Experiences with the Application of the ADIC Automatic Differentiation tool to the CSCMDO 3-D Volume Grid Generation Code*

Christian H. Bischof†

William T. Jones‡

Andrew Mauer†

Jamshid Samareh-Abolhassani‡

Extended Abstract submitted to
34th AIAA Aerospace Sciences Meeting and Exhibit
Argonne Preprint ANL-MCS-P512-0595

Abstract. Automatic differentiation (AD) is a methodology for developing reliable sensitivity-enhanced versions of arbitrary computer programs with little human effort. As such, it can vastly accelerate the use of advanced simulation codes in a multdisciplinary design optimization context, as the time for generating and verifying derivative codes is greatly reduced. In this paper, we report on the application of the recently developed ADIC automatic differentiation tool for ANSI C programs on the CSCMDO multiblock three-dimensional volume grid generator. The ADIC-generated code can easily be interfaced with FORTRAN derivative codes generated with the ADIFOR AD tool for FORTRAN 77 programs, thus providing efficient sensitivity-enhancement techniques for multilanguage, multidiscipline problems.

1 Introduction

Multidisciplinary Design Optimization (MDO) is a methodology for the design of complex engineering systems and subsystems that coherently exploits the synergism of mutually interacting phenomena. Typically, the MDO process proceeds in an iterative

*This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38, and by the National Aerospace Agency under Purchase Order L25935D.

†Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, {bischof,mauer}@mcs.anl.gov.

‡Computer Sciences Corporation, 3217 N. Armistead Ave., Hampton, Virginia 23666, {w.t.jones,j.s.abolhassani}@larc.nasa.gov.

fashion, with each MDO cycle including at least the generation of a numerical solution, determination of associated design sensitivities, and system optimization.

Of these parts, the generation of the numerical solution is the truly domain-dependent piece of this process, where in-depth knowledge of the problem at hand is used to develop the necessary simulation codes. For system optimization, on the other hand, a wide suite of existing optimization algorithms is readily available (see, for example, [11]).

An MDO problem of particular interest in the aerospace community is the design optimization of the high-speed civil transport. This problem, at the minimum, includes Computational Fluid Dynamics (CFD), and therefore numerical grid generation, as an integral part of the design process. The possible variation of simulation method employed for modeling this problem leaves the computation of design sensitivities for the simulation codes in question. Here, we should keep the following issues in mind:

Reliability: The computed derivatives should be computed accurately.

Computational Cost: The amount of memory and runtime required for the derivative code should be minimized.

Scalability: Whatever method we choose should be applicable to large codes.

Human Effort: A user should not have to spend much of his or her time on developing computational procedures for computing derivatives.

Traditionally, handcoding, finite difference approximations, and symbolic methods have been used for the computation of derivatives. However, these methods fall short with respect to the previously mentioned critera. The main drawback of finite difference approximations is their numerical unpredictability as well as their computational cost. In contrast, the handcoding and symbolic approach cannot be directly applied to large codes and require considerable human effort. In addition, significant effort has to be expended whenever the analysis code is modified.

Recently, so-called "automatic differentiation" (AD) tools have emerged as a promising approach for computing derivatives. AD techniques rely on the fact that every function, no matter how complicated, is executed on a computer as a (potentially very long) sequence of elementary operations such as additions, multiplications, and elementary functions such as `sin` and `cos` (see, for example, [6,12]). By applying the chain rule of derivative calculus over and over again to the composition of those elementary operations, one can compute, in a completely mechanical fashion, derivatives of f that are correct up to machine precision [8].

In this paper, we report on the applicaton of the recently developed ADIC automatic differentiation tool for ANSI-C programs on the CSCMDO multi-block three-dimensional volume grid generator. CSCMDO (Coordinate and Sensitivity Calculator for Multidisciplinary Design Optimization) is a general purpose grid generator taylored specifically to MDO applications. The next section gives a brief overview of CSCMDO. ADIC (Automatic Differentiation of C Programs) provides AD capability for codes written in ANSI C and the philosophy and approach underlying ADIC are described in § 3. In § 4, we report on the results obtained with sensitivity-enhanced versions of CSCMDO generated with ADIC. Lastly, we summarize our results.

