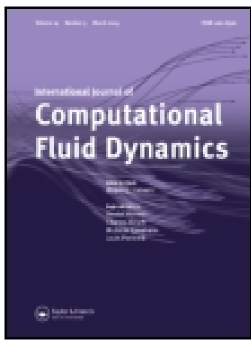

Taylor & Francis
Taylor & Francis Group

International Journal of Computational Fluid Dynamics

ISSN: 1061-8562 (Print) 1029-0257 (Online) Journal homepage: https://www.tandfonline.com/loi/gcfd20

A methodology for the development of discrete adjoint solvers using automatic differentiation tools

A. C. Marta , C. A. Mader , J. R. R. A. Martins , E. Van der Weide & J. J. Alonso

To cite this article: A. C. Marta , C. A. Mader , J. R. R. A. Martins , E. Van der Weide & J. J. Alonso (2007) A methodology for the development of discrete adjoint solvers using automatic differentiation tools, International Journal of Computational Fluid Dynamics, 21:9-10, 307-327, DOI: 10.1080/10618560701678647

To link to this article: https://doi.org/10.1080/10618560701678647

Published online: 07 Dec 2007.

Submit your article to this journal

Article views: 449

View related articles

Citing articles: 5 View citing articles

Full Terms & Conditions of access and use can be found at
https://www.tandfonline.com/action/journalInformation?journalCode=gcfd20

International Journal of Computational Fluid Dynamics, Vol. 21, Nos. 9–10, October–December 2007, 307–327

Taylor & Francis
Taylor & Francis Group

A methodology for the development of discrete adjoint solvers using automatic differentiation tools

A. C. MARTA†*, C. A. MADER‡, J. R. R. A. MARTINS‡, E. VAN DER WEIDE† and J. J. ALONSO†

†Stanford University, Stanford, CA 94305, USA
‡University of Toronto, Toronto, Ont., Canada M3H 5T6

(Received 16 May 2007; revised 29 August 2007; in final form 9 September 2007)

A methodology for the rapid development of adjoint solvers for computational fluid dynamics (CFD) models is presented. The approach relies on the use of automatic differentiation (AD) tools to almost completely automate the process of development of discrete adjoint solvers. This methodology is used to produce the adjoint code for two distinct 3D CFD solvers: a cell-centred Euler solver running in single-block, single-processor mode and a multi-block, multi-processor, vertex-centred, magneto-hydrodynamics (MHD) solver. Instead of differentiating the entire source code of the CFD solvers using AD, we have applied it selectively to produce code that computes the transpose of the flux Jacobian matrix and the other partial derivatives that are necessary to compute sensitivities using an adjoint method. The discrete adjoint equations are then solved using the Portable, Extensible Toolkit for Scientific Computation (PETSc) library. The selective application of AD is the principal idea of this new methodology, which we call the AD adjoint (ADjoint). The ADjoint approach has the advantages that it is applicable to any set of governing equations and objective functions and that it is completely consistent with the gradients that would be computed by exact numerical differentiation of the original discrete solver. Furthermore, the approach does not require hand differentiation, thus avoiding the long development times typically required to develop discrete adjoint solvers for partial differential equations, as well as the errors that result from the necessary approximations used during the differentiation of complex systems of conservation laws. These advantages come at the cost of increased memory requirements for the discrete adjoint solver. However, given the amount of memory that is typically available in parallel computers and the trends toward larger numbers of multi-core processors, this disadvantage is rather small when compared with the very significant advantages that are demonstrated. The sensitivities of drag and lift coefficients with respect to different parameters obtained using the discrete adjoint solvers show excellent agreement with the benchmark results produced by the complex-step and finite-difference methods. Furthermore, the overall performance of the method is shown to be better than most conventional adjoint approaches for both CFD solvers used.

Keywords: Partial differential equations; Adjoint solver; Automatic differentiation; Discrete adjoint; Sensitivities; Gradient-based optimization

1. Introduction

Adjoint methods have been used to perform sensitivity analysis of partial differential equations (PDEs) for over three decades. The first application to fluid dynamics was due to Pironneau (1974). The method was then extended by Jameson to perform aerofoil shape optimization Jameson (1988) and since then it has been used to design laminar flow aerofoils (Driver and Zingg 2006) and to optimize aerofoils suitable for multi-point operation (Nemec and Zingg 2004). The adjoint method has been extended to 3D problems, leading to applications such as the aerodynamic shape optimization of complete aircraft configurations (Reuther *et al.* 1999a,b), as well as aero-structural design (Martins *et al.* 2004). The adjoint theory has since been generalized for multidisciplinary systems (Martins *et al.* 2005) and for magneto-hydrodynamic (MHD) problems, using both the ideal model (Marta and Alonso 2006a) and the low magnetic Reynolds number approximation (Marta and Alonso 2006b).

The adjoint method is extremely valuable because it provides a very efficient method to compute the sensitivity

*Corresponding author. Email: acmarta@stanford.edu

ISSN 1061-8562 print/ISSN 1029-0257 online © 2007 Taylor & Francis
http://www.tandf.co.uk/journals
DOI: 10.1080/10618560701678647

of a given function of interest with respect to many parameters. When using gradient-based optimization algorithms, the efficiency and accuracy of the sensitivity computations have a significant effect on the overall performance of the optimization. Thus, having an efficient and accurate sensitivity analysis capability is very important to construct high-fidelity (and possibly multi-disciplinary) design capabilities.

Given the value of adjoint methods, it seems odd that their application to aerodynamic shape optimization is not more ubiquitous. In fact, while adjoint methods have already found their way into commercial structural analysis packages, they have yet to proceed beyond research CFD solvers. One of the main obstacles is the complexity involved in the development and implementation of adjoint methods for nonlinear PDEs. For example, the development of an approximate, continuous adjoint solver for the Reynolds-Averaged Navier–Stokes equations can require well over a year of tedious work. This is true even with significant approximations that have been shown to have a detrimental effect on the accuracy of the sensitivity derivatives (Dwight and Brezillon 2006).

The solution to this problem might be *automatic differentiation* (AD) (Griewank 2000). The development of AD tools has been done mainly in the computational and applied mathematics field, in particular as extensions to high-level programming language compilers (Horwedel 1991, Naumann and Riehme 2005). The AD approach relies on a tool that, given the original solver, creates code capable of computing the derivatives of quantities computed by the solver with respect to a number of parameters (that are typically inputs to the solver). There are two different modes of operation for AD tools: the forward and the reverse modes. The forward mode propagates the required sensitivity at the same time as the solution is being computed. To use the reverse mode directly applied to the solver, this has to be run to convergence first, with intermediate variable values stored for every iteration. These intermediate variables are then used by the reverse version of the code to find the sensitivities. The forward mode is analogous to the finite-difference method, but without problems of step-size sensitivity. The reverse mode is similar to the adjoint method and is also efficient when computing the sensitivity of a function with respect to many parameters.

The direct application of AD to CFD solvers has been tested for several years now. Forth and Evans (2002) applied AD in forward mode to an Euler code to compute sensitivities of a few functions with respect to over a dozen variables. However, they addressed the need of using the reverse mode to improve the run time if a large number of design variables had to be handled. One drawback of the reverse mode is that the memory requirements can be prohibitively expensive in the case of iterative solvers, such as those used in CFD, because they require a large number of iterations to achieve convergence and many intermediate results may need to be stored. Other approaches include the incremental iterative method (Sherman *et al.* 1996), that takes advantage of the incremental solution algorithm of the original flow solver to obtain higher computational efficiency. Although efforts have been pursued to minimize the increase in memory requirements arising from iterative solvers (Carle *et al.* 1998, Fagan and Carle 2005), the fact remains that given typical parallel computing resources, it is still very difficult to directly apply reverse mode ideas to large-scale problems. The reverse mode of AD has been applied to iterative PDE solvers by a few researchers with limited success (Mohammadi *et al.* 1996, Cusdin and Müller 2005, Giering *et al.* 2005, Heimbach *et al.* 2005). The main problem in each of these applications was the prohibitive memory requirements for the solution of 3D problems.

Our goal is to turn the development of discrete adjoint solvers into a routine and quick task that only requires the use of pre-existing code to compute the residuals of the governing equations and the cost functions (Martins *et al.* 2006a,b). The end goal of this effort is to enable the development of discrete adjoint solvers in days, rather than years, so that the adjoint information may be used in a variety of applications including sensitivity analysis and error estimation.

To achieve this goal, we propose the *AD adjoint* (ADjoint) approach, where AD is used to compute only certain terms of the discrete adjoint equations, and not to differentiate the entire solver. These differentiated terms are then used to populate the discrete adjoint system of equations that, together with standard techniques for the iterative solution of large linear systems such as the preconditioned Generalized Minimum Residual (GMRES) algorithm (Saad and Schultz 1986), allow for sensitivity analysis. This selective and careful use of AD in building the adjoint equations constitutes the novel concept that the authors believe addresses the current problems of using pure AD or adjoint methods to estimate sensitivities.

The major advantages of the ADjoint method are that it is almost entirely automatic, applicable to any PDE solver and exactly consistent. Because the process of AD allows us to treat arbitrary expressions exactly, we are able to produce sensitivities that are perfectly consistent with those that would be obtained with an exact numerical differentiation of the original solver. Thus, typical approximations made in the development of adjoint solvers (such as neglecting contributions from the variations resulting from turbulence models, spectral radii, artificial dissipation and upwind formulations) are not made here.

While the ADjoint method does not constitute a fully automatic way of obtaining sensitivities like pure AD, it is much faster in terms of execution time and drastically reduces the memory requirements.

In the next section, we review the background material that is relevant to our work, namely analytic sensitivity analysis methods and AD. We then discuss how the ADjoint approach was implemented in two distinct flow

solvers. In the results section, we establish the precision of the desired sensitivities and analyse the performance of the ADjoint method.

2. Background

2.1 Optimal design

In the context of optimization, a generic design problem can be posed as the minimization of a function of interest, I, (also called cost function or figure of merit) with respect to a vector of design variables x, while satisfying a set of constraints. The cost function depends directly on the design variables and on the state of the system, w, that may result from the solution of a governing equation. Thus, we can write the vector-valued function I as

$$I = I(x, w). \tag{1}$$

For a given vector x, the solution of the governing equations subject to appropriate boundary conditions (BC) yields a vector, w, thus establishing the dependence of the state of the system on the design variables. We denote these governing equations by

$$\mathcal{R}(x, \mathbf{w}(x)) = 0. \tag{2}$$

In mathematical terms, this design problem can be expressed as

$$\begin{aligned} &\text{Minimize } \; I(x, \mathbf{w}(x)) \;\text{ w.r.t } \; x, \\ &\text{subject to } \; \mathcal{R}(x, \mathbf{w}(x)) = 0 \\ &C_i(x, \mathbf{w}(x)) = 0 \;\; i = 1, \ldots, m, \end{aligned} \tag{3}$$

where $C_i(x, \mathbf{w}(x)) = 0$ represents m additional constraints that may or may not involve the flow solution.

When using a gradient-based optimizer to solve the design problem (3), the sensitivity of both the cost function I and the constraints C_i with respect to the design variables x are required, that is, $\mathrm{d}I/\mathrm{d}x$ and $\mathrm{d}C_i/\mathrm{d}x$ have to be determined.

2.2 Semi-analytic sensitivity analysis

Semi-analytic methods are usually capable of computing derivatives with the same precision as the quantity that is being differentiated and can also be very efficient.

Our intent is to calculate the sensitivity of the function of interest (or vector of functions) with respect to a very large number of design variables.

As a first step towards obtaining the derivatives that we ultimately want to compute, we use the chain rule to write the total sensitivity of the vector-valued function I as

$$\frac{\mathrm{d}I}{\mathrm{d}x} = \frac{\partial I}{\partial x} + \frac{\partial I}{\partial w}\frac{\mathrm{d}w}{\mathrm{d}x}, \tag{4}$$

where the sizes of the sensitivity matrices are

$$\frac{\partial I}{\partial x}(N_I \times N_x), \quad \frac{\partial I}{\partial w}(N_I \times N_w), \quad \frac{\mathrm{d}w}{\mathrm{d}x}(N_w \times N_x), \tag{5}$$

N_I is the number of functions of interest, N_x the number of design variables and N_w the size of the state vector, which for the solution of a large, 3D problem involving a system of conservation laws, can be very large.

It is important to distinguish the total and partial derivatives in these equations. The partial derivatives can be directly evaluated by varying the denominator and re-evaluating the function in the numerator *with everything else held constant*. The total derivatives however, require the solution of the governing equations. Thus, for typical cost functions, all the terms in the total sensitivity equation (4) can be computed with relatively little effort except for $\mathrm{d}w/\mathrm{d}x$.

