PAMM · Proc. Appl. Math. Mech. **2**, 45–49 (2003) / **DOI** 10.1002/pamm.200310012

Andreas Griewank and Andrea Walther

Introduction to Automatic Differentiation

Abstract: Automatic, or algorithmic, differentiation (AD) is a chain rule-based technique for evaluating derivatives of functions given as computer programs for their elimination. We review the main characteristics and application of AD and illustrate the methodology on a simple example.

1. Background and Motivation

The basic techniques of automatic, or algorithmic differentiation, namely the forward mode and the reverse mode have been known for some 50 and 30 years, respectively [11]. However, reliable and convenient software implementations have only been available for the last decade and many improvements remain to be incorporated. At the same time, the need for efficient and accurate derivative evaluations has become much more urgent as computer models grow ever more complex and are increasingly used for the purpose of optimization rather than just simulation. Without exact sensitivity information it is almost impossible to systematically adjust model parameters and design or control variables in order to fit data or optimize objectives. In rare cases a brute force approach based on simply sampling objective function values alone may lead to satisfactory and possibly even globally optimal values, but if there are more than a handful of variables the price to pay will usually be many hundreds, thousands or even millions of function evaluations. In contrast, from reasonable starting points calculus-based optimization methods typically reach local minima or roots of equations in much fewer than hundred iterations, irrespective of the number of variables.

It is a still widely held belief that, for complex computer models, the evaluation of derivatives between input and output variables is either principally impossible or prohibitively expensive. Therefore, many classical numerical methods for the treatment of nonlinear problems in scientific computing approximate derivatives by difference quotients or secant updating. Also, in some cases the differentiation capabilities of computer algebra packages followed by optimized code generation may be successful. However, this fully symbolic approach often fails on more sizable problems, either due to the sheer complexity of the formulas generated or due to the presence of program branches. Generally speaking, the difference between an *explicit* or *symbolic* representation of a function and a *numerical* one in form of an evaluation procedure, which we assume here, is not nearly as clear cut and severe as one tends to think.

2. Main products and characteristics of AD

In a symbolic programming environment derivatives are usually thought of as derivatives of specific output variables with respect to one or several input variables. Here input and output may of course refer to some subroutine so that derivatives may enter into subsequent calculations either as algebraic expressions or evaluated as numbers. The tools and techniques of AD lend themselves to more general numerical tasks and can yield for example the goodies in the following list.

- Quantitative Dependence Information (local):
  - Weighted and directed partial derivatives
  - Lipschitz constants, interval enclosures
  - Error and condition number estimates ...
  - Eigenvalues, Newton-steps ...
- Qualitative Dependence Information (regional):
  - Sparsity-structures, degrees of polynomials
  - Ranks, eigenvalue-multiplicities ...

Here "local" means in the immediate vicinity of an evaluation point, and "regional" refers to the region in the domain of independent variables in which the same program branch applies. For certain branches like min, max and other conditional assignments conservative envelope structures can be determined. In contrast to difference quotient approximations, the actual numerical derivative values computed by automatic differentiation are not affected by truncation errors and thus obtained with working accuracy. In other words, nowhere are tangent slopes replaced by secant slopes as automatic differentiation is based on the chain-rule. In contrast to fully symbolic differentiation on the other hand, this fundamental rule of calculus is applied in such a way that the costs

- Total operations count
- Maximal memory requirement
- Total memory traffic

can always be bounded a priori relative to the same costs for the function itself. For first derivatives these relative bounds consist of platform dependent constants multiplied by the number of independent or dependent variables. On particular problems the actual costs can often be further reduced by the exploitation of sparsity and other structural properties. However, as of now the user must at least assist the AD tool in detecting such structure, as for example partial separability. In short, the straightforward application of AD entails quite predictable costs so that the user is guarded from unpleasant but also from pleasant surprises.

