AIAA JOURNAL
Vol. 50, No. 12, December 2012

Derivatives for Time-Spectral Computational Fluid Dynamics Using an Automatic Differentiation Adjoint

Charles A. Mader*
University of Toronto, Toronto, Ontario M4K 2P9, Canada
and
Joaquim R. R. A. Martins†
University of Michigan, Ann Arbor, Michigan 48109

DOI: 10.2514/1.J051658

In computational fluid dynamics, for problems with periodic flow solutions, the computational cost of spectral methods is significantly lower than that of full, unsteady computations. As is the case for regular steady-flow problems, there are various interesting periodic problems, such as those involving helicopter rotor blades, wind turbines, or oscillating wings, that can be analyzed with spectral methods. When conducting gradient-based numerical optimization for these types of problems, efficient sensitivity analysis is essential. The authors developed an accurate and efficient sensitivity analysis for time-spectral computational fluid dynamics. By combining the cost advantage of the spectral-solution methodology with an efficient gradient computation, the total cost of optimizing periodic unsteady problems can be significantly reduced. The efficient gradient computation takes the form of an automatic differentiation discrete adjoint method, which combines the efficiency of an adjoint method with the accuracy and rapid implementation of automatic differentiation. To demonstrate the method, the authors computed sensitivities for an oscillating ONERA M6 wing. The sensitivities are shown to be accurate to 8–12 digits, and the computational cost of the adjoint computations is shown to scale well up to problems of more than 41 million state variables.

Nomenclature

α	=	angle of attack
C_D	=	average drag coefficient
C_L	=	average lift coefficient
C_m	=	average pitch moment coefficient
D_t	=	spectral derivative operator
e_t	=	total energy
f_i	=	flux term, function
h	=	step size
I	=	function of interest
k	=	frequency index
l	=	time instance index
M	=	freestream Mach number
N	=	number of spectral time intervals
N_{cells}	=	number of cells in mesh
N_I	=	number of functions of interest
N_x	=	number of design variables
N_ζ	=	number of states per cell
N_{ζ_T}	=	total number of states
n	=	time instance index
p	=	pressure
$\mathcal{R}$	=	steady-state flow residuals
$\mathcal{R}_{\mathcal{TS}}$	=	time-spectral flow residuals
T	=	time period of periodic flow problem
t	=	time, intermediate variables
u	=	flow velocity with respect to fixed frame
V	=	volume
v	=	flow velocity with respect to moving grid
w	=	velocity of moving grid
x	=	time interval, design variables
y	=	coordinate direction
δ_i	=	elements of identity matrix
ζ	=	flow states
ρ	=	density
ψ	=	adjoint vector

Received 6 October 2011; revision received 22 March 2012; accepted for publication 29 March 2012.

*Research Associate, Institute for Aerospace Studies. Member AIAA.
†Associate Professor, Department of Aerospace Engineering. Senior Member AIAA.

I. Introduction

FOR cost-effective gradient-based optimization, both efficient function analysis and efficient sensitivity analysis are needed. In this work, we demonstrate the application of the automatic differentiation adjoint (ADjoint) method, an efficient sensitivity analysis method, to a time-spectral computational fluid dynamics solver; the latter is an efficient method for solving periodic unsteady-flow problems. As the background section will show, significant progress has been made in both the development of adjoint techniques and the solution of periodic unsteady problems. Building on this previous work, we apply the accurate and efficient ADjoint technique to the time-spectral equations, producing sensitivities that are demonstrably more accurate than those of previous time-spectral ADjoint implementations. Further, we demonstrate the scaling of the method to cases of up to 41 million flow states, showing that it is valid and useful for practical problem sizes. A summary of related work is presented in the background section, and the implementation of our method is discussed in the implementation section.

II. Background

The adjoint method for sensitivity analysis is now commonly used in aerodynamic shape optimization. The application of the adjoint method to fluid dynamics was first developed by Pironneau [1], who demonstrated how to minimize the drag over bodies immersed in laminar viscous flows. Jameson [2] applied the adjoint method to the Euler-based aerodynamic shape optimization of airfoils and wings. Since these seminal contributions, the method has been applied to the optimization of airfoils, including viscous effects [3–6], laminar-turbulent transition prediction [7], and multipoint airfoil

optimization problems [6,8]. The adjoint method has also been extended to three dimensions and used to optimize wings for inviscid flows [9,10] and viscous flows [11,12], as well as for multipoint wing problems [13] and full-wing body configurations [14,15]. The applications of the method go beyond shape optimization with strictly aerodynamic considerations: It has been applied to sonic-boom reduction [16], hypersonic flows, including magneto-hydrodynamic effects [17,18], and coupled aerostructural design [19,20]. In each case, the adjoint method enabled efficient design optimization with large numbers of variables.

Although the adjoint method is relatively common in steady-state optimizations (at least in the research community), it is still relatively uncommon in time-dependent problems. The adjoint solution of two-dimensional time-dependent problems was demonstrated by Nadarajah and Jameson [21], Mani and Mavriplis [22], Rumpfkeil and Zingg [23], and Wang et al. [24]. A three-dimensional adjoint solution was developed by Mavriplis [25]. These methods are a significant improvement over finite difference sensitivity methods, but they still have a high computational cost. The unsteady adjoint computation requires a reverse integration in time from the final solution back to the initial condition [21,22]. Thus, a time-dependent adjoint requires the full forward solution of the unsteady problem, storing the flow states for each time step along the way, followed by a reverse sweep of the solution process to find the adjoint solution. Although this process is more efficient than computing a full unsteady solution for every design variable in the problem, it is expensive. Various methods for reducing the computational requirements have been suggested, for example writing the solution history to disk rather than storing the solution in memory [22], evaluating only a periodic portion of the time history for the adjoint problem [21,23], or using checkpointing algorithms in combination with automatic differentiation [26]. However, even with these additions, the computational cost of the full unsteady adjoint method is significant.

Recently, the use of spectral methods to discretize the time-domain portion of periodic computational fluid dynamics (CFD) problems has gained popularity. These methods exploit the periodic nature of the problem by expressing the states of the system as a Fourier series in time. The entire periodic solution can then be recovered from a small number of representative state instances spanning the time period, or frequency spectrum, of the solution. These state instances can then be solved for directly, eliminating the need to iterate through the starting transients typical of unsteady CFD solutions.

In this context, spectral-method computations can be performed in either the time domain, the frequency domain, or a combination of the two. In the time domain, the state instances represent discrete snapshots of the solution in time, whereas in the frequency domain, the state instances represent distinct frequencies present in the solution. In each case, the spectral solution is capable of representing a fundamental frequency as well as a number of higher harmonics. The number of resolved harmonics is related to the number of time or frequency instances present in the solution.

Early work on time nonlinear spectral solution techniques was conducted by Hall et al. [27], who derived a spectral formulation for the two-dimensional Navier–Stokes equations. This derivation was conducted in the frequency domain, but to facilitate the computation, they transformed the flow equations back to the time domain. This allowed a typical time-domain residual formulation to be used for the computation of the solution in each of the spectral instances. This residual was augmented by a spectral term that coupled the various solution instances. The residuals were computed in the time domain, but both the spectral operator and the boundary conditions were applied in the frequency domain, yielding a mixed time-/frequency-domain approach. In an extension of this work, Ekici and Hall [28] applied this technique, known as the harmonic balance technique, to multistage turbomachinery applications where a variety of frequencies may be present.

The time-spectral method introduced by Gopinath and Jameson [29,30] is similar to the harmonic balance method of Hall et al. [27]. However, the time-spectral method is derived completely in the time domain. This yields a purely real spectral operator, and allows for the use of the time-domain residual operator in its original form, including the boundary conditions. This is particularly advantageous in the context of the present work, because it allows us to check the newly developed time-spectral ADjoint sensitivities against those computed using the complex-step method [31–34]. This verification is not possible for codes that use frequency-domain analysis because they use complex arithmetic in the solution process.

Another nonlinear spectral solution technique is the nonlinear frequency domain (NLFD) method developed by McMullen et al. [35–37]. In this technique, the solution process takes place primarily in the frequency domain. The states of the system are stored as frequency-domain Fourier coefficients, and the solution steps are generated from the frequency-domain residual and spectral operator. To simplify the implementation, the residual is evaluated in the time domain, where the states are transformed from the frequency domain to the time domain. Then, the residual is transformed from the time domain to the frequency domain using fast Fourier transform (FFT) techniques.

The time-spectral CFD method reduces the computational cost of a periodic unsteady-flow solution relative to a full unsteady-flow solution for periodic problems. Similarly, the time-spectral adjoint method can dramatically reduce the computational cost of an optimization problem that involves a periodic unsteady problem. Just as the spectral solution technique modifies a single unsteady CFD problem into a set of coupled steady CFD problems, the time-spectral ADjoint technique converts a full unsteady adjoint problem into a single large, steady adjoint problem. Coupling this with the efficient solution of large sparse linear systems provided by modern software packages, such as Portable Extensible Toolkit for Scientific Computation (PETSc) [38], allows us to rapidly implement an ADjoint technique for periodic unsteady problems.

Adjoint methods have been developed for each of the spectral methods mentioned earlier. Thomas et al. [39] developed an adjoint for the two-dimensional viscous harmonic balance equations. They used a combination of forward- and reverse-mode automatic differentiation (AD) to generate the terms necessary for the adjoint. They computed mesh sensitivities for an airfoil and verified their implementation using finite difference sensitivities as the benchmark. Nadarajah and Jameson [40] developed an ADjoint implementation for the NLFD equations. They used analytic techniques to derive a discrete adjoint operator for the NLFD solver. The resulting technique was used to optimize an oscillating transonic wing. Finally, Choi et al. [41] developed an ADjoint implementation for the time-spectral equations. They used a manually coded adjoint method and a time-spectral flow solver to calculate the gradients required for a helicopter rotor-blade optimization. The method improved the blades, but the ADjoint implementation did not achieve the full numerical accuracy that is theoretically possible with a discrete adjoint method. This limited accuracy was the result of approximations made in the differentiation of the functions related to the spectral radius and artificial dissipation.

