NASA TECHNICAL MEMORANDUM 107661

Automatic Differentiation as a Tool in Engineering Design

Jean-Francois Barthelemy
Laura E. Hall

August 1992

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

AUTOMATIC DIFFERENTIATION AS A TOOL IN ENGINEERING DESIGN

Jean-Francois M. Barthelemy, NASA/Langley Research Center
Laura E. Hall, Unisys Corporation

Abstract

Automatic Differentiation (AD) is a tool that systematically implements the chain rule of differentiation to obtain the derivatives of functions calculated by computer programs. In this paper, it is assessed as a tool for engineering design. The paper discusses the forward and reverse modes of AD, their computing requirements as well as approaches to implementing AD. It continues with application of two different tools to two medium-size structural analysis problems to generate sensitivity information typically necessary in an optimization or design situation. The paper concludes with the observation that AD is to be preferred to finite differencing in most cases, as long as sufficient computer storage is available; in some instances, AD may be the alternative to consider in lieu of analytical sensitivity analysis.

Introduction

Automatic Differentiation (AD) is a collection of computer science techniques which permit one to automatically calculate the derivatives of information generated by a computer program with respect to any parameter intervening in its calculation. Typically, to calculate the derivative of the output of a program with respect to its input, one modifies the original program by insertion of specialized instructions which identify relevant dependent and independent variables. The program is then modified automatically by a preprocessor which enhances it to calculate derivatives. The enhanced program is compiled conventionally, linked (if necessary) with special run-time libraries and executed to generate not only the original program's dependent variables but also their derivatives with respect to the independent variables.

AD is essentially an automatic implementation of the chain rule of differentiation based on tracking the relationships between dependent and independent variables. It produces exact derivatives, limited only by machine precision. There are two modes of AD. In the first, the forward mode, the chain rule is evaluated from the input to the output; in this mode, the computational cost increases with the number of inputs. In the second mode, the reverse mode, the computational cost increases with the number of outputs. In this mode, the chain rule is evaluated from the output to the input. While it can be much faster than the forward mode, this reverse mode can place enormous demands on computer storage and requires special memory handling.

AD is distinct from finite difference or symbolic manipulation techniques. The former, based on perturbations of a programs input, generates approximate derivatives which can be affected by round-off and truncation errors (Haftka and Gurdal, 1992[15]. While an exact technique, the later tends to generate very cumbersome expressions for the derivatives. There are a number of applications of AD in the literature, although a surprisingly limited number of them have to do with engineering design.

This paper describes an effort underway to assess the applicability of AD in engineering design. It has two major sections. The first is a brief introduction to AD based on some of the most recent publications on the subject. It discusses the two modes of AD, addresses the issue of computer cost, presents different forms of AD tools and briefly discusses some results. The second section reports on applications of two different AD tools to generate sensitivity information for two representative structural applications.

Automatic Differentiation, a Brief Introduction

This section gives an introduction to AD. It draws heavily on existing literature, notably the excellent monograph by Rall (1981)[25] and papers by Iri (1984)[20] and Griewank (1991a and b)[9,10]. Another good source is the collection of papers presented at a recent symposium on the subject and which was edited by Griewank and Corliss (1991)[13].

Directed Graph Representation of a Function

The basic concepts of AD are illustrated by means of a directed graph representation of the calculations for a set of functions. The illustrative example selected is the traditional symmetric three-bar truss problem (Fig. 1). The analysis equations relating dependent variables y's to the independent variables x's are given in Eq. 1 with y_i the stress in bar i, y_4, the weight of the truss, and where x_1 is the cross-sectional area of the oblique members and x_2 that of the vertical member.

$$\begin{aligned} y_1 &= \frac{20(x_2+\sqrt{2}x_1)}{2x_1x_2+\sqrt{2}x_1^2} \\ y_2 &= \frac{20\sqrt{2}x_1}{2x_1x_2+\sqrt{2}x_1^2} \\ y_3 &= \frac{-20x_2}{2x_1x_2+\sqrt{2}x_1^2} \\ y_4 &= 10\left(2\sqrt{2}x_1+x_2\right) \end{aligned} \qquad (1)$$

Figure 2 gives a simplified directed graph representation of the calculations involved in Eq. 1. At the bottom

of the graph appears one vertex for each independent variable, and, at the top, one for each dependent variable.

Intermediate vertices correspond to intermediate variable values obtained by elementary operations on variables at lower levels*.

The arcs joining the different vertices represent the direction of information flow in the graph. To each arc, one may automatically attach the value of the partial derivative of the variable at the end of the arc with respect to the variable at the origin of the arc (Fig. 3).

The directed graph representation clearly identifies the computations involved in a given calculation. Therefore, it gives a measure of the computational cost associated with that calculation, sometimes referred to as computational complexity of the function.

* This representation is simplified in the sense that multiplications by constants are not identified as separate elementary operations, even though, they are, from a strict computational standpoint. This assumption is made for the sake of simplifying the discussion.