3. Main application areas of AD

The main present usages of AD are summarized in the following table

- "Pure" Differentiation: ○ Heterogeneous Taylor expansions ○ Surface Curvatures, Image Aberrations
- Nonlinear Optimization: ○ Unconstrained/constrained, ○ Mono/multicriteria but local
- Analysis and Solution: ○ ODEs and DAEs ○ Discretised PDEs

By heterogeneous Taylor expansions we mean the approximation of multivariate nonlinear functions by polynomials that have varying degrees in different variables or directions. Such approximations are used for example in numerical bifurcation [10]. Curvatures and related geometric properties are critical for the modeling of gear tooth flanks [15], optical lenses, and the surfaces of other three-dimensional solids. Whether parameterized explicitly or implicitly their curvature depends already on second derivatives so that the sensitivities of related performance indices with respect to design parameters are in already third derivatives. Aberrations are second and higher derivatives of mappings between objects and their images, for example in an optical instrument or a particle accelerator. Making the mapping as linear as possible requires minimizing weighted sums of the aberrations at a certain point or within a neighborhood [2].

Nonlinear optimization is probably still the prime application of differentiation tools. The first derivatives of objective functions and constraints are constitutive components of the KKT optimality conditions, and thus evaluating their linear combination as gradient of the Lagrangian is virtually indispensable for achieving high solution accuracy. To also achieve rapid convergence, the evaluation of second derivatives in form of the Hessian of the Lagrangian function is also frequently recommended, though not absolutely necessary. AD has been implemented in modeling systems like GAMS [4], AMPL [9] and the netserver NEOS [7], which include convenient interfaces to various optimization algorithms. In the reverse mode, the gradient of the Lagrangian has essentially the same temporal complexity as the combination of objective and constraint functions. However, Hessian and Jacobian may be orders of magnitude more expensive, unless partial separability or other structural properties of the problem are appropriately exploited. Despite their local nature derivative values may also be employed within global optimization strategies and they are certainly extremely useful for tracing out Pareto-optimality curves or (hyper)faces.

One of the first uses of the forward mode of AD was the classical Taylor series methods for the numerical integration of ODEs [17, 6]. While this explicit numerical integration method is not suitable for stiff ODEs, the need for evaluating the Jacobian of the right hand side, which arises in implicit numerical integrators, was one of the origins of the reverse mode [14]. The latter is also very useful for A- or L-stable implicit Taylor series methods for ODEs as well as numerical methods for Differential Algebraic Equations with unknown or variable index [5, 8]. Many traditional PDE models had rather regular domains and included no or rather simple nonlinearities. Hence, the evaluation of their Jacobians and other derivatives could be easily performed by hand. However, more realistic coefficient functions and adaptive gridding on complex domains lead to a situation where coding the Jacobian of the discretized PDEs and its boundary conditions may become very tedious and error-prone indeed. Then AD becomes a natural alternative, and there has been an effort to integrate AD into PetSc [1] and other programming environments for PDE related modeling. A particular difficulty arises in "PDE-constrained" optimization, where the decision variables are linked to the objective function via a state equation, which can usually only be solved iteratively. Then computing the total, or reduced derivatives of the objective with respect to the decision variables requires the solution of related sensitivity equations by a corresponding iterative procedure. The latter may be obtainable implicitly by applying AD in a black box fashion. However, to reduce the costs and to improve the accuracy of the derivatives computed some user intervention is still required [12].

4. Key theoretical results of AD

Rather than being just an exercise in providing efficient and reliable software AD has evolved over the last 15 years into a subdiscipline of scientific computing. The key observations that have been established so far can be summarized as follows.

- Gradients are cheap $\sim$ function costs
- Nondifferentiability only on meager set
- Costs grow only quadratically in degree
- Optimal Jacobi-Accumulation is (NP-)hard