We use the ADjoint approach of Mader et al. [42] to generate the discrete adjoint operator. Similar to the work by Thomas et al. [39], this approach combines AD and adjoint methods to generate accurate and efficient sensitivities. The characteristics of the adjoint method ensure an efficient method for computing the sensitivities of a small number of output functions of interest with respect to a large number of design variables. The use of AD ensures that the partial derivatives used in the adjoint formulation are accurate and reduces the time required to compute those derivatives. We extend our method to a three-dimensional time-spectral CFD solver, providing a detailed overview of the implementation, and demonstrate the resulting code on large-scale problems with up to 41 million flow states. Further, we conclusively demonstrate the accuracy of the method by comparing the derivatives computed with those computed with the complex-step method. In the following sections, we introduce each of the components (the time-spectral flow solver, the adjoint method, and automatic differentiation) and then discuss how these components are used in the implementation of a time-spectral ADjoint. We then demonstrate the accuracy and efficiency of the method on an oscillating-wing test case.

III. Time-Spectral Computational Fluid Dynamics

We first review the time-spectral flow solver and its relation to the steady-flow solver. The particular spectral method that we use, the time-spectral method, was derived by Gopinath and Jameson [29] and van der Weide et al [43]. As discussed in the Introduction, this method is one of a class of solution techniques based on representing the time-derivative operator in the flow equations as a Fourier series. Expressing the time-derivative operator in this fashion allows for a significant reduction in the number of time snapshots needed to model the flow, thereby reducing the computational cost of the solution. In particular, this method focuses on expressing the time derivative strictly in the time domain, which eliminates the need to use complex numbers and FFTs in the solution process. The details of this method and the resulting time-derivative operator can be found in Gopinath [30]. The specific implementation of the time-spectral method used here is that of the SUmb flow solver [44]. SUmb is a cell-centered multiblock solver for the Reynolds-averaged Navier–Stokes equations (steady, unsteady, and time-spectral) and it has options for a variety of turbulence models with one, two, and four equations. The details of the flow equations in this context are given in the following discussion.

To put the derivation of the time-spectral adjoint sensitivity equations in context, we provide a basic derivation of the time-spectral flow equations. We start by writing the governing equations for unsteady flow:

$$V\frac{\partial \zeta}{\partial t}+\frac{\partial f_i}{\partial y_i}=0 \tag{1}$$

where y_i are the coordinates in the ith direction. We assume a nondeforming mesh, so that the cell volume V can be moved outside the time derivative.

Because we are simulating periodic problems, grid motion needs to be accounted for, and so the velocity of the grid w is included in the flux term. A more detailed discussion of the formulation used in our flow solver can be found in Mader and Martins [45]. Based on this formulation, the inviscid flow states and fluxes for each cell are

$$\zeta=\begin{bmatrix}\rho\\ \rho u_1\\ \rho u_2\\ \rho u_3\\ \rho e_t\end{bmatrix},\qquad f_i=\begin{bmatrix}\rho u_i-\rho w_i\\ \rho u_i u_1-\rho w_i u_1+p\delta_{i1}\\ \rho u_i u_2-\rho w_i u_2+p\delta_{i2}\\ \rho u_i u_3-\rho w_i u_3+p\delta_{i3}\\ \rho u_i(e_t+p)-\rho w_i e_t\end{bmatrix} \tag{2}$$

We can then rewrite Eq. (1) in a concise semidiscrete form as

$$V\frac{\partial \zeta}{\partial t}+\mathcal{R}(\zeta)=0 \tag{3}$$

where $\mathcal{R}$ represents the spatially discretized residual operator implemented in the flow solver. In our case, this is a second-order cell-centered finite-volume scheme. This operator includes all of the boundary conditions and artificial dissipation operators in the flow solver.

The goal of the time-spectral approach, as for other spectral approaches, is to find a way to solve directly for the periodic steady-state solution of a given problem. This eliminates the need to iterate through the initial transients of the unsteady problem in the solution process, thus reducing the computational cost. For spectral methods, this is accomplished by expressing the states of the system as a Fourier series and then solving the problem at a finite number of frequencies (for frequency-domain approaches) or time instances (for time-domain approaches). The derivation of the time-spectral equations from the general unsteady form of the equations is described in the following discussion.

The fundamental assumption here is that the flow is periodic in time and thus the states of the system ζ can be expressed as a Fourier series. We can then write the Fourier transform of the states as

$$\hat{\zeta}_k=\frac{1}{N}\sum_{n=0}^{N-1}\zeta^n e^{-ikx_n} \tag{4}$$

with the corresponding inverse transform given by

$$\zeta^n=\sum_{k=-\frac{N-1}{2}}^{\frac{N-1}{2}}\hat{\zeta}_k e^{ikx_n} \tag{5}$$

In Eqs. (4) and (5) we have two representations of the state vector, ζ and $\hat{\zeta}$. These vectors represent the time- and frequency-domain representations, respectively, of the state vector. The time interval for the series is $x_n=2\pi n/N$, where N is the number of time intervals and n is the index of the current time interval. In the frequency domain, k represents the frequency component index of the state matrix.

Combining Eqs. (4) and (5) to express ζ^n explicitly in the time domain yields

$$\zeta^n=\sum_{k=-\frac{N-1}{2}}^{\frac{N-1}{2}}\frac{1}{N}\sum_{l=0}^{N-1}\zeta^l e^{-ikx_l}e^{ikx_n} \tag{6}$$

Rearranging the sums yields

$$\zeta^n=\frac{1}{N}\sum_{l=0}^{N-1}\zeta^l\sum_{k=-\frac{N-1}{2}}^{\frac{N-1}{2}}e^{-ikx_l}e^{ikx_n} \tag{7}$$

Now, defining another form of the time interval as $x_{ln}=x_n-x_l$, and using that in the preceding equation, we have

$$\zeta^n=\frac{1}{N}\sum_{l=0}^{N-1}\zeta^l\sum_{k=-\frac{N-1}{2}}^{\frac{N-1}{2}}e^{ikx_{ln}} \tag{8}$$

The inner sum is a geometric series and, therefore,

$$\sum_{k=-\frac{N-1}{2}}^{\frac{N-1}{2}}e^{ikx_{ln}}=e^{-i\frac{N-1}{2}x_{ln}}\frac{1-e^{iNx_{ln}}}{1-e^{ix_{ln}}}=\frac{\sin(Nx_{ln}/2)}{\sin(x_{ln}/2)} \tag{9}$$

Replacing this term in Eq. (8), we have

$$\zeta^n=\frac{1}{N}\sum_{l=0}^{N-1}\zeta^l\frac{\sin(Nx_{ln}/2)}{\sin(x_{ln}/2)} \tag{10}$$

Differentiating Eq. (10) with respect to time yields

$$D_t\zeta^n=\frac{1}{N}\sum_{l=0}^{N-1}\zeta^l\left[\frac{N\cos[(Nx_{ln}/2)]}{2\sin(x_{ln}/2)}-\frac{\sin(Nx_{ln}/2)\cos(x_{ln}/2)}{2\sin^2(x_{ln}/2)}\right]\frac{\mathrm{d}x_{ln}}{\mathrm{d}t_{ln}} \tag{11}$$

and because $Nx_{ln}/2$ is an integer multiple of π,

$$\sin\left(\frac{Nx_{ln}}{2}\right)=0\quad\text{and}\quad\cos\left(\frac{Nx_{ln}}{2}\right)=(-1)^{(n-l)} \tag{12}$$

Using these relationships in Eq. (11), we have

$$D_t\zeta^n=\frac{1}{N}\sum_{l=0}^{N-1}\zeta^l\left[\frac{N(-1)^{(n-l)}}{2\sin(x_{ln}/2)}\right]\frac{\mathrm{d}x_{ln}}{\mathrm{d}t_{ln}} \tag{13}$$

We now consider the derivative $\mathrm{d}x_{ln}/\mathrm{d}t_{ln}$ from Eq. (13). Substituting in the value of x_{ln} and evaluating gives

$$\begin{aligned}x_{ln}&=\frac{2\pi(n-l)}{N}=\frac{2\pi}{T}\frac{T(n-l)}{N}=\frac{2\pi}{T}\Delta t(n-l)\\&=\frac{2\pi}{T}(t_n-t_l)=\frac{2\pi}{T}t_{ln}\end{aligned} \tag{14}$$

Therefore, the derivative of x_{ln} with respect to time is

$$\frac{\mathrm{d}x_{ln}}{\mathrm{d}t_{ln}}=\frac{2\pi}{T} \tag{15}$$

and we can now use this relationship in Eq. (13) to get

$$D_t\zeta^n = \frac{\pi}{T}\sum_{l=0}^{N-1}\zeta^l\left[\frac{(-1)^{(n-l)}}{\sin(x_{ln}/2)}\right] \tag{16}$$

Finally, we can simplify this expression to

$$D_t\zeta^n = \frac{\pi}{T}\sum_{l=0}^{N-1} d_{ln}\zeta^l \tag{17}$$

where d_{ln} is a matrix operator defined as

$$d_{ln} = \begin{cases} \frac{(-1)^{(n-l)}}{\sin[\pi(n-l)/N]} & \text{if } l \neq n \\ 0 & \text{if } l = n \end{cases} \tag{18}$$

Thus, D_t is an operator that spans all of the time instances in the solution. By solving the N coupled time instances represented in the equation,

$$VD_t\zeta^n + R(\zeta^n) = 0 \tag{19}$$

where n represents each of the N time instances, we obtain a coupled set of solutions that represents the periodic steady-state solution to a given problem.

IV. Adjoint Equations

Having derived the governing equations of the flow solver, we can now derive the corresponding adjoint equations. To derive the time-spectral adjoint equations, we start by writing the vector-valued function of interest I as

$$I = I(x, \zeta^n(x)) \tag{20}$$

where x represents the vector of design variables, and ζ^n is the state variable vector for the nth time instance, where $n = 1, \ldots, N$, with N representing the total number of instances.