Since the governing equations must always be satisfied, the total derivative of the residuals (2) with respect to any design variable must also be zero. Expanding the total derivative of the governing equations with respect to the design variables we can write,

$$\frac{\mathrm{d}\mathcal{R}}{\mathrm{d}x} = \frac{\partial \mathcal{R}}{\partial x} + \frac{\partial \mathcal{R}}{\partial w}\frac{\mathrm{d}w}{\mathrm{d}x} = 0. \tag{6}$$

This expression provides the means for computing the total sensitivity of the state variables with respect to the design variables, $\mathrm{d}w/\mathrm{d}x$. To this end, we rewrite equation (6) as

$$\frac{\partial \mathcal{R}}{\partial w}\frac{\mathrm{d}w}{\mathrm{d}x} = -\frac{\partial \mathcal{R}}{\partial x}, \tag{7}$$

where we have defined the following sensitivity matrices,

$$\frac{\partial \mathcal{R}}{\partial w}(N_w \times N_w), \quad \frac{\partial \mathcal{R}}{\partial x}(N_w \times N_x). \tag{8}$$

Thus, the sensitivity analysis problem given by equations (4) and (7) can be written as,

$$\frac{\mathrm{d}I}{\mathrm{d}x} = \frac{\partial I}{\partial x} + \frac{\partial I}{\partial w}\frac{\mathrm{d}w}{\mathrm{d}x}, \quad \text{such that } \; \frac{\partial \mathcal{R}}{\partial w}\frac{\mathrm{d}w}{\mathrm{d}x} = -\frac{\partial \mathcal{R}}{\partial x}. \tag{9}$$

The final sensitivity can be also obtained by solving the dual of this problem (Giles and Pierce 2000). The dual problem can be derived as follows. If we substitute the solution of the linear system (7) into the total sensitivity equation (4) we obtain *hl*

$$\frac{\mathrm{d}I}{\mathrm{d}x} = \frac{\partial I}{\partial x} - \frac{\partial I}{\partial w}\left[\frac{\partial \mathcal{R}}{w}\right]^{-1}\frac{\partial \mathcal{R}}{\partial x}. \tag{10}$$

Defining $\psi^{\mathrm{T}} = (\partial I)/(\partial w)[\partial \mathcal{R}/\partial w]^{-1}$, whose size is $(N_w \times N_I)$, we can write the problem as

$$\frac{\mathrm{d}I}{\mathrm{d}x} = \frac{\partial I}{\partial w} - \psi^{\mathrm{T}} \frac{\partial \mathcal{R}}{\partial x}, \quad \text{such that} \quad \left[\frac{\partial \mathcal{R}}{\partial w}\right]^{\mathrm{T}} \psi = \left[\frac{\partial I}{\partial w}\right]^{\mathrm{T}}. \tag{11}$$

The most computationally intensive step for both of these problems (9,11) is the solution of the corresponding linear systems. In the case of the original problem (9)—the *direct method*—we have to solve a linear system of N_w equations N_x times. For the dual problem (11)—the *adjoint method*—we solve a linear system of the same size N_I times. Thus, the choice of which of these methods to use depends largely on how the number of design variables, N_x, compares to the number of functions of interest, N_I. When $N_I >> N_x$, the adjoint approach of equation (11) is the logical choice. If, instead, $N_x >> N_I$, then the direct method should be used.

When it comes to implementation, there are two main ways of obtaining the adjoint equations (11) for a given system of PDEs. The *continuous adjoint approach* forms a continuous adjoint problem from the governing PDEs and then discretizes this problem to solve it numerically. The *discrete adjoint approach* first discretizes the governing PDE and then derives an adjoint system for these discrete equations. Each of these approaches results in a different system of linear equations that, in theory, converge to the same result as the mesh is refined.

The discrete approach has the advantage that it can be applied to *any* set of governing equations, it can treat arbitrary cost functions, and the sensitivities are consistent with those produced by the discretized solver. Furthermore, it is easier to obtain the appropriate BC for the adjoint solver in a discrete fashion. In this work, we adopt the discrete approach. Although our automated way of constructing discrete adjoint solvers will usually require more memory than the continuous adjoint, it is our opinion that the advantages mentioned earlier greatly outweigh this disadvantage.

2.3 CFD adjoint equations

We will now derive the adjoint equations for the particular case of our two distinct Euler and MHD flow solvers.

The governing PDEs for the 3D Euler equations are, in conservation form,

$$\frac{\partial w}{\partial t} + \frac{\partial \mathbf{E}}{\partial x} + \frac{\partial \mathbf{F}}{\partial y} + \frac{\partial \mathbf{G}}{\partial z} = 0, \tag{12}$$

where w is the vector of conservative variables and $\mathbf{E}$, $\mathbf{F}$ and $\mathbf{G}$ are the inviscid fluxes in the x, y and z directions, respectively, defined as

$$\mathbf{w} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{pmatrix}, \quad \mathbf{E} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ \rho H u \end{pmatrix}, \quad \mathbf{F} = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ \rho H v \end{pmatrix} \text{ and } \mathbf{G} = \begin{pmatrix} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + p \\ \rho H w \end{pmatrix}, \tag{13}$$

where ρ is the density, u, v and w are the Cartesian velocity components, p is the static pressure and H is the total enthalpy, which is related to the total energy by $H = E + (p/\rho)$.

Our second solver implements a version of the MHD equations. The equations governing the 3D flow of a compressible conducting fluid in a magnetic field are obtained by coupling the Navier–Stokes equations to the Maxwell equations. If the environment of interest is characterized by a low magnetic Reynolds number, then the magnetic field induced by the current is much smaller than that imposed on the flow and therefore it can be neglected (Gaitonde 2005). In this way, there is no need to solve the three induction equations in the governing equations and the electromagnetic forces and energy show up as source terms in the Navier–Stokes equations. Furthermore, if the viscous effects and heat transfer are neglected, the Navier–Stokes equations can be simplified to the Euler equations. In this case, the non-dimensional MHD equations governing the flow are given by equation (12) with a source term $\mathbf{S}$ added to the right-hand side (RHS). $\mathbf{S}$ includes the magnetic field terms and can be shown to be

$$\mathbf{S} = \begin{pmatrix} 0 \\ Q\sigma\,[B_z(E_y + wB_x - uB_z) - B_y(E_z + uB_y - vB_x)] \\ Q\sigma\,[B_x(E_z + uB_y - vB_x) - B_z(E_x + vB_z - wB_y)] \\ Q\sigma\,[B_y(E_x + vB_z - wB_y) - B_x(E_y + wB_x - uB_z)] \\ Q\sigma\,[E_x(E_x + vB_z - wB_y) + E_y(E_y + wB_x - uB_z) \\ \quad + E_z(E_z + uB_y - vB_x)] \end{pmatrix}, \tag{14}$$

where $\mathbf{B}$ is the magnetic field, $\mathbf{E}$ is the electric field, σ is the electrical conductivity and Q is the magnetic interaction parameter defined as $Q = (\sigma B^2 L)/(\rho U) = R_b Re_\sigma$. The other non-dimensional parameters found in this formulation are the magnetic force number, R_b and the magnetic

Reynolds number, Re_σ, given by

$$R_b = \frac{B^2}{\rho U^2 \mu_\mathrm{m}} \quad \text{and} \quad Re_\sigma = \mu_\mathrm{m}\sigma UL, \tag{15}$$

where μ_m represents the magnetic permeability of the medium.

It must be noted that the derivation presented here is for both the Euler and the low magnetic Reynolds number MHD equations, but since our approach is only based on the existence of a computer program that evaluates the *residual* of the governing equations, the procedure can be extended to the full Reynolds-averaged Navier–Stokes equations and the full MHD equations, respectively, without modification. Note that the code for the residual computation is assumed to include the application of the required BC (however complex they may be) and any artificial dissipation terms that may need to be added for numerical stability.

A coordinate transformation from physical coordinates $X_i = (x, y, z)$ to computational coordinates $\xi_j = (\xi, \eta, \zeta)$ is used. This transformation is defined by the metrics

$$K_{ij} = \left[\frac{\partial X_i}{\partial \xi_j}\right], \quad K_{ij}^{-1} = \left[\frac{\partial \xi_i}{\partial X_j}\right], \tag{16}$$

$$J = \det(K^{-1}), \quad S_{ij} = \frac{1}{J}K_{ij}^{-1}, \tag{17}$$

where S_{ij} represents the areas of the face in the ξ_i direction of each cell, projected on to each of the physical coordinate directions X_j.

The governing equations in computational coordinates can then be written as

$$\frac{\partial \bar{\mathbf{w}}}{\partial t} + \frac{\partial \bar{\mathbf{E}}}{\partial \xi} + \frac{\partial \bar{\mathbf{F}}}{\partial \eta} + \frac{\partial \bar{\mathbf{G}}}{\partial \zeta} = \bar{\mathbf{S}}, \tag{18}$$

where the state, the fluxes vectors and the source terms in the computational coordinates are given by

$$\begin{cases} \bar{\mathbf{w}} = \frac{\mathbf{w}}{J} \\ \bar{\mathbf{E}} = \frac{1}{J}(\xi_x\mathbf{E} + \xi_y\mathbf{F} + \xi_z\mathbf{G}) \\ \bar{\mathbf{F}} = \frac{1}{J}(\eta_x\mathbf{E} + \eta_y\mathbf{F} + \eta_z\mathbf{G}) \\ \bar{\mathbf{G}} = \frac{1}{J}(\zeta_x\mathbf{E} + \zeta_y\mathbf{F} + \zeta_z\mathbf{G}) \\ \bar{\mathbf{S}} = \frac{1}{J}\mathbf{S} \end{cases} \tag{19}$$

In semi-discrete form the governing equations are

$$\frac{\mathrm{d}w_{ijk}}{\mathrm{d}t} + \mathcal{R}_{ijk}(w) = 0, \tag{20}$$

where $\mathcal{R}$ is the residual described earlier with all of its components (inviscid fluxes, BC, artificial dissipation, etc.) and the triad *ijk* represents the three computational directions.

The adjoint equations (11) can be re-written as

$$\left[\frac{\partial \mathcal{R}}{\partial w}\right]^\mathrm{T} \psi = \left[\frac{\partial I}{\partial w}\right]^\mathrm{T}. \tag{21}$$

where ψ is the *adjoint vector*. The total sensitivity (4) in this case is

$$\frac{\mathrm{d}I}{\mathrm{d}x} = \frac{\partial I}{\partial x} - \psi^\mathrm{T}\frac{\partial \mathcal{R}}{\partial x}. \tag{22}$$

We propose to compute the partial derivative matrices $\partial\mathcal{R}/\partial w$, $\partial I/\partial w$, $\partial I/\partial x$ and $\partial\mathcal{R}/\partial x$ using AD instead of manual differentiation or finite-differences. Where appropriate we will use the reverse mode of AD.

2.4 Adjoint-based optimization algorithm

By constructing the adjoint system of equations (21) and solving for the vector of adjoint variables, ψ, the sensitivity of the cost function is simply given by equation (22).

The exact manner in which the constraints are incorporated in the design depends on the framework used: the direct incorporation is typical for a few constraints; however, if there is an extremely large number of constraints, a lumped method is advisable to reduce the number of necessary adjoint solutions. If any of the C_i constraints is active in the design space during the optimization, then an additional adjoint system has to be solved for each additional constraint function, C_i, which includes the computation of a new RHS for the system (21).

The sensitivity obtained from (22) can then be used to find the search direction of any gradient-based optimization algorithm as indicated in the block diagram in figure 1.

2.5 Automatic differentiation

AD, also known as computational differentiation or algorithmic differentiation, is based on the application of the chain rule of differentiation to computer programs. The method relies on tools that automatically produce

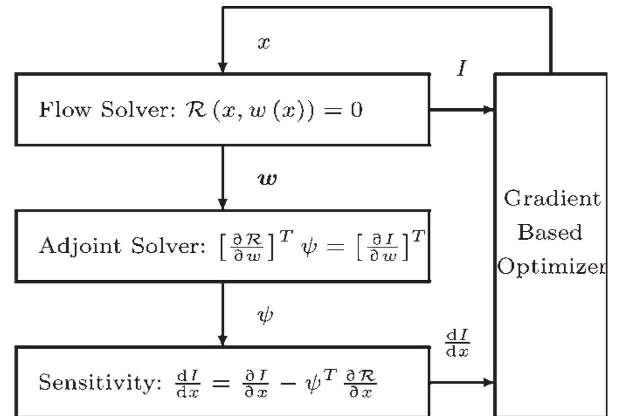

Figure 1. Schematic of the adjoint-based optimization algorithm.

a program that computes user specified derivatives based on the original program.

We denote the *independent* variables as $t_1, t_2, \ldots, t_n$, which, for our purposes, are the same as the design variables, x. We also need to consider the *dependent* variables, which we write as $t_{n+1}, t_{n+2}, \ldots, t_m$. These are all the intermediate variables in the algorithm, including the outputs, I, that we are interested in. We can then write the sequence of operations in any algorithm as

$$t_i = f_i(t_1, t_2, \ldots t_{i-1}), \quad i = n+1, n+2, \ldots, m. \quad (23)$$

The chain rule can be applied to each of these operations and is written as

$$\frac{\partial t_i}{\partial t_j} = \sum_{k=1}^{i-1} \frac{\partial f_i}{\partial t_k} \frac{\partial t_k}{\partial t_j}, \quad j = 1, 2, \ldots, n. \quad (24)$$

Using the forward mode, we choose one j and keep it fixed. We then work our way forward in the index i until we get the desired derivative. The reverse mode, on the other hand, works by fixing i, the desired quantity we want to differentiate, and working our way backwards in the index j all the way down to the independent variables.

The two modes are directly related to the direct and adjoint methods. The counterparts of the state variables in the semi-analytic methods are the intermediate variables, and the residuals are the lines of code that compute those quantities.