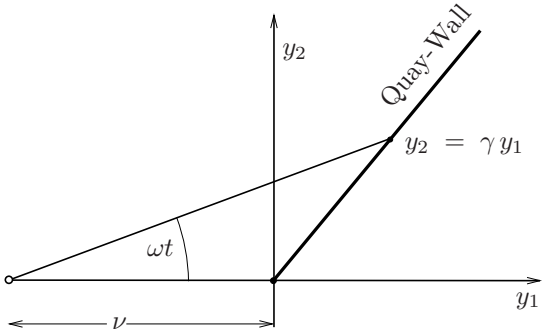
More specifically, it has been shown that the operations count for the evaluation of a gradient in the reverse mode is bounded by five time that of the underlying (scalar) function evaluation, irrespective of the number of variables. Unfortunately, the memory requirement of these basic reverse mode implementation is also proportional to the operations count, which can be a problem in certain large-scale applications. Fortunately, much more attractive trade-offs between temporal and spatial complexity can be achieved by selective recomputation of intermediate results [11]. On explicitly time-dependent problems these strategies take the form of checkpointing, which can be implemented on serial or parallel processors [16]. When higher derivatives are evaluated in a fully symbolic fashion their complexity typically grows exponentially with their degree. By applying the chain rule to floating point numbers rather than algebraic expressions, AD yields higher derivatives at a cost that grows only quadratically with the degree. Using FFT or other fast polynomial convolution methods one may reduce the cost further such that it grows essentially only linearly with respect to the degree.

For good reason there is considerable concern regarding the differentiation of programs involving non smooth elementaries like the absolute value function or outright branches in the control flow. Under realistic assumptions one can verify the intuitive notion that the output of a computer routine is repeatedly differentiable at "almost" all input values. Moreover, right on kinks or cracks of the function AD could in principle still provide one-sided derivatives and generalized gradients [11]. However, especially in a finite precision environment these constructions are a little bit dubious, and as yet most AD tools have not implemented them carefully. Even if these generalized derivatives were obtained reliably, their intelligent usage within numerical methods would remain a serious challenge. In the following section we illustrate the forward and reverse mode of AD on an example with four independent and two dependent variables. While either can be used to evaluate the whole Jacobian it can be shown [refbook] that a more general "cross-country" application of the chain rule yields the same 2×3 matrix at a lower cost in terms of arithmetic operations. In general, the search for the optimal Jacobian computation scheme leads to a rather difficult combinatorial optimization problems on directed graphs [13].

5. Forward and Reverse Mode by Example

To illustrate the basic modes of automatic differentiation we consider the following lighthouse example from [11]. The lighthouse on the left emanates a light beam that hits the quay-wall at a point whose coordinates are given by

$$y_1 = \frac{\nu * \tan(\omega * t)}{\gamma - \tan(\omega * t)} \quad \text{and} \quad y_2 = \frac{\gamma * \nu * \tan(\omega * t)}{\gamma - \tan(\omega * t)} \,. \qquad (1)$$



These two symbolic expressions might be evaluated by the procedure given on the left hand side of the box on top of the following page. First, the distance $x_1 = \nu$, the slope $x_2 = \gamma$, the angular velocity $x_3 = \omega$, and the time $x_4 = t$ are assigned as independent variables to the internal quantities v_{-3}, v_{-2}, v_{-1} and v_0. Subsequently, six quantities v_i for $i = 1 \ldots 6$ are calculated using arithmetic operations and elementary functions, and finally the last two of them are assigned to the dependent variables y_1 and y_2. In principle any functions evaluation on a digital computer can be decomposed into such a straight line code. On the right hand side of the box an additional statement computes for each variables v_i its derivative $\dot{v}_i$ in the direction $\dot{x} \in \mathbb{R}^4$, which may be initialized arbitrarily. For example setting $\dot{x}$ to the 4-th Cartesian basis vector yields the partial derivatives of all v_i with respect to time and thus in particular the velocity vector $(\dot{y}_1, \dot{y}_2)^T = F'(x)\dot{x}$ of the point of light (y_1, y_2). As one can see the derivative statements can be generated from the original statements by the mechanical application of standard differentiation rules and the computational complexity of the right half is about the same as that of the left side provided the results of the latter are already available. Since the derivatives are propagated simultaneously with the intermediate values themselves this method of computing derivatives is called the forward mode.