When deriving the adjoint equations for the steady-flow case, we can express the governing equations as

$$\mathcal{R}(x, \zeta(x)) = 0 \tag{21}$$

In the time-spectral case, following the methods of van der Weide et al. [43], we redefine the governing equations by augmenting them with the spectral derivative operator. This yields

$$\mathcal{R}_{TS} = D_t\zeta^n(x) + R(x, \zeta^n(x)) = 0 \tag{22}$$

where $\mathcal{R}(x, \zeta^n(x))$ is a spatially discretized steady-state residual for the nth time instance, and D_t is the spectral operator defined in Eq. (17). This yields a modified set of residuals

$$\mathcal{R}_{TS}(x, \zeta^n(x)) = 0 \tag{23}$$

that must be satisfied at the end of the solution process. For each vector of design variables x, these residuals yield a solution vector ζ^n. They can now be treated in the same fashion as the steady-state residual is treated in a normal adjoint formulation.

For the following derivations, we will first define some size variables. Let N_{ζ_T} be the total number of states, N_x the number of design variables, and N_I the number of functions of interest. We now discuss the derivation of the adjoint equations.

We first use the chain rule to find the total sensitivity of the vector-valued function of interest:

$$\frac{\mathrm{d}I}{\mathrm{d}x} = \frac{\partial I}{\partial x} + \frac{\partial I}{\partial \zeta^n}\frac{\mathrm{d}\zeta^n}{\mathrm{d}x} \tag{24}$$

Note that we make a distinction between total and partial derivatives. This is because the value of the state vector that satisfies $\mathcal{R}_{T_s} = 0$ is implicitly dependent on the value of the design variables x. Therefore, in the context of this paper, a partial derivative is a derivative evaluated for a constant set of states. A total derivative is a derivative evaluated including a solution of the governing equations to determine a new set of states ζ^n that satisfy $\mathcal{R}_{T_s} = 0$ for the new design variables x. Thus, in the preceding equation, the derivative $\mathrm{d}I/\mathrm{d}x$ is the total derivative that we obtain by performing a standard finite difference calculation over the entire flow solver. The partial derivative vectors $\partial I/\partial x$ and $\partial I/\partial \zeta^n$ are of size N_x and N_{ζ_T}, respectively, and are evaluated for a fixed set of states ζ^n. The total derivative of the states with respect to the design variables is represented by $\mathrm{d}\zeta^n/\mathrm{d}x$ and is of size $N_{\zeta_T} \times N_x$. Similarly, in Eq. (25), $\mathrm{d}\mathcal{R}_{TS}/\mathrm{d}x$ is the total derivative of the residuals, including the solution of the governing equations. By definition, this term must be zero to within the convergence tolerance of the flow solution. The flux Jacobian, $\partial\mathcal{R}_{TS}/\partial\zeta^n$, is a partial derivative of size $N_{\zeta_T} \times N_{\zeta_T}$. The partial derivative of the residuals of the governing equations with respect to the design variables is $\partial\mathcal{R}_{TS}/\partial x$, and is of size $N_{\zeta_T} \times N_x$. The total derivative $\mathrm{d}\zeta^n/\mathrm{d}x$ is as in Eq. (24).

As mentioned earlier, because the governing equations must always be satisfied at a converged solution, the total derivative of the residual in Eq. (23) with respect to any design variable must also be zero. This gives

$$\frac{\mathrm{d}\mathcal{R}_{TS}}{\mathrm{d}x} = \frac{\partial\mathcal{R}_{TS}}{\partial x} + \frac{\partial\mathcal{R}_{TS}}{\partial\zeta^n}\frac{\mathrm{d}\zeta^n}{\mathrm{d}x} = 0 \tag{25}$$

The derivative expressions in Eqs. (24) and (25) contain the total derivative $\mathrm{d}\zeta^n/\mathrm{d}x$, the evaluation of which requires N_x flow solutions. Because the total derivative in Eq. (25) must equal zero, we can eliminate this term from the equations. Moving the first term of this equation to the right-hand side gives

$$\frac{\partial\mathcal{R}_{TS}}{\partial\zeta^n}\frac{\mathrm{d}\zeta^n}{\mathrm{d}x} = -\frac{\partial\mathcal{R}_{TS}}{\partial x} \tag{26}$$

Substituting the solution of this system into Eq. (24) yields

$$\frac{\mathrm{d}I}{\mathrm{d}x} = \frac{\partial I}{\partial x} - \frac{\partial I}{\partial\zeta^n}\left[\frac{\partial\mathcal{R}_{TS}}{\partial\zeta^n}\right]^{-1}\frac{\partial\mathcal{R}_{TS}}{\partial x} \tag{27}$$

As in a steady adjoint solution, we now have an expression containing four partial-derivative terms and a set of linear solutions. The adjoint approach consists in factorizing the $\partial\mathcal{R}_{TS}/\partial\zeta^n$ matrix with the term to its left, yielding the adjoint system

$$\left[\frac{\partial R_{TS}}{\partial\zeta}\right]^T\psi = \frac{\partial I}{\partial\zeta} \tag{28}$$

Then, this solution is used in Eq. (27) to obtain the total sensitivity:

$$\frac{\mathrm{d}I}{\mathrm{d}x} = \frac{\partial I}{\partial x} - \psi^T\frac{R_{TS}}{\partial x} \tag{29}$$

As in the steady-state case, we now have a system of equations that requires N_I linear solutions to provide the necessary sensitivities for the optimization rather than the N_x nonlinear solutions necessary for a finite difference or complex-step approach. Because many aerodynamic optimization formulations contain many more design variables than functions of interest, this can be extremely advantageous. Note that because the time-spectral system is N times the size of the steady-state solutions, the adjoint system is also N times larger than the equivalent steady-state system.

Having derived the theory behind the adjoint equations, we will now consider the practical implementation of the partial derivatives in Eqs. (28) and (29). Specifically, we will examine the use of AD in the implementation of these partial derivatives.

V. Automatic Differentiation

The final theoretical component of our method is automatic differentiation, also known as computational differentiation or algorithmic differentiation. This is a well-known technique based on the systematic application of the chain rule of differentiation to computer programs. The method relies on tools that automatically augment the original program to compute user-specified derivatives [46,47].

The concept behind this technique is the idea that any computer program representing a function performs a series of simple operations f_i, for which simple analytic derivatives can be defined. Each of these functions produces an intermediate variable t_i and is a function of all the previous intermediate variables t_j, $j = 1, 2, \ldots, i-1$, such that

$$t_i = f_i(t_1, t_2, \ldots, t_{i-1}) \tag{30}$$

If we know the sequence of elementary functions that defines the overall function, and the derivatives of these functions, we can determine the derivative of the overall function by applying the chain rule to the derivatives of the elementary functions.

There are two common approaches to AD, forward mode and reverse mode. In forward mode, we select an input value of interest t_j and propagate the derivative with respect to this value forward as the program is evaluated. As shown by Bendtsen and Stauning [48], the operation for the forward mode can be expressed

$$\frac{\partial t_i}{\partial t_j} = \sum_{k=j}^{i-1} \frac{\partial f_i}{\partial t_k} \frac{\partial t_k}{\partial t_j} \tag{31}$$

This approach produces the derivatives of all of the output values with respect to a single input variable in one forward pass. Note that the function derivatives $\partial f_i/\partial t_k$ account only for the explicit dependence of f_i on t_k. The derivatives $\partial t_k/\partial t_j$ are total derivatives, including all of the implicit dependencies of t_k on t_j through all of the other intermediate t values.

The reverse mode can be expressed similarly. After an evaluation in the forward direction to compute all of the intermediate function values t_i, we perform a backward pass starting with a single output value t_i to accumulate the value of the derivatives of t_i with respect to all of the inputs t_j. The operation for the reverse mode (see Bendtsen and Stauning [48]) can be expressed

$$\frac{\partial t_i}{\partial t_j} = \sum_{k=j+1}^{i} \frac{\partial f_k}{\partial t_j} \frac{\partial t_i}{\partial t_k} \tag{32}$$

This approach produces the derivatives of a single t_i with respect to all of the input variables in one backward pass. Note that once again the function derivatives $\partial f_k/\partial t_j$ account only for the explicit dependence of f_k on t_j. The derivatives $\partial t_i/\partial t_k$ are total derivatives, including all of the implicit dependencies of t_i on t_k through all of the other intermediate t values.

These operations can also be expressed in matrix form. If we define

$$Df = \begin{bmatrix} 0 & \cdots & & \\ \frac{\partial f_2}{\partial t_1} & 0 & \cdots & \\ \frac{\partial f_3}{\partial t_1} & \frac{\partial f_3}{\partial t_2} & 0 & \cdots \\ \cdots & \vdots & \vdots & \ddots \\ \frac{\partial f_n}{\partial t_1} & \frac{\partial f_n}{\partial t_2} & \cdots & 0 \end{bmatrix} \tag{33}$$

and

$$Dt = \begin{bmatrix} 1 & \cdots & & \\ \frac{\partial t_2}{\partial t_1} & 1 & \cdots & \\ \frac{\partial t_3}{\partial t_1} & \frac{\partial t_3}{\partial t_2} & 1 & \cdots \\ \cdots & \vdots & \vdots & \ddots \\ \frac{\partial t_n}{\partial t_1} & \frac{\partial t_n}{\partial t_2} & \cdots & 1 \end{bmatrix} \tag{34}$$

then (see Bendtsen and Stauning [48]) the chain-rule operations can be expressed

$$Dt = I + DFDt \tag{35}$$

Rearranging this to combine the Dt variables gives

$$(I - DF)DT = I \tag{36}$$

Now, given the derivatives of the elementary functions Df, we can solve for the total derivatives Dt. To produce the reverse-mode formulation, we simply transpose the entire equation, yielding

$$(I - DF)^T Dt^T = I \tag{37}$$

A detailed example can be found in [42]. For further details see, for example, [47]. Note that the relative efficiency of the two modes depends on the ratio of inputs to outputs in the function being differentiated. In forward mode, the function must be evaluated once for each input variable being differentiated, whereas in reverse mode, it must be evaluated once for each output variable being differentiated.