2 The CSCMDO Multiblock 3-D Volume Grid Generator

CSCMDO is a general purpose, multi-block, three-dimensional, structured volume grid generator with specialized features which are highly suitable for MDO type grid modifications. The code is designed to execute in a batch environment with control provided via an ASCII user input file. The code is capable of modifying any of the six faces of a block to reflect changes in the optimized geometry as defined by an input surface(s). This section gives a brief overview of CSCMDO; a more detailed discussion can be found in [9].

With the code executing in a batch mode, it can be incorporated directly into the design loop as shown in Figure 1. As mentioned before, the "computational method" at the very least contains a CFD analysis. Information input from outside of the loop is generated one time before the loop is initiated. This information includes the baseline geometry surface(s), baseline CFD volume grid, and the user input file. The design loop is then rendered self sufficient requiring no further human intervention. CSCMDO operates within the design loop to provide automated volume grid generation/modification for each design cycle.

The surface definition(s) is provided in the form of a structured mesh of discrete point data. The number and distribution of points defining the surface are not required to match those of the desired CFD grid. However, sufficient point resolution must be provided so as to adequately define all surface curvature.

The baseline volume grid may be generated using any structured grid generation package. CSCMDO supports the file formats commonly used in the field of CFD. No restriction on grid topology is imposed. However, the topology of the volume grid must be consistent with the CFD analysis and capable of incorporating design cycle

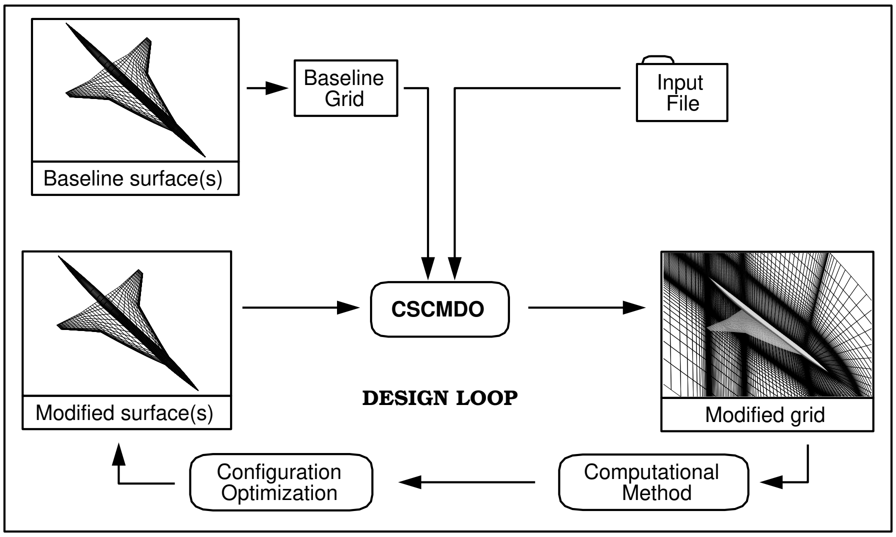

Figure 1: Integration of CSCMDO into the design loop

geometry modifications. The changes represented by modified geometry surface(s) are assumed to be small, but in the event that geometry changes do violate the volume topology, grid quality checks provide return codes to the software controlling the loop for appropriate action.

Individual block faces may be modified independently using a variety of methods including direct injection of a modified surface, parametric updates to a modified surface, projection to a modified surface, and deformations conforming to a modified surface. Volume modifications are accomplished for each block using algebraic re-initialization of the block interior, or by deformation of the original block interior based on changes defined on the six block faces.

3 The ADIC Automatic Differentiation Tool

Traditionally, two approaches to automatic differentiation have been developed: the so-called forward and reverse modes. These modes are distinguished by how the chain rule is used to propagate derivatives through the computation. We briefly summarize the main points about these two approaches, a more detailed description can be found in [2] and the references therein.