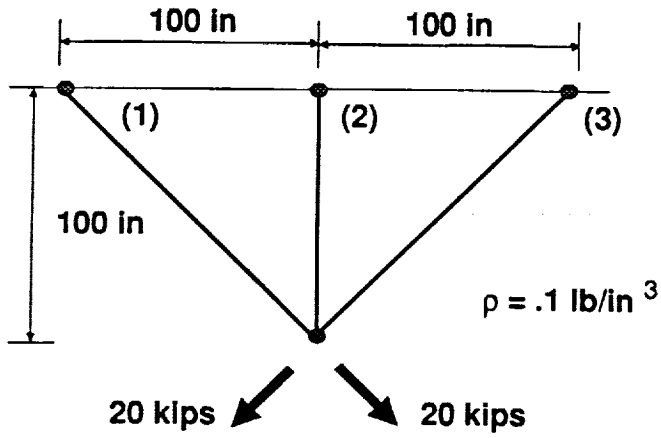

Figure 1 Three-bar truss

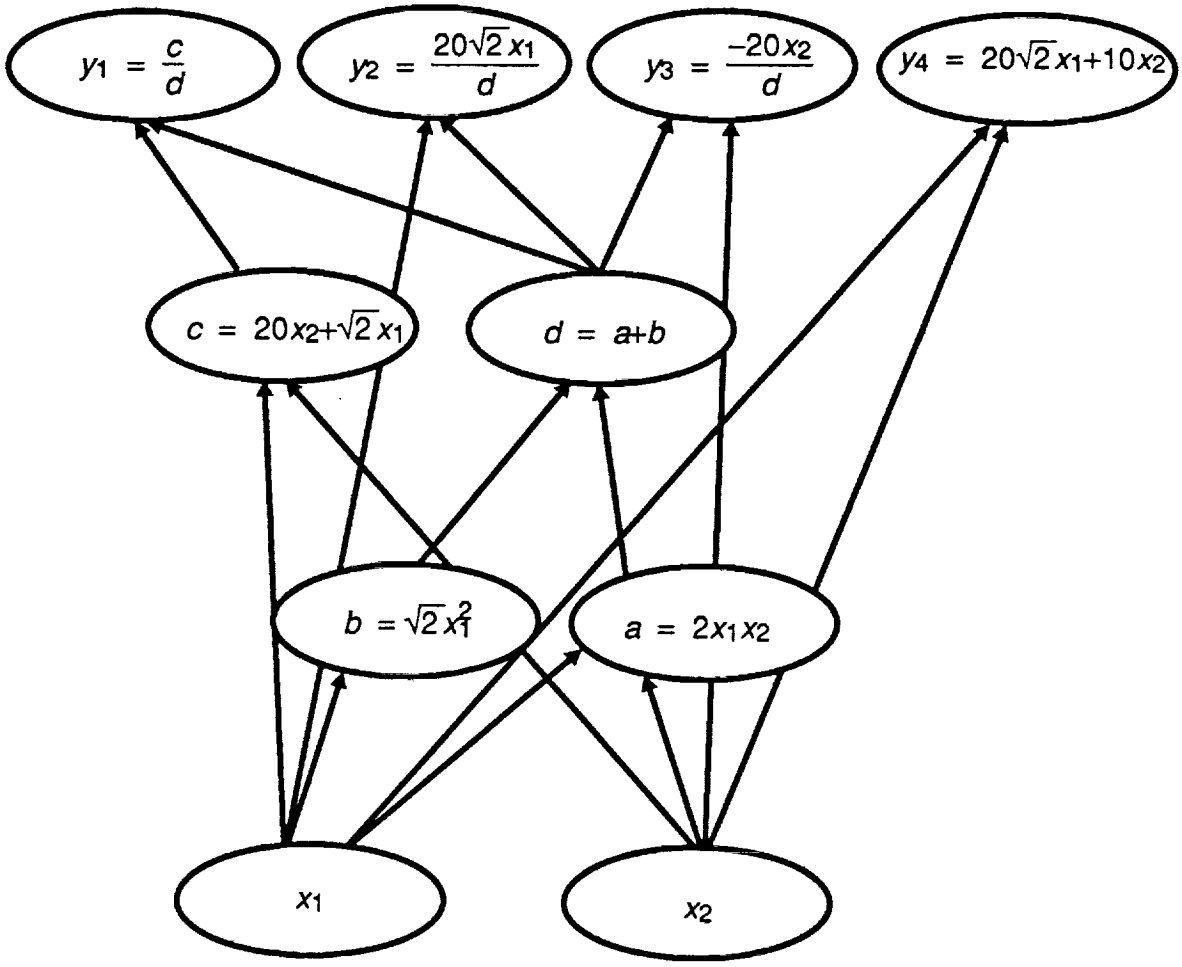

Figure 2 Directed graph representation of Eq. 1

Forward and Reverse Modes of Differentiation

The forward mode of differentiation, also called bottom-up mode, implements the chain rule of differentiation, starting with the independent variables. At each vertex is associated the numerical value of the total derivative of the intermediate variable with respect to the relevant independent variable. Referring to Fig 3, for example, to calculate the derivative of y_3 with respect to x_2, we proceed with the following calculations:

$$\begin{aligned}
\frac{da}{dx_2} &= 2x_1 \\
\frac{dd}{dx_2} &= \frac{\partial d}{\partial a}\frac{da}{dx_2} = 1 * 2x_1 \\
\frac{dy_3}{dx_2} &= \frac{\partial y_3}{\partial d}\frac{dd}{dx_2} + \frac{\partial y_3}{\partial x_2} \\
&= \frac{20x_2 * 1 * 2x_1}{\left(2x_1x_2 + \sqrt{2}x_1^2\right)^2} + \frac{-20}{\left(2x_1x_2 + \sqrt{2}x_1^2\right)}
\end{aligned} \tag{2}$$