The efficiency of the reverse mode for small numbers of output variables is a factor that we seek to exploit. As seen in the previous section, the adjoint equations contain two partial derivatives, $\partial I/\partial x$ and $\partial I/\partial \zeta^n$, for which the number of output variables is significantly smaller than the number of input variables. Further, we will show that it is more efficient to use reverse-mode differentiation when differentiating the single-cell residual routine described in Sec. VI.

Finally, there are two main approaches to implementing AD: source-code transformation and operator overloading. Tools that use source-code transformation add new statements to the original source code to compute the derivatives of the original statements. The operator-overloading approach consists in introducing a new user-defined type. This new type includes not only the value of the original variable, but the derivative as well. All the intrinsic operations and functions have to be redefined (overloaded) in order for the derivative to be computed together with the original computations. We use a source-transformation tool, TAPENADE [49,50], because such tools are typically more efficient than operator-overloading tools [47,51].

VI. ADjoint Implementation

The main idea of the ADjoint approach is to use AD to produce the routines that compute the partial-derivative terms in Eq. (27). In that sense, this work follows the procedure previously published by Mader et al. [42]. However, we have significantly enhanced the method's capability and efficiency. This section provides a brief overview of the ADjoint method and discusses the improvements and the extension to time-spectral equations.

A. Single-Cell Routine

In the previous work [42], the basis for the residual derivatives was a set of single-cell residual routines, developed from the original residual routines. These routines contained all of the functionality of the original block-based routines, including dissipation terms and boundary conditions, but were designed to operate on a $5 \times 5 \times 5$ cell cube. This is the smallest block of cells that encloses the second-order inviscid flux stencil, shown in Fig. 1. Evaluating this set of routines once produces the exact residuals for one cell in the mesh.

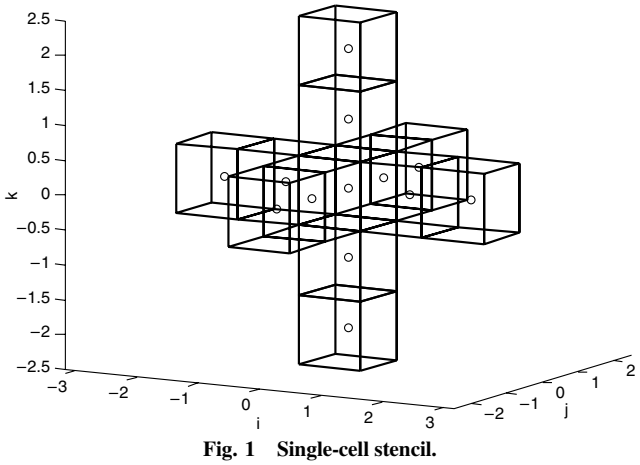

Fig. 1 Single-cell stencil.

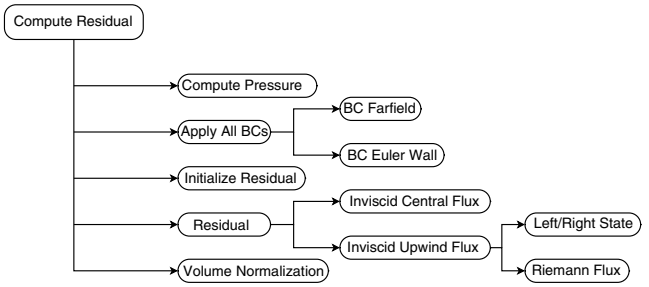

Fig. 2 Previous residual computation: steady case.

These single-cell residual routines are generated by cutting and pasting the residual computation from the original residual code, modifying the indices to limit the evaluation to a single stencil. This process, although not ideal, is less onerous than coding a derivative scheme from scratch. Further, by reducing the residual computation to a single cell at a time, we significantly reduce the size of the differentiation problem, making reverse-mode differentiation more manageable. This is advantageous, as we will discuss in more detail later, because of the 13 : 1 ratio of inputs to outputs in the residual stencil. Figure 2 shows an outline of the subroutines used in this implementation of the single-cell residual routine.

We have continued to use this single-cell residual approach in this work, extending the routines to include additional boundary conditions, discretizations, and solution modes. As shown in Fig. 3, the single-cell routines now include a second dissipation scheme, additional boundary conditions, the metric terms necessary for the mesh derivatives, and the loops required for the time-spectral implementation. Figure 3 also shows the locations of the time-instance loops needed to compute the residual for all time instances in the solution. The spectral operator from Eq. (17) is implemented in the `initializeResidual` subroutine.

B. Residual Partial Derivatives, $\partial\mathcal{R}/\partial\zeta$ and $\partial\mathcal{R}/\partial x$

All of the derivatives used in the ADjoint formulation are computed using reverse-mode AD. Specifically, the TAPENADE [49,50] AD tool is used to perform reverse-mode source-transformation AD. For the residual derivatives, $\partial\mathcal{R}/\partial\zeta$ and $\partial\mathcal{R}/\partial x$, the reverse-mode differentiation is performed on the single-cell residual routine described earlier. To explain this choice, we must discuss, in some detail, the structure of the single-cell residual routine.

For the steady case, the residual computation shown in Fig. 3 computes the value of the residual in a single-cell. This evaluation produces a result of length N_ζ, one value for each of the governing equations in that cell. The number of inputs required to generate the result is much higher. The variables of interest here are the system states ζ and the mesh coordinates x. For the second-order finite-volume discretization of the Euler equations used here, this requires the states in the nearest-neighbor cells for the central flux and the nearest-neighbor states, as well as the next-nearest-neighbor states, for the dissipation fluxes. Therefore, $13 \times N_\zeta$ states are required for the evaluation of the residual. The residual evaluation also depends on the coordinate locations of the corner nodes of each of the nearest-neighbor cells. This includes a total of 32×3 independent spatial degrees of freedom. Therefore, for the five states in the Euler equations, we have an overall ratio of $13 \times 5 + 32 \times 3 = 161{:}5$, more than 32:1, input variables to output variables. Even if we consider the fact that a single reverse-mode calculation is about 4.5 times more expensive than the equivalent forward evaluation of the single-cell routine, the reverse-mode calculation is still about 7 times faster than a forward computation for the single-cell routines.

In the time-spectral case, we must consider not only the spatial dependence of the operator, but also its temporal dependence. In this discussion, we refer to on-time-instance states and off-time-instance states. On-time-instance states are those that exist in the same time instance as the current residual evaluation. Off-time-instance states are those that exist in a time instance other than that associated with the current residual evaluation. In the case of a time-spectral solution,

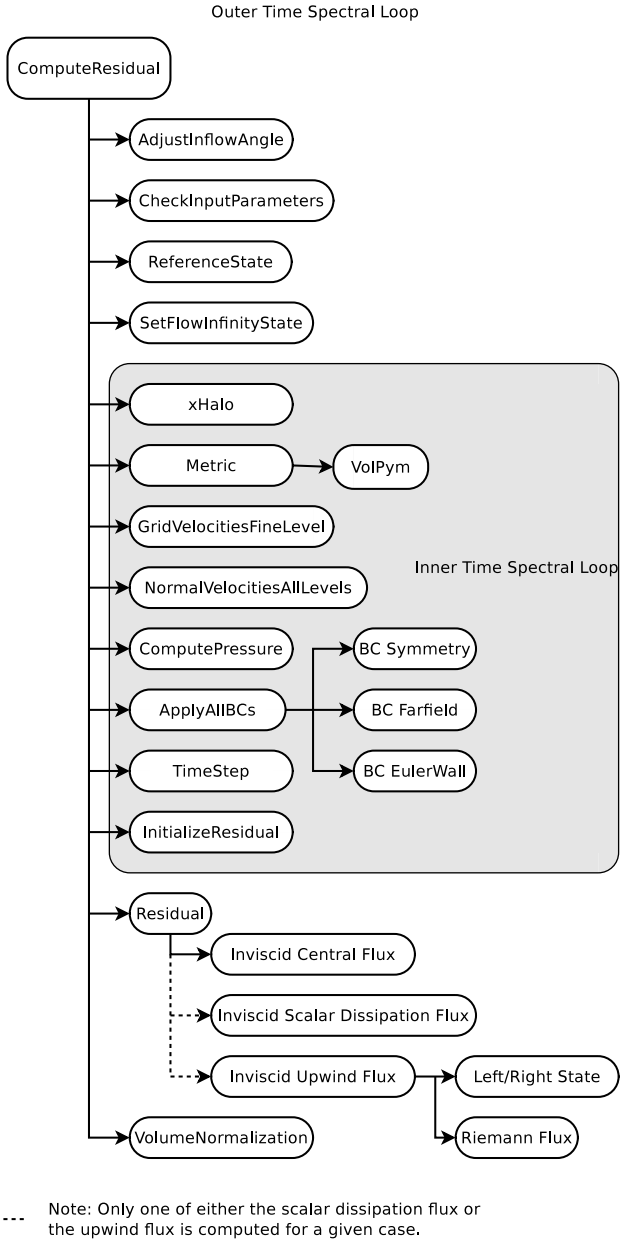

Fig. 3 Current residual computation: steady and time-spectral cases.

the residual is dependent on the same on-time-instance states and coordinates as described earlier. In addition, it is dependent on the states and coordinates of the current cell on each of the off-time instances. Thus, we now have $N_\zeta \times N$ residual values and $(13 \times N_\zeta + 32 \times 3) \times N + (N_\zeta + 8 \times 3) \times (N - 1)$ input states and coordinates. This leads to a ratio of 36:1 input variables to output variables for the Euler equations with three time instances, a ratio that increases as the number of time instances increases.