v_{-3}	=	$x_1 = \nu;$	$\dot{v}_{-3}$	$\equiv$	$\dot{x}_1 = 0$
v_{-2}	=	$x_2 = \gamma;$	$\dot{v}_{-2}$	$\equiv$	$\dot{x}_2 = 0$
v_{-1}	=	$x_3 = \omega;$	$\dot{v}_{-1}$	$\equiv$	$\dot{x}_3 = 0$
v_0	=	$x_4 = t;$	$\dot{v}_0$	$\equiv$	$\dot{x}_4 = 1$
v_1	=	$v_{-1} * v_0$;	$\dot{v}_1$	=	$\dot{v}_{-1} * v_0 + v_{-1} * \dot{v}_0$
v_2	=	$\tan(v_1);$	$\dot{v}_2$	=	$\dot{v}_1 / \cos(v_1)^2$
v_3	=	$v_{-2} - v_2;$	$\dot{v}_3$	=	$\dot{v}_{-2} - \dot{v}_2$
v_4	=	$v_{-3} * v_2;$	$\dot{v}_4$	=	$\dot{v}_{-3} * v_2 + v_{-3} * \dot{v}_2$
v_5	=	$v_4 / v_3;$	$\dot{v}_5$	=	$(\dot{v}_4 - \dot{v}_3 * v_5) * (1/v_3)$
v_6	=	$v_5 * v_{-2};$	$\dot{v}_6$	=	$\dot{v}_5 * v_{-2} + v_5 * \dot{v}_{-2}$
y_1	=	$v_5;$	$\dot{y}_1$	=	$\dot{v}_5$
y_2	=	$v_6;$	$\dot{y}_2$	=	$\dot{v}_6$

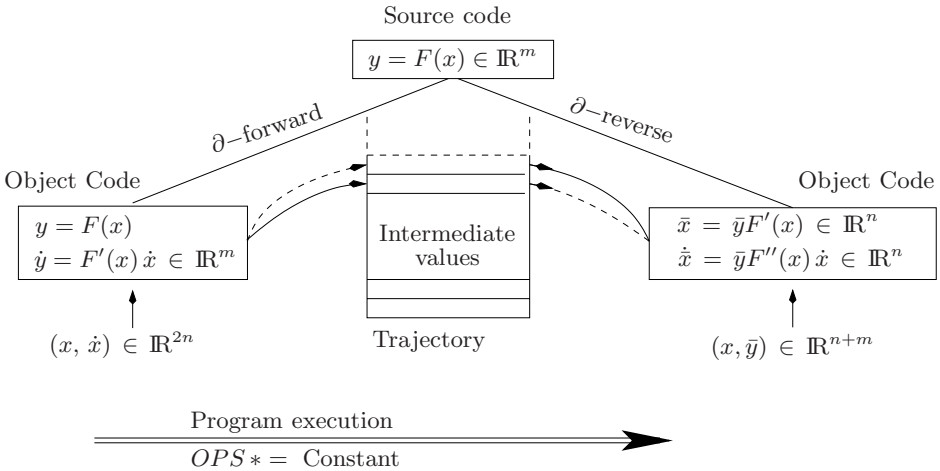
The full Jacobian of y with respect to x can be obtained column by column by executing the above procedure four times with $\dot{x}$ ranging over the Cartesian basis vectors in $\mathbb{R}^4$. In general computing the Jacobian

of a vector function of n variables in this way is about $2\,n$ times as expensive as evaluating the function by itself. Hence the cost is similar to that of approximating the Jacobian by central differences. When m the number of dependent variables is significantly smaller than n the number of independent variables then the reverse mode has a lower operations count. On the lighthouse example it takes the shown in the box below.