In this equation, the $d(.)/d(.)$ terms identify total derivatives, the $\partial(.)/\partial(.)$ are partial derivatives. At each vertex, the derivative of the corresponding intermediate variable is the sum of contributions from each incoming arc involving, for each arc, the total derivative at the vertex at the origin of the arc times the partial associated with the arc. The calculation of the derivative values may proceed along with that of the function values with the intermediate variable values being calculated along with the values of their derivatives with respect to the independent variables. At any time, the intermediate variables and their derivative values which are required in subsequent calculations must be stored.

In contrast, the reverse mode of differentiation, also known as the top-down or backward mode, implements the chain rule starting with the dependent variables. Here, at each vertex is associated the numerical value of the derivative of the relevant dependent variable with

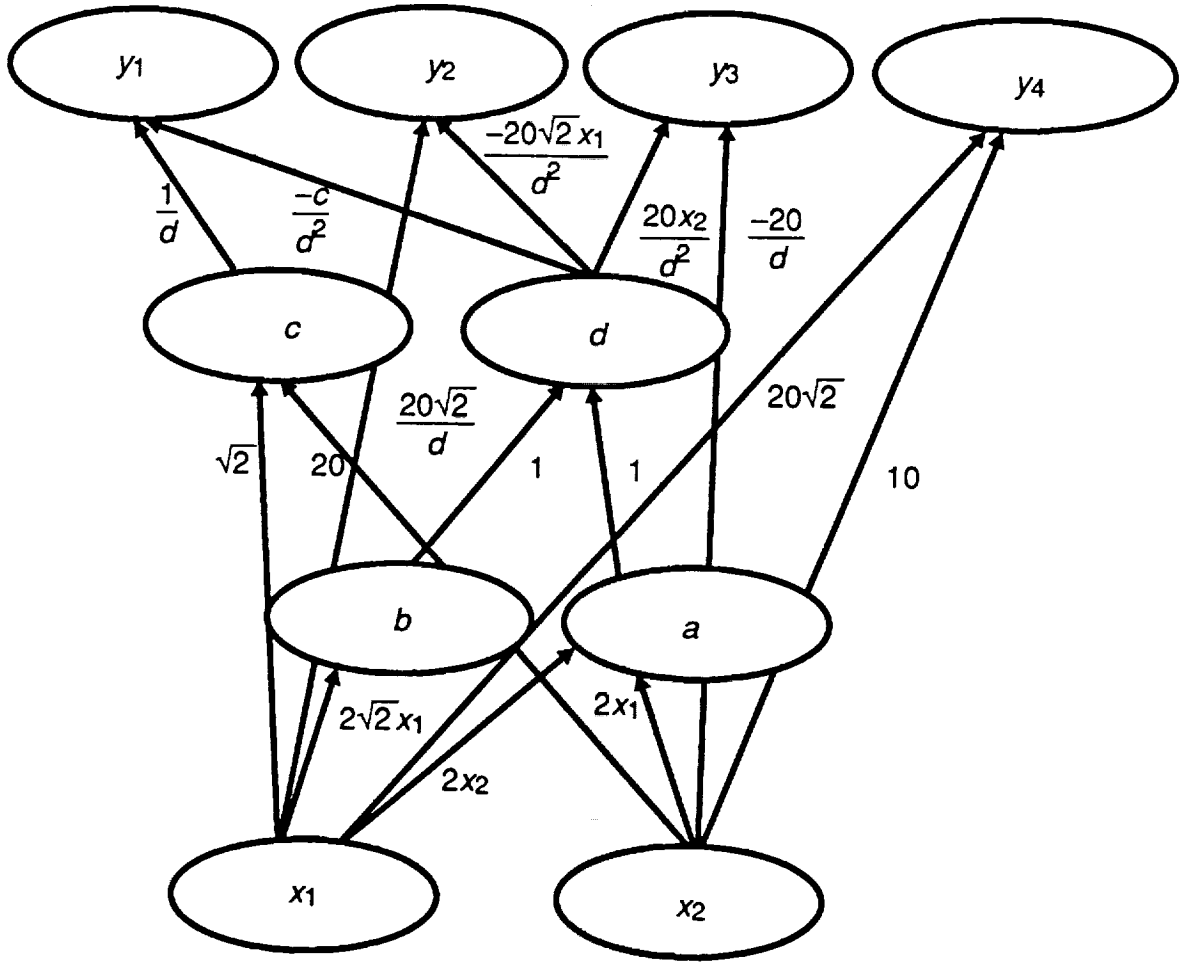

Figure 3 Partial derivatives for the arcs of the directed graph of Eq. 1

respect to the corresponding intermediate variable. If the same derivative is sought, then the following calculations result:

$$\begin{aligned}
\frac{dy_3}{dd} &= \frac{20x_2}{\left(2x_1x_2+\sqrt{2}x_1^2\right)^2} \\
\frac{dy_3}{da} &= \frac{dy_3}{dd}\frac{\partial d}{\partial a} = \frac{20x_2 * 1}{\left(2x_1x_2+\sqrt{2}x_1^2\right)^2} \\
\frac{dy_3}{dx_2} &= \frac{dy_3}{da}\frac{\partial a}{\partial x_2} + \frac{\partial y_3}{\partial x_2} \\
&= \frac{20x_2 * 1 * 2x_1}{\left(2x_1x_2+\sqrt{2}x_1^2\right)^2} + \frac{-20}{\left(2x_1x_2+\sqrt{2}x_1^2\right)}
\end{aligned} \tag{3}$$

Since this calculation traces back from the dependent variables to the independent variables, It has to occur after a bottom-up sweep through the computational graph to obtain the independent variables and the partial derivatives of the arcs. In principle, all the intermediate calculations, potentially a vast amount of data, have to be stored in memory for use in the top-down sweep.

Cost of Automatic Differentiation

Observation of the directed graph gives a sense of the cost associated with AD. As pointed out by Iri (1984)[20], for example, the cost of calculation of a function is proportional to the number of vertices of the graph, while calculating the partial derivatives of that function adds a cost proportional to the number of arcs in the graph. In addition, the cost of calculating the partial derivatives associated with the arcs is negligible, once the function has been calculated. Therefore, without distinguishing between forward or reverse mode of differentiation, we should expect the cost of calculating a derivative of a function to be of the same order as that of calculating the function.

Starting with one independent variable, the forward mode gives the partial derivatives of all dependent variables, therefore one should expect the cost of calculating derivatives by the forward mode to be proportional to the number of independent variables. On the other hand, starting with one dependent variable, the reverse mode gives its derivatives with respect to all independent variable; therefore its cost is expected to increase with the number of dependent variables.

For a single function f of a vector variable $\mathbf{x}$ of n variables, results from Iri (1984)[20] and Griewank (1991c)[11] can be combined to give, for the forward mode of differentiation,

$$1+\frac{n}{c} \le \frac{L(f,\nabla f)}{L(f)} \le 4n \tag{4}$$

where c is a constant, $L(f)$ is the cost of calculating the function and $L(f,\nabla f)$ is that of calculating the function and its gradient with respect to $\mathbf{x}$. In this case, the cost of calculating the gradient of a function increases with the number of design variables. In contrast Iri (1984)[20] shows that, for the reverse mode,

$$1 \le \frac{L(f,\nabla f)}{L(f)} \le 6 \tag{5}$$

a bound independent of the number of variables.

In engineering applications using nonlinear programming, one is most often concerned with finding the Jacobian matrix $\mathbf{J}$ for a vector of m functions $\mathbf{f}$ (dependent variables) with respect to a vector of n independent variables $\mathbf{x}$. Iri (1991)[22] gives the following bounds. For the forward mode:

$$1 \le \frac{L(\mathbf{f},\mathbf{J})}{L(\mathbf{f})} \le \phi(n) \tag{6}$$

while, for the reverse mode:

$$1 \le \frac{L(\mathbf{f},\mathbf{J})}{L(\mathbf{f})} \le 1+3m \tag{7}$$

Again, the cost of the forward method is proportional to the number of independent variables and that of the reverse method is proportional to the number of dependent variables.

Griewank and Reese (1991)[14] show that, just as in application of finite differencing, knowing the sparsity of the Jacobian in advance can reduce the cost of AD. In such a case, the upper bounds in Eqs. 6 and 7 reduce respectively to $3\hat{n}$ or $3\hat{m}$, where $\hat{n} \le n$ is the maximum number of non-zero entries in the columns of $\mathbf{J}$ and $\hat{m} \le m$ is the maximum number of non-zero entries in its rows.

From the standpoint of storage, Griewank (1991b)[10] shows that a straightforward implementation of the forward mode should require on the order of n times the random access storage and exactly the same sequential access storage as required for the calculation of the functions.

On the other hand, a straightforward implementation of the reverse mode requires on the order of m times the random access storage of the functions. Since intermediate results must be stored until the reverse sweep, the sequential access storage of the reverse mode is the sum of that required for the functions plus a term proportional to the total number of mathematical operations in the calculations of the functions. This latter term can be significant and totally negate the computational cost benefit associated with the reverse mode.

For the reverse mode, there is a direct trade-off between operation count and sequential access storage required. Indeed, the storage requirements can be reduced by not storing all intermediate information during the bottom-up sweep to calculate the functions but by regenerating it during the top-down sweep for the

derivatives. Griewank (1992)[12] discusses these trade-offs.