There are two main approaches to AD: source code transformation and operator overloading. Tools that use source code transformation add new statements to the original source code that compute the derivatives of the original statements. The operator overloading approach consists of defining a new user defined type that is used instead of real numbers. 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. The operator overloading approach results in fewer changes to the original code but is usually less efficient.

There are AD tools available for a variety of programming languages including Fortran, C/C++ and Matlab. ADIFOR (Carle and Fagan 2000), TAF (Giering and Kaminski 2002), TAMC (Gockenbach 2000) and Tapenade (Hascoët and Pascual 2004, 2005) are some of the tools available for Fortran. Of these, only TAF and Tapenade offer support for Fortran 90, which was a requirement in our case.

We chose to use Tapenade as it is the only non-commercial tool with support for Fortran 90. Tapenade is the successor of Odyssée (Faure and Papegay) and was developed at INRIA. It uses source transformation and can perform differentiation in either forward or reverse mode. Furthermore, the Tapenade team is actively developing their software and has been very responsive to a number of suggestions towards completing their support of the Fortran 90 standard.

In order to verify the results given by the ADjoint approach applied to the cell-centred Euler solver we decided to use the complex-step derivative approximation (Martins *et al.* 2003) as our benchmark. Unlike finite-differences, this method is not subject to subtractive cancellation and enables us to make much more conclusive comparisons when it comes to accuracy. However, it requires the transformation of all the real variables in the source code to complex numbers. This was easily accomplished by running a custom made Python script that automatically generated the transformed Fortran code. The complex-step formula is given by

$$\frac{dI}{dx} = \frac{\mathrm{Im}[I(x+ih)]}{h}, \quad (25)$$

where the design variable is perturbed by a small pure complex-step, which is typically less than $\mathcal{O}(10^{-20})$. A point worth noting is that the complex-step method is equivalent to the forward-mode using operator overloading (Martins *et al.* 2001).

However, the complex-step was not used to verify the ADjoint approach applied to the vertex-centred MHD solver because the script cannot handle the multi-processor functionality correctly yet. For that case, we were forced to use a forward-difference sensitivity approximation given by

$$\frac{dI}{dx} = \frac{I(x+h) - I(x)}{h}. \quad (26)$$

In this case, special care has to be taken when choosing the perturbation step h.

3. Implementation on a cell-centred Euler solver

The first solver we used as a test bed is the Stanford University multi-block (*SUmb*) flow solver (Van der Weide *et al.* 2006) that has been developed at Stanford University under the sponsorship of the Department of Energy. *SUmb* is a completely generic, finite-volume, cell-centred, multi-block solver for the Reynolds-averaged Navier–Stokes equations (steady, unsteady, and time-spectral) and it provides options for a variety of turbulence models with one, two and four equations. Since this first implementation is a proof-of-concept for the ADjoint approach, only the single-block, single-processor, Euler capabilities of the *SUmb* solver were tested.

For purposes of demonstration, the necessary steps that need to be taken to compute the derivatives of the coefficient of drag with respect to the free-stream Mach number, i.e. $I = C_D$ and $x = M_\infty$, are covered in the next subsections.

3.1 Brief summary of the discretization

The spatial discretization for the *SUmb* solver is done on a block by block basis, with each block surrounded by halo cells to facilitate the communication between the different blocks in the computational domain. Each block is discretized using a cell-centred, finite-volume scheme. The inviscid flux terms only require one level of adjacent cells, while the artificial dissipation fluxes require two levels of adjacent cells. Thus for a single cell residual calculation, this leads to the stencil shown in figure 2. The BC are enforced by altering the states in the halo cells.

3.2 Computation of $\partial \mathcal{R}/\partial w$

The flux Jacobian, $\partial R/\partial w$, is independent of the choice of function or design variable—it is simply a function of the governing equations, their discretization, and the problem BC. To compute it we need to consider the routines in the flow solver that, for each iteration, evaluate the residuals based on the values of the flow variables, w, within the discretization stencil. The computation of the residual in *SUmb* can be summarized as follows:

i) *Compute inviscid fluxes*: for our inviscid flux discretization the only flow variables, w, that influence the residual at a cell are the flow variables at that cell and at the six cells that are adjacent to the faces of the cell.
ii) *Compute artificial dissipation fluxes*: for this portion of the residual, the flow variables in the current cell and in 4 adjacent cells in each of the three directions, a total of 12, need to be considered, as shown in figure 2.
iii) *Compute viscous fluxes*: this computation requires a stencil that includes all the cells directly surrounding the cell in question, totaling 26 cells. It must be noted, however, that in this work we have chosen not to include the computation of the viscous fluxes.
iv) Apply BC.

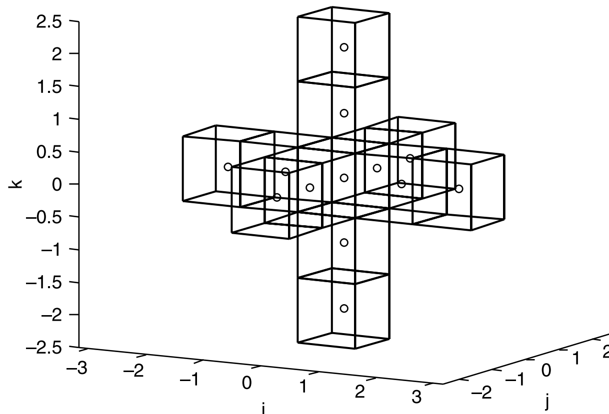

Figure 2. Stencil for the cell-centred residual computation.

In order to compute the flux Jacobian, a single cell version of the residual computation code was generated, based on the stencil shown in figure 2. Using this new single cell routine, AD was used to generate code that would compute the elements of the flux Jacobian related to the residuals in that single cell. The development of the single cell residual calculation code was straightforward and took less than two working days to obtain, including the proper BC handling. In this particular implementation, Tapenade was used to produce reverse mode differentiated code. This was done because for this stencil the number of input variables, w, is significantly larger than the number of output variables, $\mathcal{R}$, a situation for which reverse mode is far more efficient. Further, because of the sparse nature of the flux Jacobian, this computation allows us to generate an entire column of the flux Jacobian at one time. A more detailed discussion as to why the reverse mode is more efficient in this case was presented by Martins *et al.* (2006b).

3.3 Computation of $\partial C_{\mathrm{D}}/\partial w$ and other partial derivatives

The other partial derivative terms required for total sensitivities, including $\partial I/\partial w$, $\partial I/\partial x$ and $\partial \mathcal{R}/\partial x$ are calculated using a forward mode differentiation of modified versions of the original *SUmb*. In this case the forward mode was used for simplicity; however, for both cost function sensitivities, the reverse mode would be more efficient. In the case of the $\partial \mathcal{R}/\partial x$ term, forward mode is more efficient because there are fewer design variables x than residuals $\mathcal{R}$. A more detailed discussion relating to these points can be found in Martins *et al.* (2006b).

3.4 Adjoint solver

The adjoint equations (21) can be re-written for this specific case as

$$\left[\frac{\partial \mathcal{R}}{\partial w}\right]^{\mathrm{T}} \psi = \left[\frac{\partial C_{\mathrm{D}}}{\partial w}\right]^{\mathrm{T}}. \tag{27}$$

As we have pointed out, both the flux Jacobian and the RHS in this system of equations are very sparse. To solve this system efficiently, and having in mind that we want to have a parallel adjoint solver, we decided to use the PETSc library (Balay *et al.* 1997, 2004, 2006). PETSc is a suite of data structures and routines for the scalable, parallel solution of scientific applications modeled by PDEs. It employs the message passing interface (MPI) standard MPI (1994) for all interprocessor communication, it has several linear iterative solvers and preconditioners available and performs very well, provided that a careful object creation and assembly procedure is followed.

Using PETSc's data structures, $\partial \mathcal{R}/\partial w$ and $-\partial C_{\mathrm{D}}/\partial w$ were stored as sparse entities. Once the sparse matrices are

filled, PETSc's built-in preconditioned Krylov solver is used to compute the adjoint solution.

3.5 *Total sensitivity equation*

The total sensitivity (22) in this case can be written as

$$\frac{\mathrm{d}C_{\mathrm{D}}}{\mathrm{d}M_{\infty}} = \frac{\partial C_{\mathrm{D}}}{\partial M_{\infty}} - \psi^{\mathrm{T}} \frac{\partial \mathcal{R}}{\partial M_{\infty}}, \tag{28}$$

where we have chosen the free-stream Mach number, M_{∞}, to be the independent variable. Thus, with the adjoint, ψ, computed and the remaining partial terms computed, a simple matrix multiplication is performed in PETSc to compute the total derivative.

4. Implementation on a vertex-centred MHD solver

The second solver we have used is the *NSSUS* flow solver. This solver is a new finite-difference, higher-order solver that has been developed at Stanford University under the ASC (2007) program sponsored by the Department of Energy.

It is a generic node-centred, multi-block, multi-processor solver, currently only for the Euler equations, but soon to be extended to the Reynolds-Averaged Navier–Stokes equations. The finite-difference operators and artificial dissipation terms follow the work of Mattsson and Nordström (2004a), Mattsson *et al.* (2004b) and the BC are implemented by means of penalty terms, according to the work of Carpenter *et al.* (1994, 1999). The additional magnetic source terms (14) had to be included in this solver so that MHD computations could be performed. Although the solver could perform computations up to eight-order accuracy, the implementation of the adjoint solver was restricted to second-order for simplicity. The extension should be straightforward to accomplish. All the lessons learned from the previous implementation were re-used. The two major differences were the treatment of the BC and the multi-block processing, both of which are described below.

In this implementation, the lift (C_{L}) and drag (C_{D}) aerodynamic coefficients were used as the cost functions and the electrical conductivity σ in every computational node was taken as the set of design variables. As seen in the results section, this led to a total of 3,62,362 design variables for the computational mesh used.

4.1 *Brief summary of the discretization*

This section summarizes the essence of the spatial discretization, which is needed for the explanation provided in section 4.2. For more details see Mattsson and Nordström (2004a), Mattsson *et al.* (2004b) and Carpenter *et al.* (1994, 1999).

The spatial part of equation (18) is discretized on a block-by-block basis. The internal discretization is straightforward and only requires the first neighbours in each coordinate direction for the inviscid fluxes and the first and second neighbours for the artificial dissipation fluxes, see figure 3. However, the boundary treatment needs to be explained in more detail.

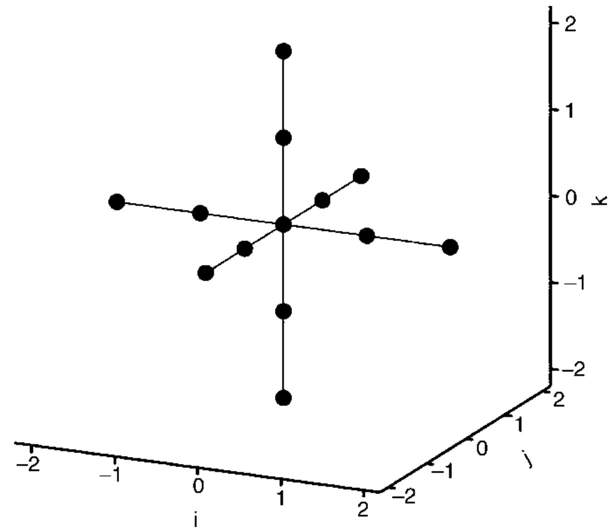

Figure 3. Stencil for the vertex-centred residual computation.

As the finite-difference scheme only operates on the nodes of a block, one-sided difference formulae are used near block boundaries (be it a physical or an internal boundary). Consequently, the nodes on internal boundaries are multiply defined, see figure 4. These multiple instances of the same physical node are driven to the same value (at convergence) by means of a penalty term, i.e. an additional term is added to the residual $\mathcal{R}$ which is proportional to the difference between the instances. This reads

$$\mathcal{R}^{i}_{\text{block A}} = \mathcal{R}^{i}_{\text{block A}} + \tau(w^{i}_{\text{block B}} - w^{i}_{\text{block A}}), \tag{29}$$

and a similar expression can be derived for $\mathcal{R}^{i}_{\text{block B}}$. In equation (29), τ controls the strength of the penalty and is a combination of a user defined parameter and the local flow conditions (see (Mattsson and Nordström 2004a) for more details). Hence, the residual of nodes on a internal block boundary is a function of its local neighbours in the block and the corresponding instance in the neighbouring block.

The BC treatment is very similar to the approach described above except that the penalty term used in equation (29) is now determined by the BC.

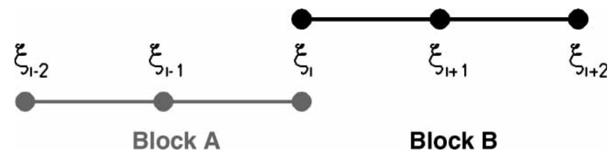

Figure 4. Block-to-block boundary stencil.

4.2 Computation of $\partial \mathcal{R}/\partial w$

In order to compute the flux Jacobian, only the routines in the flow solver that for each iteration, compute the residuals based on the flow variables w were considered. In the present case, the residual $\mathcal{R}_{ijk}$ in equation (20) is computed using the following steps:

i) *Compute inviscid fluxes*: for our inviscid flux discretization the only flow variables, w, that influence the residual at a node are the flow variables at that node and at the six nodes that are first neighbours to the node along each computational direction.
ii) *Compute artificial dissipation fluxes*: for each of the N_n nodes in the domain, compute the contributions of the flow variables to the residual at that node. For this portion of the residual, the flow variables in the current node and in the first- and second-neighbour nodes in each of the three coordinate directions need to be considered.
iii) *Compute magnetic source terms*: these only depend on the flow variables at the current node.
iv) *Apply BC*: additional penalty terms are added to enforce the BCs, see section 4.1. Note that internal block boundaries are also considered as boundaries.