Forward Sweep	Reverse sweep +=	Reverse sweep =
$v_{-3} = x_1 = \nu$	$\bar{v}_7 = \bar{y}_2$	$\bar{v}_7 = \bar{y}_2$
$v_{-2} = x_2 = \gamma$	$\bar{v}_6 = \bar{y}_1$	$\bar{v}_6 = \bar{y}_1$
$v_{-1} = x_3 = \omega$	$\bar{v}_5 \mathrel{+}= \bar{v}_7 * v_{-2}$	$\bar{v}_5 = \bar{v}_7 * \bar{v}_{-2} + \bar{v}_6$
$v_0 = x_4 = t$	$\bar{v}_{-2} \mathrel{+}= \bar{v}_7 * v_5$	$\bar{v}_4 = \bar{v}_5 / v_3$
$v_1 = v_{-1} * v_0$	$\bar{v}_5 \mathrel{+}= \bar{v}_6$	$\bar{v}_3 = -\bar{v}_5 * v_5 / v_3$
$v_2 = \tan(v_1)$	$\bar{v}_4 \mathrel{+}= \bar{v}_5 / v_3$	$\bar{v}_2 = \bar{v}_4 * v_{-3} - \bar{v}_3$
$v_3 = v_{-2} - v_2$	$\bar{v}_3 \mathrel{-}= \bar{v}_5 * v_5 / v_3$	$\bar{v}_1 = \bar{v}_2 / \cos^2(v_1)$
$v_4 = v_{-3} * v_2$	$\bar{v}_{-2} \mathrel{+}= \bar{v}_3$	$\bar{v}_0 = \bar{v}_1 * v_{-1}$
$v_5 = v_4 / v_3$	$\bar{v}_2 \mathrel{-}= \bar{v}_3$	$\bar{v}_{-1} = \bar{v}_1 * v_0$
$v_6 = v_5$	$\bar{v}_1 \mathrel{+}= \bar{v}_2 / \cos^2(v_1)$	$\bar{v}_{-2} = \bar{v}_3 + \bar{v}_7 * v_5$
$v_7 = v_5 * v_{-2}$	$\bar{v}_{-1} \mathrel{+}= \bar{v}_1 * v_0$	$\bar{v}_{-3} = \bar{v}_4 * v_2$
$y_1 = v_6$	$\bar{v}_0 \mathrel{+}= \bar{v}_1 * v_{-1}$	$\bar{x}_4 = \bar{v}_0 \;;\; \bar{x}_3 = \bar{v}_{-1}$
$y_2 = v_7$	$\bar{x}_4 = \bar{v}_0 \;;\; \bar{x}_3 = \bar{v}_{-1}$	$\bar{x}_2 = \bar{v}_{-2} \;;\; \bar{x}_1 = \bar{v}_{-3}$
	$\bar{x}_2 = \bar{v}_{-2} \;;\; \bar{x}_1 = \bar{v}_{-3}$	

There are two variants of the reverse mode in the above table. The first consists of the left and middle column and is "incremental" because, starting from an initial zero, the adjoint quantities $\bar{v}_i$ are incremented several times. The final values $\bar{v}_i$ represent the partial derivative of the inner product $\bar{y}\ y$ with respect to variations in v_i. In particular one obtains at the end the vector-matrix product $\bar{x} = \bar{y}\,F'(x) \in \mathbb{R}^n$. The weighting $\bar{y}$ can be any row vector in $\mathbb{R}^m$ and by letting it range over all m Cartesian basis vectors one obtains the Jacobian $F'(x)$ row by row rather than column-wise as in the forward mode. The second, "nonincremental" variant of the reverse mode yields the same numerical results at a slightly reduced operations count as all contributions to the $\bar{v}_i$ are collected at once. Either way the extra "adjoint" statements in the center and right column can also be derived in a completely mechanical fashion from the statements in the original code. They involve roughly twice as many operations so that a gradient in particular is obtained for $\bar{y} = 1$ at about three times the cost of the underlying scalar function value.

The key difference between the two modes of differentiation is that the $\bar{v}_i$ must be computed in the reverse order of the $\dot{v}_i$ and v_i. Hence, in some sense, the evaluation code must first be executed forward and then backward, or reverse. Unless the original procedure is a single assignment code as we have tacitly assumed in the example, this means that most intermediate values v_i must be somehow recovered for the calculation of partial derivatives on the way back. The simplest way of doing that is to simply store the "trajectory" of all intermediate values on the way forward and then to recover them on the way back. This process is displayed schematically in the following graph.



From the user supplied source code for $y = F(x)$ the AD tool generates in one way or another the object code on the left that also evaluates $\dot{y} = F(x)'\dot{x}$ in the forward mode. If also used as part of a reverse differentiation process the forward evaluation sweep must write all or most many intermediate values v_i to some sequential storage device. The object code on the right executing the reverse sweep will then pick up the values v_i in exactly opposite order to compute the adjoint vector $\bar{x} = \bar{y}F(x)'$. If the directional derivatives $\dot{v}_i$ are also saved during the forward sweep they van be used in the reverse sweep to evaluate "second order adjoints" of the form $\dot{\bar{x}} = \bar{y}\,F''\dot{x}$. The total operations count is still only a small multiple of the underlying function and they can be extremely useful in the context of Newton variants for constraint optimization. Even though the forward and reverse mode are rather special choices in a much larger class of derivative accumulation schemes they are likely to remain the main stay of AD for some time to come. Local variations on the statement and subroutine level are certainly promising and have been incorporated for example in ADIFOR [3]. Also very important are adaptions for detectiong and exploiting sparsity and other structure in Jacobians, which can reduce the cost of their evaluation by orders of magnitude.