Note that both of these ratios include the state variable derivatives of $\partial\mathcal{R}/\partial\zeta^n$, as well as the coordinate derivatives of $\partial\mathcal{R}/\partial x$. This is possible because we have used reverse-mode AD for the derivative calculation. The reverse accumulation, shown in Eq. (32), allows us to start with a single residual and, accumulating backward through the routines, calculate the derivatives of all the inputs at once. This turns out to be a significant advantage.

An important aspect of the expanded routine in the context of the time-spectral adjoint is the need for additional loops to account for the extra time instances. As shown in Fig. 3, there are two time-instance loops in the spectral computation. The outer loop accounts for the time instances in the residual, and the inner loop accounts for the time instances of the states and coordinates. Therefore, we can see that the inner part of the computation scales with the number of

instances squared, whereas the outer part scales with the number of instances. However, this is a somewhat naive implementation of the single-cell routine. Examining Eq. (19) more closely, we notice that it is only the spectral operator $VD_t\zeta^n$ that contains states from all N time instances at one time. Therefore, we can reduce the number of terms in the inner time-spectral loop to only those necessary for this term. Figure 4 shows this simplified routine. With this improved implementation, we have significantly reduced the number of computations needed in the inner time-spectral loop. Thus, the derivative computation now scales, essentially, with the number of time instances N, rather than N^2. Timing results demonstrating this are presented in Sec. VII.

C. Function of Interest Partial Derivatives, $\partial I/\partial\zeta$ and $\partial I/\partial x$

For most aerodynamic shape optimization problems, the objectives of interest are the forces and moments (or the corresponding coefficients) acting on the body being optimized. Computing the partial derivatives of these quantities with respect to the states and mesh coordinates, $\partial I/\partial\zeta$ and $\partial I/\partial x$, is significantly simpler than the computation of the residual partial derivatives. This is evident when we compare the routines required to compute the residual (Fig. 4) and the routines required to compute the forces and moments (Fig. 5). As we can see from these figures, for the inviscid equations considered here, the force computation simply requires an integration of the pressure over the surface of the body in question. This requires an application of the boundary conditions as well as a surface normal computation, but this is significantly simpler than the complicated flux computation required for the residual. Further, we can see that the ratio of input variables to output variables strongly favors a reverse-mode technique for this computation. For a typical optimization problem, we might be interested in fewer than 10 force and moment coefficients (for example, C_L and C_D for a lift-constrained drag minimization), whereas the surface needed to compute those coefficients may require hundreds, thousands, or even tens of thousands of surface cells for accurate discretization. This yields an extremely high ratio of input to output variables and thus strongly favors the reverse-mode approach. As for the residual routines, the force routines have been differentiated using the reverse-mode source-transformation capabilities of TAPENADE [49,50]. This differentiation yields a routine that computes all the state and coordinate derivatives of a specified force or moment coefficient in a single pass.

The extension of this concept to the time-spectral case is relatively straightforward. We consider simple spectral objectives, such as the average lift, drag, and moment coefficients. In these cases, the spectral objective is based on simple algebraic combinations of the corresponding coefficient values at each of the discrete time instances in the solution. For example, the average drag may be computed as

$$C_D = \frac{1}{N}C_{D_1} + \frac{1}{N}C_{D_2} + \ldots + \frac{1}{N}C_{D_N} \tag{38}$$

where each of the coefficients C_{D_i} are computed directly from the states of time instance i. The time instances of the residual computation are coupled through the spectral time derivative of Eq. (19), but once the solution is computed, the computations of the coefficients at each instance in time are independent. Therefore, provided the objective function is a function of these independent coefficients, the time-instance-based coefficients and their derivatives can be used to form the time-spectral objective derivatives. Using the chain rule, we can express this idea as

$$\frac{\partial I}{\partial \zeta^n} = \frac{\partial I}{\partial C_{D_i}}\frac{\partial C_{D_i}}{\partial \zeta^n} \tag{39}$$

In this case, we are looking to compute a derivative of size $N_i \times [N_\zeta \times N_{\text{cells}}]$ by computing a matrix–matrix product of two matrices of size $N_i \times N$ and $N \times [N_\zeta \times N_{\text{cells}}]$, respectively. However, we

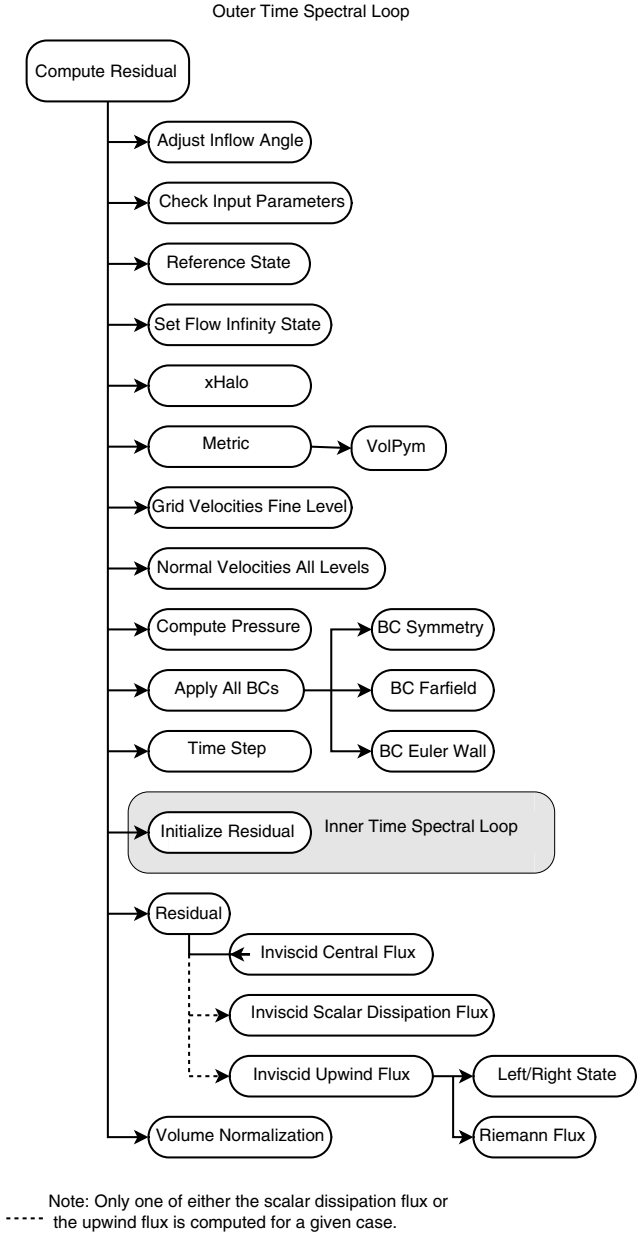

Fig. 4 Improved residual computation: time-spectral case.

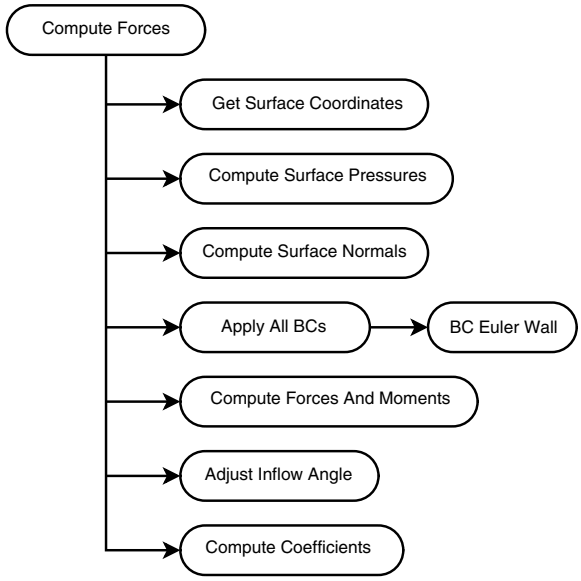

Fig. 5 Force and moment computation.

already have a reverse-mode computation for the derivative $\partial C_{D_i}/\partial \zeta^n$. Because we know that the coefficient computation in each of the time instances is independent, we can form the first derivative matrix $\partial I/\partial C_{D_i}$ and then multiply it by the result of the $\partial C_{D_i}/\partial \zeta^n$ computation, time instance by time instance, to generate the necessary derivatives without storing the second matrix. This approach yields the same ratio of inputs to outputs as in the general steady case discussed earlier. The only distinction is that in the spectral computation the coefficient derivatives are run N times, once for each time instance, whereas for the steady case, the derivatives are run once.

D. Solution of Adjoint System

Based on the characteristics of the adjoint equations and the fact that we can compute both the residual sensitivity matrices at the same time, we have chosen to explicitly store each of the terms in Eq. (27) for the solution of the adjoint equations. The time-spectral solution has a slightly different sparsity pattern than the steady case. The sparsity patterns of the steady and time-spectral cases for a 24-block H–H mesh are shown in Figs. 6 and 7, respectively. Comparing the two figures, we note that the overall sparsity pattern is similar. From the close-up views, we can see that what was a single block in the steady case has become a grouping of N blocks in the spectral case. Further, there is an additional set of off-diagonal entries present in each row for each time instance in the solution. These off-diagonal entries come from the spectral operator that couples the time instances together. The grouping apparent in Fig. 7 is indicative of our choice to order the matrix such that all time instances of a given block are adjacent. This ordering provides much better performance than the alternative, where all blocks of a given time instance are adjacent. This is because our code is parallelized by block, not by time instance. Thus, the derivatives for all time instances of a given block are computed in the same processor. Grouping the cells in the matrix by block instead of by time instance requires an immense amount of communication during the assembly of the matrix.

