hovland, ANL-MCS-TM-158, Mathematics and Computer Science Division, Argonne National Laboratory, 1991 (unpublished).
17. C. H. Bischof, G. F. Corliss, and A. Griewank, ANL-MCS-TM-159, Mathematics and Computer Science Division, Argonne National Laboratory, 1991 (unpublished).
18. C. H. Bischof, A. Carle, G. F. Corliss, A. Griewank, and P. Hovland, ANL-MCS-TM-164, Mathematics and Computer Science Division, Argonne National Laboratory, 1992 (unpublished).
19. A. Griewank, C. Bischof, G. Corliss, A. Carle, and K. Williamson, "Derivative Convergence of Iterative Equation Solvers," in *Optimization Methods and Software,* Vol. 2, p. 321; 355; Technical Report MCS-P333-1192, Mathematics and Computer Science Division, Argonne National Laboratory, 1992.
20. L. L. Sherman, M.S. thesis, Dept. of Mechanical Engineering, Old Dominion University, December 1995.
21. G. J.-W. Hou, A. C. Taylor, III, and V. M. Korivi, *Int. J. Numer. Methods Eng.* **37,** 2251 (1994); AIAA Paper 91-2259, June 1991.
22. B. Baldwin and H. Lomax, AIAA Paper 78-0257, January 1978(unpublished).
23. G. J.-W. Hou, and J. Sheen, *Int. J. Numer. Methods Eng.* **36,** 417 (1993).