6. References

1 Balay, S., Gropp, W., Curfman McInnes, L., Smith, B.: PETSc 2.0 Users Manual. Argonne National Laboratory, ANL-95/11 - Revision 2.0.29, 2000.

2 Berz, M.: Algorithms for higher derivatives in many variables with applications to bear physics. In George F. Corliss and Andreas Griewank, editors, Automatic Differentiation of Algorithms - Theory, Implementation, and Applications, SIAM (1991), 147 – 156.

3 Bischof, C., Carle, A., Khademi, P., Mauer, A., and Hovland, P.: ADIFOR 2.0 User's Guide. Argonne National Laboratory, ANL/MCS-TM-192, 1994.

4 Brook, A., Kendrick, D., and Meeraus, A.: GAMS: A User's Guide". The Scientific Press, Redwood City CA, 1988.

5 Campbell, S.L.: A computational method for general higher index nonlinear singular systems of differential equations. IMACS Trans. Sci. Comp. **1.2** (1989), 555 – 560.

6 Chang, Y.F. and Corliss, G.F.: ATOMFT: Solving ODEs and DAEs using Taylor series. Comput. Math. Appl. **28** (1994), 209–233.

7 Czyzyk, J., Mesnier, M., and Moré, J.: The NEOS Server. IEEE Journal on Computational Science and Engineering, 5 (1998), 68 – 75,

8 Enright, W.H. and Pryce, John D.: Two FORTRAN Packages for Assessing Initial Value Methods. ACM Trans. Math. Software 13, 1 – 22, 1987.

9 Fourer, R., Gay, D.M., and Kernighan, B.W.: AMPL: A Mathematical Programming Language. Computing Science Tech. Report 133, AT&T Bell Laboratories, Murray Hill, NJ, June 1989.

10 Govaerts, W.: Bordered matrices and singularities of large nonlinear systems. Int. J. Bifurcation Chaos Appl. Sci. Eng. 5, 243 – 250 (1995).

11 Griewank, A.: Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation, Frontiers in Appl. Math. 19, Phil., 2000.

12 Griewank, A. and Faure, C.: Reduced Functions, Gradients and Hessians from Fixed Point Iteration for State Equations. Numerical Algorithms 30(2): 113-139. Kluwer Academic Publishers, 2002.

13 Griewank, A. and Naumann, U.: Accumulating Jacobians by Vertex, Edge or Face Elimination. Technical Report Preprint IOKOMO-01-2002, TU Dresden, 2002. To appear in ICIAM 2002.

14 Speelpenning, B.: Compiling fast partial derivatives of functions given by algorithms. Ph.D. thesis, University of Illinois at Urbana-Champaign, 1980.

15 Vogel, O., Griewank, and Bär, G.: Direct gear tooth contact analysis for hypoid bevel gears. Computer Methods in Applied Mechanics and Engineering, Elsevier Publishers, vol 191/36, 3965 – 3982, 2002.

16 Walther, A. and Lehmann, U.: Adjoint calculation using time-minimal program reversals for multi-processor machines. Technical Report IOKOMO-09-2001, Techn. Univ. Dresden. Accepted for Publication in Proceeding of IFIP Converence 2001.

17 Wanner, G.: Integration gewöhnlicher Differentialgleichnugen, Lie Reihen,Runge-Kutta-Methoden. Vol. XI, B.I-Hochschulskripten, 831/831a, Bibliographisches Institut, Mannheim-Zürich, Germany, 1969.

Andreas Griewank, Institute of Scientific Computing, Technical University Dresden
Andrea Walther, Institute of Scientific Computing, Technical University Dresden