Proceedings of ASME Turbo Expo 2009: Power for Land, Sea and Air
GT2009
June 8-12, 2009, Orlando, Florida, USA

GT2009-59297

DEVELOPMENT OF ADJOINT SOLVERS FOR ENGINEERING GRADIENT-BASED TURBOMACHINERY DESIGN APPLICATIONS

Andre C. Marta*
General Electric - Global Research Center
85748 Garching n. Munich, Germany

Sriram Shankaran, D. Graham Holmes, Alexander Stein
General Electric - Global Research Center
Niskayuna, New York 12309, USA

ABSTRACT

High-fidelity computational fluid dynamics (CFD) are common practice in turbomachinery design. Typically, several cases are run with manually modified parameters based on designer expertise to fine-tune a machine. Although successful, a more efficient process is desired. Choosing a gradient-based optimization approach, the gradients of the functions of interest need to be estimated. When the number of variables greatly exceeds the number of functions, the adjoint method is the best-suited approach to efficiently estimate gradients. Until recently, the development of CFD adjoint solvers was regarded as complex and difficult, which limited their use mostly to academia. This paper focuses on the problem of developing adjoint solvers for legacy industrial CFD solvers. A discrete adjoint solver is derived with the aid of an automatic differentiation tool that is selectively applied to the CFD code that handles the residual and function evaluations. The adjoint-based gradients are validated against finite-difference and complex-step derivative approximations.

NOMENCLATURE

a	Hicks-Henne bump amplitude.
C	Constraint function.
E	Total energy.
$\mathbf{e}$	Unit vector.
H	Total enthalpy.
h	Perturbation step.
$\dot{m}$	Mass flow.
O	Truncation error.
PR	Pressure ratio.
p	Pressure.
$\mathbf{q}$	Conservative variables, flow solution.
$\mathcal{R}$	Residual vector of the governing equations.
t	Time.
u,v,w	Velocity Cartesian components.
$\mathbf{x}$	Vector of grid coordinates.
x,y,z	Cartesian coordinates.
Y	Objective, cost or merit function.

Greek symbols:

α	Vector of design variables.
Δ	Relative difference.
δ	Variation.
η	Isentropic efficiency.
ρ	Density.
ψ	Adjoint vector.

Subscripts:

i,j,k	Computational indexes.
x,y,z	Cartesian components.

INTRODUCTION

Turbomachinery design has changed significantly since its inception and the use of high-fidelity computational fluid dynamic (CFD) simulations have become common practice. Since the analysis tools have continued to mature in these industrial environments, the desire for efficient design tools has now emerged as the next logical step. Without a numerical optimization tool, a designer is left with the time consuming task of fine-tuning a machine, which implies running several CFD cases for a set of manually modified design parameters based on designer expertise. Although this process has proven successful, the need for a

*Corresponding author: andre.marta@ge.com

more efficient design system is often desired.

The operations research field has produced a plethoric set of optimization methods [1] that could be of potential use for optimal design. However, looking at the particular nature of a turbomachinery design problem, where a single CFD simulation can take hours, if not days, to perform, it is highly desired that the number of evaluations of the function of interest to optimize should be kept to a minimum. This fact rules out gradient-free methods, such as simulated annealing or evolutionary algorithms, as these typically require a large number of function evaluations. On the other hand, if a gradient-based optimization approach is chosen, a new problem arises to estimate the gradients of the functions of interest with respect to the design variables.

There are different approaches to evaluate derivatives [2]. Calculus teaches how to analytically obtain the derivative of a single- or multi-variable function, there are even software tools that perform symbolic differentiation [3], but for the functions of interest in turbomachinery design and their highly non-linear dependence on the design variables of interest, a numerical method is necessary. Finite-difference (FD) approximations have always been popular due to its simplicity but they rapidly become computationally prohibitive when the number of variables greatly exceeds the number of functions. In this case, an adjoint method is the best-suited approach to efficiently estimate function gradients since the cost involved in calculating sensitivities using the adjoint method is therefore practically independent of the number of design variables. While FD can be regarded as external differentiation, adjoints are evaluated by internally differentiating the function of interest.

The adjoint methods have been used in the context of partial differential equations context for a very long time. Its application to CFD was pioneered by Pironneau [4] and it was later revisited and extended by Jameson to perform airfoil [5] and wing [6] design. It has since been successfully used in constrained multi-objective and multipoint aerodynamic shape optimization problems [7–9], and even in aerostructural design optimization [10] involving the coupling between a CFD adjoint solver and a computational structural model adjoint solver.

From a design perspective, an adjoint solver can improve the design space understanding since the adjoint-based sensitivities can be easily used for design space visualization. Besides aerodynamic shape optimization, there are many other applications of adjoint solvers: mesh sensitivity [11], mesh adaptation [12], error estimation [13] or even magnetohydrodynamics flow control [14]. The same adjoint solver can be shared by any of these applications, as they only differ in the way to adjoint solution is post processed. This clearly shows the enormous potential and benefit that an adjoint solver represents any CFD or design group.

The major drawback of using adjoint-based sensitivities has always been the necessity of an additional solver – the adjoint system of equations solver. The development of CFD adjoint solvers has been thought to be both complex and difficult, which has limited their use mostly to academia. Because of that, some major turbomachinery manufacturers decided to team up with universities in research projects involving design applications using adjoint-based gradients [15–17].

This paper focuses on the problem of rapidly developing an adjoint solver for a legacy CFD solver that models the traditional flow governing equations. The methodology used to develop the adjoint solver relies on a hybrid approach [18, 19]. A discrete adjoint solver is derived with the aid of an automatic differentiation tool that is selectively applied to the CFD source code that handles the residual and function evaluations. This tool produces the routines that evaluate the individual entries of the adjoint system of equations. This hybrid approach retains the accuracy of the adjoint methods, while it adds the ease of implementation of the automatic differentiation methods. This proposed methodology is believed to be the most suitable to derive an adjoint solver for a legacy flow solver in a industrial design environment since it allows a much faster development time.

This paper is divided into four main sections. The background section revises the generic optimization problem statement, introduces the flow governing equations and details the derivation of the corresponding adjoint equations. Then, in the implementation section, the steps necessary to derive the adjoint solver using the proposed hybrid approach are explained. In addition, an integration plan of such tool in an engineering aerodynamic design framework is presented and discussed. In the results section, the discrete adjoint solver derived using the proposed methodology is tested on a rotor blade passage of a high-pressure compressor and the adjoint-based gradients of some functions of interest with respect to shape parameters are computed and verified against finite-difference and complex-step derivative approximations. Finally, the paper ends with some remarks about the findings of the work presented in the conclusions section.

BACKGROUND

The methodology presented in this paper is developed in the context of optimization. Thus, the different pieces necessary to pose and solve a design problem using a high-fidelity flow analysis are presented next.