Let us assume that we have a function f that maps an n-vector x into an m-vector y. The forward mode propagates derivatives of intermediate variables with respect to the independent variables and follows the control flow of the original program. Exploiting the linearity of differentiation, the forward mode allows us to compute arbitrary linear combinations

$$J \cdot S \tag{1}$$

of columns of the Jacobian

$$J = \begin{pmatrix} \frac{\partial y(1)}{\partial x(1)} & \cdots & \frac{\partial y(1)}{\partial x(n)} \\ \vdots & & \vdots \\ \frac{\partial y(m)}{\partial x(1)} & \cdots & \frac{\partial y(m)}{\partial x(n)} \end{pmatrix}. \tag{2}$$

For an $n \times p$ matrix S, the effort required is roughly $O(p)$ times the runtime and memory of the original program. In particular, when S is a vector s, we compute the directional derivative

$$J * s = \lim_{h \to 0} \frac{f(x + h * s) - f(x)}{h}. \tag{3}$$

In contrast, the so-called reverse mode of automatic differentiation propagates derivatives of the final result with respect to an intermediate quantity, so-called adjoint quantities. To propagate adjoints, one must be able to reverse the flow of the program, and remember or recompute any intermediate value that nonlinearly impacts the final result. This may not be easily achieved for general programs, and these issues are further discussed in [2]. In either case, AD computes derivatives accurate to machine precision [8] and is directly applicable to computer programs of arbitrary length containing branches, loops, and subroutines.

From a user's perspective, AD tools preferably should behave like "black boxes", which, given the code describing the "function" to be differentiated, and an indication of which program variables correspond to the independent and dependent variables with respect to differentiation, generate an efficient "sensitivity-augmented" code for computing the desired derivatives without any need for human intervention.

Fundamentally, there are two approaches for augmenting a computer code with derivative computations. Languages like C++ or FORTRAN 90 support a language feature called "operator overloading," which allows the redefinition of the behavior of the elementary arithmetic operations and hence can be employed to attach, "under the rug", so to speak, derivative objects to original program variables, and apply the rules of differentiation one operation at a time. The ADOL-C tool [7] employs such an approach to compute derivatives of arbitrary order.

ADIC, in contrast, employs a source transformation approach to directly rewriting the source code to compute first-order derivatives. This approach requires considerable compiler infrastructure, and ADIC employs part of the Sage++ [4] source transformation infrastructure for C++ programs to transform ANSI C programs. With minor restrictions, the current ADIC prototype accepts arbitrary ANSI C programs and can handle, for example, subroutines, dynamic memory allocation, and pointers. The code generated is portable ANSI C code that can easily be modified to print out sensitivities, say. The features and limitations of ADIC are discussed in detail in [3], here we briefly summarize the main points.

ADIC, like ADIFOR, employs a hybrid forward/reverse mode approach to generating derivatives. For each assignment statement, the reverse mode is used to generate code that computes the partial derivatives of the result with respect to the variables on the right-hand side and then the forward mode is employed to propagate overall derivatives. Hence, ADIC and ADIFOR provide the directional derivative computation possibilities (see equation 1) associated with the forward mode of automatic differentiation. As a result, derivatives generated by a FORTRAN code differentiated with ADIFOR, say, can easily be used as "seed matrix" for an ADIC-augmented C code, whose derivatives, in turn, can again easily be ingested by a FORTRAN code. As we will see in the next section, this is of vital importance in the MDO context.