It must be noted that the above bounds are only estimates, based on very general assumptions on the problem at hand. Actual performance can be significantly affected by the exact computational steps required in the problem considered and by also by the specific implementation of the AD methodology.

Developments

Automatic Differentiation Software AD software automatically transforms a description of the functions of interest into a computer program calculating the functions and their derivatives. The initial description of the functions can be either in symbolic or in computer program form. Juedes (1991)[19] provides a detailed review of 29 software tools for AD. Of those, only a handful are commercial programs, most others are research programs available from their authors. Some of those tools provide both modes of AD; some provide derivatives of higher order. Juedes describes five classes of tools, according to how the transformation is effected between the description of the original functions and the code for their derivatives.

Elemental AD tools provide the user with a set of subroutines to perform elementary numerical calculations and their derivatives. These subroutines use as input the arguments of the elementary calculation and their derivatives with respect to the relevant independent variables and return as output the result of the operation and its derivatives. The user must then use these subroutines when developing the code to calculate the functions.

Extensional AD tools work with original codes written in a conventional programing language (eg FORTRAN). These tools typically are preprocessing compilers. They take the original code, and produce an enhanced code in the same programing language. The enhanced code may then be compiled conventionally, linked with run-time libraries if necessary and executed.

Operational AD tools are similar to extensional AD tools however they apply to original codes written in a flexible modern programing language (eg C++). They define new data types for functions of which the derivatives are required and provide for the capability to automatically generate the derivatives of the functions defined in the new data types.

Integral AD tools are typically elements of special-purpose high-level computer languages that provide the capability to calculate the derivatives of expressions formulated in those languages.

Symbolic AD tools begin with a symbolic representation of the functions to be calculated, use algebraic manipulation to generate the derivatives of the functions and then automatically produce a computer program to calculate functions and derivatives.

It must be noted that some tools are actually hybrids, belonging to several of the classes. From the standpoint of applying AD to engineering optimization, both extensional and operational tools offer the best prospect for immediate application since they are likely to be directly applicable to existing analysis programs. In contrast, elemental, integral or symbolic tools should be considered only if a new analysis program is developed.

Automatic Differentiation Applications Even though AD methodology has been in development for close to 30 years, applications are remarkably few, particularly in the area of engineering design and optimization. In the volume edited by Griewank and Corliss (1991)[13] numerous potential areas of application are identified but few results are actually discussed. One notable exception is the work of Worley (1991)[26] reporting on numerous applications of the GRESS (Horwedel, 1991a and b)[17,18] computer program (see next section) in both forward and reverse modes. He systematically reports on 16 applications taken mostly from the area of contaminant transport modelling. The applications cover large programs with up to 16000 lines of code. One application of the reverse method to shallow-land disposal of radioactive waste included 69000 independent variables and 2 dependent variables and provided all necessary derivatives in 10 times the run-time of a single analysis. An application of the forward mode to a radioactive decay model with 7 independent variables and 140000 dependent variables required 25 times the run-time of a single analysis.

Bischof *et al* (1991)[4] introduced the program ADIFOR (see next section) and reported on a large number of test problems with small to moderate size (less than 1500 lines) computer codes. They show AD generally faster (up to 70%) than finite difference; in one example they show AD actually faster than analytically developed derivatives.

Other applications include the work of Garcia (1991)[6] who uses AD to fit complex models of growth in forest plantations and shows reductions in derivative computing times by factors of 4 to 6 when compared to a central difference procedure for problems with one dependent variable and up to 18 independent variables. Iri (1988)[21] demonstrates the use of AD-derived Jacobians in the solution of nonlinear equations modelling a distillation tower. In a problem with 108 independent and dependent variables, Iri demonstrates calculation of derivatives 6 to 7 times faster than by forward differencing.

An area for application of the reverse method is for models described by large numerical systems where there are typically many more inputs than outputs;

prime examples of such systems are large meteorological or oceanographic models. It can be shown that the sensitivity information required to solve typical inverse design problems (parameter estimation, data fitting or data assimilation) for such models may be found from integration of an adjoint numerical system. It turns out that the adjoint is equivalent to the reverse mode of differentiation. Since a large amount of time goes into developing such models, the developing of automatic methods to code the adjoint systems can be very beneficial. Tallagrand (1991)[23] discusses that application in the context of meteorological modelling, Thacker (1991)[24] from the perspective of the oceanographer.

Two Structural Applications

This section discusses two exploratory applications of AD to generate sensitivity information commonly used in structural optimization. The first uses the GRESS code developed by Horwedel (1991a and b)[17,18] at Oak Ridge National Laboratory to calculate derivatives of weight, displacements and stresses in trusses analyzed with a small finite element analysis program. The second uses the ADIFOR code developed by Bischof *et al* (1991)[4] at Argonne National Laboratory to find derivatives of stresses in a plate model of a supersonic transport wing.