Generic Design Problem

When some component of a turbomachine is to be tuned, several characteristics are used to monitor its performance, such as efficiency, pressure ratio or mass flow. In addition, there are several machine-defining parameters that can be adjusted to change those characteristics, such as blade stagger, camber angle and thickness distributions, axial and radial stacking. In the context of optimization, the performance monitoring characteristics are named objective functions (also cost function or figure

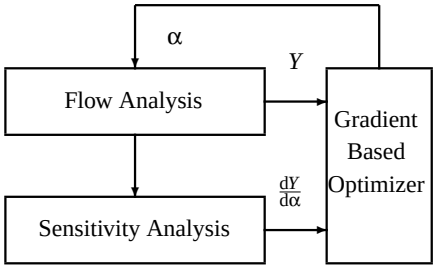

Figure 1. GRADIENT-BASED OPTIMIZATION ALGORITHM.

of merit), and the adjustable parameters are designated as design variables. In mathematical terms, an objective function can be posed as

$$Y = Y(\alpha, \mathbf{q}(\alpha)), \tag{1}$$

where α is the vector of design variables and $\mathbf{q}$ is the flow solution, which is typically of function of the design variables.

From fluid mechanics, the flow solution is given by the solution of a set of partial differential equations (PDEs) that model the flow physics. In general, such governing equations can be expressed in the form

$$\mathcal{R}(\alpha, \mathbf{q}(\alpha)) = 0, \tag{2}$$

where the first instance of α indicates the fact that the residual of the governing equations may depend explicitly on design variables.

Therefore, a generic CFD design problem can be formally described as

$$\begin{aligned} \text{Minimize} \quad & Y(\alpha, \mathbf{q}(\alpha)) \\ \text{w.r.t.} \quad & \alpha, \\ \text{subject to} \quad & \mathcal{R}(\alpha, \mathbf{q}(\alpha)) = 0 \\ & C_i(\alpha, \mathbf{q}(\alpha)) = 0 \quad i = 1, ..., m, \end{aligned} \tag{3}$$

where $C_i = 0$ 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 evaluation of the objective function Y and the constraints C_i values, their gradients with respect to the design variables α are also required, that is, $\frac{dY}{d\alpha}$ and $\frac{dC_i}{d\alpha}$ have to be estimated. The corresponding optimization algorithm is schematically shown in Fig. 1.

The main focus of present work is on the sensitivity module of such gradient-based optimization algorithm in the context of turbomachinery design using CFD tools.

Governing Flow Equations

As a proof-of-concept, the governing PDEs chosen in the present work are the Euler equations. It is important to recall that this choice is not limiting since the methodology presented to derive the discrete adjoint solver is generic and can be applied to any set of governing equations. In conservation form, the three-dimensional Euler system of equations is

$$\frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{E}}{\partial x} + \frac{\partial \mathbf{F}}{\partial y} + \frac{\partial \mathbf{G}}{\partial z} = 0, \tag{4}$$

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

$$\begin{aligned} \mathbf{q} &= \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{pmatrix}, \quad \mathbf{E} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ \rho Hu \end{pmatrix}, \\ \mathbf{F} &= \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ \rho Hv \end{pmatrix} \text{ and } \mathbf{G} = \begin{pmatrix} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + p \\ \rho Hw \end{pmatrix}, \end{aligned} \tag{5}$$

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 $H = E + \frac{p}{\rho}$.

In semi-discrete form, the governing equations (4) can be expressed as

$$\frac{dq_{ijk}}{dt} + \mathcal{R}_{ijk}(\mathbf{w}) = 0, \tag{6}$$

where $\mathcal{R}$ is the residual described earlier with all of its components (inviscid fluxes, boundary conditions, artificial dissipation, etc.), and the triad ijk represents the three computational directions. The unsteady term of Eqn. (6) is dropped out since only the steady solution of the equation is of interest in this work.

Adjoint Equations

The adjoint equations for systems of PDEs have already been well documented in the literature. A good reference is the work by Giles [20], which provides details about the continuous and discrete adjoint approaches and their derivations. Regardless

of the derivation approach adopted, the adjoint equations can be expressed as

$$\left[\frac{\partial \mathcal{R}}{\partial \mathbf{q}}\right]^T \psi = \left[\frac{\partial Y}{\partial \mathbf{q}}\right]^T, \tag{7}$$

where ψ is the adjoint vector.

Since a CFD solver, in general, does not handle the geometric parameters α directly, but rather a computational mesh defined by the coordinates of each node $\mathbf{x}$, it is convenient to use the chain rule of differentiation to express the sensitivity of the function of interest with respect to the design variables as

$$\frac{\mathrm{d}Y}{\mathrm{d}\alpha} = \frac{\mathrm{d}Y}{\mathrm{d}\mathbf{x}}\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\alpha}. \tag{8}$$

The total sensitivity of the objective function with respect to the grid coordinates, based on the adjoint solution ψ, is given by

$$\frac{\mathrm{d}Y}{\mathrm{d}\mathbf{x}} = \frac{\partial Y}{\partial \mathbf{x}} - \psi^T \frac{\partial \mathcal{R}}{\partial \mathbf{x}}. \tag{9}$$

In order to compute the sensitivity of each function of interest Y, either objective or constraint function in the optimization problem (3), a new adjoint solution is required. This implies solving Eqn. (7) with a new right-hand side vector. On the other hand, the computational cost of the total sensitivity (9) is almost independent of the number of grid coordinates $\mathbf{x}$, which is the feature that makes the adjoint method so attractive for gradient-based optimization involving a large number of variables and a few functions of interest.

The simple mathematical form of Eqn. (7) can be very misleading since, depending on the approach, their numerical implementation can be quite complex, if derived by manual differentiation, or quite costly, if derived using finite-differences.

There are two distinct ways of deriving an adjoint solver. 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. There has been some studies comparing these two distinct approaches [21] and, in general, one approach does not always prevail over the other.

The discrete approach formulation has the advantage that it can be applied to any set of governing equations and it can treat arbitrary functions of interest. As such, and in contrast to the continuous approach, no simplifications have to be made during the derivation: the effects of the viscosity and heat transfer and the turbulence equations can easily by handled when deriving the discrete adjoint. This is particularly important as not deriving the full adjoint can result in incorrect sensitivities, particularly for viscous flow, as demonstrated by Dwight [22]. Another advantage of this formulation is that the boundary conditions are handled seamlessly since the adjoint solver is derived from the discretized flow residual equations that already implement them. But the most interesting feature of the discrete approach is that it allows the use of automatic differentiation (AD) tools in its derivation, expediting considerably the process of obtaining the differentiated form of the discretized governing equations necessary to assemble the adjoint system of equations. This is the reasoning behind the approach adopted to compute the partial derivative matrices $\partial\mathcal{R}/\partial\mathbf{q}$, $\partial Y/\partial\mathbf{q}$, $\partial Y/\partial\mathbf{x}$ and $\partial\mathcal{R}/\partial\mathbf{x}$ that is presented in the implementation section.

Automatic Differentiation

Automatic differentiation (AD), also known as computational or algorithmic differentiation, is a sensitivity analysis method based on the systematic application of the chain rule of differentiation to computer programs.