We also mention that ADIC can transparently exploit sparsity in derivative computations through the use of the SparsLinC library [2, 3], which, as a byproduct of the computation, will automatically compute the sparsity pattern of large sparse Jacobians. Information on ADIC and ADIFOR as well as application highlights and reports can be found on the world-wide web at

`http://www.mcs.anl.gov/autodiff/index.html.`

4 Experimental Results

To improve the aerodynamic performance of the high-speed civil transport, we embed the system shown in Figure 2 in an optimization context. The Rapid Airplane Parametric Input Design (RAPID) code [10] is written in FORTRAN 77, and, given geometric design variables (gdv) (for example camber) produces a aircraft surface grid (sfg). From this output, CSCMDO builds a 3-D volume grid (vlg). TLNS3D, a 3-D Navier-Stokes solver for turbulent flow [14, 13] then uses the geometry information as well as the stream parameters ($strm$) to compute the flow (flw) over the aircraft and from there, measures of performance such as lift or drag. In the MDO design context, we then need $\frac{\partial flw}{\partial gdv}$ at every pass through the design cycle.

Figure 2: Structure of CFD Design Problem

We mention several issues that are important to keep in mind when approaching this problem:

1. Computationally, this process is dominated by the CFD solver. The runtime of the grid generators is almost insignificant compared to that of TLNS3D.

2. Accurate sensitivities are required for all modules, and in particular for the grid generators, as errors in their derivatives would contaminate derivatives of TLNS3D.

3. The number of design objectives (flw) is small, whereas the number of surface grid points (sfg) and certainly the number of volume grid points (vlg) usually is very large.

4. The codes employed continue to be developed and refined.

In our experiments, we only concerned ourselves with computing the derivatives $\frac{\partial\, vlg}{\partial\, sfg}$, employing the RAPID and CSCMDO codes. The ADIFOR tool [1, 2] was applied to RAPID and a file containing $\frac{\partial\, sfg}{\partial\, gdv}$ was written by the sensitivity-enhanced version of RAPID. We also mention that ADIFOR has also been successfully applied to the single-block version of TLNS3D, and the results are summmarized in [5].

The ADIC prototype was applied to CSCMDO. The fact that the ADIC interface allows for "derivative chaining" is essential in this context. That is, instead of computing $\frac{\partial\, vlg}{\partial\, sfg}$, which would be a huge (albeit sparse) matrix, which then would be multiplied with $\frac{\partial\, sfg}{\partial\, gdv}$ to obtain $\frac{\partial\, vlg}{\partial\, gdv}$, we can *directly* compute $\frac{\partial\, vlg}{\partial\, gdv}$ by using $\frac{\partial\, sfg}{\partial\, gdv}$ as the "derivative seed matrix" associated with the inputs corresponding to the surface grid (sfg). As a result, the complexity of CSCMDO.AD, the ADIC-generated derivative

	Number of Design Variables						
	1	2	3	4	5	6	7
Best-Case FD	39	59	79	99	118	138	158
CSCMDO.AD (1)	130	174	223	281	324	526	560
CSCMDO.AD (2)	69	84	103	119	144	192	212

Table 1: Timing Results for RS/6000 (Times in User Seconds)

	Number of Design Variables						
	1	2	3	4	5	6	7
Best-Case FD	57	86	115	144	172	201	230
CSCMDO.AD (1)	149	205	278	351	465	546	637
CSCMDO.AD (2)	83	104	131	160	184	213	242

Table 2: Timing Results for Sun-4 (Times in User Seconds)

code for CSCMDO, depends on the number of geometric design variables (gdv), **not** on the number of surface grid points (sfg).

Varying the number of geometric design variables from 1 to 7, we compare the runtime performance of CSCMDO.AD with that of a one-sided finite difference approximation on a Sun Sparcstation 5 and IBM RS/6000 platform. Employing version 2.5.8 of the GNU C compiler with the `-O2 -ffast-math` compiler flags, we obtain the performance shown in Tables 1 and 2.

To generate a grid, CSCMDO can be thought of as using a two-step approach. First, it generates a grid, secondly, it validates the quality of the generated grid. In the line labelled "CSCMDO.AD (1)", we differentiated through the entire CSCMDO code, including the validation part. The line labelled "CSCMDO.AD (2)" refers to a somewhat more judicious use of CSCMDO, where derivatives are only propagated through the grid generation part, and not through the validation part. Note, however, that since ADIC intersperses derivatives and original program variables (see [3] for details), the validation part also had to be modified by ADIC to have data structures compatible with the sensitivity-enhanced grid generation part, but it did not include code for propagating derivatives. The line labelled "Best-case FD" lists the time required for a one-sided finite difference (FD) approximation of the derivatives, *under the assumption that the correct stepsize was chosen.* This is an optimistic assumption, as typically, several perturbation sizes had to be tried before a good match of the with FD derivative approximations with the values obtained by CSCMDO.AD was obtained.