Once the various terms are generated, we use the PETSc [38] to store the sparse derivatives and solve the linear system of equations. Specifically, the linear solution is computed using a restarted Generalized Minimum Residual Method iterative solver (restarted after 150 subspace vectors) with an additive-Schwarz global preconditioner and Incomplete Lower-Upper [ILU(1)] local preconditioning. The preconditioning matrix is computed using the lumped dissipation technique of Hicken and Zingg [52]. This technique reduces the bandwidth of the preconditioning matrix, thereby reducing both the cost of generating the matrix and the memory requirements for storing it. In addition, we neglect the off-time-instance entries of the matrix when forming the preconditioner. This further reduces the fill-in of the ILU preconditioner and contributes to a reduction in the memory required to solve the system. Note that, because the preconditioning matrix contains derivatives with respect to the states only, it is computed with a reduced residual computation similar to that shown in Fig. 2. Also note that because all the terms are explicitly stored, the cost of computing adjoint solutions for multiple objectives is reduced.

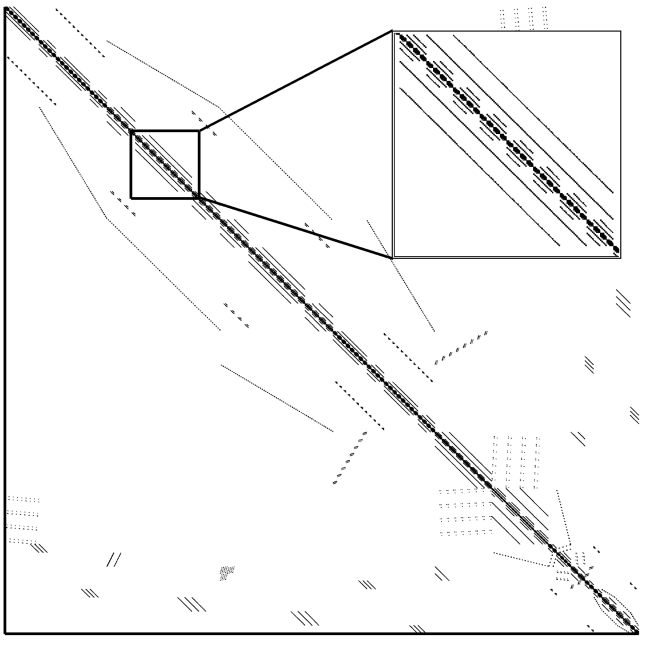

Fig. 6 Steady case sparsity pattern: 24-block H–H mesh.

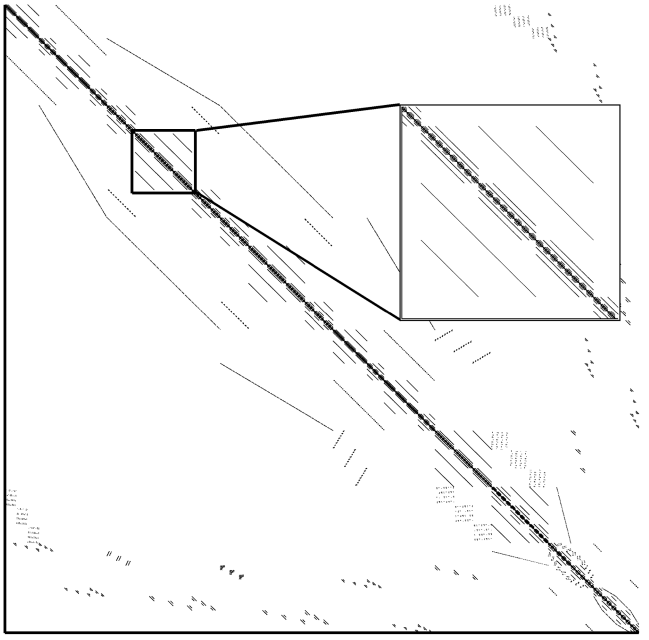

Fig. 7 Time-spectral sparsity pattern: 24-block H–H mesh; three time instances.

VII. Results

To demonstrate the effectiveness of the time-spectral ADjoint, we present results showing the numerical accuracy and computational efficiency of the method. Numerical accuracy is demonstrated by comparing the ADjoint sensitivities to sensitivities computed using the complex-step method [31]. In the complex-step method, the sensitivity of a function $I(x)$ is computed as

$$\frac{\mathrm{d}I}{\mathrm{d}x} = \frac{\mathrm{Im}(I(x+ih))}{h} + \mathcal{O}(h^2) \tag{40}$$

where $i = \sqrt{-1}$ and h is an extremely small step, in this case 10^{-20}. Because the perturbation is carried through the code in the complex portion of the variable, the subtractive cancellation issues associated with finite difference sensitivities are not present. Thus, h can be made very small, reducing the $\mathcal{O}(h^2)$ truncation error to negligible levels and yielding sensitivities that have the same level of precision as the function I. To demonstrate the computational efficiency, we show weak scaling results, showing how the implementation scales as the numbers of time instances and processors increase. The ideal scaling in this sense would be constant time for equal increases in problem size and number of processors, N^1. As we will show, the ADjoint implementation scales with $N^{1.2}$.

A. Test Case Description

To benchmark the time-spectral ADjoint for accuracy, we compute the sensitivities for a 917,000 cell ONERA M6 [53] wing mesh. The wing is simulated with Euler wall surfaces and a symmetry plane at the root. The mesh has an H–H topology with a nominal off-wall spacing of 0.002.

B. Accuracy

The design variables are those corresponding to a simple wing planform optimization, specifically, a simple linear twist distribution and a single value of the sweep, as well as the angle of attack. The sensitivities of the average lift, drag, and moment coefficients are shown for both pitching and plunging motions. Table 1 shows the sensitivities for the pitching motion, and Table 2 shows the sensitivities for the plunging motion.

As can be seen from Tables 1 and 2, the time-spectral ADjoint implementation is extremely accurate. For both the pitching and plunging motions, the sensitivities of the coefficients C_L, C_D, and C_m match to between 8 and 12 digits, which is consistent with the accuracy of the flow solution and ADjoint solution. Further, this level of accuracy allows for tightly converged optimization results, which provides greater certainty for designers.

C. Computational Cost

To assess the computational cost of the method, we compare the cost of the time-spectral ADjoint with the cost of the time-spectral flow solver and the steady-state ADjoint. The flow solver is a Newton–Krylov solver implemented using the nonlinear solvers in PETSc. The steady-state ADjoint is also solved using PETSc and implemented with the methods discussed in this work. The results of these comparisons are shown in Table 3.

Table 3 shows that the time-spectral ADjoint implementation is cost competitive with the steady-state case and that the overall cost of the ADjoint method is reasonable. The cost of the adjoint solution is between 1.6 and 2.4 times the cost of the flow solution. Further, the highest ratios occur between the steady case and the $N = 3$ case, indicating that our implementation scales well.

To quantify this, we plot the log of the total work done (scaled time multiplied by the number of flow states) versus the log of the number of flow states in the problem ($N_{\text{cell}} \times N_\zeta \times N$). In the ideal case, this plot would have a slope of one, indicating that the amount of work required to solve the problem scales exactly with the problem size. Figure 8 shows that the slope for the flow solver is 1.38 and that for the adjoint is 1.21. There are several possible reasons for the lack of ideal scaling. The time-spectral formulation requires the

Table 1 Sensitivity verification for pitching motion case with 10^{-12} relative convergence of the norm of the density residual, 10^{-20} complex-step size[a]

I	Design variable	ADjoint	Complex step
C_L		$9.630674822\underline{3}94 \times 10^{-2}$	$9.630674822\underline{4}17 \times 10^{-2}$
C_D	α	$7.800095886\underline{6}13 \times 10^{-3}$	$7.800095886\underline{0}03 \times 10^{-3}$
C_m		$4.840706880\underline{0}27 \times 10^{-2}$	$4.840706880\underline{0}28 \times 10^{-2}$
C_L		$-1.216072\underline{0}0\underline{2}894 \times 10^{-3}$	$-1.216072\underline{0}5\underline{2}039 \times 10^{-3}$
C_D	Sweep	$-2.509546\underline{5}6804 \times 10^{-4}$	$-2.509546\underline{6}2672 \times 10^{-4}$
C_m		$5.594081\underline{5}95955 \times 10^{-3}$	$5.594081\underline{5}90141 \times 10^{-3}$
C_L		$2.991291\underline{6}2\underline{6}9936 \times 10^{-2}$	$2.991291\underline{6}2\underline{7}7759 \times 10^{-2}$
C_D	Twist	$2.754766857\underline{7}997 \times 10^{-3}$	$2.754766858\underline{9}21 \times 10^{-3}$
C_m		$1.794675891\underline{4}073 \times 10^{-2}$	$1.794675891\underline{4}994 \times 10^{-2}$

[a]Underlined numbers facilitate the comparison of the numbers in the two columns.

Table 2 Sensitivity verification for plunging motion case with 10^{-12} relative convergence of the norm of the density residual, 10^{-20} complex-step size[a]

I	Design variable	ADjoint	Complex step
C_L		$9.652812\underline{3}6\underline{1}2840 \times 10^{-2}$	$9.652812\underline{3}6\underline{3}9616 \times 10^{-2}$
C_D	α	$8.484790352\underline{3}76 \times 10^{-3}$	$8.484790352\underline{7}48 \times 10^{-3}$
C_m		$5.035898\underline{4}2\underline{3}4680 \times 10^{-2}$	$5.035898\underline{4}2\underline{5}6455 \times 10^{-2}$
C_L		$-1.20734500\underline{4}567 \times 10^{-1}$	$-1.20734505\underline{9}580 \times 10^{-3}$
C_D	Sweep	$-3.0400606\underline{1}1953 \times 10^{-4}$	$-3.0400606\underline{8}085 \times 10^{-4}$
C_m		$5.490542\underline{5}1\underline{6}99 \times 10^{-3}$	$5.490542\underline{4}4506 \times 10^{-3}$
C_L		$2.997406710\underline{4}863 \times 10^{-2}$	$2.997406711\underline{3}723 \times 10^{-2}$
C_D	Twist	$2.950611281\underline{9}51 \times 10^{-3}$	$2.950611282\underline{7}71 \times 10^{-3}$
C_m		$1.857973481\underline{2}958 \times 10^{-2}$	$1.857973481\underline{3}456 \times 10^{-2}$

[a]Underlined numbers facilitate the comparison of the numbers in the two columns.