The sequence of operations in any computational algorithm can be cast in the form

$$t_i = f_i(t_1, t_2, \ldots, t_{i-1}), \quad i = n+1, n+2, \ldots, m, \tag{10}$$

where each function f_i is either a unary or binary operation. $t_1, t_2, \ldots, t_n$ are the independent variables, which in this work assume the role of grid coordinates $\mathbf{x}$, and $t_{n+1}, t_{n+2}, \ldots, t_m$ are the dependent variables, that include all the intermediate variables in the algorithm, among which we can find the outputs of interest, Y. Applying the chain rule to the algorithm (10) yields

$$\frac{\partial t_i}{\partial t_j} = \sum_{k=1}^{i-1} \frac{\partial f_i}{\partial t_k}\frac{\partial t_k}{\partial t_j}, \quad j = 1, 2, \ldots, n. \tag{11}$$

AD tools can automatically generated new code that computes user-specified derivatives as accurately as an analytic method.

There are two different modes of operation for AD. The forward mode propagates the required sensitivity at the same time as the solution is being computed. In terms of index notation, one independent variable is selected, choosing j and keeping it fixed, and then the expression is worked forward in the index i until the desired derivative is obtained. Thus, this mode is well-suited when evaluating the sensitivity of many functions with respect to one parameter. The reverse mode requires the function to be computed first, with intermediate variable values stored. These intermediate variables are then used by the reverse version of the code to find the sensitivities. This mode works by fixing i, corresponding to the desired output to be differentiated, and working the way backward in the index j all the way down to the independent variables. As such, it is the desired mode to compute the sensitivity of one function with respect to many parameters.

There are two methods for implementing AD – source code transformation and operator overloading. The AD implementation using source transformation requires the whole source code to be processed by a parser. The parser introduces additional lines of code corresponding to the derivative calculations while generating the differentiated version of the source code. Therefore, the resulting code is greatly enlarged and it becomes practically unreadable. The AD implementation using operator overloading defines 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) for the new type in order for the derivative to be computed together with the original computations. This results in a very elegant implementation since very few changes are required in the original code, but it is usually less efficient.

There are a number of software tools available for automatic differentiation, supporting different programming languages, such as Fortran 77/90 or C/C++, and implementing either the source code or the operator overloading methods. A short comparison of some AD tools has been compiled by Cusdin [23]. The AD tool chosen in this work was Tapenade [24–26] because it supports Fortran 90, which was a requirement taking into account the programming language used in the flow solver implementation, it uses source transformation and it can perform differentiation in either forward or reverse mode.

In the context of CFD, the code that evaluates the function of interest is typically an iterative solver. It has been shown that the direct application of AD tools to CFD solvers generates differentiated code that requires too much memory to store the intermediate variable values for every iteration, and can take up to ten times longer to run [27].

Hybrid approach: AD adjoint

The approach used in this work is hybrid. On one hand, it relies on the discrete adjoint method to compute the sensitivities, making use of the adjoint (7) and the total sensitivity (9) equations to compute the gradients of the function of interest with respect to the grid coordinates. However, rather than using AD to differentiate the entire source code of the CFD solver, AD is selectively applied to produce only the code that computes the individual entries in the flux Jacobian matrix and the other partial derivatives – $\frac{\partial \mathcal{R}}{\partial \mathbf{q}}$, $\frac{\partial Y}{\partial \mathbf{q}}$, $\frac{\partial Y}{\partial \mathbf{x}}$ and $\frac{\partial \mathcal{R}}{\partial \mathbf{x}}$ – that are necessary to compute sensitivities using an adjoint method.

Verification of Gradients

The adjoint-based gradients obtained using the proposed hybrid approach are verified against both finite-difference (FD) and complex-step (CS) derivative approximations.

To minimize the necessary number of evaluations of the flow solver, the 1^{st}-order forward-difference formula for the first derivative, obtained from the Taylor series expansion of function Y for a perturbation about a point x, is used. That is,

$$\frac{\mathrm{d}Y}{\mathrm{d}x} = \frac{Y(x+h) - Y(x)}{h} + O(h)\,, \tag{12}$$

where x corresponds to each individual entry of the vector of grid coordinates $\mathbf{x}$,and h is the real perturbation step.

As in FD approximation, this expression suffers from a large sensitivity to the choice of step size. If h is chosen too large, the derivative estimate might be inaccurate because of the large truncation error; if it is made too small, then subtractive cancellation might occur and the estimate is again inaccurate. Finding the sweet spot of h is often problem dependent so several values have to be tried.

The complex-step derivative approximation can also be derived using a Taylor series expansion [28, 29], similar to the finite-difference approximation, but instead of using a real perturbation step, it uses a pure imaginary step,

$$\frac{\mathrm{d}Y}{\mathrm{d}x} = \frac{\mathrm{Im}\left[Y(x+ih)\right]}{h} + O(h^2)\,. \tag{13}$$

Comparing the CS approximation (13) to the FD approximation (12), it can be seen that the former has 2^{nd}-order accuracy, even though it only requires $N_x + 1$ function evaluations, and it is not subject to subtractive cancellation. This enables us to make much more conclusive comparisons when it comes to accuracy.

To implement the CS method in a flow solver coded in a language that supports complex-arithmetic, it is necessary to make a few changes to the code, namely substitute all real type variable declarations with complex declarations and define all functions and operators that are not defined for complex arguments. This was easily accomplished by running a custom made script that automatically generated the transformed Fortran code. A point worth noting is that the complex-step method is equivalent to the forward-mode of automatic differentiation using operator overloading [30].

This method has already been proved to be very accurate, extremely robust and surprisingly easy to implement in design problems [31]. Nevertheless, the cost of estimating the derivative using this method is still proportional to the number of design variables, which is further aggravated by the fact that the run time of the complex-valued code might take up to three times longer to run when compared to the original real-valued code.

IMPLEMENTATION

Following the methodology outlined in the previous section, the development of the discrete adjoint solver and its integration into a design system followed several well-defined steps, which are described next.

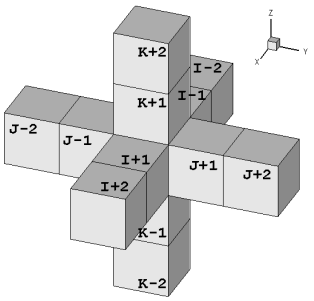

Figure 2. COMPUTATIONAL FLOW STENCIL: 13 CELLS.

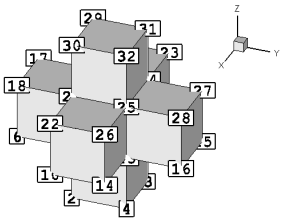

Figure 3. COMPUTATIONAL GRID STENCIL: 32 NODES.

Flow Solver

The implementation of the proposed hybrid adjoint solver had a legacy flow solver as its starting point. This flow solver is multi-block, three-dimensional, finite-volume, structured grid, non-linear and linear, Euler/Navier-Stokes solver for turbomachinery blade rows. It is capable of efficiently performing three-dimensional analysis for aeromechanics, aerodynamic design, parametric studies, and robust design applications.

As typical for most iterative CFD flow solvers, the residual calculation is done in a subroutine that loops through the three-dimensional domain and accumulates the several fluxes and boundary conditions contributions in the residual $\mathcal{R}$. However, the residual at each cell only depends on the flow variables at that cell and at the cells adjacent to it, which define the stencil of dependence. This stencil is shown in Fig. 2 for the case of an inviscid flow analysis. For this same case, the cell residual depends only on grid metrics defined by the grid stencil shown in Fig. 3.