We see that the code generated by ADIC when applied to all of CSCMDO is considerably slower than a "best-case" finite-difference approximation of derivatives. On the other hand, if we do avoid the (useless) derivative computation in the verification stage, CSCMDO.AD exhibits runtimes that are close to that of a best-case finite difference approximation. Note that we cannot avoid the verification stage in finite difference approximations, as the new volume grid arising from a parameter perturbation must be verified. We also mention that ADIC is still in the prototype stage and further improvements will narrow this gap. In either case, though, these runtimes are dwarfed by the runtime of a CFD flow code such as TLNS3D. However, the accuracy of the grid sensitivities is essential for the accuracy of the overall design sensitivities.

When examining our results, considering the sensitivity of the CFD volume grid X coordinate with respect to a change in wing root chord, we obtain the results shown in Figures 3 - 5. Note that our RAPID output assumes the wing trailing edge remains fixed with respect to the fuselage. Therefore, a change in the wing root chord should only be propogated forward of the wing trailing edge/fuselage intersection.

The volume grid shown consists of two blocks and over 525,000 grid points. Contour lines represent the sensitivity of the volume grid X component with respect to a change in the root chord. The expected symmetry can be noted in both figures. The grid X component can be seen as most sensitive to changes in the wing root chord around the leading edge of the wing/fuselage intersection. As mentioned, the surface grid points (sfg) and $\frac{\partial\, sfg}{\partial\, gdv}$ were obtained from the sensitivity-enhanced version of RAPID. This data was used by CSCMDO.AD to produce the volume grid points (vlg) and the sensitivities $\frac{\partial\, vlg}{\partial\, gdv}$ shown in the figures.

5 Conclusions

In this paper, we reported on the applicaton of the ADIC automatic differentiation tool for ANSI-C programs on the CSCMDO multi-block three-dimensional volume grid generator. CSCMDO (Coordinate and Sensitivity Calculator for Multi-disciplinary Design Optimization) is a general purpose grid generator tailored specifically for MDO applications. ADIC (Automatic Differentiation of C) is a prototype

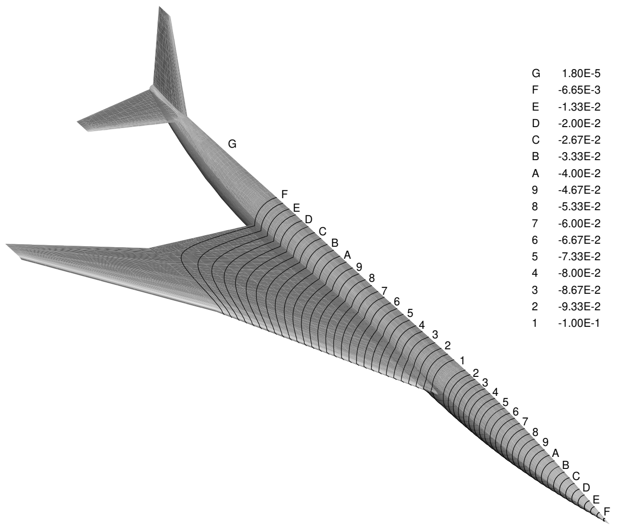

Figure 3: Sensitivity on the surface of CFD grid X coordinate wrt wing root chord

tool for the automatic sensitivity enhancement of codes written in ANSI-C. In our experience, ADIC allowed us to obtain accurate derivatives with a minimum of human effort. Without ADIC, we would have been forced to spend considerable effort developing a sensitivity-enhanced version by hand. Instead, this effort was more profitably spent improving the features of CSCMDO. Together with the ADIFOR tool for FORTRAN 77 programs, ADIC provides efficient support for multilanguage multidisciplinary design optimization where codes written in FORTRAN and C are embedded in the design loop.