GRESS Applied to Finite Element Analysis with STAP

GRESS is a hybrid AD tool which has characteristics of both extensional and symbolic tools. GRESS offers the two modes of AD. The CHAIN option implements the forward mode and produces derivatives of intermediate variables with respect to selected independent variables, as they are calculated. The ADGEN option implements the reverse mode. As the analysis is performed, the ADGEN option generates partial derivatives for all assignment statements in the model and stores those. Then that information is read back and processed to generate the derivative of selected dependent variables with respect to all independent variable. This storage of intermediate information is in-core for a small enough problem but can be moved out-of-core for larger problems. For the ADGEN option, GRESS uses several techniques to reduce the amount of intermediate information retained, including retaining only derivative information depending on selected independent variables and affecting selected dependent variables

Given a FORTRAN program performing an analysis, the user must augment it with statements identifying dependent and independent variables as well as the AD mode required. GRESS precompiles this modified code to produce another FORTRAN code enhanced with derivative taking capabilities. The enhanced code is then compiled and linked with run-time libraries. GRESS is available for both VAX/VMS and UNIX computers; the results given here were obtained with the UNIX operating system. GRESS accepts most ANSI standard FORTRAN 77 statements but disallows functions that may be discontinuous and complex functions; it does not allow the use of scratch files during execution (Horwedel 1991a)[17].

Table 1 shows timing results obtained using GRESS to obtain the derivatives of volume, stresses and displacements in trusses analyzed with the simple finite element program STAP (Bathe and Wilson, 1976)[3]. The results were validated by comparing them with finite difference derivatives; for the smallest example, analytical results were available as well and compared exactly with the GRESS-generated results. The table shows that, except for the largest problem, both the forward mode and the reverse mode with in-core storage of intermediate results are noticeably faster than the finite difference alternative, with the reverse mode being fastest. The reverse mode with out-of-core storage of intermediate results requires considerably more time than the other two approaches due to its high I/O requirements. However, for the largest problem, it is the only AD alternative and it requires much more time even than the finite difference alternative.

ADIFOR Applied to Equivalent Plate Analysis with ELAPS

ADIFOR is a recent development. It is an extensional tool that implements a hybrid combination of the forward and reverse modes of AD. The program operates primarily in the forward mode, but implements the reverse mode for each complex assignment statement. Since it is based primarily on the forward mode of AD, ADIFOR's cost increases with the number of independent variables in the problem treated. However, it is capable of exploiting known sparsity of the Jacobian matrix so that the cost increases only proportional to the maximum number of structurally orthogonal# columns of the Jacobian.

Recognizing that the development of derivative code is an application of program translation, ADIFOR is based on tools from the ParaScope programming environment (Callahan *et al*, 1988)[5] which was developed for automatic parallelization of FORTRAN programs. Although operating ADIFOR is somewhat similar to operating GRESS, ADIFOR does not require any run-time library.

Table 2 lists timing results for structural analysis and sensitivity analysis in a plate model of a Mach 2.4 supersonic transport wing. The details of the model and analysis are given by Barthelemy *et al* (1992)[2]. This

column J_m and J_n of the Jacobian are structurally orthogonal if $J_{in}{}^{*}J_{im}=0$, for all i.

Table 1 Timing for generation of derivatives of volume, stress and displacements in trusses with respect to member cross-sectional areas (SPARCstation 1+, 16Mb CPU)

Number of bars, n_b	3	25	52	200
Number of nodes, n_n	4	10	20	77
Number of load cases, n_l	2	2	1	3
Reference	Haug & Arora (1979)	Haug & Arora (1979)	Barthelemy & Riley (1988)	Haug & Arora (1979)
Independent variables, n	3	25	52	200
Dependent variables, m[a]	31	111	113	1294
Function calculation time, secs (STAP)	.2	.3	.4	2.3
Derivative calculation time, secs:				
Finite differences	.4	8.6	28	592
Forward mode, in-core	.2	3.3	7.0	-
Reverse mode, in-core	.5	.6	.8	-
Reverse mode, out-of-core	1.	41	56	7259

[a] $m=1+n_l*(3*n_n+n_b)$

analysis is based on Giles' (1986, 1989)[7,8] program ELAPS which uses a Rayleigh-Ritz approach to analyze wing structures. The problem independent variables are skin thicknesses and spar and rib cap cross-sectional areas; the dependent variables are maximum strains and stresses in the wing covers in five different load cases. A preprocessor program transforms the 44 input skin thicknesses and cap areas into 136 ELAPS inputs; ELAPS is by far the longest running of the two codes. Applying AD to the preprocessor and to ELAPS separately and then using the chain rule to calculate the derivatives of the output of ELAPS with respect to the inputs of the preprocessor would yield a cost driven by the number of inputs to ELAPS, that is136. Instead, the preprocessor and ELAPS are merged into one program and the cost of applying AD is now driven by the number of inputs to the preprocessor which is 44.

Table 2 show results selecting only 4 of the independent variables of the problem or all 44 or them. No special purpose finite difference code was written for this example but the results show a reduction of 40% to 60% of time with respect to a simple-minded implementation of the differencing process amounting to rerunning the basic analysis program, once for the baseline analysis and once for each independent variable. Here the derivatives were validated by comparison with hand-calculated finite difference results.