Adjoint Solver

The gradients of the functions of interest with respect to the grid coordinates are computed by first assembling the discrete adjoint equations (7), solving them, and then using the sensitivity equation (9). The sizes of the matrices involved in this process are

$$\frac{\partial \mathcal{R}}{\partial \mathbf{q}} \quad (N_q \times N_q) \ , \quad \frac{\partial Y}{\partial \mathbf{q}} \quad (N_Y \times N_q) \ ,$$
$$\frac{\partial \mathcal{R}}{\partial \mathbf{x}} \quad (N_q \times N_x) \ , \quad \frac{\partial Y}{\partial \mathbf{x}} \quad (N_Y \times N_x) \ ,$$

where N_Y is the number of functions of interest, N_x the number of grid coordinates and N_q the size of the state vector. The size of the vector $\mathbf{q}$ depends on the number of governing equations, N_e, and the number of cells of the computational mesh, N_c, that discretize the physical domain, according to the relation $N_q = N_e \times N_c$, which for the solution of a large, three-dimensional problem involving a system of conservation laws, can be very large. The size of the grid coordinates vector $\mathbf{x}$, is given by dimensionality of the problem times the number of vertices corresponding to the computational mesh used, that is, $N_x = 3 \times N_v$ for three-dimensional problems.

According to the proposed hybrid adjoint approach, automatic differentiation tools are used to generate code that computes the non-zero entries of these matrices of partial sensitivities.

Construction of the Jacobian $\frac{\partial \mathcal{R}}{\partial \mathbf{q}}$ If one would directly apply AD to the original nested-loop residual code, that would translate into enormous computational inefficiencies. If the forward mode were used, then the cost of computing $\frac{\partial \mathcal{R}}{\partial \mathbf{q}}$ would be roughly $N_c \times N_q$ times the cost of the original residual computation. If the reverse mode were used, then there would be a large memory penalty associated with the storage of all the intermediate variables generated by the series of nested loops, which is exactly what needs to be avoided.

Therefore, to avoid the AD of nested loops over the whole computational domain, a re-engineered set of routines that mimic the original computation of the residual, but only at a given cell location in the computational domain, was created. That code was easily constructed from the original residual evaluation routines in the flow solver by removing the loops over all the cells in the domain and making necessary adjustments so that the boundary conditions were handled properly. The new residual routine computes the N_e residuals, `rAdj`, at a specified cell `(i,j,k)`, getting contributions from all $N_e \times N_{cs}$ flow variables, `qAdj`, and from all $3 \times N_{vs}$ grid variables in the stencil, `xAdj`,`yAdj`,`zAdj`, where N_{cs} denotes the number of cells of the flow stencil (see Fig. 2), and N_{vs} denotes the number of vertices of the grid stencil (see Fig. 3),

```
subroutine residualAdj(i,j,k,xAdj,yAdj,zAdj,qAdj,rAdj).
```
(14)

There are $N_e \times (N_e \times N_{cs})$ sensitivities to be computed for each cell, corresponding to N_e rows in the Jacobian adjoint matrix, $\frac{\partial \mathcal{R}}{\partial \mathbf{q}}$, where each of these rows contains no more than $N_e \times N_{cs}$

non-zero entries. Due to the way residuals are computed, the reverse mode is much more efficient in this case and, on this basis, it was used to produce adjoint code for the set of residual evaluation routines. All derivatives in the stencil can be calculated from a single call to the differentiated residual routine at a given computational cell.

Construction of the vector $\frac{\partial Y}{\partial \mathbf{q}}$ The RHS vector of the adjoint equations (7) (or matrix, in the case of multiple functions of interest) represents the direct effect of the flow variables on the function of interest. Similarly to the residual evaluation, it was necessary to obtain modified versions of the original functions to use Tapenade to produce the AD code that computes the derivatives. However, since the functions of interest were all global parameters of some sort, the dependence on the flow (and grid) could not be reduced to a stencil. In this case, the whole computational domain was specified for both the flow variables,`qAdj`, and grid coordinates,`xAdj`,`yAdj`,`zAdj`,

```
subroutine functionAdj(xAdj,yAdj,zAdj,qAdj,fyAdj). (15)
```

AD was used in reverse mode because the dimension of inputs `qAdj`, N_q, clearly outnumbered the dimension of the outputs `fyAdj`, N_Y. As such, the whole RHS vector could be computed from a single call to the differentiated function routine.

Solution of the Adjoint System The adjoint linear system of equations (7) has to be solved N_Y times because ψ is valid for all grid coordinates $\mathbf{x}$, but must be recomputed for each function Y. Each adjoint solver run requires the solution of a system of N_q equations but both the Jacobian matrix and the RHS vector in this system of equations are very sparse.

In order to solve this large sparse discrete adjoint problem, the Portable, Extensible Toolkit for Scientific Computation (PETSc) [32, 33] was used. 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 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.

All the adjoint and partial sensitivity matrices and vectors – $\frac{\partial \mathcal{R}}{\partial \mathbf{q}}$, $\frac{\partial Y}{\partial \mathbf{q}}$, $\frac{\partial \mathcal{R}}{\partial \mathbf{x}}$ and $\frac{\partial Y}{\partial \mathbf{x}}$ – were created as PETSc's data structures and, due to their structure, stored as sparse entities. Once the sparse data structures were assembled, the adjoint system of equations was solved using a PETSc built-in Krylov subspace (KSP) method, more specifically, a Generalized Minimum Residual (GMRES) method [34] was used.

Function Gradient Evaluation

Once the adjoint solution, ψ, is found, the gradient of the function of interest with respect to the grid coordinates is obtained from Eqn. (9).

Similarly to the partial derivatives with respect to the flow, the terms $\frac{\partial \mathcal{R}}{\partial \mathbf{x}}$ and $\frac{\partial Y}{\partial \mathbf{x}}$ were also computed using automatically differentiated routines. The same re-engineered routines of the flow solver residual (14) and objective function (15) used earlier were again differentiated automatically, this time with respect to the grid coordinates `xAdj`,`yAdj`,`zAdj`. This generated code that evaluated the entries of the matrix $\frac{\partial \mathcal{R}}{\partial \mathbf{x}}$ and vector $\frac{\partial Y}{\partial \mathbf{x}}$, respectively.

Using these AD codes, it was possible to compute the non-zero entries of the matrix and vector of partial derivatives in the total sensitivity equation (9). After these had been assembled, the adjoint-based sensitivity of the function of interest Y was then evaluated using the matrix-vector multiplication and the vector addition built-in operation routines provided in PETSc.

Gradient-Based Optimization Framework

For the reasons enumerated in the introduction, an efficient turbomachinery design framework should be controlled by a gradient-based optimizer. Such optimizer has to be provided with both the objective and constraint function values and gradients with respect to the design variables, as illustrated in Fig. 1.

From a design perspective, a turbomachine is geometrically represented not by the surface nodes coordinates but rather by some higher-level descriptors, such as stagger, camber angle distribution and thickness distribution. Let α denote the high-level geometric parameters that form the set of design variables.

While the function values can easily be computed by post processing the flow solution obtained from running the flow solver, the corresponding gradients require an additional solver – the adjoint solver. After the sensitivity of the objective function with respect to the grid coordinates is computed by Eqn. (9), it is still necessary to evaluate the sensitivity of the computational mesh with respect to those high-level parameters, $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\alpha}$, according to Eqn. (8).

