

Extended Abstract for Semiautomatic Differentiation for Efficient Gradient Computations

David M. Gay
Sandia National Labs *

March 25, 2004

This extended abstract is for a talk proposed for presentation at the *Fourth International Conference on Automatic Differentiation*, to be held 19–23 July 2004 at The University of Chicago.

Many large-scale computations concern partial differential equations (PDEs) based on physical systems and thus involve discretizations that approximate physical objects on meshes. Such discretizations generally yield systems of non-linear equations whose residuals involve the elements of a mesh. As a PDE model matures, interest often grows in optimizing some aspects of the model, i.e., of solving optimization problems with PDE constraints. Like the constraint residuals, the objectives are generally sums of contributions from functions of the mesh elements.

For solving both discretized PDEs themselves and optimization problems involving them, partial derivatives (or approximations to them) are required. Conventionally, these partials are often approximated by finite differences, but finite differences have several drawbacks. Finding suitable step sizes that balance truncation and round-off error can be tricky, and overall error in a finite-difference approximation can contribute to computational difficulties. Moreover, when partials with respect to many variables are required, finite differences can be slow. Automatic differentiation (AD) provides a more accurate and often faster alternative to finite differences.

As Griewank [8] showed in a survey that appeared in 1989, AD has been reinvented many times. His more recent book [9] tells much more about AD than we will discuss here. Of primary interest here are first derivatives, which may be computed either by *forward AD*, in which one recurs the desired partials while carrying out each operation in the expression of interest, or by *backward AD*, in which one first evaluates an expression, then visits its operations again in reverse order to recur partials (so-called *adjoints*) of the final expression result with respect to the result of each operation.

*Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

Forward AD works well when only a few independent variables are involved, but its complexity can grow with the number of independent variables. In particular, when there is only one independent variable, forward AD can efficiently and conveniently compute derivatives of high order. Nonlinear equations can be solved by matrix-free Newton-Krylov methods, which simply use Jacobian-vector products. Such computations effectively involve just one independent variable, and are well handled by forward AD. (For example, TFAD [4] works well in some applications at Sandia [3].)

When there are many independent variables (as is often the case in optimization problems), backward AD is attractive for computing gradients. It delivers function and gradient in time proportional to that required for a function evaluation alone. Unfortunately, when used straightforwardly, backward AD may require memory proportional to the number of operations in the function evaluation, which can be prohibitive in large-scale computations. (Straightforward use of AD is facilitated by many tools, such as those listed in the *autodiff* web site [2].)

As mentioned above, many large-scale computations involve meshes. For optimization problems whose objectives and constraints involve sums of functions of mesh elements, one can use backward AD to compute (separately) the contributions of each mesh element to the objective and constraint gradients and manually assemble these pieces into the overall gradients, an approach sketched in [1]. Preliminary investigations suggest that this approach should work well in some problems of interest at Sandia, such as mesh optimization (moving interior mesh points to improve the quality of a given mesh) and PDE-constrained optimization. Only a few kinds of mesh elements appear in such problems, making it feasible to treat each separately, either by operator overloading or by source transformation and optimization of routines that compute the element functions. This results in what might be called semiautomatic differentiation.

Our work on this approach is ongoing. One preliminary result is that for an objective relevant to mesh optimization [5], a sum of terms of the form

$$\frac{3 \det(AW^{-1})^{2/3}}{\|AW^{-1}\|_F^2} \tag{1}$$

where the 3×3 matrix A is determined by the vertices of a tetrahedral mesh element and W is a fixed weighting matrix, on some systems AD computations optimized by the *nlc* program (see [6] and [7]) and an optimizing C compiler (gcc) can deliver function and gradient in less than twice the time for the function alone, apparently because the divisions and exponentiations in (1) mask the AD overhead.

I hope to be able to present more computational results by the time I would talk on this subject in the summer of 2004. I also hope to say more about optimizing AD computations.

References

- [1] Jason Abate, Steve Benson, Lisa Grignon, Paul D. Hovland, Lois C. McInnes, and Boyana Norris. Integrating AD with object-oriented toolkits for high-performance scientific computing. In George Corliss, Christèle Faure, Andreas Griewank, Laurent Hascoët, and Uwe Naumann, editors, *Automatic Differentiation of Algorithms: From Simulation to Optimization*, Computer and Information Science, chapter 20, pages 173–178. Springer, New York, NY, 2001.
- [2] Autodiff web site. <http://www.autodiff.org/Tools/index.php>.
- [3] Roscoe A. Bartlett. Private communication, 2004.
- [4] Nicolas Di Cesare. Fad: Automatic differentiation library in forward mode using expression template, 1999. <http://acm.emath.fr/dicesare/cpp.php3>.
- [5] Lori Freitag, Patrick Knupp, Todd Munson, and Suzanne Shontz. A comparison of optimization software for mesh shape-quality improvement problems, 2002. <http://www.imr.sandia.gov/papers/imr11/freitag.pdf>.
- [6] D. M. Gay. Automatic differentiation of nonlinear AMPL models. In A. Griewank and G. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 61–73. SIAM, 1991.
- [7] David M. Gay. Hooking your solver to AMPL. Numerical Analysis Manuscript No. 93-10, AT&T Bell Laboratories, Murray Hill, NJ, 1993, revised 1997. <http://www.ampl.com/REFS/hooks2.ps.gz>.
- [8] A. Griewank. On automatic differentiation. In M. Iri and K. Tanabe, editors, *Mathematical Programming*, pages 83–107. Kluwer Academic Publishers, 1989.
- [9] A. Griewank. *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation*. SIAM, 2000.