Table 2 Timing for generation of derivatives of strains and stresses in a plate model of a wing with respect to skin thicknesses and rib and spar cap cross-sectional areas (SPARCstation IPX, 16 Mb CPU)

Independent variables	4	44
Dependent variables	16500	16500
Analysis time, secs	46	46
Derivative time, secs	141	771
Finite differences (est.), secs	230	2070

Discussion

The results discussed in this paper as well as others reported on in the literature establish clearly that AD generates accurate derivatives, generally faster than the finite difference alternative. Speed-up factors of up to one order of magnitude have been reported with existing AD tools, speed-up factors of 2 are not unusual. While some of the examples reported on are quite large, most are of moderate size, with the original non-modified code seldom larger than 3000 lines of codes. While the estimates discussed in the paper and other available in the literature indicate useful trends in computational cost and storage requirements, they are not sharp enough to decide unequivocally when AD is cheaper than finite differencing.

In general, AD is simpler to implement than analytical sensitivity analysis. However, it is unlikely that pure systematic (even clever) application of the chain rule of differentiation will prove in general faster than analytical sensitivity analysis. Indeed, when calculating derivatives analytically, all possible simplifications can be effected prior to doing any coding and very com-

pact formulations can be derived. A simple example is that of a function found as the solution of a single nonlinear equation (for example $y = sin(xy)$) and obtained by some sort of iterative process. AD will generate an iterative process for the derivative that mirror exactly that for the calculation of the function. It turns out, however, that the derivative can be found analytically, without iteration, from a linear sensitivity equation. Convergence of the iterative process for the derivative may require more or less iteration than that for the function and the AD procedure used must insure convergence of that process. In addition, the iterative calculation for the derivative will be significantly more costly than the analytical solution. In general, AD will not be able to overcome such difficulty, unless some symbolic manipulation capability is added. There are counterexamples however where AD proves faster that analytical sensitivity analysis, as reported by Bischof *et al* (1991)[4].

From these observations, and provided that the source code is available for the analysis program, AD is recommended over finite differencing for moderately sized computer programs. For larger programs, application of AD may not be possible or its performance may be degraded as it requires at least a small multiple of the storage necessary for the original program. AD remains the alternative of choice to analytical sensitivity analysis as long as execution time can be traded for coding and debugging time. This is especially true when prototyping a computer system or conducting a brief study. When adding subroutines to a computer program which already performs sensitivity analysis analytically or otherwise, the derivatives of the added subroutines can be obtained by means of AD and inserted in the original code.

Applying AD to any analysis problem definitely requires some development time. First, the original program must be written in standard ANSI FORTRAN. The usage of capabilities offered by extensions to ANSI FORTRAN may or may not be permitted. Also, the original program must be modified to meet the constraints of the specific AD tool used. For example, the tools considered in this paper do not permit usage of scratch files (although this is not a general restriction of AD). If those are used in the original program, the user must somehow address that problem. If one writes a new analysis program to be enhanced later by means of AD, one must keep those restrictions in mind.

This paper has focused only on the subset of AD techniques dealing with first order sensitivity analysis. Many extensions exist which have significant potential for engineering design. These include the use of higher-order derivatives and the generation of Taylor series coefficients, for example, to generate high-order approximations to functions and also to make use of second order optimization algorithms. Also, only one of the types of AD tools has been explored and others could certainly prove useful as well. For example, integral AD tools are attractive in that they recast engineering analysis programs in terms of higher level functions. In turn, those tools offer corresponding AD capabilities which may be worth exploring.

Acknowledgments

The authors are grateful to J. Horwedel of Oak Ridge National Laboratory who assisted them in the installation and use of GRESS. C. Bischof, G. Corliss, and A. Griewank at Argonne National Laboratory made the ADIFOR program available and are thanked for their untiring support.

References

1. Barthelemy, J.-F.M., and Riley, M.F., (1988), "Improved Multilevel Optimization Approach for the Design of Complex Engineering Systems," *AIAA J.*, Vol. 26, No. 3, Mar., pp. 353–360.
2. Barthelemy, J.-F.M., Wrenn, G.A., Dovi, A.R., and Coen, P.G. (1992), "Integrating Aerodynamics and Structures in the Minimum Weight Design of a Supersonic Transport Wing," AIAA Paper 92–2372, Presented at 33rd AIAA/ASME/AHS/ASC Structures, Structural Dynamics and Materials Conference," Apr. 13–15, Dallas, TX.
3. Bathe, K.-J., and Wilson, E.L. (1976), *Numerical Methods in Finite Element Analysis*, Prentice Hall.
4. Bischof, C., Carle, A., Corliss, G., Griewank, A., and Hovland, P., (1991), ADIFOR—Generating Derivative Codes for Fortran Programs, Argonne Preprint, MCS-P263–0991.
5. Callahan, D., Cooper, K., Hood, R.T., Kennedy, K., and Torcson, L.M., (1988) "ParaScope: A Parallel Programming Environment," *Int.J. Supercomputer Applications*," Vol. 2, No., 4, Dec.
6. Garcia, O., (1991), "A System for the Differentiation of FORTRAN Codes and an Application to Parameter Estimation in Forest Growth Models," in Griewank and Corliss (1991), pp. 273–285.
7. Giles, G.L., (1986) "Equivalent Plate Analysis of Aircraft Wing Box Structures with General Planform Geometry," *J. Aircraft*, Vol, 23, No. 11, Nov., pp. 859-864.
8. Giles, G.L., (1989) "Further Generalization of an Equivalent Plate Representation for Aircraft Structural Analysis," *J. Aircraft*, Vol. 26, No. 1, Jan. , pp. 67–74.
9. Griewank, A. (1991a), "The Chain Rule Revisited in Scientific Computing," *SIAM News*, May, pp. 20.