Unless the source code of every tool involved in the grid generation process is available, it is necessary to use an approximation to estimate $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\alpha}$. In this work, a simple finite-difference approximation was used to accomplish that. This meant that for every design variable, it was necessary to re-grid the computational domain. However, since the grid topology was kept constant, it was possible to accelerate this process by means of grid morphing.

The computed final sensitivity $\frac{\mathrm{d}Y}{\mathrm{d}\alpha}$ is then used by the gradient-based optimizer to find the search direction and to determine the step size during the line search.

The schematic of such adjoint-based optimization algorithm is illustrated in Fig. 4.

It is important to notice that 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 active constraint

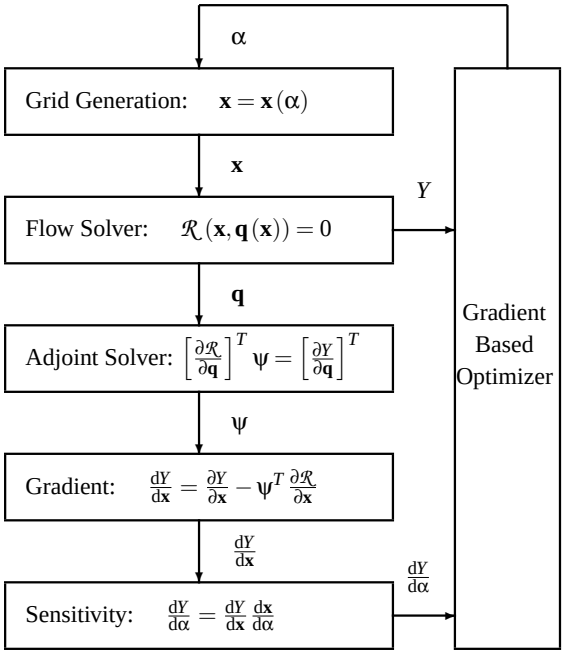

Figure 4. SCHEMATIC OF THE ADJOINT-BASED OPTIMIZATION ALGORITHM.

function, C_i, which includes the computation of a new right-hand side (RHS) for the system (7).

In terms of computational cost, the flow solver and the adjoint solver are the two main blocks of the process, being that the cost of solution of the adjoint equations is similar to that of the solution of the governing equations since they are of similar size and complexity. The sensitivity block consists of some linear algebra involving matrix-vector multiplication, thus relatively cheap to evaluate.

The advantage of the adjoint approach can be seen from Eqn. (9), which is independent of $\delta\mathbf{q}$, meaning that the gradient of Y with respect to an arbitrarily large vector of grid coordinates $\mathbf{x}$ can be determined without the need for additional solutions of the governing flow PDE. This capability to effectively handle design problems involving a large number of variables is what makes the adjoint methods well known for.

RESULTS

A transonic blade passage of a high-pressure compressor stage was used to demonstrate the capabilities of the discrete adjoint solver developed using the proposed hybrid approach. The rotor design pressure ratio is 1.5 at a mass flow rate of 145 lbm/s. The design rotational speed is 9,000 rpm. The rotor has 60 blades and an aspect ratio of 1.75 (based on average span/root axial chord). The three-dimensional geometry is shown in Fig. 5, where the casing wall has been removed for clarity.

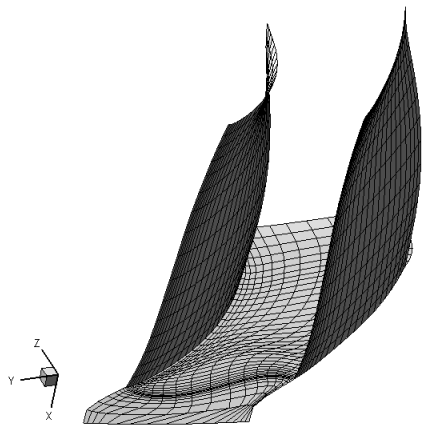

Figure 5. ROTOR BLADE PASSAGE.

The flow solver implements the boundary conditions by means of halo cells, that is, auxiliary cell layers that sit on the exterior of the physical domain. The boundary conditions of the case presented were made simple for demonstration purposes. As such, the inlet boundary had the absolute tangential velocity fixed and the pressure extrapolated from the interior. The exit pressure was fixed and the total internal energy and the velocity were extrapolated from the interior cells to the auxiliary cells at the exit. All solid walls were considered inviscid, being the momentum vector mirrored so that the resulting flux was null. The remaining faces were periodic, in which the state vector, transferred from the master to the slave faces, undergone a coordinate transformation according to rotationally periodicity of the geometry.

Figure 6 shows the contour of pressure and the momentum vector of the flow on the hub and blade surface planes corresponding to the baseline blade geometry. The values are given in the absolute reference frame. In the relative reference frame, the no-permeability condition is satisfied at the solid walls.

Having obtained the baseline flow solution, the corresponding adjoint solution is computed using Eqn. (7). Although an adjoint solution *per se* is of little use to a designer, for completeness, the adjoint solution is shown in Fig. 7 for pressure ratio ($Y = PR$). The contour plot corresponds to the adjoint of the continuity equation and the vector plot is for the adjoint of the momentum equation. As typical for the adjoint solution, the vector plot shows an adjoint flow some how reverse of the real flow.

The adjoint-based sensitivity of pressure ratio with respect to the grid coordinates evaluated using Eqn. (9) is illustrated in Fig. 8. The vector plot components correspond to the gradients of pressure ratio with respect to each coordinate component,

$$\frac{dPR}{d\mathbf{x}} = \frac{dPR}{dx}\mathbf{e}_x + \frac{dPR}{dy}\mathbf{e}_y + \frac{dPR}{dz}\mathbf{e}_z \qquad (16)$$

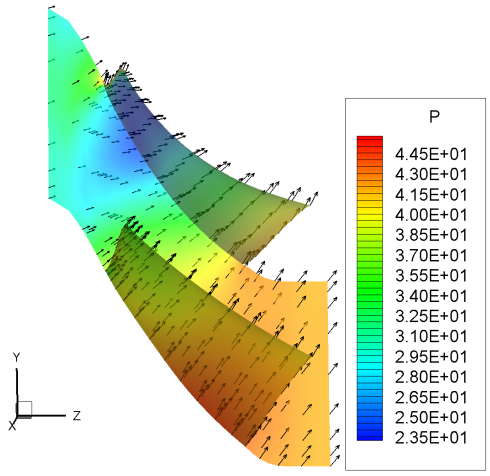

Figure 6. PRESSURE DISTRIBUTION AND MOMENTUM VECTOR.

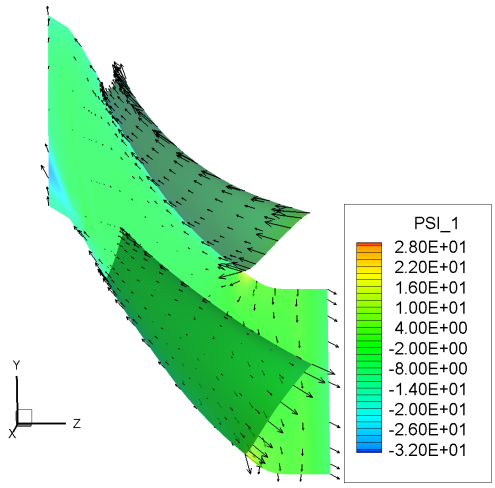

Figure 7. ADJOINT SOLUTION OF THE CONTINUITY EQUATION ($Y = PR$).