Table 3 Time-spectral ADjoint computational cost breakdown for ONERA M6 (times normalized to the total computational cost of the flow solution, which takes 160.3 s)

		Time spectral			
	Steady state	$N = 3$	$N = 5$	$N = 7$	$N = 9$
No. of processors	24	80	112	176	224
No. of flow states	4.587520×10^6	1.376256×10^7	2.29376×10^7	3.211264×10^7	4.128768×10^7
Average no. of flow states per processor	191,147	172,032	204,800	182,458	184,320
Percentage load imbalance	7.14	7.15	0	17.86	0
Flow solution time (10^{-10} convergence)	1.08	1.56	1.87	2.46	2.34
Jacobian matrix assembly time	0.26	0.46	0.58	0.68	0.66
Preconditioner assembly time	0.03	0.04	0.08	0.1	0.13
Algebraic volume/surface sensitivity time	0.14	0.14	0.14	0.14	0.14
Adjoint solution time, 10^{-10} convergence	1.89	2.89	2.53	2.98	2.75
Total sensitivity time	0.23	0.23	0.23	0.23	0.23
Total adjoint time	2.54	3.76	3.56	4.12	3.92
Flow solution: normalized total computational cost	1	4.82	8.71	14.82	21.82
Adjoint solution: normalized total computational cost	1.76	8.95	11.81	17.93	25.71

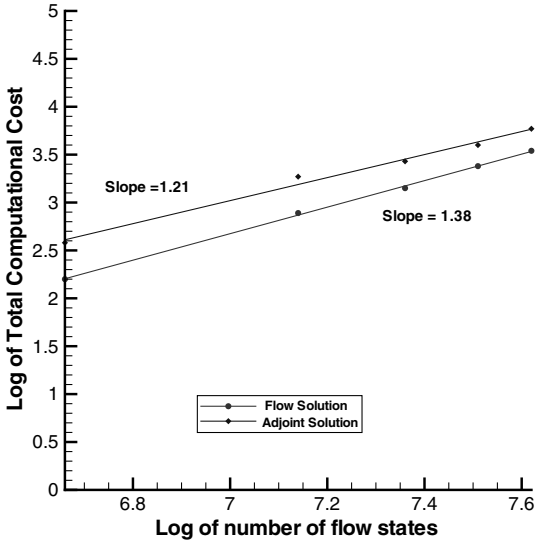

Fig. 8 Time-spectral ADjoint scaling.

computation of off-time-instance coupling terms. The computation of these terms increases the cost of the time-spectral flow solution with respect to the steady case, and computing the derivatives of these terms increases the cost of computing the Jacobian for the adjoint system. Although the number of terms in the on-time-instance blocks of the Jacobian scales linearly with the number of time instances, the number of off-time-instance terms scales with the number of time instances squared. Because the cost of the on-time-instance terms is much higher than that of the off-time-instance terms, the overall impact on the scaling of the algorithm is muted. A second potential reason for the imperfect scaling is the increased bandwidth of the time-spectral matrices. The addition of the off-time-instance terms changes the sparsity pattern of the matrix and causes additional fill-in in the preconditioner. This can impact the performance of the linear solution process and impact the solution efficiency. Finally, the number of processors changes for each case. Because no parallel implementation is perfect, this will impact the scaling to some extent as well. Although not perfect, the scaling is satisfactory. A single adjoint solution is shown here, but this method increases in efficiency when multiple adjoint solutions are required. Because we store the full Jacobian and the preconditioner, they are calculated just once for each group of adjoint solutions. Therefore, the cost can be amortized over multiple adjoint solutions, minimizing the cost per adjoint.

Finally, if we look at the cost in an absolute sense, we have demonstrated an implementation of the ADjoint that allows for the solution of problems with over 41 million flow states. Furthermore, the solution of that system, for both the flow and the adjoint, was computed in under 30 min on 224 Intel Nehalem processors. Although this is not a trivial computational cost, it does bring the optimization of periodic unsteady problems into the realm of possibility.

VII. Conclusions

The authors have developed and verified an adjoint method for the three-dimensional time-spectral Euler equations. The time-spectral flow equations and their differences from the steady equations have been discussed. Those differences that affected the implementation of the adjoint method were the focus of a particularly detailed discussion. Based on this theory, the ADjoint method was extended to the time-spectral equations. A detailed discussion of this implementation, including details of the computation of the derivatives in the adjoint equations, as well as details related to the solution of the adjoint system have been presented.

The results show that this method is extremely accurate, yielding derivatives accurate to between 8 and 12 digits for a solution converged to a relative tolerance of 10^{-12}. The computational efficiency of the method was demonstrated by comparing the steady-state case with time-spectral cases with 3, 5, 7, and 9 time instances. The results show that the method scaled well in a weak scaling sense up to more than 41 million flow states. The total cost of a single adjoint was approximately twice that of a flow solution. For a workload of approximately 180,000 flow states per processor, a flow solution and an adjoint solution can be computed in under 30 min. Thus, the optimization of periodic unsteady problems is now feasible.

Acknowledgments

The authors are grateful for the funding provided by the Natural Sciences and Engineering Research Council. The computations were performed on the general purpose cluster supercomputer at the SciNet high performance computing consortium. SciNet is funded by the Canada Foundation for Innovation under the auspices of Compute Canada, the Government of Ontario, Ontario Research Fund–Research Excellence, and the University of Toronto. The authors would like to thank Edwin van der Weide and Juan J. Alonso for their assistance in the early stages of this project, in particular with respect to the time-spectral computational fluid dynamics formulation in the SUmb flow solver. The authors would also like to thank Gaetan Kenway for his assistance in improving the efficiency of the steady-state ADjoint.

References

[1] Pironneau, O., "On Optimum Design in Fluid Mechanics," *Journal of Fluid Mechanics*, Vol. 64, No. 01, 1974, pp. 97–110. doi:10.1017/S0022112074002023

[2] Jameson, A., "Aerodynamic Design via Control Theory," *Journal of Scientific Computing*, Vol. 3, No. 3, Sept. 1988, pp. 233–260. doi:10.1007/BF01061285

[3] Nemec, M., and Zingg, D. W., "Newton—Krylov Algorithm for Aerodynamic Design Using the Navier—Stokes Equations," *AIAA Journal*, Vol. 40, No. 6, June 2002, pp. 1146–1154. doi:10.2514/2.1764

[4] Anderson, W. K., and Bonhaus, D. L., "Airfoil Design on Unstructured Grids for Turbulent Flows," *AIAA Journal*, Vol. 37, No. 2, 1999, pp. 185–191. doi:10.2514/2.712

[5] Anderson, W. K., and Venkatakrishnan, V., "Aerodynamic Design Optimization on Unstructured Grids with a Continuous Adjoint Formulation," *Computers and Fluids*, Vol. 28, Nos. 4–5, 1999, pp. 443–480. doi:10.1016/S0045-7930(98)00041-3

[6] Soemarwoto, B. I., and Labrujére, T. E., "Airfoil Design and Optimization Methods: Recent Progress At NLR," *International Journal for Numerical Methods in Fluids*, Vol. 30, No. 2, 1999, pp. 217–228. doi:10.1002/(SICI)1097-0363(19990530)30:2<217::AID-FLD830>3.0.CO;2-6

[7] Driver, J., and Zingg, D. W., "Numerical Aerodynamic Optimization Incorporating Laminar-Turbulent Transition Prediction," *AIAA Journal*, Vol. 45, No. 8, 2007, pp. 1810–1818. doi:10.2514/1.23569

[8] Nemec, M., Zingg, D. W., and Pulliam, T. H., "Multipoint and Multi-Objective Aerodynamic Shape Optimization," *AIAA Journal*, Vol. 42, No. 6, 2004, pp. 1057–1065. doi:10.2514/1.10415

[9] Hicken, J. E., and Zingg, D. W., "Induced-Drag Minimization of Nonplanar Geometries Based on the Euler Equations," *AIAA Journal*, Vol. 48, No. 11, 2010, pp. 2564–2575. doi:10.2514/1.J050379

[10] Jameson, A., "Optimum Aerodynamic Design Using Control Theory," *Computational Fluid Dynamics Review*, Wiley, New York, 1995, pp. 495–528.

[11] Jameson, A., Martinelli, L., and Pierce, N. A., "Optimum Aerodynamic Design Using the Navier-Stokes Equations," *Theoretical and Computational Fluid Dynamics*, Vol. 10, Nos. 1–4, 1998, pp. 213–237. doi:10.1007/s001620050060

[12] Nielsen, E. J., and Anderson, W. K., "Aerodynamic Design Optimization on Unstructured Meshes Using the Navier-Stokes Equations," *AIAA Journal*, Vol. 37, No. 11, 1999, pp. 1411–1419. doi:10.2514/2.640

[13] Buckley, H. P., Zhou, B. Y., and Zingg, D. W., "Airfoil Optimization Using Practical Aerodynamic Design Requirements," *Journal of Aircraft*, Vol. 47, No. 5, 2010, pp. 1707–1719. doi:10.2514/1.C000256

[14] Reuther, J. J., Jameson, A., Alonso, J. J., Rimlinger, M. J., and

Saunders, D., "Constrained Multipoint Aerodynamic Shape Optimization Using an Adjoint Formulation and Parallel Computers, Part 1," *Journal of Aircraft*, Vol. 36, No. 1, 1999, pp. 51–60. doi:10.2514/2.2413

[15] Reuther, J. J., Jameson, A., Alonso, J. J., Rimlinger, M. J., and Saunders, D., "Constrained Multipoint Aerodynamic Shape Optimization Using an Adjoint Formulation and Parallel Computers, Part 2," *Journal of Aircraft*, Vol. 36, No. 1, 1999, pp. 61–74. doi:10.2514/2.2414

[16] Nadarajah, S. K., Jameson, A., and Alonso, J. J., "Sonic Boom Reduction Using an Adjoint Method for Wing-Body Configurations in Supersonic Flow," AIAA Paper 2002-5547, 2002.