Acknowledgments

We thank Larry Green of the Multidisciplinary Optimization Branch of NASA Langley for providing us with sensitivities from the RAPID code and Brad Homann of the Mathematics and Computer Science Division of Argonne National Laboratory for performing the CSCMDO benchmark runs. We are also indebted to Tom Zang of the Multidisciplinary Optimization Branch of NASA Langley for initiating this collaboration and to Dennis Gannon and his group at Indiana University for their support with Sage++.

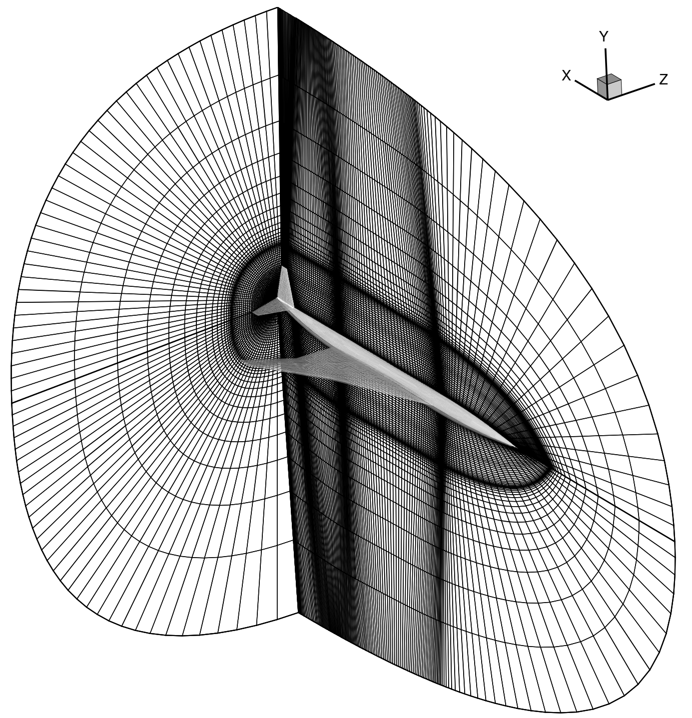

Figure 4: CFD volume grid produced by CSCMDO.AD

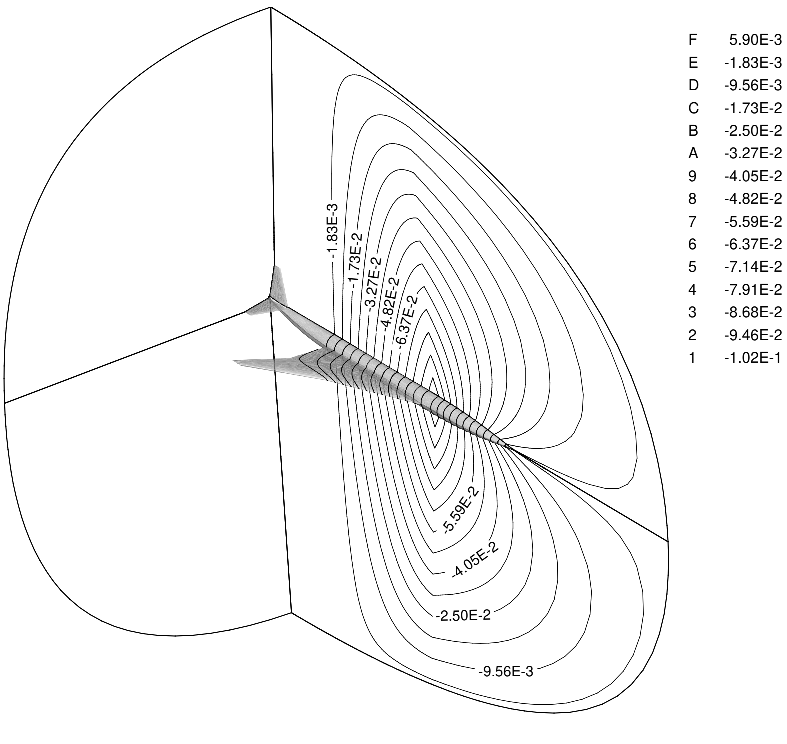

Figure 5: Sensitivity in the field of CFD grid X coordinate w.r.t wing root chord