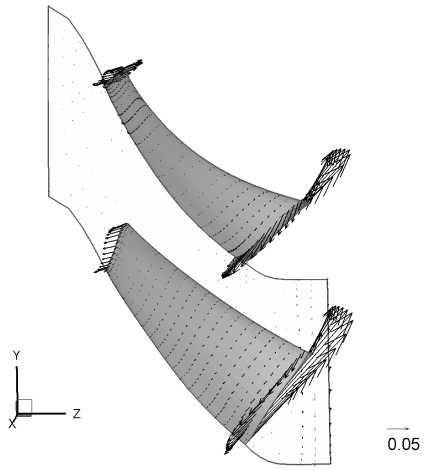

Figure 8. GRADIENT OF PRESSURE RATIO W.R.T. SURFACE NODES: $\frac{\mathrm{d}PR}{\mathrm{d}\mathbf{x}}$.

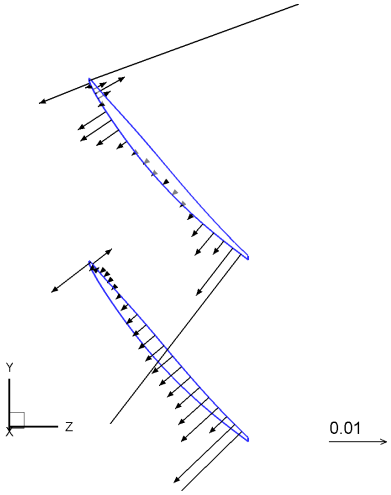

Figure 9. GRADIENT OF PRESSURE RATIO W.R.T. SURFACE NODES: $\frac{\mathrm{d}PR}{\mathrm{d}\mathbf{x}}$ AT MIDSPAN.

Each vector points into the direction of increased pressure ratio and its magnitude is the improvement per unit change in the grid node coordinate. Consequently, a designer can easily infeer from Fig. 8 how to tune the blade and/or endwalls for increased PR, since those vectors tell him how the surface geometry should change to accomplish it. The large vectors at the blade leading and trailing edges reveal how sensitive the machine performance is relative to these regions.

A cross-section located at blade midspan is presented in Fig. 9. The sensitivity of pressure ratio with respect to the blade shape is greater closer to the leading and trailing edges on the suction side, and gradually increases towards the trailing edge on the pressure side.

In the current implementation, the memory use was the largest because the full Jacobian matrix $\frac{\partial \mathcal{R}}{\partial \mathbf{q}}$ was pre-assembled. If an alternative free-matrix adjoint computations would have been used, this would considerably mitigated this issue, at the expense of larger runtimes, though. In addition, the coordinates of every node of the whole three-dimensional grid were taken as variables in the adjoint sensitivity computation. A change in the grid perturbation module to make it local to the nodes of the blade would also translate into memory savings.

To verify the adjoint-based gradients, the sensitivity of the

function of interest was compared to values obtained using both finite-difference and complex-step derivative approximations, using Eqn. (12) and Eqn. (13). While the adjoint approach allows for an efficient computation of the sensitivity with respect to every grid coordinate, it would have been computationally prohibitive to perform the same computation using either of the two approximations because it would imply running the flow solver again for each node, for each coordinate component. Consequently, only a few grid nodes were selected for verification. Figure 10 shows an example of such verification study for the grid node located on the suction side of the blade, at the 14.2% chord, at the midspan plane. Two functions of interest are shown, pressure ratio and isentropic efficiency. The vertical axis represent the gradient value and the horizontal axis the different step sizes h used for the approximations.

As expected using finite-differences, the gradient value is highly dependent of the step size chosen. The best matches were for $h = 10^{-5} - 10^{-4}$. Values greater than that produced significant truncation errors, whereas smaller lead to subtraction cancellation. The gradients estimated using the complex-step derivative approximation showed a much better accuracy, and because the approximation expression Eqn. (13) does not involved any subtraction, it was possible to use extremely small perturbation steps, in the order of $h = 10^{-20}$. The agreement between gradient values using the different sensitivity analysis methods is excellent, which proves the correct implementation of the adjoint method. Similar findings were obtained for every other grid node tested, and for different functions of interest.

After the successful verification of the adjoint solver, Hicks-Henne bump functions [35] were used to test the integration with the grid generation module and compute higher-level sensitivities of the form $\frac{dY}{d\alpha}$. By superimposing these shape functions on the baseline blade surface, smooth perturbations, that mimic the effect produced by geometric design variables α, are introduced in the grid. Normalizing the radial and axial coordinates with respect to the blade dimensions as $R_n = (R - R_{hub})/(R_{tip} - R_{hub})$ and $Z_n = (Z - Z_{LE})/(Z_{TE} - Z_{LE})$, respectively, the bump function can be expressed by

$$\delta(R\theta) = a[sin(\pi R_n^{log(0.5)/log(R_c)})]^{R_e}[sin(\pi Z_n^{log(0.5)/log(Z_c)})]^{Z_e}, \quad (17)$$

where a is the bump maximum height, R_c and Z_c are the coordinates of the bump center, and R_e and Z_e are the bump extension in the radial and axial directions, respectively.

Figures 11 and 12 show the perturbation corresponding to a bump of amplitude $a = 10^{-5}$, centered at $R_c = 0.5$ and $Z_c = 0.142$, with extension parameters $R_e = 5.0$ and $Z_e = 5.0$. The horizontal axes are the radial and axial coordinates of the rotor blade and the vertical axis is the perturbation on the combined variable $(R\theta)$, where θ is the circumferential angle defined in polar coordinates. This bump is centered at the same node used for