Similarly to the *SUmb* solver, the baseline implementation of *NSSUS* computes the residual using nested loops over the nodes of each computational block on each of the processors used in the calculation. To make the implementation of the discrete adjoint solver more efficient, it was necessary to re-write the flow residual routine such that it computed the residual for a single specified node. These routines were developed by cutting and pasting from the original flow solver routines, a process which turned out to be straightforward and that could be completed in roughly one day of work. The re-engineered residual routine is a function with the residuals at a given node, `rAdj`, is returned as an output argument, and the stencil of flow variables, `wAdj`, that affect the residuals at that node is provided as an input argument,

```
subroutine residualAdj (wAdj,rAdj,i,j,k),
```

for each node in each block on each processor. Any required Fortran pointers were set before calling the routine due to the current pointer handling limitations in Tapenade.

The stencil of flow variables `wAdj` that affects the residual `rAdj` in a given node `(i,j,k)` extends two nodes in each direction to allow for a second-order discretization as described in section 4.1. In this case, the number of nodes in the stencil whose variables affect the residual of a given node is $N_s = 13$. This re-engineered residual routine `residualAdj` computes N_w residuals in a given node that depend on all $(N_w \times N_s)$ flow variables in the stencil.

The BC penalty terms were moved to a separate routine,

```
subroutine residualPenaltyAdj (wAdj,rAdj,mm,
wDonorAdj,i,j,k).
```

This routine has two additional input parameters: the boundary sub-face, `mm` and the penalty state of the corresponding donor node, `wDonorAdj`. Naturally, this penalty residual routine was only called when the given node `(i,j,k)` was located on a boundary face, see section 4.1.

In order to properly treat the multi-block grid, it was necessary to setup a global node numbering scheme and preserve the node connectivity. This allowed for the proper block boundary treatment when assembling the flux Jacobian matrix.

Having verified the re-written residual routines by comparing the residual values to the ones computed with the original code, these routines were then fed into Tapenade for differentiation. For the same reasons as the ones explained in the previous section, the AD was performed using the reverse mode on the set of routines `residualAdj` and `residualPenaltyAdj` that, given a node location, compute the residuals of that node only. The AD process produces the differentiated routines

```
subroutine   residual_B   (wAdj,wAdjB,rAdj,
rAdjB,i,j,k)
```

and

```
subroutine              residualPenaltyAdj_B
(wAdj,wAdjB,rAdj,rAdjB,mm,wDonorAdj,
wDonorAdjB,i,j,k)
```

.

The $N_w \times (N_w \times N_s)$ sensitivities that need to be computed for each node, corresponding to N_w rows in the $(\partial \mathcal{R})/(\partial W)$ matrix, are readily computed by these automatically differentiated routines in reverse mode. This is accomplished because all the derivatives in the stencil can be calculated from one residual evaluation `residualAdj_B` since, for each node (i,j,k) and governing equation m,

$$\mathrm{wAdjB(ii,jj,kk,n)} = \frac{\partial \mathcal{R}(i,j,k,m)}{\partial w(i+\mathrm{ii}, j+\mathrm{jj}, k+\mathrm{kk}, \mathrm{n})}. \quad (30)$$

where the triad `(ii,jj,kk)` spans the stencil and `n` spans the N_w flow variables. Similarly, the penalty term contributions are given by the outputs `wAdjB` and `wDonorAdjB` of `residualPenaltyAdj_B`.

This allows for an easy assembly of the flux Jacobian matrix $(\partial \mathcal{R})/(\partial w)$ as shown in the routine pseudo-code in figure 5.

4.3 Computation of $\partial C_{\mathrm{D}}/\partial w$

The RHS of the adjoint equations (21) for the functions of interest C_{D} and C_{L} is easily computed for this flow solver. Because this specific flow solver works with primitive variables $w = (\rho, u, v, w, p)$ and since, for inviscid flow, C_{D} and C_{L} are simple surface integrations of the pressure, then the derivatives $\partial C_{\mathrm{D}}/\partial w$ and $\partial C_{\mathrm{L}}/\partial w$ are always zero except for $w_5 (= p)$. Therefore, it became trivial to derive analytically the expression for these partial derivatives

```
subroutine setupADjointMatrix
(...)
! Loop over the local computational blocks.
do nn=1,nDom
  ! Loop over location of output (R) cell of residual
  do k=1,kl
    do j=1,jl
      do i=1,il
        ! Global node number
        idxmgb = globalNode(i,j,k)
        ! Transfer state w to auxiliar stencil array wAdj(:,:,:,:)
        call copyADjointStencil(wAdj,i,j,k)
        ! Loop over the outputs (R)
        do m=1,nwFlow
          ! Initialize the seed for the reverse mode to return dR(m)/dw
          rAdjB(:)=zero; rAdjB(m)=one; rAdj(:)=zero; wAdjB(:,:,:,:)=zero
          ! Call reverse mode of residual computation
          call residualAdj_B(wAdj,wAdjB,rAdj,rAdjB,i,j,k)
          ! Store block Jacobians (by rows).
          Aad(m,:) = wAdjB(0,0,0,:); Bad(m,:) = wAdjB(-1,0,0,:)
          (...)
        enddo
        ! Transfer block Jacobians to PETSc matrix.
        ! >>> center block dR(i,j,k)/W(i,j,k)
        idxngb = idxmgb
        call MatSetValuesBlocked(dRdW,1,idxmgb,1,idxngb,Aad,INSERT_VALUES,ierr)
        ! >>> west block B < W(i-1,j,k)
        idxngb = globalNode(i-1,j,k)
        call MatSetValuesBlocked(dRdW,1,idxmgb,1,idxngb,Bad,INSERT_VALUES,ierr)
        (...)
```

Figure 5. Flux Jacobian matrix assembly routine.

from the flow solver routine that calculated the functions of interest.

4.4 Adjoint solver

With both the Jacobian flux matrix $(\partial\mathcal{R})/(\partial w)$ and the RHS $\partial C_{\mathrm{D}}/\partial w$ (or $\partial C_{\mathrm{L}}/\partial w$) computed, we then solved the adjoint equations (27).

Similarly to section 3.4, the large discrete adjoint system of equations was solved using PETSc, taking advantage of its seamless MPI implementation. All the partial sensitivity matrices and vectors were created as PETSc objects, and the adjoint system of equations (27) was solved using a Krylov subspace method. More specifically, a GMRES method was used, preconditioned with the block Jacobi method, with one block per processor, each of which solved with ILU(0) preconditioning.

4.5 Total sensitivity equation

The total sensitivity (22) in this case can be written as

$$\frac{\mathrm{d}C_{\mathrm{D}}}{\mathrm{d}\sigma} = \frac{\partial C_{\mathrm{D}}}{\partial \sigma} - \psi^{\mathrm{T}}\frac{\partial \mathcal{R}}{\partial \sigma}, \tag{31}$$

where we have chosen the electrical conductivity, σ, to be the independent variable. Thus, with the adjoint ψ computed, it is necessary to evaluate two additional terms to form the total sensitivity equation—$\partial C_{\mathrm{D}}/\partial\sigma$ and $\partial\mathcal{R}/\partial\sigma$.

The first term is easy to evaluate since there is no direct dependence on C_{D} with respect to the electrical conductivity σ. As such, that term is identically zero.

The second term is also easy to obtain by looking at the MHD governing equations, in particular to the magnetic source term (14)—the residual $\mathcal{R}$ shows a linear dependence of σ through the source term S. Thus, this term was computed analytically, the result being a very sparse matrix, with non-zero entries only along the diagonal since the residual at a given node depends only on the conductivity of that same node.

The total sensitivity (31) was then computed using the matrix-vector multiplication and the vector addition routines provided in PETSc, as implemented in the structure of the adjoint solver code, despite the simple algebra involved in the present test case.

5. Results

The ADjoint approach is used in two different flow solvers: a cell-centred Euler solver and a vertex-centred MHD solver. Specific test cases are used to demonstrate our method for each solver.

5.1 Cell-centred single-processor Euler solver

5.1.1 Description of test case. The test case solved with *SUmb* is a 3D bump intended to simulate the top canopy of a solar car. The mesh contains a pair of bumps in the

middle of a rectangular flow domain. The larger, flatter bump represents the main body of the vehicle while the smaller, more pronounced bump in the middle represents the driver's canopy. The bottom face of the flow domain is set as an Euler (inviscid) wall, while the other five faces of the flow domain are modeled with far-field BC. The free-stream flow conditions are set at Mach 0.5 at sea level standard conditions.

The mesh for this test case is shown in figure 6. This case has a relatively small mesh of 72,000 cells ($30 \times 30 \times 80$) for a total of 3,60,000 flow variables. The run was made on a parallel processor machine, with 32 processors of 1.5 GHz each and a total of 32GB shared RAM. Due to the adjoint solver implementation, the test case was run only on a single-processor.

5.1.2 Flux Jacobian. As mentioned in section 3, to reduce the cost of the AD procedure, we wrote a set of subroutines that compute the residual of a single, specified cell. This set of subroutines closely resembles the original code used to compute the residuals over the whole domain, except that it considers only a single cell at a time.

We differentiated the single cell residual calculation routines with Tapenade using the reverse mode, with the residuals specified as the outputs and the flow variables specified as the inputs. The resulting code computes the sensitivity of one residual of the chosen cell with respect to all the flow variables in the stencil of that cell, i.e. all the nonzero terms in the corresponding row of $\partial R/\partial w$. To obtain the full flux Jacobian, a loop over each of the residual and cell indexes in each of the three coordinate directions is necessary.

5.1.3 Other partial derivative terms. In the process of developing our adjoint solver we visualized the various partial derivative terms that appear in the adjoint and total sensitivity equations. Figure 7 shows the sensitivity of a force coefficient with respect to the density, a state variable, on the surface of the bump. Only the surface is shown because the only non-zero terms in the grid are located in the two layers of cells adjacent to the Euler wall surface, which is what we expected.

The term $\partial R/\partial M_\infty$, which is needed for the total sensitivity equation (28), is shown in figure 8. The figure shows the sensitivity corresponding to the residual of the continuity equation. Since the flow is purely in the x direction, M_∞ affects only the cells near the inflow and outflow planes.

5.1.4 Lift and drag coefficient sensitivities. The benchmark sensitivity results were obtained using the complex-step derivative approximation (25), which is numerically exact. That is to say that the precision of the

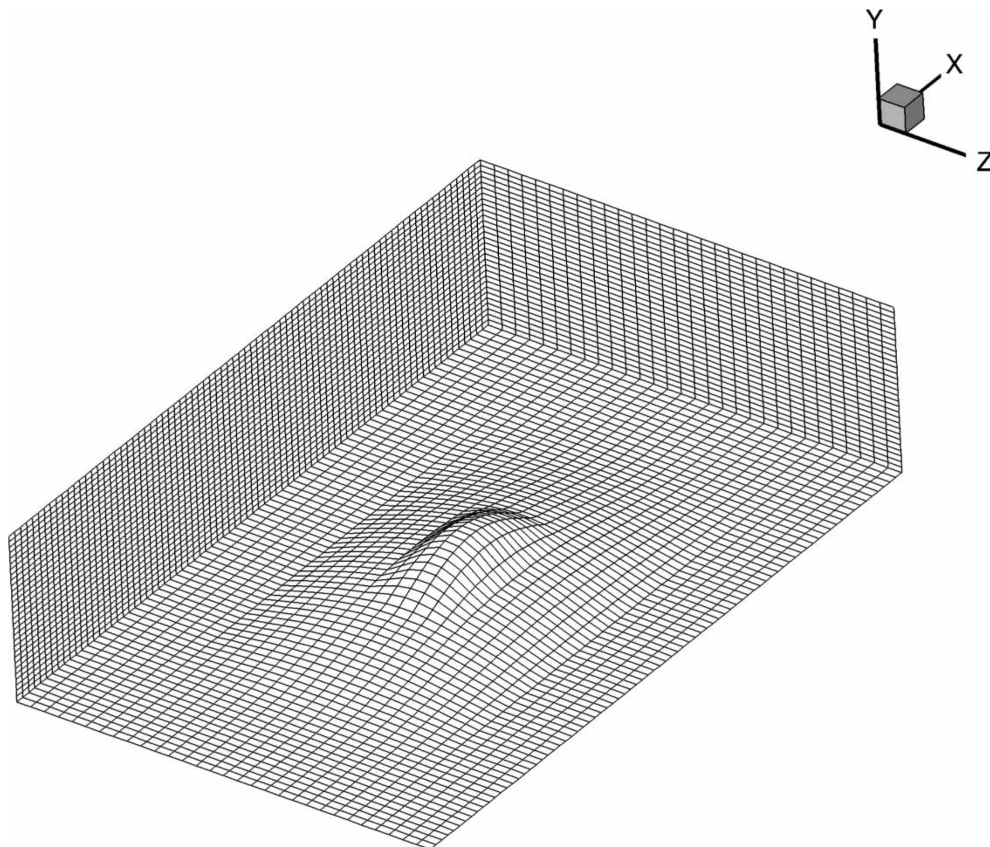

Figure 6. Mesh for 3D bump test case.

