Sensitivity analysis and design optimization through automatic differentiation

Paul D. Hovland, Boyana Norris, Michelle Mills Strout
Mathematics and Computer Science Division, Argonne National Laboratory, 9700 S Cass Ave, Argonne IL 60439, USA

Sanjukta Bhowmick
Columbia University, New York NY, USA and Argonne National Laboratory

Jean Utke
University of Chicago, Chicago IL, USA

E-mail: hovland@mcs.anl.gov, norris@mcs.anl.gov, mstrout@mcs.anl.gov, bhowmick@mcs.anl.gov, utke@mcs.anl.gov

Abstract.
Automatic differentiation is a technique for transforming a program or subprogram that computes a function, including arbitrarily complex simulation codes, into one that computes the derivatives of that function. We describe the implementation and application of automatic differentiation tools. We highlight recent advances in the combinatorial algorithms and compiler technology that underlie successful implementation of automatic differentiation tools. We discuss applications of automatic differentiation in design optimization and sensitivity analysis. We also describe ongoing research in the design of language-independent source transformation infrastructures for automatic differentiation algorithms.

1. Introduction
Automatic differentiation (AD) is a family of techniques for computing the derivatives of a function defined by a computer program. The basis for AD is the assumption that the computation of a vector function \( \mathbf{y} = \mathbf{f}(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^m \) is accomplished by a sequence of \( p \) elemental operations \( \mathbf{v}_i = \phi_i(\ldots, \mathbf{v}_j, \ldots), i = 1, \ldots, p \) as found in a computer program implementing an evaluation procedure for \( \mathbf{f} \). Each of the \( \phi \) is differentiable at least in open subdomains. The derivatives of these elemental operations are combined according to the chain rule of differential calculus. The associativity of the chain rule leads to the two major modes of computing derivatives with AD, the so-called forward mode and reverse (or adjoint) mode.

The forward mode multiplies derivatives starting with the independent variables and proceeding toward the dependent variables. Because the order of the derivative computation parallels that of the function computation, intermediate function values can be used as they are computed, and the control flow of the derivative computation follows that of the original program. The reverse mode multiplies derivatives starting with the dependent variables and proceeding toward the independent variables. In this case, the control flow reverses that of the
original program, and therefore intermediate function values and control flow decisions may need to be recorded. This increases the storage requirements of the reverse mode.

In Section 2, we highlight recent advances in the combinatorial algorithms and compiler technology that underlie successful implementation of automatic differentiation tools. We also describe ongoing research in the design of language-independent source transformation infrastructures for automatic differentiation algorithms. In Section 3, we discuss applications of automatic differentiation in design optimization and sensitivity analysis. We conclude with a summary and a discussion of future research opportunities.

2. Foundations
The implementation of robust and effective automatic differentiation tools requires advances in compiler technology, graph algorithms, and automatic differentiation theory. A robust compiler infrastructure is required to support the source-to-source transformation process. Compiler analysis is needed to reduce the cost of the derivative code. Graph algorithms play a role both in derivative accumulation strategies and in compression techniques for computing sparse Jacobians and Hessians.

OpenAD is an automatic differentiation tool built from components; this facilitates the implementation of code analysis and transformation in a language-independent fashion, as illustrated in Figure 1. OpenAD uses code analysis results implemented in the OpenAnalysis component. The interface to the language-independent transformation engine is an xml abstract interface format, called XAIF that is specified through an xml schema. The transformation algorithms implemented in the xaifBooster component enable efficient derivative computations. The Fortran-specific front and back end to this transformation engine is the Open64-based OpenAD Fortran toolkit, OpenADFortTk. OpenAD is a relatively new tool, but already provides state-of-the-art algorithms for computing derivatives in the forward and reverse mode, built on a robust compiler architecture.

![Figure 1. Schematic of the OpenAD tool for Fortran 90 and the ADIC tool for C/C++.](image)

Automatic differentiation tools rely on compiler analyses, including traditional analyses such as alias analysis and side effect analysis, as well as domain-specific analyses such as activity analysis and linearity analysis, to improve the efficiency of the generated derivative code. Accurate analysis can improve performance by a factor of 2 or more. The OpenAnalysis toolkit, see [1], separates program analysis from language-specific or front-end specific intermediate
representations. This separation enables a single implementation of domain-specific analyses such as activity analysis, to-be-recorded analysis, and linearity analysis in OpenAD. Standard analyses implemented within OpenAnalysis such as CFG construction, call graph construction, alias analysis, reaching definitions, ud- and du-chains, and side-effect analysis are also available. OpenAnalysis implements an important AD-specific analysis called activity analysis, based on the formulation in [2]. Activity analysis is implemented in a context-insensitive, flow-sensitive interprocedural fashion.

The associativity of the chain rule leads to exponentially many ways to combine derivatives. Choosing an order can be interpreted as choosing an elimination order on the directed acyclic graph (DAG) representing the computation. The vertices in this computational graph represent variables or intermediate expressions and the edge weights correspond to partial derivatives. Choosing an optimal elimination order is conjectured to be NP-hard, except in certain special cases, so heuristics are used. Typically, heuristics are designed to minimize the number of floating point operations. We have developed new heuristics that seek instead to increase reuse of data in the cache and reduce register pressure by choosing elimination sequences that favor locality. In addition, we have developed algorithms for assembling computational graphs for basic blocks from the computational graphs for individual statements. This procedure (known as flattening) uses alias and def/use chain information provided by OpenAnalysis.