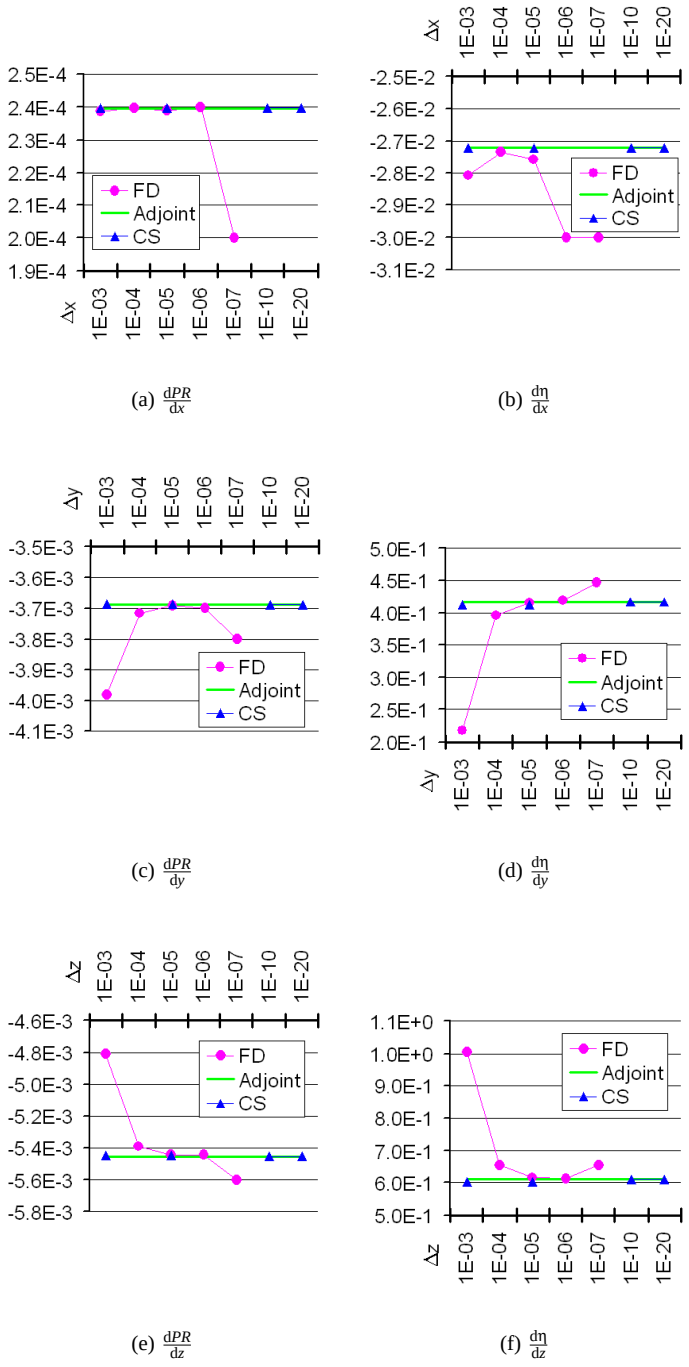

Figure 10. FUNCTION GRADIENT W.R.T. NODE COORDINATES.

the verification of the sensitivity with respect to the grid coordinates shown previously. When applied to the rotor blade, the bump produced the perturbation shown in Fig. 13.

The sensitivity of different functions of interest, namely mass flow, $\dot{m}$, efficiency, η and pressure ratio, PR, with respect to

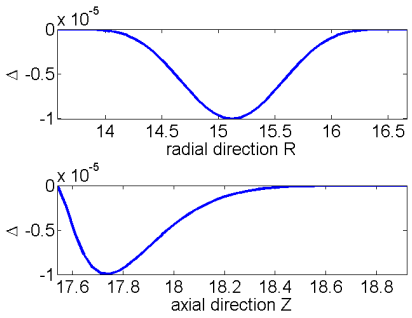

Figure 11. HICKS-HENNE BUMP: 1D PARAMETRIZATION.

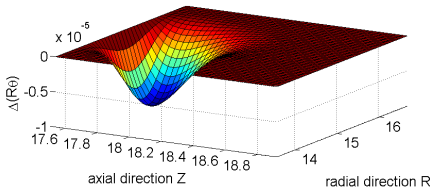

Figure 12. HICKS-HENNE BUMP: 2D PARAMETRIZATION.

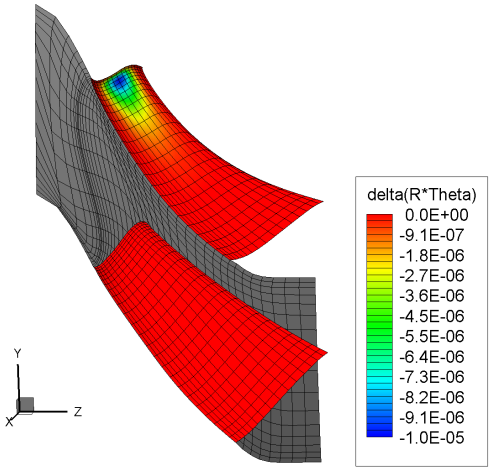

Figure 13. HICKS-HENNE BUMP: APPLIED TO THE BLADE

the bump amplitude, a, were computed using the adjoint solver. The term $\frac{d\mathbf{x}}{d\alpha}$ in Eqn. (8) was approximated by finite-differences using the baseline and perturbed computational grids. Table 1 summarizes these results, together with the comparison using full finite-difference derivative approximation using a perturbation step of $h = 10^{-5}$ on the bump amplitude. As shown in Tab. 1, there is again an excellent agreement between the adjoint-based gradient and the finite-difference derivative approximation.

Table 1. COMPARISON OF FUNCTION GRADIENTS.

	Adjoint	Finite-difference	Δ
$\frac{d\dot{m}}{da}$	$3.5540E+0$	$3.5550E+0$	-0.03%
$\frac{d\eta}{da}$	$2.9184E-2$	$2.9186E-2$	-0.01%
$\frac{dPR}{da}$	$4.5107E+0$	$4.5113E+0$	-0.01%

At this point, the adjoint solver implementation was considered successful and ready to be integrated in a gradient-based optimization framework.

CONCLUSIONS

A methodology for developing an adjoint solver for a legacy CFD solver that models the traditional flow governing equations has been presented. The adjoint approach enables large computational savings, at the expense of a more complex implementation, when compared to traditional sensitivity analysis methods such as finite differences.

The issue of rapidly developing the adjoint solver was tackled in this work. Instead of differentiating the entire flow solver using automatic differentiation, the discrete adjoint solver was derived with the aid of an automatic differentiation tool that was selectively applied to the CFD source code 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.

This approach has the advantages that it is applicable to arbitrary governing equations and functions and it eliminates errors that would have resulted from the necessary approximations if a manual differentiation had been used for the derivation. Furthermore, because this approach is largely automatic, it accelerates development time considerably and makes an adjoint solver now also accessible to the industry. These advantages come at the cost of increased memory requirements for the discrete adjoint solver. Nevertheless, the memory penalty is regarded as small given not only the significant advantages enumerated but also the amount of memory typically available in parallel high-performance computers.

The discrete adjoint solver developed was tested on a high-pressure rotor blade passage and the adjoint-based gradients of some functions of interest with respect to blade node coordinates and high-level shape parameters were verified against finite-difference and complex-step derivative approximations. Following the successful preliminary implementation of the adjoint solver, future efforts will concentrate on expanding it to fully support the flow solver capabilities and on integrating it into a gradient-based optimization framework. Upon completion of

such development tasks, the final framework should provide the designer with a turbomachinery tuning tool that, given a set of baseline geometric parameters, re-shapes the blade and endwall surfaces to meet or exceed the design goals. When such state-of-the-art tool is made available to the designers, an improved understanding of the design space could lead to highly-tuned machines in a much quicker timeframe.

The authors believe that the proposed hybrid methodology to develop discrete adjoint solvers makes turbomachinery design using gradient-based optimizers based on CFD analysis feasible in industrial environments.

ACKNOWLEDGMENT

The authors would like to thank Steven Ray, from GE Aviation, for the selection of the test case presented, his feedback during the revision of this paper and his overall support. Moreover, the authors are thankful to General Electric for giving permission to publish this paper.

REFERENCES

[1] Nocedal, J., and Wright, S. J., 1999. *Numerical optimization*. Springer.

[2] Griewank, A., 2000. *Evaluating Derivatives*. SIAM, Philadelphia.

[3] Wolfram, 2008. Mathematica 6. On the WWW. URL `http://www.wolfram.com`.