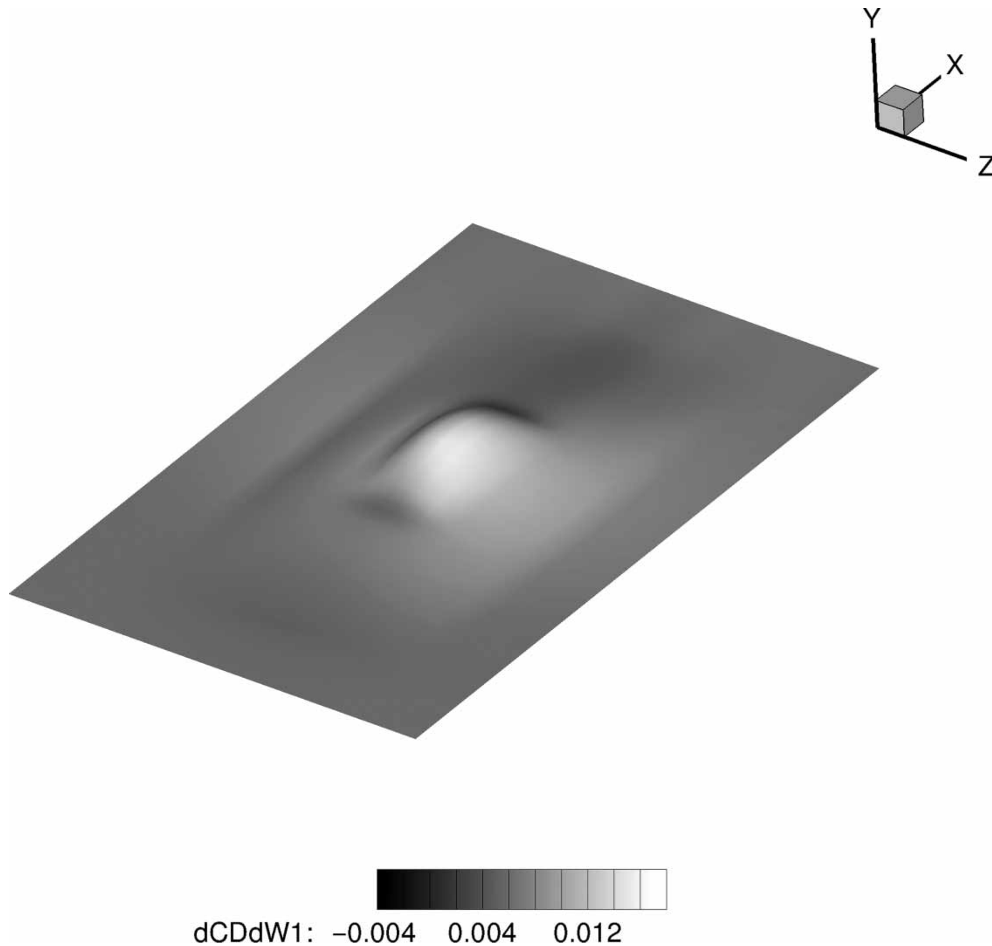

Figure 7. Partial sensitivity of C_D with respect to density.

sensitivity is of the same order as the precision of the solution. The derivative in this case is given by,

$$\frac{dC_D}{dM_\infty} = \frac{\text{Im}[C_D(M_\infty + ih)]}{h}, \tag{32}$$

where h represents the magnitude of the complex-step, for which a value of $h = 10^{-20}$ was used. The sensitivities given by the complex-step were spot-checked against finite-differences by doing a step size study.

Table 1 shows the sensitivities of the force coefficients with respect to free-stream Mach number, at an inlet angle of 0°. We can see that the adjoint sensitivities are extremely accurate, yielding nine and seven digits agreement, respectively, when compared to the complex-step results. This is consistent with the convergence tolerance that was specified in PETSc for the adjoint solution—in this test case, both the flow and adjoint solvers were converged ten orders of magnitude.

The timing results for the ADjoint solver on the 3D bump case are shown in table 2. The total cost of the adjoint solver, including the computation of all the partial derivatives and the solution of the adjoint system, is approximately one fifth of the cost of the flow solution for the bump case. This is even better than what is usually observed in the case of adjoint solvers developed by conventional means, showing that the ADjoint approach is indeed very efficient.

When it comes to comparing the performance of the various components in the adjoint solver, we found that most of the time was spent in the solution of the adjoint equations, thus showing that the AD sections are very efficient. The costliest of the AD routines was the computation of the flux Jacobian. When one takes into consideration the number of terms in this matrix, spending only 3% of the flow solution time in this computation is very impressive.

The computation of the RHS ($\partial C_D/\partial w$) is not much cheaper than computing the flux Jacobian. However, this is likely due to the use of forward mode instead of reverse mode differentiation in the computation of this term. We expect to improve the computation of this term significantly in the future.

The measurement of the memory requirements of this code, shown in table 3, indicate that the memory required for the ADjoint code is approximately twenty times that required for the original flow solver. No special care was taken regarding this issue at this point but, as seen in the multi-processor implementation this ratio can be considerably improved.

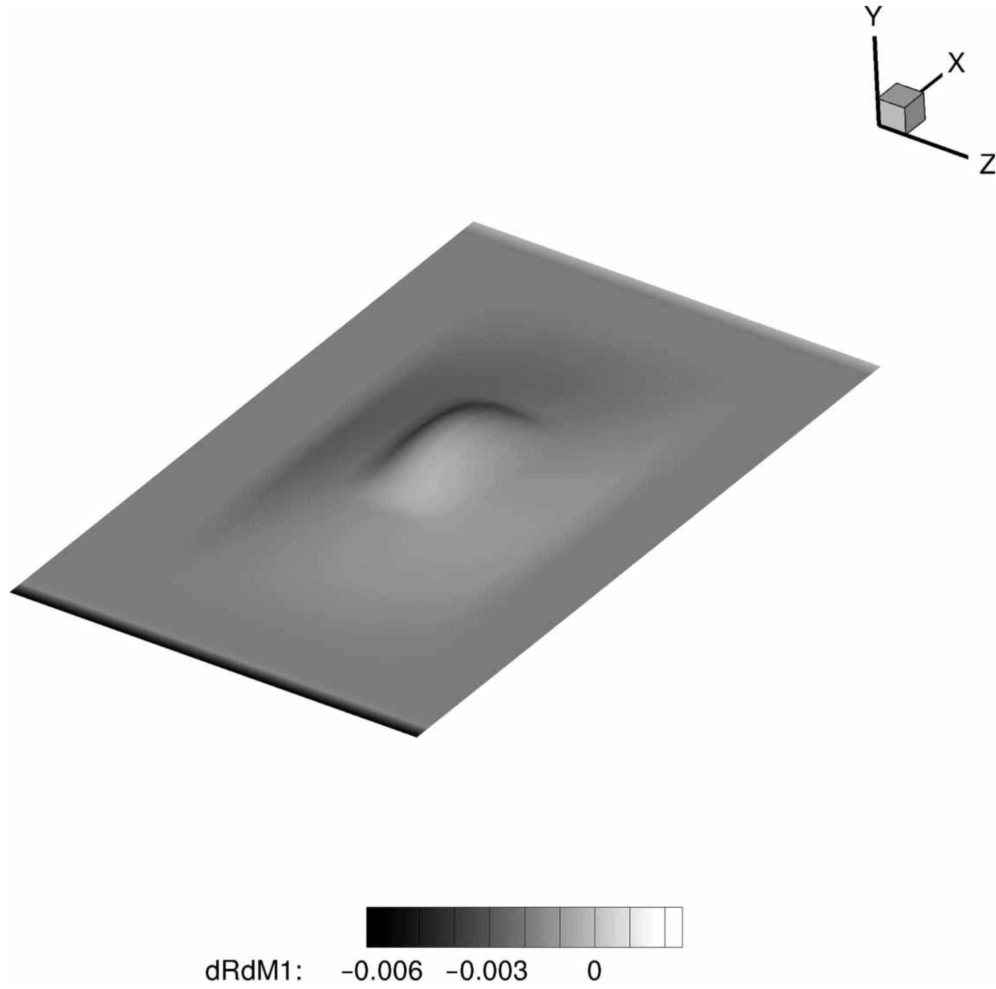

Figure 8. Partial sensitivity of the continuity residual with respect to M_∞.

5.2 Vertex-centred multi-processor MHD solver

5.2.1 Description of test case. The hypersonic vehicle geometry used for the second test case is a generic vehicle inspired by the NASA X-43A aircraft (NASA 2006), which is a scramjet-powered research aircraft capable of flying at Mach 10 and is shown in figures 9 and 10.

Since the simulation was run without any side-slip angle, only half the body had to be modeled, with a symmetry BC imposed on the centre plane. The body wall was set to be an impermeable Euler wall and the inflow and outflow surfaces of the surrounding domain had supersonic BC imposed on them. The free-stream flow conditions chosen were Mach 5 and an angle of attack of five degrees.

To simulate the MHD interaction a collection of five hypothetical dipoles was placed inside the body, at the locations indicated in table 4, which imposed a magnetic field on the flow. The vehicle is 20 m long and the coordinate origin is located at the nose of the vehicle, as shown in figure 12. Each dipole produced a magnetic field given by $B = (\mu_{\rm m} m)/(4\pi r^3)[2\cos\theta \mathrm{e}_r + \sin\theta \mathrm{e}_\theta]$, where r and θ define the dipole orientation and m is the dipole

Table 1. Sensitivities of drag and lift coefficients with respect to M_∞.

Coefficient	ADjoint	Complex step
$C_{\rm D}$	1.1211866278	1.1211866250
$C_{\rm L}$	−0.82534905*650*	−0.82534904*598*

Table 2. ADjoint computational cost breakdown (times in seconds): bump test case.

Flow solution	946.039
Adjoint	205.567
Breakdown	
Setup PETSc variables	0.022
Compute flux Jacobian	29.506
Compute RHS	14.199
Solve the adjoint equations	154.551
Compute the total sensitivity	7.289

Table 3. Memory usage comparison (in MB): bump test case.

Flow	Adjoint	Ratio
62	1326	21.4 ×

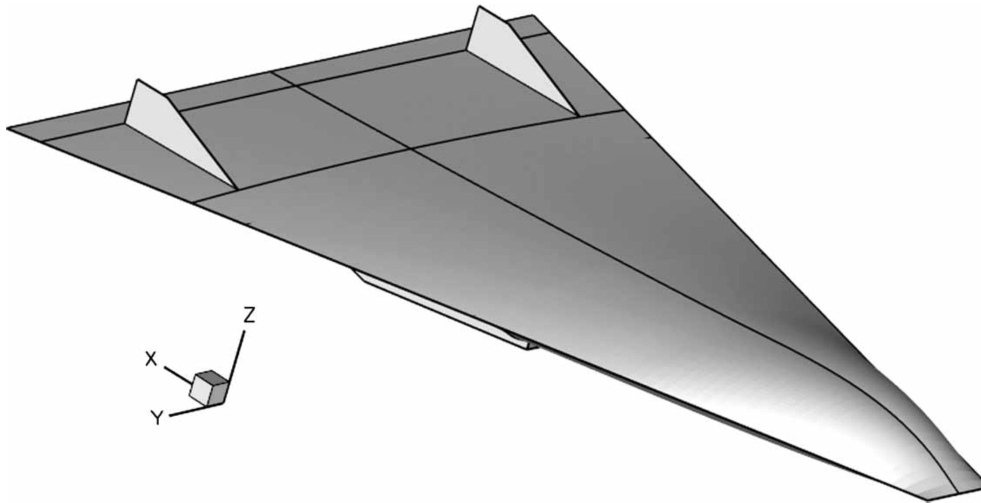

Figure 9. Generic hypersonic vehicle: top view.

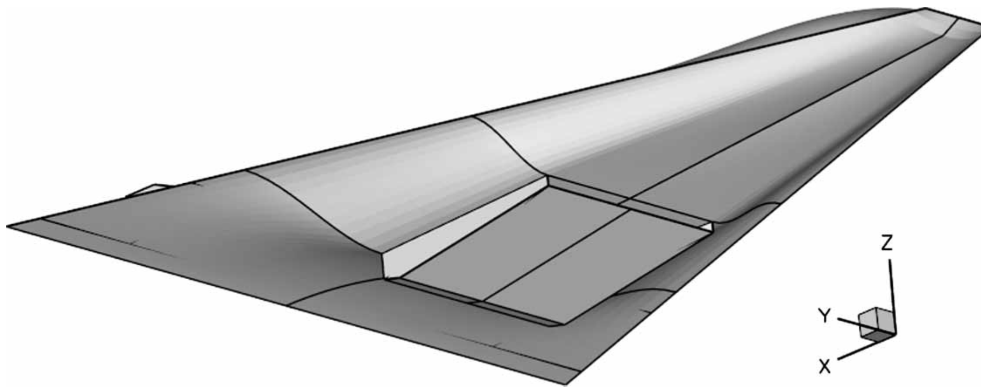

Figure 10. Generic hypersonic vehicle: bottom view.

strength. The resulting field given by the superposition of dipoles is shown in figure 11. Even though only half the domain was simulated, all of the dipoles were taken into account when computing the imposed magnetic field. All the dipoles were set to the same strength and the baseline electrical conductivity was such that a magnetic Reynolds number of $Re_\sigma = 0.57$ and a magnetic interaction parameter $Q = 0.3$ were used.