10. Griewank, A. (1991b), "The Chain Rule Revisited in Scientific Computing, Part II" *SIAM News*, Jul., pp. 8.
11. Griewank, A. (1991c), "Automatic Evaluation of First and Higher Derivative Vectors," in Proc. of *1990 Wurzburg Conference on Bifurcation and Chaos: Analysis, Algorithms, Applications*, Seidel, Schneider, Dupper and Troger, Eds, Birkauser, Basel, pp. 124–137
12. Griewank, A. (1992), "Achieving Logarithmic Growth of Temporal and Spatial Complexity in Reverse Automatic Differentiation," *Optimization Methods and Software*, Vol 1, pp. 35–54.
13. Griewank, A., and Corliss, G.F., Eds., (1991) *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, SIAM, Philadelphia.
14. Griewank, A., and Reese, S. (1991), "On the Calculation of Jacobian Matrices by the Markowitz Rule," in Griewank and Corliss (1991), pp. 126–135.
15. Haftka, R.T., and Gurdal, Z., (1992) *Elements of Structural Optimization*, 3rd Ed., Kluwer Academic Publishers Group, the Netherlands.
16. Haug. E.J., and Arora, J.S., (1979), *Applied Optimal Design*, J. Wiley and Son.
17. Horwedel, J.E., (1991a), GRESS Version 2.0 User's Manual, Report ORNL/TM-11951, Nov.
18. Horwedel, J.E., (1991b), "GRESS, A Preprocessor for Sensitivity Analysis of FORTRAN programs," in Griewank and Corliss (1991), pp. 243–250.
19. Juedes, D.W., (1991) "A Taxonomy of Automatic Differentiation Tools," in Griewank and Corliss (1991), pp. 315–329.
20. Iri, M. (1984), "Simultaneous Computation of Functions, Partial Derivatives and Estimates of Rounding Errors — Complexity and Practicality —," *Japan J. Appl. Math.*, Vol. 1, pp. 223–252.
21. Iri, M. (1988), "Automatic Computation of Partial Derivatives and Rounding Error Estimates with Applications to Large-Scale Systems of Nonlinear Equations," *J. Comp Appl. Math.*, Vol 24, pp. 365–392.
22. Iri, M., (1991), "History of Automatic Differentiation and Rounding Error Estimation", in Griewank and Corliss (1991), pp. 3–16.
23. Talagrand, O., (1991), "The Use of Adjoint Equations in Numerical Modeling of the Atmospheric Circulation," in Griewank and Corliss, (1991), pp. 169–180.
24. Thacker, W.M., (1991), "Automatic Differentiation form an Oceanographer's Perspective," in Griewank and Corliss (1991), pp. 191–201.
25. Rall, L.B., (1981), *Automatic Differentiation: Techniques and Applications*, Lecture Notes in Computer Science No. 120, Springer Verlag, New York.
26. Worley, B.A. (1991), "Experience with the Forward and Reverse Mode of GRESS in Contaminant Transport Modeling and Other Applications," in Griewank and Corliss (1991), pp. 307–314.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 1992	Technical Memorandum

4. TITLE AND SUBTITLE
Automatic Differentiation as a Tool in Engineering Design

5. FUNDING NUMBERS
WU 505-63-50-06

6. AUTHOR(S)
Jean-Francois Barthelemy
Laura E. Hall

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23665-5225

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA TM-107661

11. SUPPLEMENTARY NOTES
Presented at Fourth AIAA/Air Force/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization, September 21-23, 1992 in Cleveland, Ohio. Laura Hall-Unisys Corporation, Inc., Hampton, VA
Jean-Francois Barthelemy-Langley Research Center, Hampton, VA

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category 61

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)
Automatic Differentiation (AD) is a tool that systematically implements the chain rule of differentiation to obtain the derivatives of functions calculated by computer programs. In this paper, it is assessed as a tool for engineering design. The paper discusses the forward and reverse modes of AD, their computing requirements as well as approaches to implementing AD. It continues with application of two different tools to two medium-size structural analysis problems to generate sensitivity information typically necessary in an optimization or design situation. The paper concludes with the observation that AD is to be preferred to finite differencing in most cases, as long as sufficient computer storage is available; in some instances, AD may be the alternative to consider in lieu of analytical sensitivity analysis.

14. SUBJECT TERMS
Sensitivity Analysis, Automatic Differentiation
Structural Optimization

15. NUMBER OF PAGES
10

16. PRICE CODE
A02

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	