Another important strategy for reducing the cost of computing derivatives is to exploit sparsity. Graph coloring is used to reduce the cost of computing sparse Jacobians and Hessians via compression techniques [3]. We have developed a new backtracking heuristic for graph coloring. The number of colors used is to a large extent determined by the order in which the vertices are colored. We present a backtracking correction algorithm that can be used in conjunction with a top level heuristic. The backtracking method attempts to dynamically rearrange the coloring assignment thus further decreasing the number of colors.

The backtracking algorithm is as follows. The user defines a threshold roughly corresponding to a lower bound on the expected number of colors. The set of colors within the threshold gives the total number of colors allowed until now. The algorithm is invoked whenever the threshold is exceeded. The backtracking algorithm tries to determine whether there is an alternate assignment of colors, that would keep the number of colors within the given threshold. If such an assignment is found, then we have a coloring within the limits of the threshold. If no such arrangement can be obtained, then the original assignment is implemented and the threshold is increased by one.

Given below are results for distance-1 colorings of graphs arising in molecular dynamics. We experimented with the following ordering heuristics; Natural(NT), Largest First(LF), Smallest Last(SL), Incidence Degree(ID), Saturation Degree (SD) [3] and Depth First Search(DF). For each heuristic we conducted three sets of experiments using: i) only the heuristic, ii) the heuristic with backtracking(B) and iii) the heuristic with another correction algorithm, Culberson’s Greedy Iterative (I) method [4]. The results given in Figure 2 for six matrices show that use of the backtracking algorithm almost always gives the smallest number of colors.

3. Applications
The ADIC, ADIFOR, and OpenAD tools have been applied to a broad range of applications, including breast cancer modeling, atmospheric chemistry, climate modeling, weather modeling, computational fluid dynamics, semiconductor device modeling, the network-enabled optimization server (NEOS), power network modeling, water reservoir simulation, chemical kinetics, and groundwater modeling. The widespread use of the tools helps ensure their robustness and guides future work on theory and implementation.

Automatic differentiation is used to compute gradients and Hessians for the parallel solution of optimization problems in TAO [5]. TAO focuses on scalable optimization software, including
nonlinear least squares, unconstrained minimization, bound constrained optimization, and general nonlinear optimization. The TAO optimization algorithms use high-level abstractions for matrices and vectors and place strong emphasis on the reuse of external tools where appropriate, including support for using the linear algebra components provided by PETSc and related tools.

Many of the algorithms employed by TAO require first and sometimes second derivatives. For example, unconstrained minimization solvers that require the gradient, \( f'(u) \), of a function, \( f(u) \), include a limited-memory variable metric method and a conjugate gradient method, while solvers that require both the gradient, \( f'(u) \), and Hessian, \( f''(u) \), (or Hessian-vector products) include line search and trust region variants of Newton methods. We have used automatic differentiation to compute these derivatives. Figure 3 shows a sequence of solutions and their deviation from the optimal solution for a bound constrained minimization problem.

The OpenAD tool for Fortran 90 is being applied to the MIT General Circulation Model. Figure 4 displays a map of sensitivities of zonal volume transport through the Drake Passage to changes in bottom topography everywhere in a barotropic ocean model. The model is based on the shallow water model used by [6], and extended to a global configuration at 2x2 degree horizontal resolution with realistic topography. Enhanced sensitivities are manifest both locally and remotely, for example, over the Kerguelen Plateau, over the South Pacific Ridge, and in the Indonesian Throughflow. Sensitivities are mediated through the flow field represented by the model dynamics. This sensitivity map was achieved through one single adjoint model integration. While OpenAD has had minimal tuning for the efficiency of generated code, consisting of a manually specified 2-level checkpointing strategy, an improved storage strategy, and activity
analysis via OpenAnalysis, the preliminary results in Figure 4 indicate an overhead factor of only 10 in the adjoint computation over one forward function computation, compared with the 14401 computations performed by finite difference methods, or 22 minutes versus 23 days.

4. Conclusions and Future Work

Automatic differentiation is a powerful technique for computing the derivatives required in sensitivity analysis and design optimization, among other applications. Development of effective automatic differentiation tools requires advances in compiler infrastructure and graph algorithms. Motivated by the needs of applications such as uncertainty quantification in climate models, future research will investigate efficient algorithms for the computation of Hessians and Hessian-vector products. Such algorithms should exploit the symmetry of the Hessian (and, perhaps, the symmetry in the corresponding computational graph). Future work also includes development of new locality-aware accumulation algorithms, enhancements to our compiler infrastructure, and development of a robust, reverse mode tool for C++.

Acknowledgments

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. We thank Patrick Heimbach of MIT for generating the map of sensitivities.

References

[1] OpenAnalysis. http://www-unix.mcs.anl.gov/OpenAnalysisWiki/moin.cgi.
[2] U. Naumann. Reducing the memory requirement in reverse mode automatic differentiation by solving TBR flow equations. In P. M. A. Sloot, C. J. K. Tan, J. J. Dongarra, and A. G. Hoekstra, editors, Proceedings of ICCS 2002, volume 2330 of Lecture Notes in Computer Science, pages 1039–1048, Berlin, 2002. Springer.
[3] A. Gebremdhin, F. Manne, and A. Pothen. What color is your Jacobian? Graph coloring for computing derivatives. accepted by SIAM Review, Dec 2004.
[4] Joseph C. Culberson. Iterated Greedy Graph Coloring and the Difficulty Landscape. Technical Report, 1992.
[5] Steve Benson, Lois Curfman McInnes, and Jorge Moré. GPCG: A case study in the performance and scalability of optimization algorithms. Technical Report ANL/MCS-P768-0799, Mathematics and Computer Science Division, Argonne National Laboratory, 1999.
[6] M. Losch and C. Wunsch. Bottom topography as a control variable in an ocean model. J. Atmos. Ocean. Technol., 20:1685–1696, 2003.