The mesh for this test case consisted of 14 blocks and a total of 3,62,362 nodes and is shown in figure 12. The multi-block computational domain is shown in figure 13. The runs were made on a parallel processor workstation, with four 3.2 GHz nodes and 8GB of RAM.

Figures 14 and 15 show the pressure contours for the flow solution on the top and bottom body surfaces, respectively, as well as on the plane of symmetry. As expected there is a large pressure increase close to the dipoles due to the imposed magnetic field. This effect is caused by the additional source terms in the MHD equations. The baseline cost function values are $C_\mathrm{L} = 0.07753$ and $C_\mathrm{D} = 0.02038$, where the reference area was taken as $96.6\,\mathrm{m}^2$.

5.2.2 Flux Jacobian. Similarly to the previous test case, the ADjoint approach was used on a set of subroutines that compute the residual of a single, specified node, obtained mainly through "cut-and-paste" from the original flow solver code.

5.2.3 Other partial derivative terms. The assembly time of the Jacobian matrix was 20.34 s, including all the calls to the automatically differentiated routines and PETSc matrix functions, whereas the adjoint vector assembly time was negligible.

Once the adjoint system of equation (27) was set up, the GMRES solver provided by PETSc was used. To be

Table 4. Dipole locations.

Dipole #	Location	Orientation
1	(0.5,0,0)	(− 1,0,0)
2/3	(1.5, ± 0.96,0)	(0, ± 1,0)
4/5	(3.5, ± 1.24,0)	(0, ± 1,0)

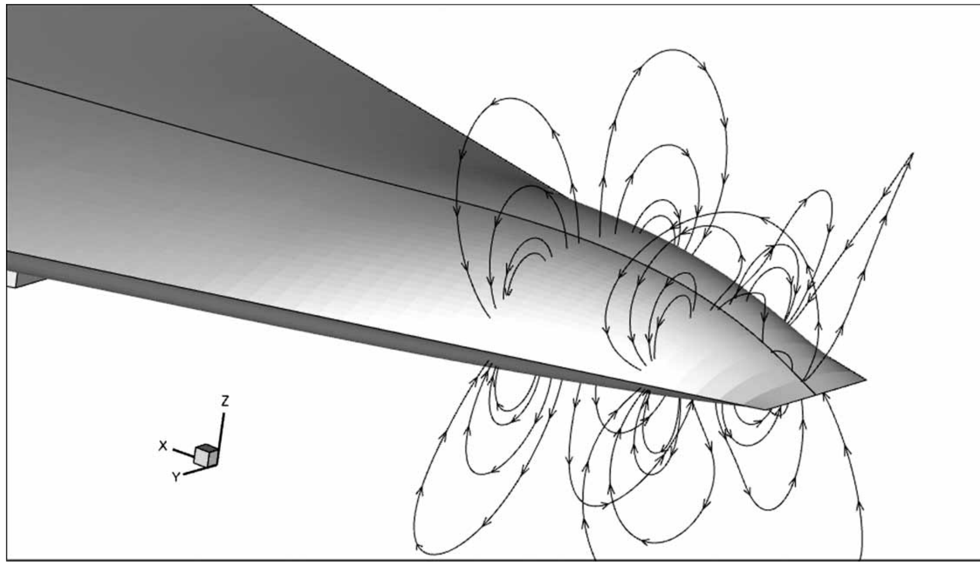

Figure 11. Imposed magnetic field.

consistent with the flow solver, the adjoint solution residual convergence criterion was also set to 10^{-10}. Side tests showed that even when the convergence was relaxed to 4–6 orders of magnitude, the total sensitivities computed with the ADjoint hardly changed. This becomes particularly useful if this tool is to be used in a design engineering framework, where fast turnaround time is crucial. The iterative solver consistently shows very good robustness and convergence properties. For the different cost functions tested, the convergence was typically achieved after about 100 iterations, which took 187 s to run. It is relevant to note that no restart was used in the GMRES solver, meaning all 100 Krylov subspaces were stored during the iterative procedure. The residual history of the adjoint solution using PETSc for both $I = C_{\mathrm{L}}$ and $I = C_{\mathrm{D}}$ is illustrated in figure 16.

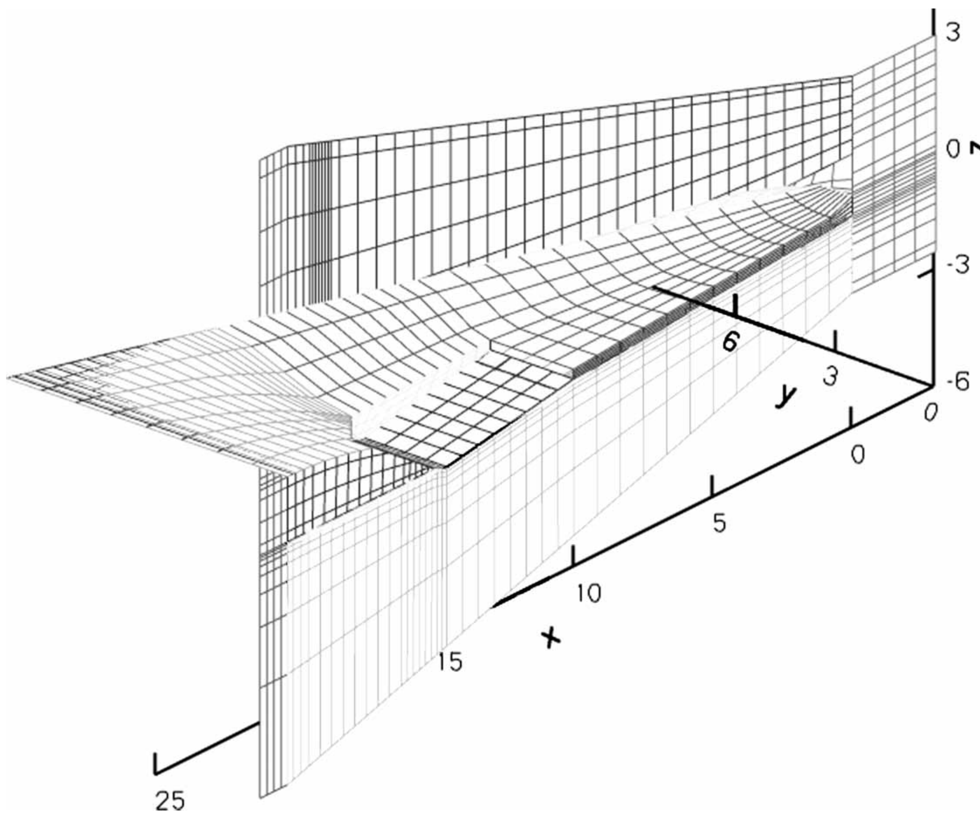

Figure 12. Mesh: bottom view (coarsened three levels for visualization).

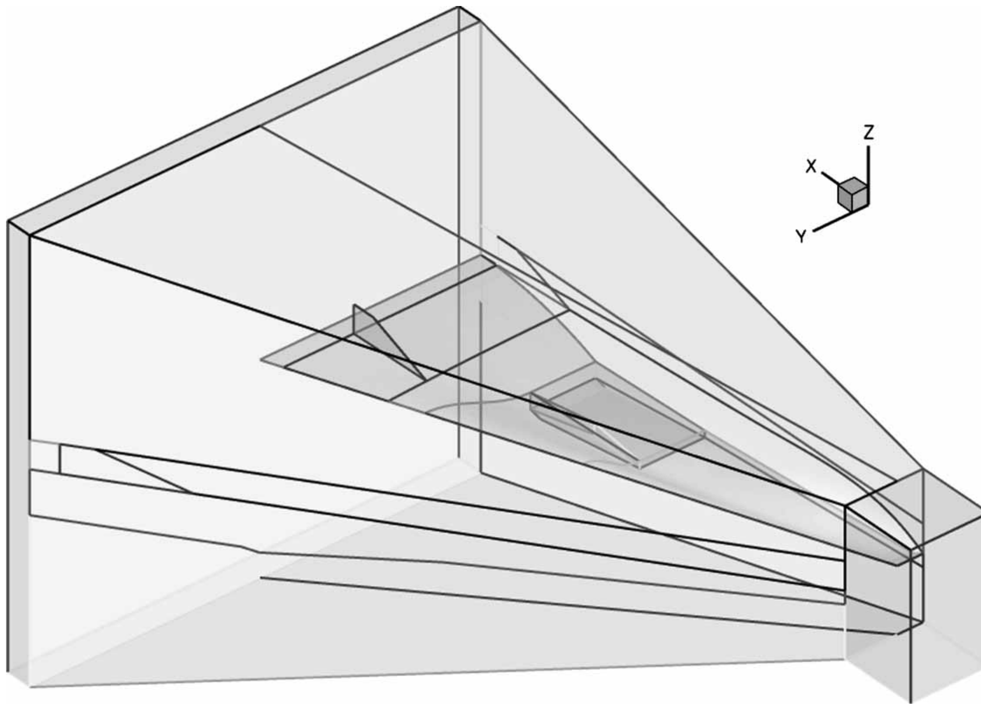

Figure 13. Multi-block domain.

The adjoint solution corresponding to the flow pressure for $I = C_D$ is shown in figures 17 and 18. It is interesting to point out that the adjoint solution, for the cost functions used in this work is similar to the flow solution, except that the flow direction is reversed. This occurrence has already been observed in other works involving the adjoint method (Lee *et al.* 2006).

5.2.4 Lift and drag coefficient sensitivities. Once the adjoint solution is obtained, the total sensitivity can then be computed using equation (28), thus requiring the calculation of the partial derivatives $\partial\mathcal{R}/\partial\alpha$ and $\partial I/\partial\alpha$. However, since the design variables chosen were the electrical conductivity in each computational node, these partial derivatives were computed analytically: $\mathcal{R}$ depends linearly on σ due to its MHD source term and the aerodynamic coefficients used as cost function I do not depend explicitly on σ.

The total sensitivity was then computed for the different cost functions and exists everywhere in the volume since the design variable σ spanned the entire problem domain. For visualization purposes, the values are shown only at the body surface and symmetry plane.

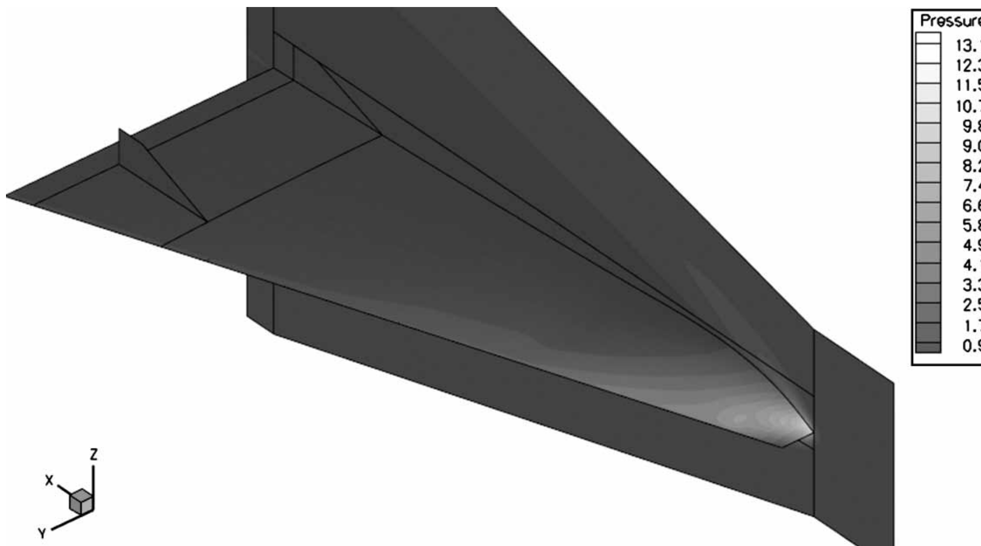

Figure 14. Contour plot of pressure: top view.

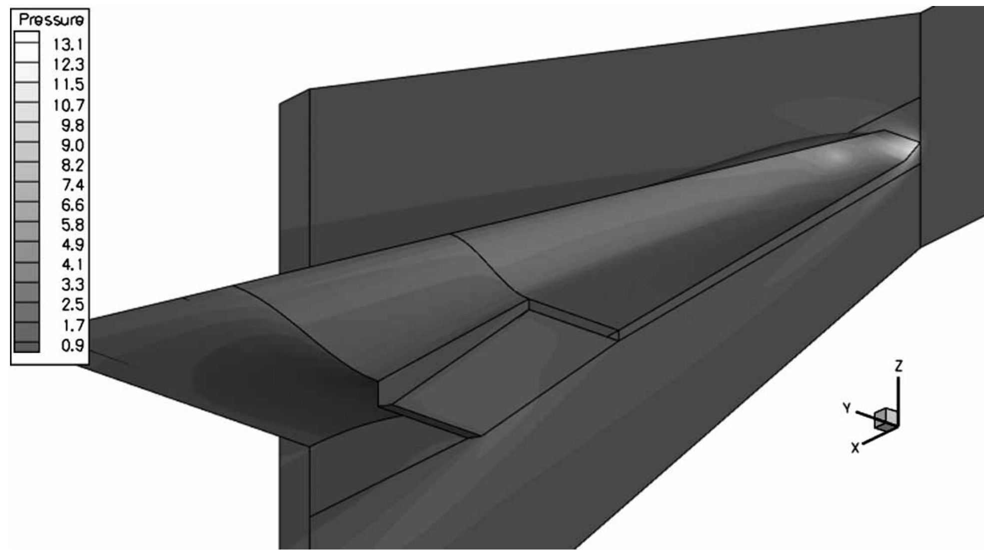

Figure 15. Contour plot of pressure: bottom view.