[4] Pironneau, O., 1974. "On optimum design in fluid mechanics". *Journal of Fluid Mechanics*, **64**, pp. 97–110.

[5] Jameson, A., 1988. "Aerodynamic design via control theory". *Journal of Scientific Computing*, **3**(3), Sept., pp. 233–260.

[6] Jameson, A., Pierce, N. A., and Martinelli, L., 1998. "Optimum aerodynamic design using the Navier–Stokes equations". In *Theoretical and Computational Fluid Dynamics*, Vol. 10. Springer-Verlag GmbH, Jan., pp. 213–237.

[7] Nemec, M., and Zingg, D. W., 2004. "Multipoint and multi-objective aerodynamic shape optimization". *AIAA Journal*, **42**(6), June, pp. 1057–1065.

[8] Reuther, J. J., Alonso, J. J., Jameson, A., Rimlinger, M. J., and Saunders, D., 1999. "Constrained multipoint aerodynamic shape optimization using an adjoint formulation and parallel computers, part 1". *Journal of Aircraft*, **36**(1), pp. 51–60.

[9] Reuther, J. J., Alonso, J. J., Jameson, A., Rimlinger, M. J., and Saunders, D., 1999. "Constrained multipoint aerodynamic shape optimization using an adjoint formulation and parallel computers: Part II". *Journal of Aircraft*, **36**(1), pp. 61–74.

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

[11] Nielsen, E. J., and Park, M. A., 2005. "Using an adjoint approach to eliminate mesh sensitivities in computational design". In Proceedings of the 43^{rd} AIAA Aerospace Sciences Meeting & Exhibit, no. AIAA 2005-0491.

[12] Lee, K.-H., Alonso, J. J., and van der Weide, E., 2006. "Mesh adaptation criteria for unsteady periodic flows using a discrete adjoint time-spectral formulation". In Proceedings of the 44^{th} AIAA Aerospace Sciences Meeting & Exhibit, no. AIAA 2006-0692.

[13] Jones, W. T., Nielsen, E. J., and Park, M. A., 2006. "Validation of 3D adjoint based error estimation and mesh adaptation for sonic boom prediction". In Proceedings of the 44^{th} AIAA Aerospace Sciences Meeting & Exhibit, no. AIAA 2006-1150.

[14] Marta, A. C., and Alonso, J. J., 2006. "High-speed MHD flow control using adjoint-based sensitivities". In Proceedings of the 14^{th} AIAA/AHI International Space Planes and Hypersonic Systems and Technologies Conference, no. AIAA 2006-8009.

[15] Duta, M. C., Shahpar, S., and Giles, M. B., 2007. "Turbomachinery design optimization using automatic differentiated adjoint code". In Proceedings of the ASME Turbo Expo 2007: Power for Land, Sea and Air, no. GT2007-28329.

[16] Wang, D. X., and He, L., 2008. "Adjoint aerodynamic design optimization for blades in multi-stage turbomachinery: Part I - methodology and verification". In Proceedings of the ASME Turbo Expo 2008: Power for Land, Sea and Air, no. GT2008-50208.

[17] Wang, D. X., He, L., Li, Y. S., Wells, R. G., and Chen, T., 2008. "Adjoint aerodynamic design optimization for blades in multi-stage turbomachinery: Part II - validation and application". In Proceedings of the ASME Turbo Expo 2008: Power for Land, Sea and Air, no. GT2008-50209.

[18] Marta, A. C., 2007. "Rapid Development of Discrete Adjoint Solvers with Applications to Magnetohydrodynamic Flow Control". Ph.D. Dissertation, Stanford University, Stanford, CA, USA, June.

[19] Marta, A. C., Mader, C. A., Martins, J. R. R. A., van der Weide, E., and Alonso, J. J., 2007. "A methodology for the development of discrete adjoint solvers using automatic differentiation tools". *International Journal of Computational Fluid Dynamics*, **21**(9–10), Oct., pp. 307–327.

[20] Giles, M. B., and Pierce, N. A., 2000. "An introduction to the adjoint approach to design". In *Flow, Turbulence and Combustion*, Vol. 65. Kluwer Academic Publishers, pp. 393–415.

[21] Nadarajah, S. K., and Jameson, A., 2000. "A comparison of the continuous and discrete adjoint approach to automatic aerodynamic optimization". In Proceedings of the 38^{th}

AIAA Aerospace Sciences Meeting & Exhibit, no. AIAA 2000-0667.

[22] Dwight, R. P., and Brezillon, J., 2006. "Effect of various approximations of the discrete adjoint on gradient-based optimization". In Proceedings of the 44^{th} AIAA Aerospace Sciences Meeting & Exhibit, no. AIAA 2006-0690.

[23] Cusdin, P., and Müller, J.-D., 2005. "On the performance of discrete adjoint CFD codes using automatic differentiation". *International Journal of Numerical Methods in Fluids*, **47**(6–7), pp. 939–945.

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

[25] Hascoët, L., and Pascual, V., 2005. "Extension of TAPENADE towards Fortran 95". In *Automatic Differentiation: Applications, Theory, and Tools*, H. M. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, eds., Lecture Notes in Computational Science and Engineering. Springer.

[26] Dervieux, A., Hascoët, L., Pascual, V., Koobus, B., and Vazquez, M., 2008. TAPENADE: web page. `http://www-sop.inria.fr/tropics/tapenade.html`. Accessed in October.

[27] Kroll, N., Becker, K., Rieger, H., and Thiele, F., 2006. "Ongoing activities in flow simulation and shape optimization within the German MEGADESIGN project". In Proceedings of the 25^{th} ICAS Congress, no. ICAS 2006-3.11.1.

[28] Lyness, J. N., and Moler, C. B., 1967. "Numerical differentiation of analytic functions". *SIAM Journal on Numerical Analysis*, **4**(2), pp. 202–210.

[29] Squire, W., and Trapp, G., 1998. "Using complex variables to estimate derivatives of real functions". *SIAM Review*, **40**(1), pp. 110–112.

[30] Martins, J. R. R. A., Sturdza, P., and Alonso, J. J., 2001. "The connection between the complex-step derivative approximation and algorithmic differentiation". In Proceedings of the 39^{th} AIAA Aerospace Sciences Meeting & Exhibit, no. AIAA 2001-0921.

[31] Martins, J. R. R. A., Sturdza, P., and Alonso, J. J., 2003. "The complex-step derivative approximation". *ACM Transactions on Mathematical Software*, **29**(3), Sept., pp. 245–262.

[32] Balay, S., Buschelman, K., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., and Zhang, H., 2004. PETSc users manual. Tech. Rep. ANL-95/11 - Revision 2.3.0, Argonne National Laboratory.

[33] Balay, S., Buschelman, K., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., and Zhang, H., 2008. PETSc: Web page. `http://www.mcs.anl.gov/petsc`. Accessed in October.

[34] Saad, Y., and Schultz, M. H., 1986. "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems". *SIAM Journal for Scientific and Statistical Computing*, **7**(3), July, pp. 856–869.

[35] Hicks, R. M., and Henne, P. A., 1978. "Wing design by numerical optimization". *AIAA Journal*, **15**(7), July, pp. 407–412.