References

[1] Christian Bischof, Alan Carle, George Corliss, Andreas Griewank, and Paul Hovland. ADIFOR: Generating derivative codes from Fortran programs. *Scientific Programming*, 1(1):11–29, 1992.

[2] Christian Bischof, Alan Carle, Peyvand Khademi, and Andrew Mauer. The ADIFOR 2.0 system for the automatic differentiation of Fortran 77 programs, 1994. Preprint MCS-P481-1194, Mathematics and Computer Science Division, Argonne National Laboratory, and CRPC-TR94491, Center for Research on Parallel Computation, Rice University.

[3] Christian Bischof and Andrew Mauer. ADIC – A tool for the automatic differentiation of C programs. Preprint MCS-P499-0295, Mathematics and Computer Science Division, Argonne National Laboratory, 1995.

[4] Francois Bodin, Peter Beckman, Dennis Gannon, Jacob Goutwals, Srinivas Narayana, Suresh Srinivas, and Beata Winnicka. Sage++: an object-oriented toolkit and class library for building Fortran and C++ restructuring tools. In *Proceedings of the Second Annual Object-Oriented Numerics Conference*. IEEE, 1994.

[5] Alan Carle, Lawrence Green, Christian Bischof, and Perry Newman. Applications of automatic differentiation in CFD. In *Proceedings of the 25th AIAA Fluid Dynamics Conference, AIAA Paper 94-2197*. American Institute of Aeronautics and Astronautics, 1994.

[6] Andreas Griewank. On automatic differentiation. In *Mathematical Programming: Recent Developments and Applications*, pages 83–108, Amsterdam, 1989. Kluwer Academic Publishers.

[7] Andreas Griewank, David Juedes, and Jay Srinivasan. ADOL-C, a package for the automatic differentiation of algorithms written in C/C++. Preprint MCS-P180-1190, Mathematics and Computer Science Division, Argonne National Laboratory, 1990.

[8] Andreas Griewank and Shawn Reese. On the calculation of Jacobian matrices by the Markowitz rule. In Andreas Griewank and George F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 126–135. SIAM, Philadelphia, 1991.

[9] William T. Jones and Jamshid Samareh-Abolhassani. A grid generation system for multidisciplinary design optimization. In *Proceedings of the Workshop on Surface Modeling, Grid Generation, and Related Issues in CFD Solutions*, pages 11–21, 1995. NASA-CP3291.

[10] Robert E. Smith, Malcolm G. I. Bloor, Michael Wilson, and Almuttil M. Thomas. Rapid Airplane Parametric Input Design (RAPID). In *Proceedings of the 12th AIAA Computational Fluid Dynamics Conference, San Diego, AIAA 95-1687*. American Institute of Aeronautics and Astronautics, June 1995.

[11] Jorge J. Moré and Stephen J. Wright. *Optimization Software Guide*. SIAM, Philadelphia, 1993.

[12] Louis B. Rall. *Automatic Differentiation: Techniques and Applications*, volume 120 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, 1981.

[13] V. N. Vatsa, M. D. Sanetrik, and E. B. Parlette. Development of a flexible and efficient multigrid-based multiblock flow solver. In *Proceedings of the 31st AIAA Aerospace Sciences Meeting, AIAA 93-0677*. American Institute of Aeronautics and Astronautics, 1993.

[14] V. N. Vatsa and B. W. Wedan. Development of a multigrid code for 3-D Navier-Stokes equations and its application to a grid-refinement study. *Computers & Fluids*, 18(4):391–403, 1990.