Figures 19 and 20 show the drag coefficient sensitivity with respect to the electrical conductivity ($\mathrm{d}C_\mathrm{D}/\mathrm{d}\sigma$). As expected, these sensitivities are greatest close to the location of the dipoles, where the imposed magnetic field is stronger.

A performance analysis was also conducted for this multi-block implementation and the detailed computational costs are summarized in table 5. It is important to notice that the flow solver has not been optimized for MHD computations yet. The additional source terms in the MHD equations make the numerical solution much less stable, and because we used an explicit 5-stage Runge–Kutta time integration scheme, the runs had to be made at significantly lower CFL numbers (0.1). Consequently, it took almost 6 h for the flow solver residual to converge ten orders of magnitude. This clearly rules out the use of finite-differences to compute cost function gradients and highlights the importance of an alternative approach such as the discrete adjoint-based gradients. Moreover, the slow convergence highlights the need for an implicit treatment of the source terms in the MHD solution that we intend to pursue in the near future.

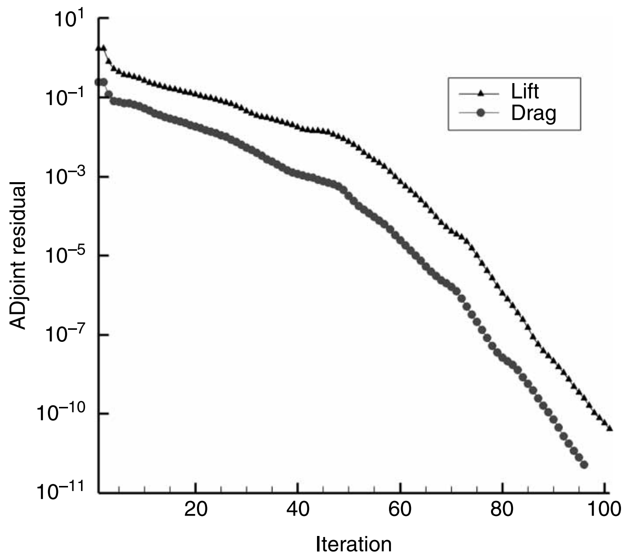

Figure 16. Adjoint residual convergence history.

The total cost of the adjoint solver, including the computation of all the partial derivatives and the solution of the adjoint system, is only one tenth the cost of the flow solution for this case. Again, this is not truly representative of reality as the flow solver can still be optimized for MHD, but it clearly shows once again that the ADjoint approach is very efficient. Considering the fact that this test case is approximately 30 times larger than the previous one and that it runs on four processors, the adjoint solver cost scales roughly linearly with problem size and number of processors, which is a much desired property. Once again, the solution of the adjoint equations was the component that took most of the time in the adjoint solver, whereas the AD sections represented less than 10% of the time, proving once again its efficiency.

The memory usage running this multi-processor test case was assessed by monitoring the virtual memory used by each processor. The observations summarized in table 6 confirm that the ADjoint solver required approximately ten times more memory than the original flow solver. The point is that our current computations use 1/10 of the memory available because we (and all designers) want to decrease the turnaround time. Also, given the pattern of use on most parallel computers, this is well within the acceptable limits for an adjoint code.

The sensitivities obtained using the discrete adjoint approach were matched against values obtained using the forward finite-difference solution. The comparison was made using two control nodes located on the body surface,

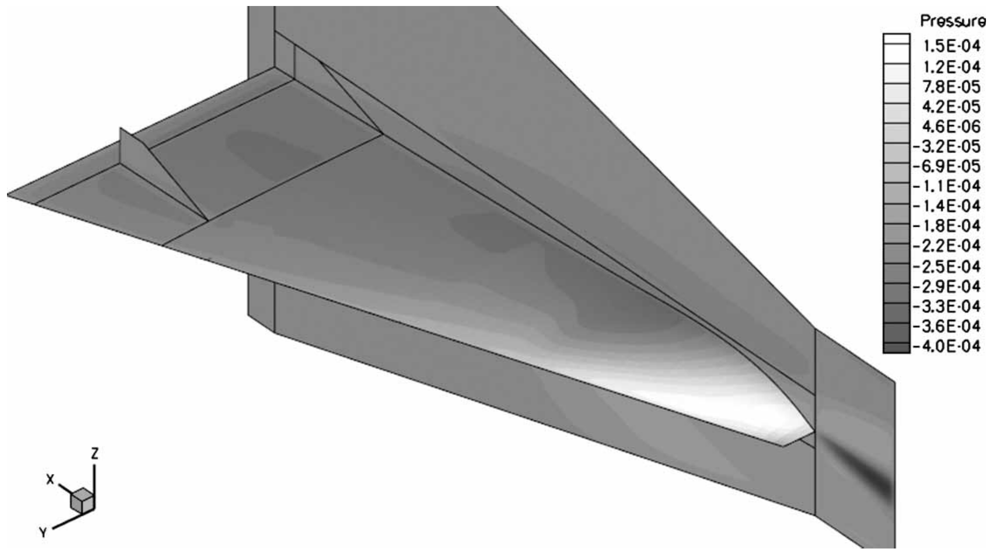

Figure 17. Adjoint of pressure for $I = C_D$: top view.

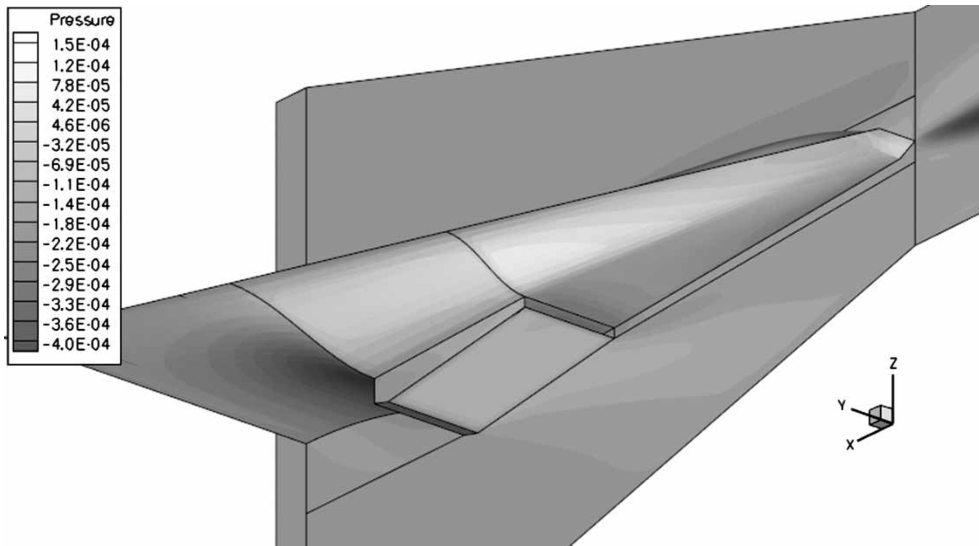

Figure 18. Adjoint of pressure for $I = C_D$: bottom view.

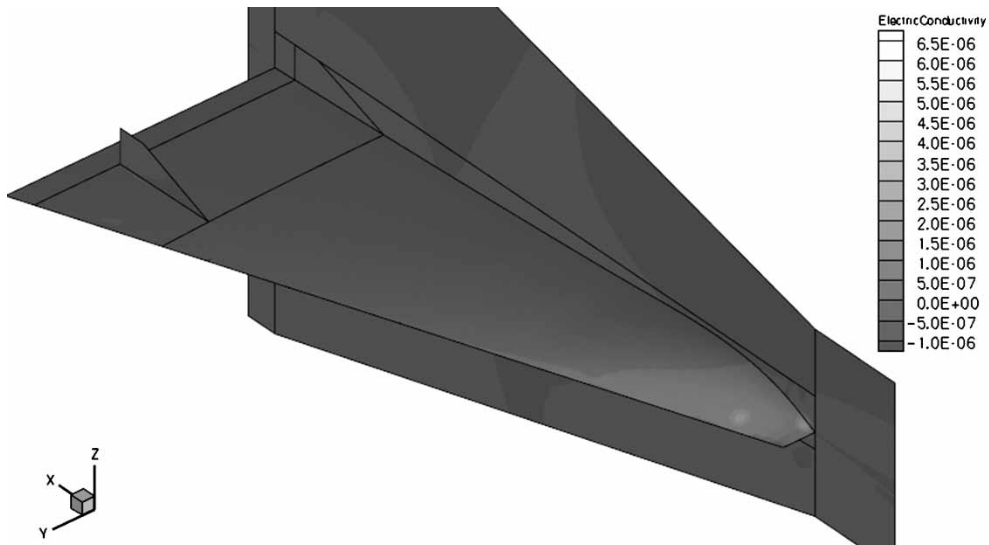

Figure 19. Drag coefficient sensitivity with respect to σ : top view.

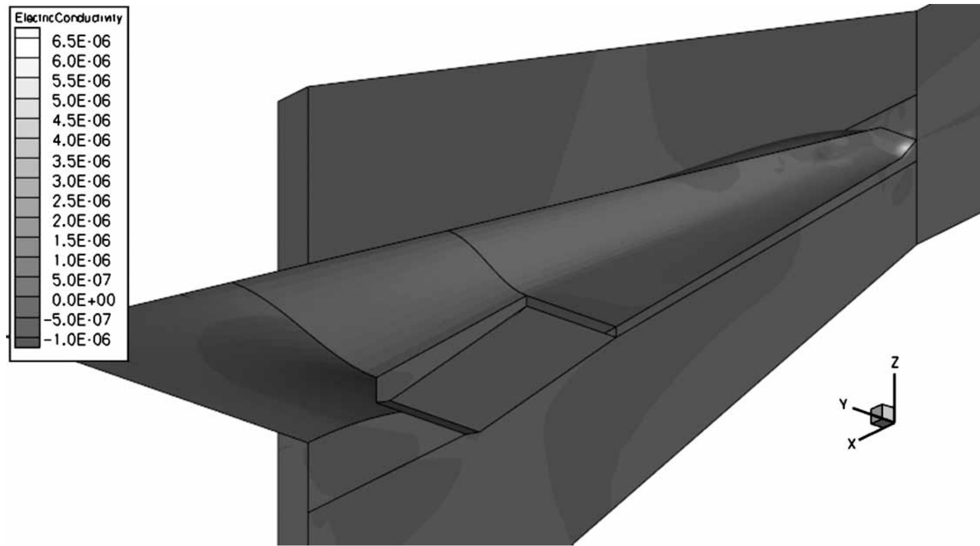

Figure 20. Drag coefficient sensitivity with respect to σ : bottom view.

over the magnetic dipoles location, the first control node located at coordinates (0.26, 0.14, 0.12) and the second at (1.35, 1.22, 0.25). The results are summarized in table 7 using two different finite-difference perturbation step sizes: 0.05 and 0.1%.

The values in table 7 demonstrate two things: firstly, the agreement between the two different approaches is excellent, successfully verifying the adjoint-based gradient values; secondly, it shows how the finite-difference approach is sensitive to the chosen perturbation step. This verification also revealed that it would have been computationally prohibitive to compute the sensitivities with respect to such large numbers of design variables using anything but the adjoint method—to get the flow solver to converge (starting from the baseline solution) every time the electrical conductivity was perturbed in a single node in the domain, took roughly 2 h. Extrapolating to all nodes, corresponding to the 3,62,362 design variables, it would have taken over eighty years to obtain the same results that took less than 4 min for the adjoint method described. However, in realistic design problems with optimized flow and adjoint solvers, 5–10 cost functions and about 100 design variables, the automatic discrete adjoint-based gradients are expected to be at least 20–50 times faster compared with finite-difference approximations.

Table 5. ADjoint computational cost breakdown (wall clock time in seconds): hypersonic test case.

Flow solution	20,812
Adjoint	208.27
Breakdown	
Setup PETSc variables	0.83
Compute flux Jacobian	20.34
Compute RHS	0.01
Solve the adjoint equations	186.76
Compute the total sensitivity	0.33

6. Conclusions

We have presented a new methodology for developing discrete adjoint solvers for arbitrary governing equations. The implementation of the ADjoint method was largely automatic, since no hand differentiation of the equations or adjoint BC was necessary.

This approach has the advantage that it uses the reverse mode of differentiation on the code that computes the residuals on a cell-per-cell (or node-per-node) basis for the governing equations and it is therefore time-efficient. The automatically differentiated code is used to populate only the non-zero entries of the large discrete adjoint system, treating one complete row at the time, given the stencil of dependence of the flow residual.