[17] Marta, A. C., Mader, C. A., Martins, J. R. R. A., van der Weide, E., and Alonso, J. J., "A Methodology for the Development of Discrete Adjoint Solvers Using Automatic Differentiation Tools," *International Journal of Computational Fluid Dynamics*, Vol. 21, No. 9, 2007, pp. 307–327. doi:10.1080/10618560701678647

[18] Marta, A. C., and Alonso, J. J., "High-Speed MHD Flow Control Using Adjoint-Based Sensitivities," *Proceedings of the 14th AIAA/AHI Space Planes and Hypersonic Systems and Technologies Conference*, AIAA Paper 2006-8009, 2006.

[19] Martins, J. R. R. A., Alonso, J. J., and Reuther, J. J., "High-Fidelity Aerostructural Design Optimization of a Supersonic Business Jet," *Journal of Aircraft*, Vol. 41, No. 3, 2004, pp. 523–530. doi:10.2514/1.11478

[20] Martins, J. R., Alonso, J. J., and Reuther, J. J., "A Coupled-Adjoint Sensitivity Analysis Method for High-Fidelity Aero-Structural Design," *Optimization and Engineering*, Vol. 6, No. 1, 2005, pp. 33–62. doi:10.1023/B:OPTE.0000048536.47956.62

[21] Nadarajah, S. K., and Jameson, A., "Optimum Shape Design for Unsteady Flows with Time-Accurate Continuous and Discrete Adjoint Methods," *AIAA Journal*, Vol. 45, No. 7, 2007, pp. 1478–1491. doi:10.2514/1.24332

[22] Mani, K., and Mavriplis, D. J., "Unsteady Discrete Adjoint Formulation for Two-Dimensional Flow Problems with Deforming Meshes," *AIAA Journal*, Vol. 46, No. 6, 2008, pp. 1351–1364. doi:10.2514/1.29924

[23] Rumpfkeil, M. P., and Zingg, D. W., "The Optimal Control of Unsteady Flows with a Discrete Adjoint Method," *Optimization and Engineering*, Vol. 11, No. 1, 2010, pp. 5–22. doi:10.1007/s11081-008-9035-5

[24] Wang, L., Mavriplis, D. J., and Anderson, W. K., "Adjoint Sensitivity Formulation for Discontinuous Galerkin Discretizations in Unsteady Inviscid Flow Problems," *AIAA Journal*, Vol. 48, No. 12, 2010, pp. 2867–2883. doi:10.2514/1.J050444

[25] Mavriplis, D. J., "Solution of the Unsteady Discrete Adjoint for Three-Dimensional Problems on Dynamically Deforming Unstructured Meshes," *Proceedings of the 46th AIAA Aerospace Sciences Meeting and Exhibit*, AIAA Paper 2008-727, Jan. 2008.

[26] Griewank, A., and Walther, A., "Algorithm 799: Revolve: An Implementation of Checkpointing for the Reverse or Adjoint Mode of Computational Differentiation," *ACM Transactions on Mathematical Software*, Vol. 26, No. 1, March 2000, pp. 19–45. doi:10.1145/347837.347846

[27] Hall, K. C., Thomas, J. P., and Clark, W. S., "Computation of Unsteady Nonlinear Flows in Cascades Using a Harmonic Balance Technique," *AIAA Journal*, Vol. 40, No. 5, 2002, pp. 879–886. doi:10.2514/2.1754

[28] Ekici, K., and Hall, K. C., "Nonlinear Analysis of Unsteady Flows in Multistage Turbomachines Using Harmonic Balance," *AIAA Journal*, Vol. 45, No. 5, 2007, pp. 1047–1057. doi:10.2514/1.22888

[29] Gopinath, A. K., and Jameson, A., "Time Spectral Method for Periodic Unsteady Computations over Two-Three Dimensional Bodies," AIAA Paper 2005-1220, 2005.

[30] Gopinath, A. K., "Efficient Fourier-Based Algorithms for Time-Periodic Unsteady Problems," Ph.D. Thesis, Stanford Univ., Stanford, CA, April 2007.

[31] Martins, J. R. R. A., Sturdza, P., and Alonso, J. J., "The Complex-Step Derivative Approximation," *ACM Transactions on Mathematical Software*, Vol. 29, No. 3, 2003, pp. 245–262. doi:10.1145/838250.838251

[32] Lyness, J. N., and Moler, C. B., "Numerical Differentiation of Analytic Functions," *SIAM Journal on Numerical Analysis*, Vol. 4, No. 2, 1967, pp. 202–210. doi:10.1137/0704019

[33] Squire, W., and Trapp, G., "Using Complex Variables to Estimate Derivatives of Real Functions," *SIAM Review*, Vol. 40, No. 1, 1998, pp. 110–112. doi:10.1137/S003614459631241X

[34] Anderson, W. K., Newman, J. C., Whitfield, D. L., and Nielsen, E. J., "Sensitivity Analysis for Navier-Stokes Equations on Unstructured Meshes Using Complex Variables," *AIAA Journal*, Vol. 39, No. 1, 2001, pp. 56–63. doi:10.2514/2.1270

[35] McMullen, M., Jameson, A., and Alonso, J. J, "Application of a Nonlinear Frequency Domain Solver to the Euler and Navier-Stokes Equations," *Proceedings of the 40th AIAA Aerospace Sciences Meeting and Exhibit*, AIAA Paper 2002-0120, Jan. 2002.

[36] McMullen, M., Jameson, A., and Alonso, J., "Demonstration of Nonlinear Frequency Domain Methods," *AIAA Journal*, Vol. 44, No. 7, 2006, pp. 1428–1435. doi:10.2514/1.15127

[37] McMullen, M., and Jameson, A., "The Computational Efficiency of Non-Linear Frequency Domain Methods," *Journal of Computational Physics*, Vol. 212, No. 2, 2006, pp. 637–661. doi:10.1016/j.jcp.2005.07.021

[38] Balay, S., Buschelman, K., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., and Zhang, H., "PETSc Users Manual," Argonne National Lab., TR ANL-95/11, Rev. 2.1.5, 2004.

[39] Thomas, J. P., Hall, K. C., and Dowell, E. H., "Discrete Adjoint Approach for Modeling Unsteady Aerodynamic Design Sensitivities," *AIAA Journal*, Vol. 43, No. 9, 2005, pp. 1931–1936. doi:10.2514/1.731

[40] Nadarajah, S. K., and Jameson, A., "Optimum Shape Design for Unsteady Three-Dimensional Viscous Flows Using a Nonlinear Frequency-Domain Method," *Journal of Aircraft*, Vol. 44, No. 5, 2007, pp. 1513–1527. doi:10.2514/1.27601

[41] Choi, S., Potsdam, M., Lee, K., Iaccarino, G., and Alonso, J. J., "Helicopter Rotor Design Using a Time-Spectral and Adjoint-Based Method," *Proceedings of the 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, AIAA Paper 2008-5810, 2008.

[42] Mader, C. A., Martins, J. R. R. A., Alonso, J. J., and van der Weide, E., "ADjoint: An Approach for the Rapid Development of Discrete Adjoint Solvers," *AIAA Journal*, Vol. 46, No. 4, 2008, pp. 863–873. doi:10.2514/1.29123

[43] van der Weide, E., Gopinath, A. K., and Jameson, A., "Turbomachinery Applications with the Time Spectral Method," AIAA Paper 2005-4905, 2005.

[44] van der Weide, E., Kalitzin, G., Schluter, J., and Alonso, J. J., "Unsteady Turbomachinery Computations Using Massively Parallel Platforms," *Proceedings of the 44th AIAA Aerospace Sciences Meeting and Exhibit*, AIAA Paper 2006-0421, 2006.

[45] Mader, C. A., and Martins, J. R. R. A., "Computation of Aircraft Stability Derivatives Using an Automatic Differentiation Adjoint Approach," *AIAA Journal*, 2011,Vol. 49, No. 12, pp. 2737–2750.

[46] Corliss, G., Faure, C., Griewank, A., Hascoet, L., and Naumann, U. (eds.), *Automatic Differentiation: From Simulation to Optimization*, Springer, New York, 2001.

[47] Griewank, A., and Walther, A., *Evaluating Derivatives*, 2nd ed., Society for Industrial and Applied Mathematics, Philadelphia, 2008, pp. 1–142.

[48] Bendtsen, C., and Stauning, O., "FADBAD, a Flexible C++ Package for Automatic Differentiation Using the Forward and Backward Methods," Technical Univ. of Denmark Rept. 1996-x5-94, 1996.

[49] Pascual, V., and Hascoët, L., "Extension of TAPENADE Towards Fortran 95," *Automatic Differentiation: Applications, Theory, and Tools*, edited by H. M. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, Lecture Notes in Computational Science and Engineering, Springer, New York, 2005, pp. 171–180.

[50] Hascoët, L., and Pascual, V., "TAPENADE 2.1 User's Guide," Inst. national de recherche en informatique et en automatique (INRIA) TR 300, 2004.

[51] Pryce, J. D., and Reid, J. K., "AD01, a Fortran 90 Code for Automatic Differentiation," Rutherford Appleton Lab. Rept. RALTR-1998-057, Chilton, Didcot, Oxfordshire, United Kingdom, 1998.

[52] Hicken, J. E., and Zingg, D. W., "Parallel Newton—Krylov Solver for the Euler Equations Discretized Using Simultaneous-Approximation Terms," *AIAA Journal*, Vol. 46, No. 11, 2008, pp. 2773–2786. doi:10.2514/1.34810

[53] Barche. J. (ed.), "Experimental Data Base for Computer Program Assessment," AGARD Advisory Rept. 138, May 1979.

W. Anderson
Associate Editor