The timings presented show that the ADjoint approach is much faster than pure AD, and even “manual adjoint” solvers, even though our codes have yet to be fully optimized. In multi-processor test cases up to $\mathcal{O}(10^5)$ nodes, the linear solve in the adjoint equations took approximately one tenth of the flow solve time itself and is shown to scale well with both problem size and number of processors.

The major advantage of the ADjoint approach when compared to traditional approaches is the fact that it drastically reduces the implementation time—while the typical development of an adjoint solver could take up to a year of work, the ADjoint can take as little as a week,

Table 6. Memory usage comparison (in MB): hypersonic test case.

Proc.	Flow	Adjoint	Ratio
1	102	1295	12.7 ×
2	97	1063	11.0 ×
3	93	1040	11.2 ×
4	86	856	10.0 ×
Total	378	4254	11.2 ×

Table 7. Electrical conductivity sensitivity verification: ($dI/d\alpha$).

Control Node	Cost function J	Adjoint	Finite-difference (5×10^{-4})	Δ (%)	Finite-difference (1×10^{-3})	Δ (%)
1	C_L	-1.4850×10^{-5}	-1.4906×10^{-5}	0.38	-1.5041×10^{-5}	1.29
	C_D	2.0896×10^{-6}	2.0652×10^{-6}	−1.16	2.0646×10^{-6}	−1.19
2	C_L	-9.8145×10^{-6}	-9.8914×10^{-6}	0.78	-1.0037×10^{-5}	2.26
	C_D	1.5272×10^{-6}	1.5066×10^{-6}	−1.35	1.5053×10^{-6}	−1.44

provided a well written flow solver is already available. However, if this approach is to be used with legacy CFD codes (typically found in industry), it is crucial that a complete knowledge of the subroutine call structure of the CFD solver is acquired prior to any attempt to use AD. Once a re-engineered node-based residual routine is built, as exemplified in section 4.2, the ADjoint approach can then be rapidly implemented.

Besides expediting the development time, this method also produces gradients that are exactly consistent with the flow solver discretization and permits the use of arbitrary cost functions and design variables.

The ADjoint approach was successfully applied to two distinct flow solvers—a cell-centred single-block Euler solver and a vertex-centred multi-block MHD solver—and the total sensitivities obtained from the corresponding discrete adjoint solvers showed excellent agreement with the benchmark results produced by complex-step and finite-difference methods. This also highlighted the large computational savings that result from using the discrete adjoint method when compared to other methods, such as finite-differences and complex-stepping.

The ADjoint approach described is particularly well suited to compute the gradients of any cost function in an optimization problem involving any set of governing equations that can be cast in the form $\mathcal{R}(x, w(x)) = 0$, when the number of design variables considerably outnumbers the number of cost functions or when the solution of the governing equations is computationally too expensive to allow the use of finite-differences.

Currently, few research groups have been able to develop adjoint codes, largely due to the sheer effort required. It is our opinion that the ADjoint approach will facilitate the implementation of adjoint methods and that it will be a popular choice among researchers wishing to perform efficient sensitivity analysis and optimization. The benefits of the ADjoint approach outweigh its drawbacks and the increased memory requirement is definitely worthwhile.

Acknowledgements

We would like to thank San Gunawardana for providing the generic hypersonic vehicle multi-block mesh. The first and fifth authors are grateful for the Air Force Office of Scientific Research contract No. FA9550-04-C-0105 under tasks monitored by John Schmisseur. The first author also acknowledges the support of the *Fundação para a Ciência e a Tecnologia* from Portugal. In addition, the second and third authors are grateful for the funding provided by the Canada Research Chairs program and the Natural Sciences and Engineering Research Council.

References

ASC, Advanced simulation and computing, http://www.llnl.gov/asc (accessed in March 2007).

Balay, S., Gropp, W.D., McInnes, L.C. and Smith, B.F., Efficient management of parallelism in object oriented numerical software libraries. In *Proceedings of the Modern Software Tools in Scientific Computing*, edited by E. Arge, A.M. Bruaset and H.P. Langtangen, pp. 163–202, 1997 (Birkhäuser Press: Boston).

Balay, S., Buschelman, K., Eijkhout, V., Gropp, W.D., Kaushik, D., Knepley, M.G., McInnes, L.C., Smith, B.F. and Zhang, H., PETSc user manual, Technical report ANL-95/11—Revision 2.3.0, Argonne National Laboratory. 2004.

Balay, S., Buschelman, K., Gropp, W.D., Kaushik, D., Knepley, M.G., McInnes, L.C., Smith, B.F. and Zhang, H., PETSc: web page, http://www.mcs.anl.gov/petsc (accessed in April 2006).

Carle, A. and Fagan, M., ADIFOR 3.0 overview, Technical report CAAM-TR-00-02, Rice University 2000.

Carle, A., Fagan, M. and Green, L.L., Preliminary results from the application of automated adjoint code generation to CFL3D. *AIAA Paper 1998–4807*, pp. 1998–4807, 1998 (7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization: St. Louis, MO).

Carpenter, M.H., Gottlieb, D. and Abarbanel, S., Time-stable BC for finite-difference schemes solving hyperbolic systems: methodology and application to high-order compact schemes. *J. Comput. Phys.*, 1994, **111**(2), 220–236.

Carpenter, M.H., Nordström, J. and Gottlieb, D., A stable and conservative interface treatment of arbitrary spatial accuracy. *J. Comput. Phys.*, 1999, **148**(2), 341–365.

Cusdin, P. and Müller, J.D., On the performance of discrete adjoint CFD codes using automatic differentiation. *Int. J. Numer. Methods Fluids*, 2005, **47**(6–7), 939–945.

Driver, J. and Zingg, D.W., Optimized natural-laminar-flow airfoils. *Proceedings of the 44th AIAA Aerospace Sciences Meeting and Exhibit*, AIAA 2006-0247, January 2006 Reno, NV).

Dwight, R.P. and Brezillon, J., *Proceedings of the 44th AIAA Aerospace Sciences Meeting and Exhibit*, 2006-0690, January 2006 Reno, NV).

Fagan, M. and Carle, A., Reducing reverse-mode memory requirements by using profile-driven check-pointing. *Future Generation Comp. Syst.*, 2005, **21**(8), 1380–1390.

Faure, C. and Papegay, Y., Chapter title, Odyssée Version 1.6: The Language Reference Manual, INRIA Rapport Technique 211.

Forth, S.A. and Evans, T.P., Aerofoil optimisation via automatic differentiation of a multigrid cell-vertex Euler flow solver. In *Automatic Differentiation: From Simulation to Optimization*, edited by G. Corliss, C. Faure, A. Griewank, L. Hascoet and U. Naumann, pp. 153–160, 2002 (Springer-Verlag: New York).

Gaitonde, D.V., Simulation of local and global high-speed flow control with magnetic fields. *Proceedings of the 43rd AIAA Aerospace Sciences Meeting and Exhibit*, AIAA 2005-0560, Janaury 2005 Reno, NV).

Giering, R. and Kaminski, T., Applying TAF to generate efficient derivative code of Fortran 77–95 programs. *Proceedings of the GAMM 2002*, 2002 (Augsburg: Germany).

Giering, R., Kaminski, T. and Slawig, T., Generating efficient derivative code with TAF: adjoint and tangent linear Euler flow around an airfoil. *Future Generation Comp. Syst.*, 2005, **21**(8), 1345–1355.

Giles, M.B. and Pierce, N.A., An introduction to the adjoint approach to design. *Flow Turb. Comb.*, 2000, **65**, 393–415.

Gockenbach, M.S., *Understanding Code Generated by TAMC*, IAAA Paper TR00-29 2000 (Department of Computational and Applied Mathematics, Rice University: Texas, USA).

Griewank, A., *Evaluating Derivatives*, 2000 (SIAM: Philadelphia).

Hascoët, L. and Pascual, V., TAPENADE 2.1 user's guide, Technical report 300, INRIA 2004.

Hascoët, L. and Pascual, V., Extension of TAPENADE towards Fortran 95. In *Automatic Differentiation: Applications, Theory, and Tools*, Lecture Notes in Computational Science and Engineering edited by H.M. Bücker, G. Corliss, P. Hovland, U. Naumann and B. Norris, 2005 (Springer-Verlag: New York).

Heimbach, P., Hill, C. and Giering, R., An efficient exact adjoint of the parallel MIT general circulation model, generated via automatic differentiation. *Future Generation Comp. Syst.*, 2005, **21**(8), 1356–1371.

Horwedel, J.E., GRESS: a preprocessor for sensitivity studies on Fortran programs. In *Automatic Differentiation of Algorithms: Theory, Implementation and Application*, edited by A. Griewank and G.F. Corliss, pp. 243–250, 1991 (SIAM: Philadelphia, PA).

Jameson, A., Aerodynamic design via control theory. *J. Sci. Comput.*, 1988, **3**(3), 233–260.

Lee, K.H., Alonso, J.J. and Van der Weide, E., Mesh adaptation criteria for unsteady periodic flows using a discrete adjoint time-spectral formulation. *Proceedings of the 44th AIAA Aerospace Sciences Meeting and Exhibit*, AIAA 2006-0692, January 2006 Reno, NV).

Marta, A.C. and Alonso, J.J., Discrete adjoint formulation for the ideal MHD equations. *Proceedings of the 3rd AIAA Flow Control Conference*, AIAA 2006-3345, Junuary 2006a San Francisco, CA).

Marta, A.C. and Alonso, J.J., High-speed MHD flow control using adjoint-based sensitivities. *Proceedings of the 14th AIAA/AHI International Space Planes and Hypersonic Systems and Technologies Conference*, AIAA 2006-8009 November 2006b Canberra, Australia).

Martins, J.R.R.A., Sturdza, P. and Alonso, J.J., The connection between the complex-step derivative approximation and algorithmic differentiation. *Proceedings of the 39th Aerospace Sciences Meeting*, AIAA 2001-0921 2001 Reno, NV).

Martins, J.R.R.A., Sturdza, P. and Alonso, J.J., The complex-step derivative approximation. *ACM Trans. Math. Softw.*, 2003, **29**(3), 245–262.

Martins, J.R.R.A., Alonso, J.J. and Reuther, J.J., High-fidelity aerostructural design optimization of a supersonic business jet. *J. Aircraft*, 2004, **41**(3), 523–530.

Martins, J.R.R.A., Alonso, J.J. and Reuther, J.J., A coupled-adjoint sensitivity analysis method for high-fidelity aero-structural design. *Optim. Eng.*, 2005, **6**(1), 33–62.

Martins, J.R.R.A., Alonso, J.J. and Van der Weide, E., An automated approach for developing discrete adjoint solvers. *Proceedings of the 2nd AIAA Multidisciplinary Design Optimization Specialist Conference*, AIAA 2006-1608 2006a Newport, RI).

Martins, J.R.R.A., Mader, C.A. and Alonso, J.J., ADjoint: an approach for rapid development of discrete adjoint solvers. *Proceedings of the 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, AIAA 2006-7121 2006b Portsmouth, VA).

Mattsson, K. and Nordström, J., Summation by parts operators for finite difference approximations of second derivatives. *J. Comput. Phys.*, 2004a, **199**(2), 503–540.

Mattsson, K., Svärd, M. and Nordström, J., Stable and accurate artificial dissipation. *J. Sci. Comput.*, 2004b, **21**(1), 57–79.

Mohammadi, B., Malé, J.M. and Rostaing-Schmidt, N., Automatic differentiation in direct and reverse modes: application to optimum shapes design in fluid mechanics. In *Computational Differentiation: Techniques, Applications, and Tools*, edited by M. Berz, C.H. Bischof, G.F. Corliss and A. Griewank, pp. 309–318, 1996 (SIAM: Philadelphia, PA).

MPI, A message-passing interface standard. *Int. J. Supercomputer Appl. High Perform. Comput.*, 1994, **8**(3/4), 159–416.

NASA, X-43A hypersonic scramjet-powered research aircraft, http://www.nasa.gov/missions/research/x43-main.html Accessed in October 2006.

Naumann, U. and Riehme, J., A differentiation-enabled Fortran 95 compiler. *ACM Trans. Math. Softw.*, 2005, **31**(4), 458–474.

Nemec, M. and Zingg, D.W., Multipoint and multi-objective aerodynamic shape optimization. *AIAA J.*, 2004, **42**(6), 1057–1065.

Pironneau, O., On optimum design in fluid mechanics. *J. Fluid Mech.*, 1974, **64**, 97–110.

Reuther, J., Alonso, J.J., Jameson, A., Rimlinger, M. and Saunders, D., Constrained multipoint aerodynamic shape optimization using an adjoint formulation and parallel computers: part I. *J. Aircraft*, 1999a, **36**(1), 51–60.

Reuther, J., Alonso, J.J., Jameson, A., Rimlinger, M. and Saunders, D., Constrained multipoint aerodynamic shape optimization using an adjoint formulation and parallel computers: part II. *J. Aircraft*, 1999b, **36**(1), 61–74.

Saad, Y. and Schultz, M.H., GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 1986, **7**(3), 856–869.

Sherman, L.L., Arthur, C., Taylor, I., Green, L.L., Newman, P.A., Hou, G.W. and Korivi, V.M., First- and second-order aerodynamic sensitivity derivatives via automatic differentiation with incremental iterative methods. *J. Comput. Phys.*, 1996, **129**(2), 307–331.

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 2006-0421 2006 Reno, NV).